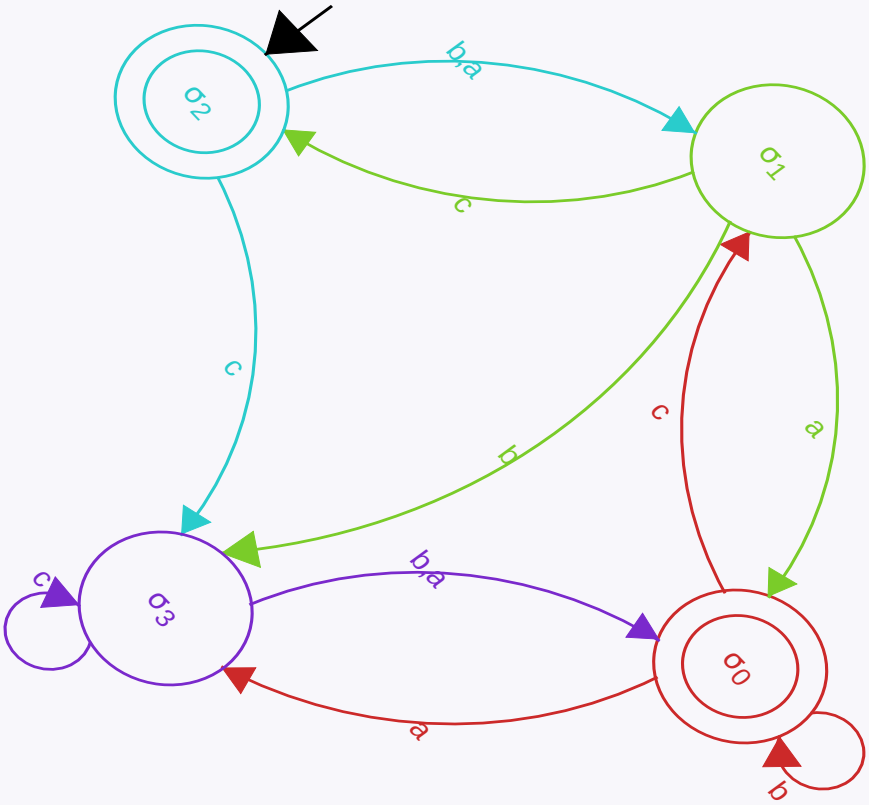
2018
Revista digital
Matemática Educación *e Internet*
https://tecdigital.tec.ac.cr/revistamatematica

Matemática discreta a través del uso del

Paquete Vilcretas

Enrique Vílchez Q.



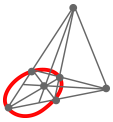
Enrique Vílchez Q.

Matemática discreta a través del uso del

Paquete Vilcretas

Wolfram Mathematica

1ra Edición, 2018



Revista digital

Matemática, Educación e Internet. (https://tecdigital.tec.ac.cr/revistamatematica/).

Correo Electrónico: wmora2@itcr.ac.cr
Escuela de Matemática
Instituto Tecnológico de Costa Rica
Apdo. 159-7050, Cartago
Teléfono: (506) 25502225
Fax: (506) 25502493

Vílchez Quesada, Enrique.
Matemática discreta a través del uso del Paquete *VilCretas*/Enrique Vílchez Quesada – 1ra ed.
– Escuela de Informática, Universidad Nacional, Costa Rica. 2018.
521 p.
ISBN 978-9930-541-16-6
1. Matemática discreta. 2. *Wolfram Mathematica*. 3. *VilCretas*.

Licencia.

Revista digital
Matemática, Educación e Internet.
https://tecdigital.tec.ac.cr/revistamatematica/.

Índice general

Prólogo v

1 Instalación del paquete *VilCretas* Página 1

2 Comandos generales Página 2

3 Recursividad con *VilCretas* Página 4

4 Relaciones de recurrencia con *VilCretas* Página 22

5 Análisis de algoritmos con *VilCretas* Página 38

6 Relaciones binarias con *VilCretas* Página 58

7 Teoría de grafos con *VilCretas* Página 134

8 Teoría de árboles con *VilCretas* Página 348

9 Máquinas y autómatas de estado finito con *VilCretas* Página 437

10 Gramáticas y lenguajes con *VilCretas* Página 496

Bibliografía 516

Prólogo

El uso de software y tecnología para la enseñanza y el aprendizaje en la educación superior, se ha convertido en los últimos años en un pilar fundamental dentro del currículum de las carreras a nivel universitario. Mesa (2012) justifica este hecho al admitir: "las tecnologías de información y comunicación (TIC) son sinónimo de modernización, calidad, productividad, mejores servicios y apoyo a los procesos educativos" (p. 1). La inserción de la informática en la universidad, debe ser entendida como un reto de formación profesional ante la demanda de un mercado cada vez más competitivo. Bournissen (2012) así lo expone: "existe hoy la necesidad de formar a nuestros alumnos para que cuando se inserten en el mercado laboral lo hagan conociendo las nuevas tecnologías existentes" (p. 1). Gutiérrez y Tyner (2012) con una orientación de pensamiento más integradora, atribuyen al fenómeno de las TIC formas de codificación y estructuración de la información tan imprescindibles, como lo fue en su momento la incorporación de la lectoescritura en el currículo escolar.

Las universidades más importantes tanto a nivel nacional como internacional, actualmente reconocen en sus modelos pedagógicos el insumo tecnológico como un elemento medular en la docencia. A este respecto en el documento "Modelo Pedagógico" (s.a.), la Universidad Nacional de Costa Rica (UNA) circunscribe dentro de los procesos de enseñanza y aprendizaje el uso de las tecnologías de información y comunicación, como un aspecto consistente con las demandas de un nuevo tipo de estudiante y más aún, un nuevo tipo de modelo universitario. La UNA en este sentido manifiesta refiriéndose a las TIC: "constituyen un agente de cambio que incide en el trabajo pedagógico y en las relaciones educando-educador y educando-educando" (p. 7).

Pese a estos enfoques de reforma y el análisis mediador que conduciría en el mejor de los casos a transformar las prácticas de enseñanza en la educación superior, la realidad histórica sobre la marcha de alternativas múltiples es testigo de la confusión pedagógica que provee la aceleración continua de los avances tecnológicos y el error de centrar la didáctica en la tecnología como un fin en sí misma. La verdad del engaño puede trastornar las mejores intenciones formativas en fracasos rotundos. En esta dirección, es preciso reconocer que no todo software puede ser empleado con fines educativos. A este respecto Peña (2010), bien lo apunta: "en cuanto a software: no sólo lo constituye lo que hace que los ordenadores corran, se extiende a aquello donde la clase va mucho más allá de sus paredes y sus cursos" (p. 3), exaltando con ello, la necesidad de valorar en el ámbito pedagógico, aquellas aplicaciones informáticas que brinden un valor agregado hacia el desarrollo de habilidades y competencias digitales. Bajo esta perspectiva, en el contexto del presente libro y los resultados obtenidos por Vílchez y González (2014), se reconoce en el programa comercial *Wolfram Mathematica* una poderosa herramienta de investigación científica que puede contribuir con la generación de aprendizajes más profundos, aproximaciones más adecuadas a los intereses disciplinarios del alumno y al alcance de un mayor nivel de motivación en correlación directa con la solución de problemas más significativos. *Mathematica* es un software que, dentro de un planeamiento didáctico consistente, puede contribuir con el mejoramiento de los procesos educativos. La mayor dificultad en esta acepción, no reside en las potencialidades técnicas de *Mathematica*, sino en sus características semánticas. Vílchez y González (2014) lo recalcan al recomendar: "no enfatizar tanto en el lenguaje de programación de la herramienta" (p. 11), al referirse al empleo de *Mathematica* como un apoyo en la docencia.

La creación del paquete *VilCretas* facilita comandos de uso rápido y directo con fines pedagógicos y se vislumbra como un medio que solventa el obstáculo del avance efectivo en clase, en un área de conocimiento con poco desarrollo de software, como lo es la matemática finita. Este paquete entendido como un software didáctico, se clasifica en la tipología expuesta por Couturejuzón (2003), como interactivo, al pretender constituirse en un puente entre conocimientos de naturaliza abstracta y el empleo de manipulaciones concretas para el análisis conceptual y procedimental.

Otro factor determinante como un juicio de valor sobre el paquete *VilCretas*, se basa en la creación de la herramienta dentro del contexto de un ambiente didáctico específico, dirigido a una población concreta con características muy particulares, como lo es la cátedra de profesores y alumnos de un curso introductorio de matemática discreta. El desarrollo de un paquete de software en escenarios claramente preconcebidos, constituye para algunos investigadores un indicador muy positivo sobre sus posibles alcances. Cañizares, Febles y Estrada (2012) advierten en este rumbo que: "una tecnología puede tener éxito en un contexto y no así en otros … Una solución viable es … realizar desarrollos propios que sí satisfagan las necesidades presentes" (p. 103).

VilCretas además, intenta sistematizar una metodología asistida por computadora como un recurso didáctico a través del uso del software *Wolfram Mathematica*, que busca sublevar el privilegio de la abstracción en el campo discreto, por una "matemática viva puesta en escena con el desarrollo del pensamiento crítico" (Rodríguez, 2013, p. 2).

El software didáctico y las TIC tienen en este sentido mucho que aportar en el inhóspito camino de la originalidad pedagógica. Serrano y Narváez (2010) así lo confirman: "el vertiginoso avance de las Tecnologías de la Información y la Comunicación … ha servido como referente para impulsar novedosas formas de apoyar los procesos de transformación de contenidos" (p. 2). Este es el principio base del cual parte esta propuesta didáctica: buscar nuevas formas de hacer educación y esencialmente educación matemática, facilitando mediante el uso de software y en este caso el paquete *VilCretas*, nuevas formas de interpretación, análisis y representación del conocimiento.

Heredia, 2018. E. VÍLCHEZ

Capítulo 1

Instalación del paquete *VilCretas*

VilCretas es un paquete elaborado por el autor de este libro que añade al software comercial *Wolfram Mathematica*, doscientos treinta comandos para desarrollar distintos procedimientos vinculados con un curso introductorio de matemática discreta. En general, las instrucciones integradas en *VilCretas* abordan áreas de contenido vinculadas con: recursividad, relaciones de recurrencia, análisis de algoritmos, relaciones binarias, teoría de grafos, teoría de árboles, máquinas y autómatas de estado finito y, gramáticas y lenguajes. El trabajo es producto de un proyecto de investigación adscrito a la Escuela de Informática de la Universidad Nacional de Costa Rica.
VilCretas se ha creado para versiones de *Mathematica* superiores o iguales a la 9. Para instalar dicho paquete se accede al menú en el software *Mathematica*: *Archivo/Instalar*. Lo anterior, despliega la siguiente ventana:

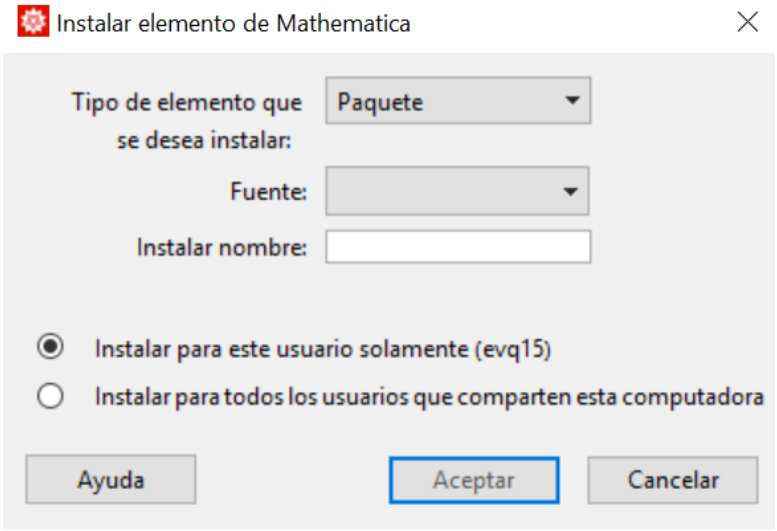


Figura 1: Instalación de *VilCretas*

La opción *Fuente* en la figura 1 permite direccionar el archivo de la librería que debe estar descargado localmente en la computadora del usuario (en el sitio Web: `http://www.escinf.una.ac.cr/discretas/index.php/archivos/category/7-packages` se encuentra disponible la última versión de *VilCretas*). Se recomienda en *Instalar nombre* dejar por defecto el nombre *VilCretas*.
Si se desea levantar el paquete en un *cuaderno* de *Mathematica* se utiliza la instrucción: <<**`VilCretas`**`.
Esto le permite al profesor o al estudiante tener acceso a todas las funciones incluidas en la biblioteca.
El código fuente de los ejemplos expuestos en este libro se encuentra disponible en el vínculo: "*Ejemplos de uso del paquete VilCretas por comando*" publicado en la *URL* donde se aloja *VilCretas*.

Explicación en video

Capítulo 2

Comandos generales

En este capítulo se introducen **dos** comandos de descripción general, sobre los objetivos de creación del paquete *VilCretas* y el autor de esta librería. Las instrucciones preveen recursos con la intención de describir las áreas de contenido que abarca el uso de *VilCretas* y la afiliación de su autor.

1) **VilCretasPackage**: presenta una **descripción general** del contenido del paquete *VilCretas* y los **objetivos** de su creación.

Sintaxis: `VilCretasPackage[]`.

Ejemplo 2.1

Ejecute la instrucción `VilCretasPackage[]`

Solución:

En *Mathematica*:

In[ ] :=
```
VilCretasPackage[]
```

Out[ ] :=
El paquete “VilCretas” se ha desarrollado con la principal intención de trabajar desde un punto de vista educativo en el campo de la matemática discreta, utilizando como apoyo el conocido software comercial “ Mathematica”. “VilCretas” está constituido por una serie de funciones que por defecto no están integradas en “Mathematica”, abarcando las principales áreas de contenido de un curso clásico de estructuras discretas, particularmente: recursividad, relaciones de recurrencia, análisis de algoritmos, relaciones binarias, teoría de grafos, teoría de árboles, máquinas y autómatas de estado finito y, gramáticas y lenguajes. Desde un punto de vista didáctico, el paquete puede ser empleado como una herramienta de verificación de resultados, o bien, como un medio para profundizar el ambiente de programación del software. “VilCretas” incorpora métodos de solución automatizados en un campo de conocimiento muchas veces catalogado como abstracto.

El comando brinda una breve descripción sobre los objetivos y funciones que caracterizan a *VilCretas*.

Explicación en video

2) **AutorVilCretas**: muestra los **datos** del **autor** del paquete *VilCretas*.

Sintaxis: `AutorVilCretas[]`.

Ejemplo 2.2

Ejecute el comando **`AutorVilCretas[]`**.

Solución:

En el software:

In[ ] :=

```
AutorVilCretas[]
```

Out[ ] :=

Autor: Enrique Vílchez Quesada
Afiliación: Escuela de Informática de la Universidad Nacional de Costa Rica
Puesto: Profesor catedrático
Correo electrónico: enrique.vilchez.quesada@una.cr
Para mayor información se sugiere consultar el libro:
Vílchez, E. (2015). Estructuras discretas con Mathematica. México: Editorial Alfaomega.
Sitios Web de apoyo:
http://www.escinf.una.ac.cr/discretas
http://www.alfaomega.com.mx/default/estructuras-discretas-con-mathematica-6344.html

Como se observa la instrucción despliega la información personal del autor del paquete.

Explicación en video

Capítulo 3

Recursividad con *VilCretas*

En el presente capítulo se compartirán **trece** instrucciones incluidas en *VilCretas* para trabajar en el área de contenido de recursividad. El tema de recursividad se concibe como un lenguaje de naturaleza científica y una técnica de programación. Desde este punto de vista, los comandos de *VilCretas* suministran herramientas para comprender la recursividad como técnica de programación y analizar diferentes ejemplos típicos de recursividades normales y de cola.

1) **Factoriales**: calcula de forma recursiva el **factorial** de un número **natural** o **cero**. Presenta dos opciones: "code" y "steps", "code" con un valor lógico "`True`" muestra el **código** de la función recursiva que realiza el cálculo y "**`steps ->` `True`**"arroja **paso a paso** las iteraciones que recorre la recursividad.

Sintaxis: **`Factoriales[n]`**, o bien, **`Factoriales[n, code ->Valor, steps ->Valor]`**.

Ejemplo 3.1

Calcule 20! y muestre el procedimiento recursivo paso a paso.

Solución:

En *Mathematica*:

```
In[ ] :=
Factoriales[20, steps -> True]

Out[ ] :=
Factoriales[20] = 20 * Factoriales[19] = 2432902008176640000
Factoriales[19] = 19 * Factoriales[18] = 121645100408832000
Factoriales[18] = 18 * Factoriales[17] = 6402373705728000
Factoriales[17] = 17 * Factoriales[16] = 355687428096000
Factoriales[16] = 16 * Factoriales[15] = 20922789888000
Factoriales[15] = 15 * Factoriales[14] = 1307674368000
Factoriales[14] = 14 * Factoriales[13] = 87178291200
Factoriales[13] = 13 * Factoriales[12] = 6227020800
Factoriales[12] = 12 * Factoriales[11] = 479001600
Factoriales[11] = 11 * Factoriales[10] = 39916800
Factoriales[10] = 10 * Factoriales[9] = 3628800
Factoriales[9] = 9 * Factoriales[8] = 362880
Factoriales[8] = 8 * Factoriales[7] = 40320
Factoriales[7] = 7 * Factoriales[6] = 5040
Factoriales[6] = 6 * Factoriales[5] = 720
Factoriales[5] = 5 * Factoriales[4] = 120
Factoriales[4] = 4 * Factoriales[3] = 24
Factoriales[3] = 3 * Factoriales[2] = 6
Factoriales[2] = 2 * Factoriales[1] = 2
Factoriales[1] = 1
```

```
2432902008176640000
```

Ejemplo 3.2

Calcule 20!, muestre el procedimiento paso a paso y el código de la función que permite obtener el resultado.

Solución:

```
In[ ] :=
Factoriales[20, code -> True, steps -> True]

Out[ ] :=
{2432902008176640000,Factoriales[n_]:=If[Or[n==0, n==1], Return[1], n*Factoriales[n-1]]}
Factoriales[20] = 20 * Factoriales[19] = 2432902008176640000
Factoriales[19] = 19 * Factoriales[18] = 121645100408832000
Factoriales[17] = 17 * Factoriales[16] = 355687428096000
⋮
Factoriales[3] = 3 * Factoriales[2] = 6
Factoriales[2] = 2 * Factoriales[1] = 2
Factoriales[1] = 1
2432902008176640000
```

Explicación en video

2) **NFibonacci**: encuentra de forma recursiva un **número** de la sucesión de ***Fibonacci,*** además a través de la opción "**`code -> True`**" muestra el **código** de *Mathematica* que realiza el cálculo y al emplear "**`steps -> True`**", ejecuta **paso a paso** las iteraciones de la recursividad.

Sintaxis: **`NFibonacci[n]`**, o bien, **`NFibonacci[n, code -> Valor, steps -> Valor]`**.

Ejemplo 3.3

Halle el número 20 de *Fibonacci* donde se muestre el código de la función recursiva que realiza el cálculo.

Solución:

Al emplear el comando **`NFibonacci`** se obtiene:

```
In[ ] :=
NFibonacci[20, code -> True]

Out[ ] :=
{6765, NFibonacci[n_]:=If[Or[n==1, n==2], Return[1], NFibonacci[n-1]+NFibonacci[n-2]]}
```

Ejemplo 3.4

Encuentre el número 20 de *Fibonacci* donde se muestre el procedimiento paso a paso y el código del programa recursivo.

Solución:

En el software:

In[ ] :=

```
NFibonacci[20, code->True, steps->True]
```

Out[ ] :=

{6765, NFibonacci[n_]:=If[Or[n==1, n==2], Return[1], NFibonacci[n-1]+NFibonacci[n-2]]}
NFibonacci[20] = NFibonacci[19] + NFibonacci[18] = 4181 + 2584 = 6765
NFibonacci[19] = NFibonacci[18] + NFibonacci[17] = 2584 + 1597 = 4181
⋮
NFibonacci[3] = NFibonacci[2] + NFibonacci[1] = 1 + 1 = 2
NFibonacci[2] = 1
NFibonacci[1] = 1
6765

Explicación en video

3) **PotenciaPos**: determina de manera recursiva la **potencia** "n" de un **número real** "a", siendo el exponente "n" un **número natural** o **cero**. La función permite visualizar los resultados devueltos **paso a paso** y el **código** de programación.

Sintaxis: `PotenciaPos[a, n]`, o bien, `PotenciaPos[a, n, code->Valor, steps->Valor]`, "`code->True`" genera el contenido de la función y "`steps->True`" las iteraciones invocación a invocación.

Ejemplo 3.5

Calcule mediante un procedimiento recursivo 6^{10}. Muestre el código de la función.

Solución:

En *Mathematica*:

In[ ] :=

```
PotenciaPos[6, 10, code->True]
```

Out[ ] :=

{60466176, Potencia[a_,n_]:=If[And[a==0, n==0], Return["Indefinido"], If[a==0, Return[0], If[n==0, Return[1], a*Potencia[a,n-1]]]]}

Ejemplo 3.6

Determine de forma recursiva paso a paso 6^{10}.

Solución:

Al emplear la opción "steps":

In[ ] :=

```
PotenciaPos[6, 10, steps -> True]
```

Out[ ] :=

```
PotenciaPos[6, 10] = 6*PotenciaPos[6, 9] = 60466176
PotenciaPos[6, 9] = 6*PotenciaPos[6, 8] = 10077696
PotenciaPos[6, 8] = 6*PotenciaPos[6, 7] = 1679616
⋮
PotenciaPos[6, 2] = 6*PotenciaPos[6, 1] = 36
PotenciaPos[6, 1] = 6*PotenciaPos[6, 0] = 6
PotenciaPos[6, 0] = 1
60466176
```

Explicación en video

4) **PotenciaNeg**: encuentra de manera recursiva la **potencia** "n" de un **número real** "a", siendo el exponente "n" un **número entero negativo** o **cero**. El comando permite visualizar los resultados devueltos **paso a paso** y el **código** de programación.

Sintaxis: `PotenciaNeg[a, n]`, o bien, `PotenciaNeg[a, n, code -> Valor, steps -> Valor]`, `code -> True` retorna el contenido de la función y `steps -> True` muestra las iteraciones una a una.

Ejemplo 3.7

Determine 6^{-10} mediante el uso de un programa recursivo. Utilice la opción "code" para mostrar el código interno.

Solución:

In[ ] :=

```
PotenciaNeg[6, -10, code -> True]
```

Out[ ] :=

```
{1/60466176, Potencia[a_,n_]:=If[And[a==0, n==0], Return["Indefinido"], If[a==0, Return["Indefinido"],
If[n==0, Return[1], 1/a*Potencia[a,n+1]]]]}
```

Ejemplo 3.8

Resuelva el ejemplo anterior mostrando paso a paso la solución recursiva.

Solución:

In[ ] :=

```
PotenciaNeg[6, -10, code -> True, steps -> True]
```

Out[ ] :=

{1/60466176, Potencia[a_,n_]:=If[And[a==0, n==0], Return[“Indefinido”], If[a==0, Return[“Indefinido”], If[n==0, Return[1], 1/a*Potencia[a,n+1]]]]}
PotenciaNeg[6, -10] = 1/6*PotenciaNeg[6, -9] = 1/60466176
PotenciaNeg[6, -9] = 1/6*PotenciaNeg[6, -8] = 1/10077696
PotenciaNeg[6, -8] = 1/6*PotenciaNeg[6, -7] = 1/1679616
⋮
PotenciaNeg[6, -2] = 1/6*PotenciaNeg[6, -1] = 1/36
PotenciaNeg[6, -1] = 1/6*PotenciaNeg[6, 0] = 1/6
PotenciaNeg[6, 0] = 1
1/60466176

Explicación en video

5) **Dato**: realiza la **búsqueda** de forma **recursiva** de un dato “a” sobre una lista “L”. El comando retorna los resultados **paso a paso** y permite, además, visualizar el **código** que lo conforma mediante el uso de las opciones “`steps -> True`” y “`code -> True`”, respectivamente.

Sintaxis: `Dato[a, L]`, o bien, `Dato[a, L, code -> Valor, steps -> Valor]`.

Ejemplo 3.9

Utilice el comando `Dato` para buscar el elemento 6 en la lista {8, 6, -9, a, -11, 9}.

Solución:

In[ ] :=

```
Dato[6, {8, 6, -9, a, -11, 9}]
```

Out[ ] :=

Dato encontrado

Ejemplo 3.10

Emplee el comando **`Dato`** para buscar el elemento 6 en la lista {8, 6, -9, a, -11, 9}. Genere el procedimiento paso a paso y muestre el código de la función **`Dato`**.

Solución:

En *Mathematica*:

In[ ] :=

```
Dato[6, {8, 6, -9, a, -11, 9}, code -> True, steps -> True]
```

Out[ ] :=

```
{Dato encontrado, Dato[a_,L_List]:=Module[{Lista=L}, If[Lista=={}, "Dato no encontrado",  Vl=Last[Lista];
If[ToString[a]==ToString[Vl], "Dato encontrado",  Lista=Delete[Lista,Length[Lista]]; Dato[a,Lista]]]]}
Dato[6, {8, 6, -9, a, -11, 9}]
Dato[6, {8, 6, -9, a, -11}]
Dato[6, {8, 6, -9, a}]
Dato[6, {8, 6, -9}]
Dato[6, {8, 6}]
Dato encontrado
```

Explicación en video

6) **MCD**: calcula el **máximo común divisor** entre **dos números enteros positivos** o **cero** usando una lógica **recursiva**. A través de esta instrucción el resultado se visualiza **iteración por iteración** y permite retornar el **código** de *Mathematica* por medio del cual se programó. Las opciones "**`code -> True`**" y "**`steps -> True`**" respectivamente, muestran el código y las ejecuciones paso a paso.

Sintaxis: **`MCD[n, m]`**, o bien, **`MCD[n, m, code -> Valor, steps -> Valor]`**.

Ejemplo 3.11

Encuentre de forma recursiva paso a paso el máximo común divisor entre 680 y 240.

Solución:

En el software:

In[ ] :=

```
MCD[680, 240, steps -> True]
```

Out[ ] :=

```
MCD[680, 240] = MCD[Mod[240, 680], 680] = MCD[240, 680]
MCD[240, 680] = MCD[Mod[680, 240], 240] = MCD[200, 240]
MCD[200, 240] = MCD[Mod[240, 200], 200] = MCD[40, 200]
MCD[40, 200] = MCD[Mod[200, 40], 40] = MCD[0, 40]
```

```
MCD[0, 40] = 40
40
```

N **Mod** en *Mathematica* resuelve la operación módulo.

Ejemplo 3.12

Resuelva el ejemplo anterior mostrando además, el código de la función **MCD**.

Solución:

Para ello se recurre al uso de la opción "code":

```
In[ ] :=
MCD[680, 240, code -> True, steps -> True]

Out[ ] :=
{40, MCD[n_,m_]:=If[n==0, Return[m], MCD[Mod[m,n],n]]}
MCD[680, 240] = MCD[Mod[240, 680], 680] = MCD[240, 680]
MCD[240, 680] = MCD[Mod[680, 240], 240] = MCD[200, 240]
⋮
MCD[200, 240] = MCD[Mod[240, 200], 200] = MCD[40, 200]
MCD[40, 200] = MCD[Mod[200, 40], 40] = MCD[0, 40]
MCD[0, 40] = 40
40
```

Explicación en video

7) **MCM**: calcula el **mínimo común múltiplo** de manera **recursiva** entre **dos números enteros positivos**. Por medio de esta instrucción el resultado se visualiza **paso a paso** y permite retornar el **código** de *Mathematica* a través del cual se programó.

Sintaxis: **MCM[n, m, d]**, o bien, **MCM[n, m, d, code -> Valor, steps -> Valor]**, con "**d=1**" por defecto, "**code->True**" exhibe el código fuente y "**steps -> True**" el procedimiento de ejecución iteración por iteración.

Ejemplo 3.13

Halle con una lógica recursiva el mínimo común múltiplo entre 58855 y 25012. Muestre además, el código de programación.

Solución:

Al utilizar el comando **MCM** se obtiene:

In[ ] :=

```
MCM[58855, 25012, 1, code -> True]
```

Out[ ] :=

```
{1472081260, MCM[n_,m_,d_]:=If[n==1 && m==1, Return[1], on=n; om=m; od=Prime[d]; If[Mod[on,od]==0 || Mod[om,od]==0, If[Mod[on,od]==0 && Mod[om,od]==0, on=on/od; om=om/od, If[Mod[on,od]==0, on=on/od]; If[Mod[om,od]==0, om=om/od]]; Return[od*MCM[on,om,d]], Return[MCM[on,om,d+1]]]]}
```

Ejemplo 3.14

Resuelva paso a paso el cálculo del ejemplo anterior.

Solución:

Para ello se emplea la opción "steps" del comando **MCM**:

In[ ] :=

```
MCM[58855, 25012, 1, steps -> True]
```

Out[ ] :=

```
MCM[58855, 25012, 1] = 2*MCM[58855, 12506, 1] = 1472081260
MCM[58855, 12506, 1] = 2*MCM[58855, 6253, 1] = 736040630
MCM[58855, 6253, 1] = MCM[58855, 6253, 2] = 368020315
⋮
MCM[149, 1, 34] = MCM[149, 1, 35] = 149
MCM[149, 1, 35] = 149*MCM[1, 1, 35] = 149
MCM[1, 1, 35] = 1
1472081260
```

Explicación en video

8) **Multipli**: realiza de forma recursiva la **multiplicación** entre **dos números enteros positivos**. Mediante su uso el usuario puede observar los resultados **paso a paso** y el **código** de programación en el software *Mathematica*, a través de las opciones "**code -> True**" y "**steps -> True**", respectivamente.

Sintaxis: **Multipli[n, m, d]**, o bien, **Multipli[n, m, d, code -> Valor, steps -> Valor]**, con "**d=1**" por defecto.

Ejemplo 3.15

Emplee el comando **`Multipli`** para encontrar el resultado de: 2416 · 36058. Genere la solución iteración a iteración.

Solución:

En el software:

```
In[ ] :=
Multipli[2416, 36058, 1, steps -> True]

Out[ ] :=
Multipli[2416, 36058, 1] = 2*Multipli[1208, 36058, 1] = 87116128
Multipli[1208, 36058, 1] = 2*Multipli[604, 36058, 1] = 43558064
Multipli[604, 36058, 1] = 2*Multipli[302, 36058, 1] = 21779032
Multipli[302, 36058, 1] = 2*Multipli[151, 36058, 1] = 10889516
Multipli[151, 36058, 1] = 2*Multipli[151, 18029, 1] = 5444758
Multipli[151, 18029, 1] = Multipli[151, 18029, 2] = 2722379
Multipli[151, 18029, 2] = Multipli[151, 18029, 3] = 2722379
Multipli[151, 18029, 3] = Multipli[151, 18029, 4] = 2722379
Multipli[151, 18029, 4] = Multipli[151, 18029, 5] = 2722379
Multipli[151, 18029, 5] = 11*Multipli[151, 1639, 5] = 2722379
Multipli[151, 1639, 5] = 11*Multipli[151, 149, 5] = 247489
⋮
Multipli[151, 149, 5] = Multipli[151, 149, 6] = 22499
Multipli[151, 149, 6] = Multipli[151, 149, 7] = 22499
⋮
Multipli[151, 1, 36] = 151*Multipli[1, 1, 36] = 151
Multipli[1, 1, 36] = 1
87116128
```

Ejemplo 3.16

Resuelva el ejemplo anterior mostrando el código del comando **`Multipli`**.

Solución:

```
In[ ] :=
Multipli[2416, 36058, 1, code -> True, steps -> True]

Out[ ] := {87116128, Multipli[n_, m_, d_]:=If[n==1 && m==1, Return[1], on=n; om=m; od=Prime[d];
If[Mod[on, od]==0 || Mod[om, od]==0, If[Mod[on, od]==0, on=on/od, om=om/od]; Return[od*Multipli[on,
om, d]], Return[Multipli[on, om, d + 1]]]]}
Multipli[2416, 36058, 1] = 2*Multipli[1208, 36058, 1] = 87116128
Multipli[1208, 36058, 1] = 2*Multipli[604, 36058, 1] = 43558064
⋮
Multipli[151, 1, 36] = 151*Multipli[1, 1, 36] = 151
Multipli[1, 1, 36] = 1
87116128
```

Explicación en video

9) **SumaDigi**: **suma** de manera **recursiva** los **dígitos** de un **número natural**. El comando facilita el **código** con el que fue programado y permite observar los valores retornados en **cada iteración** mediante el uso de las opciones "**code -> True**" y "**steps -> True**", respectivamente.

Sintaxis: **SumaDigi[n]**, o bien, **SumaDigi[n, code -> Valor, steps -> Valor]**.

Ejemplo 3.17

Sume mediante un programa recursivo los dígitos de: 1980025487. Emplee la opción "code" para mostrar el código de programación.

Solución:

In[ ] :=
```
SumaDigi[1980025487, code -> True]
```

Out[ ] :=
```
{44, SumaDigi[n_]:=Module[{suma=Mod[n,10]}, If[n==0, Return[suma], suma+SumaDigi[(n-Mod[n,10])/10]]]}
```

Ejemplo 3.18

Resuelva el cálculo del ejemplo anterior paso a paso.

Solución:

En el software:

In[ ] :=
```
SumaDigi[1980025487, steps -> True]
```

Out[ ] :=
```
SumaDigi[1980025487] = 7+SumaDigi[(1980025487-Mod[1980025487,10])/10]
= 7+SumaDigi[(1980025487-7)/10] = 44
SumaDigi[198002548] = 8+SumaDigi[(198002548-Mod[198002548,10])/10]
= 8+SumaDigi[(198002548-8)/10] = 37
SumaDigi[19800254] = 4+SumaDigi[(19800254-Mod[19800254,10])/10]
= 4+SumaDigi[(19800254-4)/10] = 29
SumaDigi[1980025] = 5+SumaDigi[(1980025-Mod[1980025,10])/10]
= 5+SumaDigi[(1980025-5)/10] = 25
SumaDigi[198002] = 2+SumaDigi[(198002-Mod[198002,10])/10]
 = 2+SumaDigi[(198002-2)/10] = 20
SumaDigi[19800] = 0+SumaDigi[(19800-Mod[19800,10])/10]
= 0+SumaDigi[(19800-0)/10] = 18
```

```
SumaDigi[1980] = 0+SumaDigi[(1980-Mod[1980,10])/10]
= 0+SumaDigi[(1980-0)/10] = 18
SumaDigi[198] = 8+SumaDigi[(198-Mod[198,10])/10]
= 8+SumaDigi[(198-8)/10] = 18
SumaDigi[19] = 9+SumaDigi[(19-Mod[19,10])/10]
= 9+SumaDigi[(19-9)/10] = 10
SumaDigi[1] = 1+SumaDigi[(1-Mod[1,10])/10]
= 1+SumaDigi[(1-1)/10] = 1
SumaDigi[0] = 0
44
```

Explicación en video

10) **Multiplos**: muestra mediante un recorrido recursivo **todos** los **múltiplos** de un número natural "n" **menor** que otro **número natural** "m", visualizando si el usuario lo desea, los resultados **paso a paso** ("`steps->True`"). El comando retorna el **código** con el cual fue programado mediante la opción "`code->True`".

Sintaxis: `Multiplos[n, m, d]`, o bien, `Multiplos[n, m, d, code->Valor, steps->Valor]`, con "`d=1`" por defecto.

Ejemplo 3.19

Ejecute la función `Multiplos` con los argumentos 6 y 100.

Solución:

En *Mathematica*:

```
In[ ] :=
Multiplos[6, 100, 1]

Out[ ] :=
{6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90, 96}
```

Ejemplo 3.20

Ejecute el comando `Multiplos` con los argumentos 6 y 100, mostrando el código de programación interno y las iteraciones recursivas paso a paso.

Solución:

```
In[ ] :=
Multiplos[6, 100, 1, code->True, steps->True]

Out[ ] :=
```

{{6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90, 96}, Multiplos[n_,m_,d_]:=Module[{i=d}, If[i==1,L={}; If[n*i<m, L=Append[L,n*i]]; i++; Multiplos[n,m,i], If[n*i<m, L=Append[L,n*i]; i++; Multiplos[n,m,i], Return[L]]]]}
Multiplos[6, 100, 1] = {6}
Multiplos[6, 100, 2] = {6, 12}
Multiplos[6, 100, 3] = {6, 12, 18}
Multiplos[6, 100, 4] = {6, 12, 18, 24}
Multiplos[6, 100, 5] = {6, 12, 18, 24, 30}
⋮
Multiplos[6, 100, 14] = {6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84}
Multiplos[6, 100, 15] = {6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90}
Multiplos[6, 100, 16] = {6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90, 96}
{6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90, 96}

Explicación en video

11) **Divisores**: determina mediante un recorrido **recursivo** todos los **divisores** de un **número natural** "n". El comando retorna el **código** con el cual fue programado mediante la opción "**`code -> True`**" y permite visualizar los resultados **paso a paso** a través de "**`steps -> True`**".

Sintaxis: **`Divisores[n, d]`**, o bien, **`Divisores[n, d, code -> Valor, steps -> Valor]`**, con "**`d=1`**" por defecto.

Ejemplo 3.21

Utilice el comando **`Divisores`** para obtener todos los divisores de 246. Muestre el código interno de esta función.

Solución:

En *Mathematica*:

In[ ] :=
```
Divisores[246, 1, code -> True]
```

Out[ ] :=
{{1, 2, 3, 6, 41, 82, 123, 246}, Divisores[n_,d_]:=Module[{i=d}, If[i==1, L={}; L=Append[L,i]; i++; Divisores[n,i], If[i<=Floor[n/2]+1, If[Mod[n,i]==0, L=Append[L,i]; i++; Divisores[n,i], i++; Divisores[n,i]], L=Append[L,n]; Return[L]]]]}

Ejemplo 3.22

Encuentre de forma recursiva los divisores de 246 y ejecute el procedimiento paso a paso.

Solución:

In[ ] :=
```
Divisores[246, 1, steps->True]
```

Out[ ] :=
Divisores[246, 1] = {1}
Divisores[246, 2] = {1, 2}
Divisores[246, 3] = {1, 2, 3}
Divisores[246, 6] = {1, 2, 3, 6}
Divisores[246, 41] = {1, 2, 3, 6, 41}
Divisores[246, 82] = {1, 2, 3, 6, 41, 82}
Divisores[246, 123] = {1, 2, 3, 6, 41, 82, 123}
{1, 2, 3, 6, 41, 82, 123, 246}

Explicación en video

12) **Sumatoria**: construye de manera **automática** un **programa recursivo** para calcular una **sumatoria** ingresada por el usuario, la instrucción además, facilita el cálculo **paso a paso** para un valor de "n" (índice superior de la suma) especificado como parámetro.

Sintaxis: `Sumatoria[suma, n]`, suma es un vector con tres componentes en el siguiente orden: el **índice inferior**, el **índice superior** (en función de "n") y la **expresión interna** de la sumatoria (en función de "i").

Ejemplo 3.23

Genere un programa recursivo para calcular $\sum_{i=3}^{n-6} \frac{i^2}{i^3+1}$. Evalué además, el programa en $n = 26$.

Solución:

Como el índice superior de la sumatoria es $n-6$, al evaluar en $n = 26$, dicho índice queda igual a 20. En *Mathematica*:

In[ ] :=
```
Sumatoria[{3, n - 6, i^2/(i^3 + 1)}, 20]
```

Out[ ] :=
Sumatoria[n_]:=If[n==9, 9/28, Sumatoria[n-1]+(-6+n)^2/(1+(-6+n)^3)]
Sumatoria[20] = Sumatoria[19] + $\frac{196}{2745}$

$$= \frac{175282134242834789}{101180684916356760}$$

$$\text{Sumatoria}[19] = \text{Sumatoria}[18] + \frac{169}{2198}$$

$$= \frac{2755042218358121}{1658699752727160}$$

$$\text{Sumatoria}[18] = \text{Sumatoria}[17] + \frac{144}{1729}$$

$$= \frac{117150037278191}{73954766045160}$$

$$\text{Sumatoria}[17] = \text{Sumatoria}[16] + \frac{121}{1332}$$

$$= \frac{110990704536431}{73954766045160}$$

$$\text{Sumatoria}[16] = \text{Sumatoria}[15] + \frac{100}{1001}$$

$$= \frac{2818178146073}{1998777460680}$$

$$\text{Sumatoria}[15] = \text{Sumatoria}[14] + \frac{81}{730}$$

$$= \frac{238045461643}{181707041880}$$

$$\text{Sumatoria}[14] = \text{Sumatoria}[13] + \frac{64}{513}$$

$$= \frac{2984704759}{2489137560}$$

$$\text{Sumatoria}[13] = \text{Sumatoria}[12] + \frac{49}{344}$$

$$= \frac{46915247}{43669080}$$

$$\text{Sumatoria}[12] = \text{Sumatoria}[11] + \frac{36}{217}$$

$$= \frac{473197}{507780}$$

$$\text{Sumatoria}[11] = \text{Sumatoria}[10] + \frac{25}{126}$$

$$= \frac{12547}{16380}$$

$$\text{Sumatoria}[10] = \text{Sumatoria}[9] + \frac{16}{65}$$

$$= \frac{1033}{1820}$$

$$\text{Sumatoria}[9] = \frac{9}{28}$$

$$\frac{175282134242834789}{101180684916356760}$$

Ejemplo 3.24

Construya un programa recursivo para calcular $\sum_{i=2}^{n+2} \frac{i - i^2}{\text{sen}\,(i) + 1}$, con $n = 3$.

Solución:

En este ejemplo, al evaluar $n = 3$ en el índice superior de la sumatoria da como resultado 5, luego:

In[ ] :=
```
Sumatoria[{2, n + 2, (i - i^2)/(Sin[i] + 1)}, 5]
```

Out[ ] :=
```
Sumatoria[n_]:=If[n==0, -(2/(1+Sin[2])), Sumatoria[n-1] + (2+n-(2+n)^2) / (1+Sin[2+n])]
Sumatoria[5] = Sumatoria[4] + -(42/(1+Sin[7]))
= -(2/(1+Sin[2]))-6/(1+Sin[3])-12/(1+Sin[4])-20/(1+Sin[5])-30/(1+Sin[6])-42/(1+Sin [7])
Sumatoria[4] = Sumatoria[3] + -(30/(1+Sin[6]))
= -(2/(1+Sin[2]))-6/(1+Sin[3])-12/(1+Sin[4])-20/(1+Sin[5])-30/(1+Sin[6])
Sumatoria[3] = Sumatoria[2] + -(20/(1+Sin[5]))
= -(2/(1+Sin[2]))-6/(1+Sin[3])-12/(1+Sin[4])-20/(1+Sin[5])
Sumatoria[2] = Sumatoria[1] + -(12/(1+Sin[4]))
= -(2/(1+Sin[2]))-6/(1+Sin[3])-12/(1+Sin[4])
Sumatoria[1] = Sumatoria[0] + -(6/(1+Sin[3]))
 = -(2/(1+Sin[2]))-6/(1+Sin[3])
Sumatoria[0] = -(2/(1+Sin[2]))
-(2/(1 + Sin[2])) - 6/(1 + Sin[3]) - 12/(1 + Sin[4]) - 20/(1 + Sin[5])  - 30/(1 + Sin[6]) - 42/(1 + Sin[7])
```

Explicación en video

13) **Quicksort**: implementación del algoritmo "**quicksort**" desarrollado por el científico británico **C.A.R Hoare** en 1960 (ganador del premio *Turing* en 1980), para **ordenar** de forma **recursiva** una lista "L" con "n" datos. La instrucción permite visualizar el **código** de programación del comando y las ejecuciones **paso a paso**, mediante el uso de "**code -> True**" y "**steps -> True**", respectivamente. En adición, la opción "**ascendente->True**" **ordena** la lista de forma **ascendente** y "**ascendente->False**" de manera **descendente**.

Sintaxis: **Quicksort[begin, end, L]**, o bien, **Quicksort[begin, end, L, code -> Valor, steps -> Valor, ascendente->Valor]**, con "**begin=1**" y "**end=longitud de la lista**" por defecto.

Ejemplo 3.25

Emplee el algoritmo **quicksort** para ordenar paso a paso la lista: {9, -7, 5, 4, 10, -8, 10, 1}.

Solución:

In[ ] :=
```
Quicksort[1, 8, {9, -7, 5, 4, 10, -8, 10, 1}, steps -> True]
```

Out[ ] :=
```
Quicksort[1, 8] = {9, -7, 5, 4, 10, -8, 10, 1}
```

⋮
Quicksort[1, 3] = {1, -7, -8}
⋮
Quicksort[3, 8] = {1, 4, 10, 5, 10, 9}
⋮
Quicksort[3, 6] = {1, 4, 9, 5}
⋮
Quicksort[5, 6] = {9, 5}
⋮
Quicksort[6, 8] = {9, 10, 10}
⋮
Quicksort[6, 7] = {9, 10}
⋮
Quicksort[7, 8] = {10, 10}
⋮
L = {-8, -7, 1, 4, 5, 9, 10, 10}
{-8, -7, 1, 4, 5, 9, 10, 10}

Ejemplo 3.26

Resuelva el ejemplo anterior mostrando además, el código de implementación de la función **`Quicksort`** y un orden descendente.

Solución:

En *Mathematica*:

In[ ] :=

```
Quicksort[1, 8, {9, -7, 5, 4, 10, -8, 10, 1}, code->True,
steps->True, ascendente->False]
```

Out[ ] :=

⋮
{10, 10, 9, 5, 4, 1, -7, -8}
{{10, 10, 9, 5, 4, 1, -7, -8}, Quicksort[begin_,end_]:=Module[{}, If[begin<end, i=begin; j=end; piv=L[[Floor[(begin+end)/2]]]; While[i<=j, While[L[[i]]piv, i++]; While[L[[j]]<piv,j–]; If[i<=j, aux=L[[i]]; L[[i]]=L[[j]]; L[[j]]=aux; i++; j–]]; Print[ReplacePart[L,Style[L[[Floor[(begin+end)/2]]], Blue], Floor[(begin+end)/2]]]; If[begin<j, Quicksort[begin,j]]; If[i<end, Quicksort[i, end]]]; L]}
Quicksort[1, 8] = {9, -7, 5, 4, 10, -8, 10, 1}
⋮
Quicksort[1, 4] = {9, 10, 5, 10}
⋮
{10, 10, 9, 5, 4, 1, -7, -8}

Explicación en video

Aporte pedagógico

Los comandos expuestos en esta sección tienen un enfoque **didáctico** al permitirle al estudiante y al profesor observar el **código** de programación de las distintas funciones y las ejecuciones **paso a paso**. Los procedimientos iteración a iteración son esenciales en este tema, pues brindan la posibilidad a los alumnos de interiorizar la forma en **cómo** se realiza un **proceso recursivo** en cualquier ambiente de programación.

Aporte de investigación

Se sugiere emplear estas funciones para comparar **tiempos de ejecución** esto con la finalidad de comprender que ciertos problemas resueltos con un enfoque recursivo, no siempre ofrecen buenos **tiempos de salida**. Para ello, el estudiante puede recurrir al empleo de la función **`Timing`** en *Mathematica*.

Ejercicios

Resuelva los siguientes ejercicios utilizando como apoyo el paquete *VilCretas*.

1) Halle 15! paso a paso usando un razonamiento recursivo.
2) Modifique el código de programación del comando **`NFibonacci`** para obtener una función recursiva que calcule cada uno de los elementos de la sucesión de *Lucas*.
3) Determine el valor de las siguientes potencias de manera recursiva: $(2.2)^{-9}$ y $\left(\frac{3}{5}\right)^{10}$. Muestre el código de las funciones recursivas y el procedimiento iteración por iteración.
4) Realice la búsqueda del **`dato=RandomInteger[{1, 20}]`** sobre la lista **`L=RandomInteger[{1, 20}, 20]`** de manera recursiva mostrando el proceso paso a paso.
5) Encuentre de forma recursiva iteración por iteración el mínimo común múltiplo y el máximo común divisor entre:
   a) 100 y 562.
   b) 822 y 672.
6) Multiplique recurriendo al uso de la instrucción **`Multipli`** de *VilCretas*, los número enteros del ejercicio anterior, paso a paso y mostrando el código del comando.
7) Sume los dígitos de los números enteros obtenidos en el ejercicio 6 a través de un razonamiento recursivo, iteración por iteración y mostrando el código.

8) Halle de manera recursiva iteración por iteración los divisores de los números enteros 645 y 2022 ¿Qué sucede con el cálculo de los divisores de 2022?, explique.

9) Encuentre un programa recursivo que evalúe cada una de las sumatorias dadas a continuación:

   a) $\sum_{i=3}^{n-6} \frac{i-9}{i^2-8}$, con $n = 9$.

   b) $\sum_{i=5}^{n+2} 3^{i-1} \ln(i)$, con $n = 15$, exprese el resultado en un formato decimal.

   c) $\sum_{i=1}^{n-1} \ln(n \cdot i)$, con $n = 3$ ¿Qué ocurre en este ejercicio?, explique.

10) Emplee el algoritmo *Quicksort* para ordenar la lista `L=RandomInteger[{1, 20}, 20]`, mostrando el código de programación y el procedimiento paso a paso. Corra el algoritmo para ordenar también, de forma descendente.

Capítulo 4

Relaciones de recurrencia con *VilCretas*

En este capítulo se introducen distintos comandos que permiten estudiar el tema de relaciones de recurrencia. Se han implementado en *VilCretas* **nueve** instrucciones orientadas a evaluar, resolver y graficar relaciones de recurrencia lineales de cualquier tipo (homogéneas, no homogéneas, con coeficientes constantes o funcionales), además, de mostrar paso a paso procedimientos de resolución clásicos. A este respecto, estas sentencias proporcionan al estudiante mecanismos de revisión de ejecución de algoritmos, iteración por iteración, facilitando un aprendizaje de los contenidos más autónomo y autoregulado.

1) **CDFResolRecurrencias**: aplicación *CDF* empleada para **resolver** una relación de recurrencia **lineal**, **homogénea** o **no homogénea**, **con** o **sin coeficientes constantes**, utilizando como base lo expuesto en el artículo: Vílchez, E. (2015). Resolución de relaciones de recurrencia con apoyo de *Wolfram Mathematica*. Revista UNICIENCIA, No. 1, Vol. 29. Publicado en: http://dx.doi.org/10.15359/ru.29-1.2.

   Sintaxis: **`CDFResolRecurrencias[m, n]`** con "m" la cantidad de términos a evaluar en la relación de recurrencia correspondiente. El segundo argumento de esta función especifica el parámetro mediante el cual se visualizará la **solución** de la relación de recurrencia. Solo procesa relaciones de orden **mayor** o **igual** a **dos**.

Ejemplo 4.1

Mediante la aplicación *CDF* mostrada por el comando **`CDFResolRecurrencias`** resuelva la relación de recurrencia que genera los números de *Fibonacci*. Las siglas *CDF* significan en el software *Mathematica* ***Computable Document Format*** y representan un tipo especial de documento que es posible construir mediante este programa.

Solución:

Recordando la relación de recurrencia de interés en este ejemplo, corresponde a:

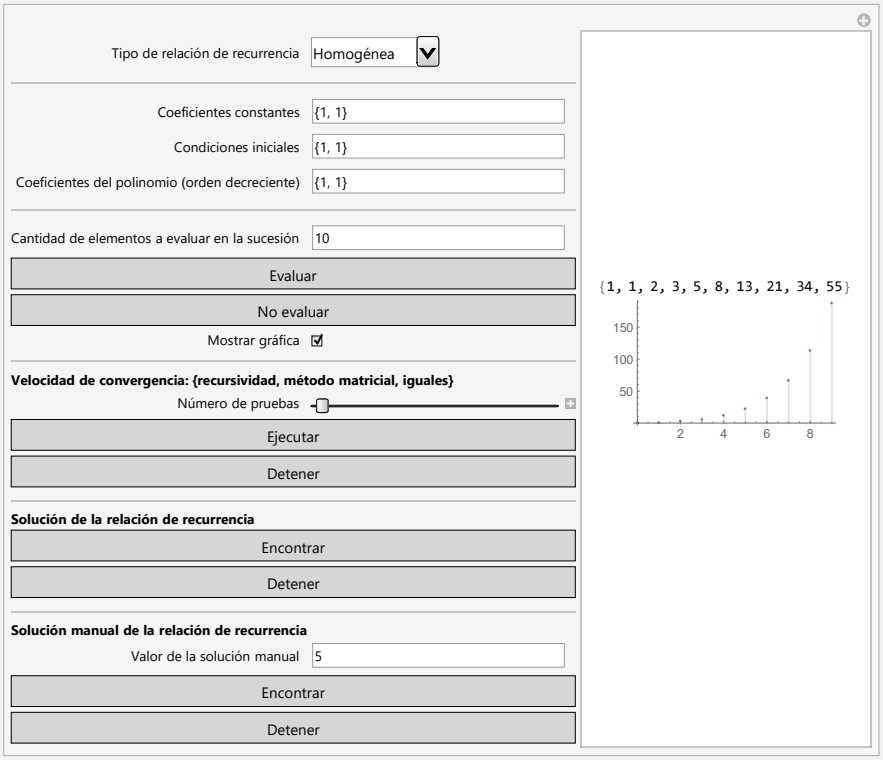
$$a_n = a_{n-1} + a_{n-2}, \text{ con } a_1 = a_2 = 1$$

Por consiguiente, en el software:

In[ ] :=

```
CDFResolRecurrencias[10, n]
```

Out[ ] :=



Al presionar el botón "Encontrar" halla la solución buscada. El parámetro 10 del argumento indica diez términos a evaluar en la relación de recurrencia y **n** constituye la variable con la que se muestra el resultado.

Ejemplo 4.2

Utilice el comando **CDFResolRecurrencias** para evaluar los veinte primeros elementos de la sucesión representada por la relación de recurrencia:

$$a_n = a_{n-1} + a_{n-2} + n + 1, \text{ con } a_1 = 1, a_2 = 2$$

Además, grafique la relación y muestre su solución.

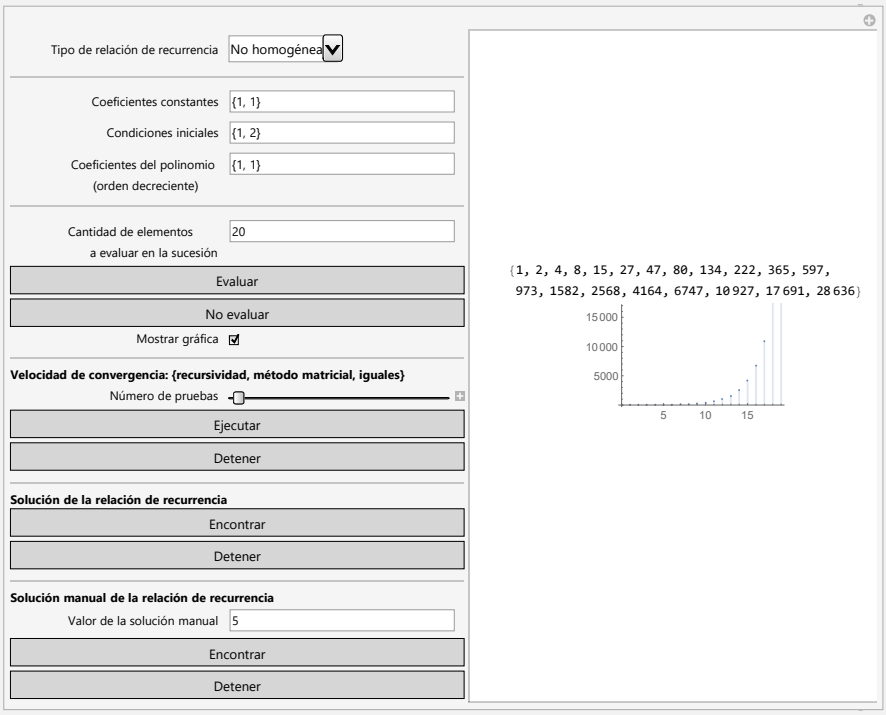
Solución:

Al emplear la instrucción **CDFResolRecurrencias** es necesario seleccionar en el primer combo "No homogénea":

In[ ] :=

```
CDFResolRecurrencias[20, n]
```

Out[ ] :=



Explicación en video

2) **RT**: instrucción que permite encontrar una **lista** con "m" **términos** de una sucesión representada por una **relación de recurrencia lineal**.

Sintaxis: `RT[Coeficientes, Condiciones, m]` con "Coeficientes" un **vector de coeficientes en orden descendente** y/o que contiene la parte no homogénea (en función de "n") de la relación de recurrencia como su última componente y "Condiciones" la lista de condiciones iniciales en orden **ascendente**. Por otra parte, "m" corresponde a la cantidad de términos a evaluar. Por defecto, se asume el **dominio** como el conjunto de los **números naturales** (no incluye el cero). La opción "`inicio->Valor`" comienza las condiciones iniciales en "Valor".

Ejemplo 4.3

Determine los primeros diez elementos de la sucesión representada por:

$$a_n = 5a_{n-1} + 6a_{n-2} + 3a_{n-3} + 9a_{n-4} + a_{n-5} + \frac{n-1}{n^2+9}$$

donde $a_1 = 9$, $a_2 = 8$, $a_3 = 1$, $a_4 = 1$ y $a_5 = 1$.

Solución:

En el software al emplear el comando **RT**:

In[ ] :=

```
RT[{5, 6, 3, 9, 1, (n - 1)/(n^2 + 9)}, {9, 8, 1, 1, 1}, 10]
```

Out[ ] :=
{9, 8, 1, 1, 1, 856/9, 130933/261, 58912973/19053, 1787710993/95265, 238295053588/2076777}

En formato decimal:

In[ ] :=
```
RT[{5, 6, 3, 9, 1, (n - 1)/(n^2 + 9)}, {9, 8, 1, 1, 1}, 10] // N
```

Out[ ] :=
{9., 8., 1., 1., 1., 95.1111, 501.659, 3092.06, 18765.7, 114743.}

Ejemplo 4.4

Halle los primeros diez términos de la sucesión dada por:

$$a_n = 5a_{n-1} + 6a_{n-2} + 3a_{n-3} + 9a_{n-4} + a_{n-5} + 2^{n+6}$$

donde $a_1 = 9$, $a_2 = 8$, $a_3 = 1$, $a_4 = 1$ y $a_5 = 1$.

Solución:

In[ ] :=
```
RT[{5, 6, 3, 9, 1, 2^(n + 6)}, {9, 8, 1, 1, 1}, 10]
```

Out[ ] :=
{9, 8, 1, 1, 1, 4191, 29173, 187408, 1157429, 7102368}

Explicación en video

3) **RR**: **resuelve** una **relación de recurrencia lineal** recibiendo como parámetros los **coeficientes**, la **parte no homogénea** si existiera y las **condiciones iniciales**.

Sintaxis: `RR[Coeficientes, Condiciones, n]` con "Coeficientes" un **vector de coeficientes en orden descendente** y/o que contiene la parte no homogénea (en función de "n") de la relación de recurrencia como su última componente y "Condiciones" la lista de condiciones iniciales en orden **ascendente**. El tercer argumento indica la variable mediante la cual se visualizará la **solución**. La opción "`inicio->Valor`" comienza las condiciones iniciales en "Valor". En principio "`Valor=1`" pues se asume el **dominio** como el conjunto de los **números naturales** (no incluye el cero).

Ejemplo 4.5

Resuelva mediante el uso del software *Mathematica* la relación de recurrencia:

$$a_n = 5a_{n-1} + 9a_{n-2} + n - 4n^2, \text{ con } a_1 = 1 \text{ y } a_2 = -1$$

Solución:

In[ ] :=
```
RR[{5, 9, n - 4 n^2}, {1, -1}, n]
```

No se mostrará la salida por sus dimensiones, pues el resultado por defecto se retorna de manera exacta. Si se desea obtener la respuesta con un formato decimal se procede así:

In[ ] :=
```
RR[{5, 9, n - 4 n^2}, {1, -1}, n] // N
```

Out[ ] :=
-1.38529*10^-9 (-2.81025)^(-1. n) (-1.)^(-2. n) 2.^(7. - 1. n) 3.^(-2. - 4. n) (6.77254*10^6 (-36.)^n (-1.)^(2. n) 3.^(4. n) - 792152. (-6.40512)^n (-2.81025)^(2. n) 3.^(3. + 2. n) - 2.24663*10^6 (-2.)^(2. n) (-1.40512)^n 3.^(2. + 4. n) - 380799. (-5.6205)^n (-1.40512)^n (-1.)^(2. n) 3.^(3. + 4. n) - 2.20159*10^6 (-6.40512)^n (-2.81025)^(2. n) 3.^(2. + 2. n) n - 3.50475*10^6 (-2.)^(2. n) (-1.40512)^n 3.^(2. + 4. n) n - 534365. (-6.40512)^n (-2.81025)^(2. n) 3.^(2. + 2. n) n^2 - 300225. (-1.40512)^n (-1.)^(2. n) 2.^(2. + 2. n) 3.^(2. + 4. n)n^2)

Ejemplo 4.6

Encuentre la solución mediante el uso de *VilCretas* de la relación de recurrencia:

$$a_n = 5a_{n-1} + 6a_{n-2} + \frac{n-1}{n^2+9}, \text{ con } a_1 = a_2 = 1$$

Solución:

In[ ] :=
```
RR[{5, 6, (n - 1)/(n^2 + 9)}, {1, 1}, n]
```

Out[ ] :=
(1/3276 + i/3276) ((-1188 + 1188 i) (-1)^n + (25 - 25 i) 2^n 3^(1 + n) + (13 + 13 i) 2^n 3^(1 + n)Hypergeometric2F1[1, 1 - 3 i, 2 - 3 i, 1/6] - (13 + 13 i) 2^n 3^(1 + n) Hypergeometric2F1[1, 1 + 3 i, 2 + 3 i, 1/6] - (156 - 78 i) (-1)^(2 n)LerchPhi[-1, 1, (2 - 3 i) + n] - (78 - 156 i) (-1)^(2 n)LerchPhi[-1, 1, (2 + 3 i) + n] - (26 - 13 i) LerchPhi[1/6, 1, (2 - 3 i) + n] - (13 - 26 i) LerchPhi[1/6, 1, (2 + 3 i) + n] + (78 - 39 i) (-1)^n PolyGamma[0, 1/2 - (3 i)/2] + (39 - 78 i) (-1)^n PolyGamma[0, 1/2 + (3 i)/2] - (78 - 39 i) (-1)^n PolyGamma[0, 1 - (3 i)/2] - (39 - 78 i) (-1)^n PolyGamma[0, 1 + (3 i)/2])

Las expresiones **`Hypergeometric2F1`**, **`LerchPhi`** y **`PolyGamma`** son funciones intergradas por defecto en *Mathematica*, además, $i = (0,1)$.

Explicación en video

4) **FindRRHL**: comando que **encuentra** si existe, una **relación de recurrencia homogénea lineal** con **coeficientes constantes** y su **solución**, dada una sucesión de números reales a través de una **lista** "L".

Sintaxis: `FindRRHL[L, a, n]` siendo "L" el conjunto de datos, "a" el **nombre** de la relación de recurrencia y "n" el parámetro con el que se mostrará su **solución**. Por defecto, retorna "NaD" si **no hay resultado** en el proceso por datos insuficientes o no existencia.

Ejemplo 4.7

Encuentre una relación de recurrencia para la sucesión: {-1, 2, 0, -1, 9, 7, 20, 79, 154, 412, ...}.

Solución:

In[ ] :=
```
FindRRHL[{-1, 2, 0, -1, 9, 7, 20, 79, 154, 412}, a, n]
```

Out[ ] :=
```
{a[n] == 5 a[-3 + n] + 2 a[-2 + n] + a[-1 + n], a[1] == -1, a[2] == 2, a[3] == 0, Root[-5 - 2 #1 - #1^2 + #1^3 &, 1]^n Root[-7 + 193 #1 + 839 #1^3 &, 1] + Root[-5 - 2 #1 - #1^2 + #1^3 &, 2]^n Root[-7 + 193 #1 + 839 #1^3 &, 2] + Root[-5 - 2 #1 - #1^2 + #1^3 &, 3]^n Root[-7 + 193 #1 + 839 #1^3 &, 3]}
```

En general, `Root[f, k]` representa una raíz exacta de la ecuación $f(x) = 0$. Un resultado más claro puede obtenerse así:

In[ ] :=
```
FindRRHL[{-1, 2, 0, -1, 9, 7, 20, 79, 154, 412}, a, n] // N
```

Out[ ] :=
```
{a[n] == 5. a[-3. + n] + 2. a[-2. + n] + a[-1. + n], a[1.] == -1., a[2.] == 2., a[3.] == 0., (-0.0180328 - 0.480636 i) (-0.775851 - 1.16513 i)^n - (0.0180328 - 0.480636 i) (-0.775851 + 1.16513 i)^n + 0.0360655 2.5517^n}
```

Ejemplo 4.8

Determine una relación de recurrencia y su solución, que represente: {1, 1, -1, 1, 10, 35, 144, 628, 2695, 11527, ...}.

Solución:

In[ ] :=
```
FindRRHL[{1, 1, -1, 1, 10, 35, 144, 628, 2695, 11527}, a, n]
```

Out[ ] :=
```
{a[n] == 6 a[-4 + n] + 5 a[-3 + n] + 4 a[-2 + n] + 3 a[-1 + n], a[1] == 1, a[2] == 1, a[3] == -1, a[4] == 1, Root[-6 - 5 #1 - 4 #1^2 - 3 #1^3 + #1^4 &, 1]^n Root[13 - 5843 #1 + 621902 #1^2 + 1579075 #1^3 + 1894890 #1^4 &, 1] + Root[-6 - 5 #1 - 4 #1^2 - 3 #1^3 + #1^4 &, 2]^n Root[13 - 5843 #1 + 621902 #1^2 + 1579075 #1^3 + 1894890 #1^4 &, 2] + Root[-6 - 5 #1 - 4 #1^2 - 3 #1^3 + #1^4 &, 4]^n Root[13 - 5843 #1 + 621902 #1^2 + 1579075 #1^3 + 1894890 #1^4 &, 3] + Root[-6 - 5 #1 - 4 #1^2 - 3 #1^3 + #1^4 &, 3]^n Root[13 - 5843 #1 + 621902 #1^2 + 1579075 #1^3 + 1894890 #1^4 &, 4]}
```

> **N** El lector percibirá que la respuesta no es nada trivial, por lo que, sin el uso de software la tarea se tornaría realmente compleja.

Explicación en video

5) **GraficaRRL**: instrucción que **grafica** una **relación de recurrencia lineal** permitiendo al usuario las opciones de especificar la **cantidad de puntos** a representar y un **valor máximo** de **observación** sobre el **eje de las ordenadas**.

Sintaxis: **`GraficaRRL[Coeficientes, Condiciones]`**, siendo "Coeficientes" un **vector de coeficientes en orden descendente** y/o que contiene la parte no homogénea (en función de "n") de la relación de recurrencia como su última componente y "Condiciones" la lista de condiciones iniciales en orden **ascendente**. Además, **`GraficaRRL[Coeficientes, Condiciones, npuntos->Valor, ymax->Valor, inicio->Valor]`** da las opciones al usuario de escoger la cantidad de puntos a graficar, el valor máximo de visualización sobre el eje "y" y el inicio del dominio de la relación de recurrencia, respectivamente. "ymax" acepta el **atributo** "All" que muestra **todos los puntos** involucrados en la graficación. Por defecto, "**`npuntos->10`**", "**`ymax->All`**" e "**`inicio->1`**".

Ejemplo 4.9

Grafique sobre sus primeros veinte términos, la relación de recurrencia dada por:

$$a_n = a_{n-1} + a_{n-2} + 3a_{n-3} + \frac{n}{n\sqrt{n} + 2n} - 50, \text{ con } a_{10} = 9,\ a_{11} = 8 \text{ y } a_{12} = 1$$

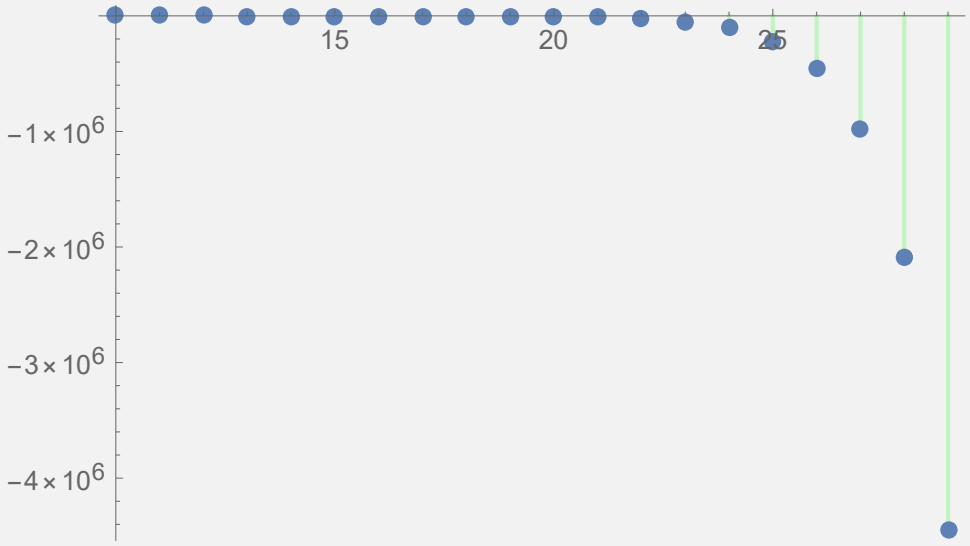
Solución:

En *Mathematica*:

In[ ] :=

```
GraficaRRL[{1, 1, 3, n/(n Sqrt[n] + 2 n) - 50}, {9, 8, 1},
npuntos->20, inicio->10]
```

Out[ ] :=



Ejemplo 4.10

Represente en el plano cartesiano sobre sus primeros veinte elementos, la relación de recurrencia:

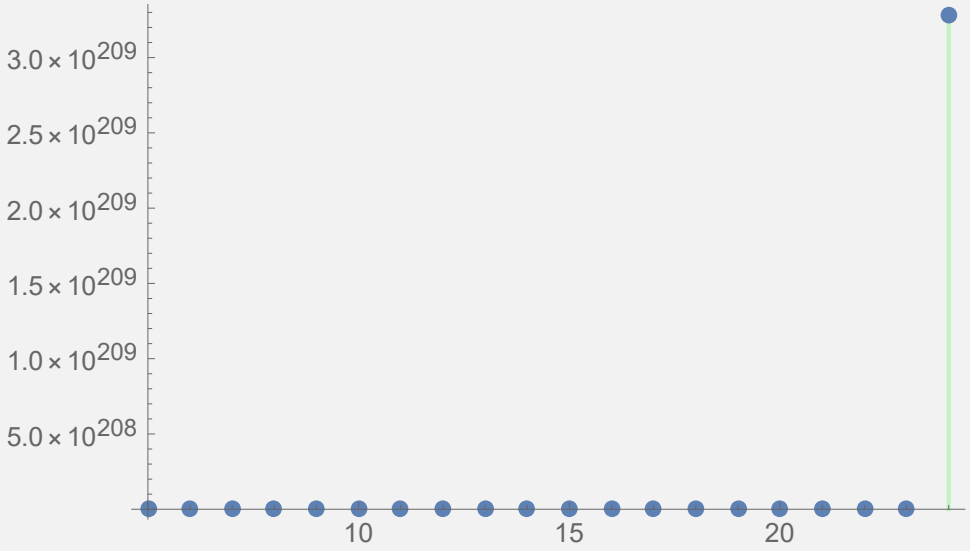
$$a_n = a_{n-1} + a_{n-2} + 3a_{n-3} + 2^{5n+n^2}, \text{ con } a_5 = 9,\ a_6 = 8 \text{ y } a_7 = 1$$

Solución:

In[ ] :=

```
GraficaRRL[{1, 1, 3, 2^(5 n + n^2)}, {9, 8, 1}, npuntos->20,
inicio->5]
```

Out[ ] :=



Explicación en video

6) **RTL**: sentencia que permite **encontrar** una **lista** con "m" términos de una **sucesión** representada por una **relación de recurrencia homogénea lineal con coeficientes constantes**.

Sintaxis: `RTL[Coeficientes, Condiciones, m]` con "Coeficientes" un **vector de coeficientes en orden descendente** y "Condiciones" las condiciones iniciales en orden **ascendente**. Por otra parte, "m" corresponde a la cantidad de términos a evaluar. Por defecto, se asume el **dominio** como el conjunto de los **números naturales** (no incluye el cero).

Ejemplo 4.11

Encuentre los primeros diez términos de la relación de recurrencia:

$$a_n = 5a_{n-1} + \sqrt{3}a_{n-2} + 3a_{n-3} + 9a_{n-4} + a_{n-5}$$

donde $a_1 = 9$, $a_2 = \frac{8}{2}$, $a_3 = 1$, $a_4 = 1$ y $a_5 = 1$.

Solución:

In[ ] :=
```
RTL[{5, Sqrt[3], 3, 9, 1}, {9, 8/2, 1, 1, 1}, 10]
```

Out[ ] :=
$\{9, 4, 1, 1, 1, 53 + \sqrt{3}, 281 + 6\sqrt{3}, 1421 + 83\sqrt{3}, 7292 + 699\sqrt{3}, 38030 + 4943\sqrt{3}\}$

En formato decimal:

In[ ] :=
```
RTL[{5, Sqrt[3], 3, 9, 1}, {9, 8/2, 1, 1, 1}, 10] // N
```

Out[ ] :=
{9., 4., 1., 1., 1., 54.7321, 291.392, 1564.76, 8502.7, 46591.5}

Ejemplo 4.12

Determine los primeros diez términos de la relación de recurrencia dada a continuación:

$$a_n = 5a_{n-1} + 6a_{n-2} + 3a_{n-3} + 9xa_{n-4}$$

donde $a_1 = 9$, $a_2 = 8$, $a_3 = 1$ y $a_4 = m$.

Solución:

In[ ] :=
```
RTL[{5, 6, 3, 9x}, {9, 8, 1, m}, 10]
```

Out[ ] :=

{9, 8, 1, m, 30 + 5 m + 81 x, 153 + 31 m + 477 x, 945 + 188 m + 2880 x, 5733 + 1141 m + 17505 x + 9 m x, m (6926 + 90 x) + 9 (3866 + 11834 x + 81 x^2), m (42040 + 783 x) + 9 (23467 + 71953 x + 882 x^2)}

Explicación en video

7) **MetodoRRHL**: muestra **paso a paso** la aplicación de uno de los **métodos clásicos** para **resolver** una **relación de recurrencia homogénea lineal con coeficientes constantes**, a través del uso de una ecuación denominada "ecuación característica".

Sintaxis: `MetodoRRHL[Coeficientes, Condiciones, n, b]` siendo "Coeficientes" un **vector de coeficientes numéricos en orden descendente**, "Condiciones" las condiciones iniciales en orden **ascendente**, "n" el parámetro con el que se visualizará la **solución** de la relación de recurrencia y "b" el **nombre** de la variable del sistema de ecuaciones que emplea el procedimiento. Por defecto, el **dominio** corresponde al conjunto de los **números naturales** (no incluye el cero). La instrucción integra las alternativas "`decimal->True`" que exhibe el resultado con un **formato decimal** e "`inicio->Valor`" que define el **primer elemento** ("Valor") del **dominio** si se deseara cambiar. Originalmente, "decimal" tiene el valor lógico "`False`" e "`inicio->1`".

Ejemplo 4.13

Resuelva paso a paso la relación de recurrencia:

$$a_n = 11a_{n-1} - 49a_{n-2} + 113a_{n-3} - 142a_{n-4} + 92a_{n-5} - 24a_{n-6}$$

donde $a_0 = 9$, $a_1 = 8$, $a_2 = 1$, $a_3 = 1$, $a_4 = 1$ y $a_5 = 1$.

Solución:

In[ ] :=

```
MetodoRRHL[{11, -49, 113, -142, 92, -24}, {9, 8, 1, 1, 1, 1}, n, b,
inicio->0]
```

Out[ ] :=

La ecuación característica corresponde a: $n^6 - 11n^5 + 49n^4 - 113n^3 + 142n^2 - 92n + 24 = 0$

Raíz o raíces de la ecuación característica: $\{1,1,2,2,2,3\}$

La forma que toma la solución de la relación de recurrencia es: $b_2 n + b_1 + 2^n b_3 + (2^n n)\, b_4 + \left(2^n n^2\right) b_5 + 3^n b_6$

El sistema de ecuaciones a resolver corresponde a: $\{b_1 + b_3 + b_6 = 9,\ b_1 + b_2 + 2b_3 + 2b_4 + 2b_5 + 3b_6 = 8,\ b_1 + 2b_2 + 4b_3 + 8b_4 + 16b_5 + 9b_6 = 1,\ b_1 + 3b_2 + 8b_3 + 24b_4 + 72b_5 + 27b_6 = 1,\ b_1 + 4b_2 + 16b_3 + 64b_4 + 256b_5 + 81b_6 = 1,\ b_1 + 5b_2 + 32b_3 + 160b_4 + 800b_5 + 243b_6 = 1\}$

La solución del sistema de ecuaciones es:

$\left\{b_1 \to 440, b_2 \to 142, b_3 \to -464, b_4 \to \frac{301}{2}, b_5 \to -23, b_6 \to 33\right\}$

La solución de la relación de recurrencia corresponde a:

$\frac{1}{2}\left(-\left(23\ 2^{n+1}\right) n^2 + \left(301\ 2^n\right) n + 284n + 22\ 3^{n+1} - 29\ 2^{n+5} + 880\right)$

Ejemplo 4.14

Encuentre la solución de la relación de recurrencia siguiente:

$$a_n = 21a_{n-1} - 192a_{n-2} + 1002a_{n-3} - 3285a_{n-4} + 7005a_{n-5} - 9698a_{n-6} + 8388a_{n-7} - 4104a_{n-8} + 864a_{n-9}$$

con $a_5 = 1$, $a_6 = -2$, $a_7 = 3$, $a_8 = -4$, $a_9 = 5$, $a_{10} = -6$, $a_{11} = 7$, $a_{12} = 1$ y $a_{13} = 2$.

Solución:

In[ ] :=

```
MetodoRRHL[{21, -192, 1002, -3285, 7005, -9698, 8388, -4104, 864},
{1, -2, 3, -4, 5, -6, 7, 1, 2}, n , b, inicio->5]
```

Out[ ] :=

La ecuación característica corresponde a:

$n^9 - 21n^8 + 192n^7 - 1002n^6 + 3285n^5 - 7005n^4 + 9698n^3 - 8388n^2 + 4104n - 864 = 0$

Raíz o raíces de la ecuación característica: $\{1,1,2,2,2,3,3,3,4\}$

La forma que toma la solución de la relación de recurrencia es:

$b_2 n + b_1 + 2^n b_3 + (2^n n)\, b_4 + \left(2^n n^2\right) b_5 + 3^n b_6 + (3^n n)\, b_7 + \left(3^n n^2\right) b_8 + 4^n b_9$

El sistema de ecuaciones a resolver corresponde a:

$\{b_1 + 5b_2 + 32b_3 + 160b_4 + 800b_5 + 243b_6 + 1215b_7 + 6075b_8 + 1024b_9 = 1,$
$b_1 + 6b_2 + 64b_3 + 384b_4 + 2304b_5 + 729b_6 + 4374b_7 + 26244b_8 + 4096b_9 = -2,$
$b_1 + 7b_2 + 128b_3 + 896b_4 + 6272b_5 + 2187b_6 + 15309b_7 + 107163b_8 + 16384b_9 = 3,$
$b_1 + 8b_2 + 256b_3 + 2048b_4 + 16384b_5 + 6561b_6 + 52488b_7 + 419904b_8 + 65536b_9 = -4,$
$b_1 + 9b_2 + 512b_3 + 4608b_4 + 41472b_5 + 19683b_6 + 177147b_7 + 1594323b_8 + 262144b_9 = 5,$
$b_1 + 10b_2 + 1024b_3 + 10240b_4 + 102400b_5 + 59049b_6 + 590490b_7 + 5904900b_8 + 1048576b_9 = -6,$
$b_1 + 11b_2 + 2048b_3 + 22528b_4 + 247808b_5 + 177147b_6 + 1948617b_7 + 21434787b_8 + 4194304b_9 = 7,$
$b_1 + 12b_2 + 4096b_3 + 49152b_4 + 589824b_5 + 531441b_6 + 6377292b_7 + 76527504b_8 + 16777216b_9 = 1,$ $b_1 + 13b_2 + 8192b_3 + 106496b_4 + 1384448b_5 + 1594323b_6 + 20726199b_7 + 269440587b_8 + 67108864b_9 = 2\}$

La solución del sistema de ecuaciones es:

$\{b_1 \to \frac{212303}{144}, b_2 \to -\frac{59429}{24}, b_3 \to \frac{343895}{128}, b_4 \to -\frac{128913}{256}, b_5 \to \frac{20743}{256}, b_6 \to -\frac{5557183}{11664}, b_7 \to \frac{966829}{17496},$
$b_8 \to -\frac{31799}{17496}, b_9 \to \frac{805}{2304}\}$

La solución de la relación de recurrencia corresponde a:

$$\frac{(45364941\ 2^n)\, n^2 - (1017568\ 3^n)\, n^2 - (281932731\ 2^n)\, n + (30938528\ 3^n)\, n}{559872} + \frac{-1386359712n + 195615\ 2^{2n} + 752098365\ 2^{n+1} - 88914928\ 3^{n+1} + 825434064}{559872}$$

Explicación en video

8) **RRCP**: resuelve distintos **casos** de una **relación de recurrencia lineal no homogénea**, donde la parte no homogénea depende de un parámetro "j".

Sintaxis: `RRCP[Coeficientes, Condiciones, n, m]` con "Coeficientes" un **vector de coeficientes en orden descendente**, que contiene la parte no homogénea (en función de "n") de la relación de recurrencia como su última componente, "Condiciones" la lista de condiciones iniciales en orden **ascendente** y "m" la **cantidad de casos** a evaluar. El tercer argumento indica la variable mediante la cual se visualizarán las **soluciones**. Por defecto, se admite el **dominio** como el conjunto de los **números naturales** (no incluye el cero) y el parámetro involucrado con el identificador "j". El comando posee la opción "`decimal->True`" que muestra el resultado con un **formato decimal**, en principio "decimal" tiene el valor lógico "`False`". Además, "`inicio->Valor`" **comienza** las condiciones iniciales con **índice** en "Valor". En arranque "`Valor=1`".

Ejemplo 4.15

Retorne las soluciones en formato decimal, de la relación de recurrencia dada, haciendo variar j de uno a cinco:

$$a_n = 5a_{n-1} + 6a_{n-2} + 6a_{n-3} + 9a_{n-4} + 6j$$

donde $a_{10} = 9$, $a_{11} = 8$, $a_{12} = 6$ y $a_{13} = 1$.

Solución:

In[ ] :=

```
RRCP[{5, 6, 6, 9, 6j}, {9, 8, 6, 1}, n, 5, decimal->True, inicio->10]
```

Out[ ] :=

Caso 1: 0.04 (-6.+22.3624 (-1.1862)^n-(22.7988 +51.902 i) (0.00876387 -1.109 i)^n-(22.7988 -51.902 i) (0.00876387 +1.109 i)^n+3.47763*10^-8 6.16867^n)

Caso 2: 0.04 (-12.+22.9987 (-1.1862)^n-(23.3423 +53.3888 i) (0.00876387 -1.109 i)^n-(23.3423 -53.3888 i) (0.00876387 +1.109 i)^n+3.60388*10^-8 6.16867^n)

Caso 3: 0.04 (-18.+23.6351 (-1.1862)^n-(23.8858 +54.8755 i) (0.00876387 -1.109 i)^n-(23.8858 -54.8755 i) (0.00876387 +1.109 i)^n+3.73012*10^-8 6.16867^n)

Caso 4: 0.04 (-24.+24.2715 (-1.1862)^n-(24.4293 +56.3622 i) (0.00876387 -1.109 i)^n-(24.4293 -56.3622 i) (0.00876387 +1.109 i)^n+3.85637*10^-8 6.16867^n)

Caso 5: 0.2 (-6.+4.98157 (-1.1862)^n-(4.99455 +11.5698 i) (0.00876387 -1.109 i)^n-(4.99455 -11.5698 i) (0.00876387 +1.109 i)^n+7.96522*10^-9 6.16867^n)

(N) Se esperaría inferir de los resultados obtenidos, una conjetura sobre la solución de la recursividad en cualquier caso.

Ejemplo 4.16

Encuentre mediante el uso de la instrucción **RRCP** las soluciones en formato decimal, de la relación de recurrencia dada, haciendo variar j de uno a cinco:

$$a_n = 5a_{n-1} + 6a_{n-2} + 6a_{n-3} + 9a_{n-4} + 2^n j$$

con $a_{10} = 9$, $a_{11} = 8$, $a_{12} = 6$ y $a_{13} = 1$.

Solución:

```
In[ ] :=
RRCP[{5, 6, 6, 9, 2^n j}, {9, 8, 6, 1}, n, 3, decimal->True,
inicio->10]

Out[ ] :=
Caso 1: (267.847 +0. i) (-1.1862)^(-10.+n)- ...
⋮
```

No se muestra por su tamaño, la salida completa arrojada por *Mathematica*.

Explicación en video

9) **MetodoI**: muestra sobre una **relación de recurrencia lineal** de orden **uno**, sus **evaluaciones** en "k" **iteraciones** de forma **ascendente o descendente**, según lo escoja el usuario. El comando es una aplicación directa del método de resolución de relaciones de recurrencia **denominado**: "iterativo".

Sintaxis: `MetodoI[Coeficientes, k]`, o bien,

```
MetodoI[Coeficientes, k, ascendente->True, inicio->Valor]
```

siendo "Coeficientes" **un vector de dimensión uno o dos que contiene un coeficiente** y/o **la parte no homogénea** (en función de "n") como su última componente, "`ascendente->True`" opción que exhibe los resultados de forma ascendente (por defecto es descendente) e "`inicio->Valor`", el valor del **índice** con el que **empieza** la condición inicial. "`MetodoI`" solo acepta **números enteros**.

Ejemplo 4.17

Emplee el método iterativo para evaluar cinco veces de forma descendente, la relación de recurrencia:

$$a_n = 2a_{n-1} + 2 + 3^{n-1}, \text{ con } a_1 = 4$$

Solución:

In[ ] :=

```
MetodoI[{2, 2 + 3^(n - 1)}, 5]
```

Out[ ] :=

Por el método iterativo:

n $\to n-1$: $\left(2+3^{n-2}\right)2^1+3^{n-1}+a(n-2)2^2+2$

n $\to n-2$: $\left(2+3^{n-2}\right)2^1+3^{n-1}+\left(2+3^{n-3}\right)2^2+a(n-3)2^3+2$

n $\to n-3$: $\left(2+3^{n-2}\right)2^1+3^{n-1}+\left(2+3^{n-3}\right)2^2+\left(2+3^{n-4}\right)2^3+a(n-4)2^4+2$

n $\to n-4$: $\left(2+3^{n-2}\right)2^1+3^{n-1}+\left(2+3^{n-3}\right)2^2+\left(2+3^{n-4}\right)2^3+\left(2+3^{n-5}\right)2^4+a(n-5)2^5+2$

n $\to n-5$: $\left(2+3^{n-2}\right)2^1+3^{n-1}+\left(2+3^{n-3}\right)2^2+\left(2+3^{n-4}\right)2^3+\left(2+3^{n-5}\right)2^4+\left(2+3^{n-6}\right)2^5+a(n-6)2^6+2$

N El comando pretende ser un apoyo, donde el estudiante sea capaz de deducir alguna generalización de los resultados expuestos, con respecto a la forma que toma la solución de la relación de recurrencia.

Ejemplo 4.18

Utilice el comando **MetodoI** para evaluar cinco veces de forma descendente y ascendente, la relación de recurrencia:

$$a_n = 2n^2a_{n-1} + 2n + 3^{n-1}, \text{ con } a_3 = 6$$

Solución:

De manera descendente tenemos:

In[ ] :=

```
MetodoI[{2 n^2, 2 n + 3^(n - 1)}, 5]
```

Out[ ] :=

Por el método iterativo:

n $\to n-1: 2n+3^{n-1}+\left(\left(2(n-1)+3^{n-2}\right)n^2\right)2^1+\left(\left((n-1)^2n^2\right)a(n-2)\right)2^2$

n $\to n-2: 2n+3^{n-1}+\left(\left(2(n-1)+3^{n-2}\right)n^2\right)2^1+\left(\left(\left(2(n-2)+3^{n-3}\right)(n-1)^2\right)n^2\right)2^2+$
$\left(\left(\left((n-2)^2(n-1)^2\right)n^2\right)a(n-3)\right)2^3$

n $\to n-3: 2n+3^{n-1}+\left(\left(2(n-1)+3^{n-2}\right)n^2\right)2^1+\left(\left(\left(2(n-2)+3^{n-3}\right)(n-1)^2\right)n^2\right)2^2+$
$\left(\left(\left(\left(2(n-3)+3^{n-4}\right)(n-2)^2\right)(n-1)^2\right)n^2\right)2^3+\left(\left(\left(\left((n-3)^2(n-2)^2\right)(n-1)^2\right)n^2\right)a(n-4)\right)2^4$

n $\to n-4: 2n+3^{n-1}+\left(\left(2(n-1)+3^{n-2}\right)n^2\right)2^1+\left(\left(\left(2(n-2)+3^{n-3}\right)(n-1)^2\right)n^2\right)2^2+$
$\left(\left(\left(\left(2(n-3)+3^{n-4}\right)(n-2)^2\right)(n-1)^2\right)n^2\right)2^3+$
$\left(\left(\left(\left(2(n-4)+3^{n-5}\right)(n-3)^2\right)(n-2)^2\right)(n-1)^2\right)n^2\Big)\,2^4+$
$\left(\left(\left(\left(\left((n-4)^2(n-3)^2\right)(n-2)^2\right)(n-1)^2\right)n^2\right)a(n-5)\right)2^5$

n $\to n-5: 2n+3^{n-1}+\left(\left(2(n-1)+3^{n-2}\right)n^2\right)2^1+\left(\left(\left(2(n-2)+3^{n-3}\right)(n-1)^2\right)n^2\right)2^2+$
$\left(\left(\left(\left(2(n-3)+3^{n-4}\right)(n-2)^2\right)(n-1)^2\right)n^2\right)2^3+$
$\left(\left(\left(\left(2(n-4)+3^{n-5}\right)(n-3)^2\right)(n-2)^2\right)(n-1)^2\right)n^2\Big)\,2^4+$
$\left(\left(\left(\left(\left(2(n-5)+3^{n-6}\right)(n-4)^2\right)(n-3)^2\right)(n-2)^2\right)(n-1)^2\right)n^2\Big)\,2^5+$
$\left(\left(\left(\left(\left(\left((n-5)^2(n-4)^2\right)(n-3)^2\right)(n-2)^2\right)(n-1)^2\right)n^2\right)a(n-6)\right)2^6$

De forma ascendente:

In[ ] :=

```
MetodoI[{2 n^2, 2 n + 3^(n - 1)}, 5, ascendente->True, inicio->3]
```

Out[ ] :=

Por el método iterativo:

n $\to$ 4: $5^1 \cdot 7^1 + a(3)2^5$

n $\to$ 5: $a(3)2^6 \cdot 5^2 + 7^1 \cdot 13^1 + 2^1 \cdot 5^3 \cdot 7^1$

n $\to$ 6: $a(3)2^9 \cdot 3^2 \cdot 5^2 + 3^1 \cdot 5^1 \cdot 17^1 + 2^3 \cdot 3^2 \cdot 7^1 \cdot 13^1 + 2^4 \cdot 3^2 \cdot 5^3 \cdot 7^1$

n $\to$ 7: $2^4 \cdot 3^2 \cdot 7^3 \cdot 13^1 + 2^5 \cdot 3^2 \cdot 5^3 \cdot 7^3 + a(3)2^{10} \cdot 3^2 \cdot 5^2 \cdot 7^2 + 2^1 \cdot 3^1 \cdot 5^1 \cdot 7^2 \cdot 17^1 + 743^1$

n $\to$ 8: $2^7 \cdot 743^1 + 2^{11} \cdot 3^2 \cdot 7^3 \cdot 13^1 + 2^{12} \cdot 3^2 \cdot 5^3 \cdot 7^3 + a(3)2^{17} \cdot 3^2 \cdot 5^2 \cdot 7^2 + 2^8 \cdot 3^1 \cdot 5^1 \cdot 7^2 \cdot 17^1 + 2203^1$

Explicación en video

Aporte pedagógico

Los comandos abordados en esta sección pueden contribuir con un **autoaprendizaje** por parte de la población estudiantil, específicamente en los **métodos** de **resolución** de relaciones de recurrencia: *iterativo* y por *ecuación característica*. Las instrucciones **`MetodoI`** y **`MetodoRRHL`** a este respecto, proporcionan recursos de **consulta directo** donde el alumno podría de manera autónoma **profundizar** el aprendizaje de estos procedimientos.

Aporte de investigación

Un comando interesante que ofrece posibilidades de investigación lo constituye **`RRCP`**. La instrucción abre una ventana de posibilidades con el objetivo de resolver ejercicios donde se logren **construir conjeturas**. La construcción de conjeturas va más allá de un enfoque de enseñanza y aprendizaje basado netamente en la transmisión de contenido, de allí su importancia y énfasis cuando se emplea **tecnología** en el salón de clase.

Ejercicios

Resuelva los siguientes ejercicios utilizando como apoyo el paquete *VilCretas*.

1) Encuentre los primeros veinte elementos de la sucesión representada por la relación de recurrencia:

   a) $a_n = 2na_{n-2} + 6a_{n-1}$, con $a_2 = \sqrt{2}$ y $a_1 = -1$.

   b) $a_{n+2} = a_n + n$, con $a_0 = \frac{1}{2}$ y $a_1 = 9\pi$.

   c) $a_n = a_{n-1} + 1 + \left(\frac{2}{3}\right)^{n-1}$, con $a_6 = -7$.

   d) $a_{n-2} = \ln(2)\, a_{n-3} + \ln(5)\, a_{n-4}$ con $a_0 = 0$ y $a_1 = 6$.

   e) $a_n = -a_{n-1} + 5a_{n-2} - 3a_{n-3}$, con $a_2 = a_3 = 9$ y $a_4 = 8\sqrt[3]{3}$.

f) $a_n = -a_{n-1} + 5a_{n-2} - 3a_{n-3} + (-5)^n$, con $a_2 = a_3 = a_4 = 1$.

2) Grafique las relaciones de recurrencia anteriores mediante el uso del comando **`GraficaRRL`**. Manipule las opciones que brinda esta instrucción.

3) Resuelva a través del uso de la sentencia **`RR`**, las relaciones de recurrencia del ejercicio 1.

4) Encuentre si es posible utilizando el comando **`FindRRHL`**, una relación de recurrencia para las siguientes sucesiones:

   a) $\{3,7,11,15,19,23,\ldots\}$

   b) $\{3,6,9,15,24,39,\ldots\}$

   c) $\{1,16,256,4096,65536,...\}$

5) Conjeture la forma que toma la solución de $a_n = -a_{n-1} + 5a_{n-2} - 3a_{n-3} + j$, con $a_2 = a_3 = a_4 = 1$ y $j \in \mathbb{N}$. Sugerencia: emplee el comando **`RRCP`**.

6) Utilizando la instrucción **`MetodoRRHL`**, resuelva:

   a) $a_n = \dfrac{a_{n-1} + a_{n-2}}{2}$, con $a_1 = 0$ y $a_2 = 1$.

   b) $2a_n = 7a_{n-1} - 3a_{n-2}$, con $a_0 = a_1 = 1$.

   c) $a_n = 2a_{n-1} - a_{n-2} + a_{n-3}$, con $a_1 = -6$ y $a_2 = a_3 = -1$.

   d) $9a_n = 6a_{n-1} - a_{n-2} + a_{n-3} - 2a_{n-4}$, con $a_2 = 1, a_3 = 2, a_4 = 3$ y $a_5 = 4$. Exprese con un formato decimal.

7) Recurra al comando **`MetodoI`** para resolver:

   a) $a_n = a_{n-1} + 4n$, con $a_0 = \frac{5}{2}$.

   b) $a_n = 6a_{n-1} + 3^n$, con $a_2 = 3$.

   c) $a_n = 3^n a_{n-1} + 7^n n$, con $a_1 = -1$.

Capítulo 5

Análisis de algoritmos con *VilCretas*

El análisis de algoritmos es un tema crucial cuando se estudia el nivel de eficiencia de un programa y puede abarcar dos enfoques: uno experimental midiendo tiempos de ejecución localmente y comparando distintos algoritmos que resuelven el mismo problema y otro, teórico mediante el uso de ciertas notaciones denominadas notaciones asintóticas. En este capítulo, se explica el funcionamiento de **doce** instrucciones que forman parte del paquete *VilCretas*, favoreciendo el estudio de la materia con ambas orientaciones. Los comandos proveen recursos para efectuar comparaciones en tiempo de ejecución de dos o tres programas que resuelven el mismo problema y se ofrecen objetos de manipulación dinámica (*CDF's*) enfocados en la comprensión conceptual sobre la definición de notación asintótica.

1) **Burbuja**: **ordena** una **lista** de datos de forma **ascendente** o **descendente** utilizando el "algoritmo de la burbuja". Las opciones "**code->True**" y "**steps->True**" le facilitan al usuario visualizar el **código** de programación interno de la función y las ejecuciones **paso a paso**, respectivamente. Por otra parte, "**ascendente->True**" ordena de forma ascendente y "**ascendente->False**" de manera descendente.

Sintaxis: **Burbuja[L]**, o bien, **Burbuja[L, code -> Valor, steps -> Valor, ascendente -> Valor]**.

Ejemplo 5.1

Ordene la lista de datos {9, -7, 5, 4, 10, -8, 10, 1} mediante el *algoritmo de la burbuja*. Muestre el código de la función y el procedimiento iteración por iteración.

Solución:

En *Mathematica* al emplear el comando **Burbuja** se obtiene:

```
In[ ] :=
Burbuja[{9, -7, 5, 4, 10, -8, 10, 1}, code -> True, steps -> True]
Out[ ] :=
{{-8, -7, 1, 4, 5, 9, 10, 10}, Burbuja[L_List]:=Module[{Datos=L}, For[i=Length[Datos], i=1, For[j=1, j<i,
If[Datos[[j]]Datos[[j+1]], aux=Datos[[j]]; Datos[[j]]=Datos[[j+1]]; Datos[[j+1]]=aux]; j++]; i− =1]; Datos]}
Iteración 1:
{9, -7, 5, 4, 10, -8, 10, 1}
{-7, 9, 5, 4, 10, -8, 10, 1}
{-7, 5, 9, 4, 10, -8, 10, 1}
⋮
{-7, 5, 4, 9, -8, 10, 1, 10}
Iteración 2:
{-7, 5, 4, 9, -8, 10, 1}
{-7, 5, 4, 9, -8, 10, 1}
{-7, 4, 5, 9, -8, 10, 1}
⋮
```

{-7, 4, 5, -8, 9, 1, 10}
Iteración 3:
{-7, 4, 5, -8, 9, 1}
{-7, 4, 5, -8, 9, 1}
{-7, 4, 5, -8, 9, 1}
⋮
{-7, 4, -8, 5, 1, 9}
⋮
Iteración 7:
{-8, -7}
{-8, -7}
Iteración 8:
{-8}
{-8, -7, 1, 4, 5, 9, 10, 10}

Ejemplo 5.2

Ordene de forma descendente la lista {9, -7, 5, 4, 10, -8, 10, 1} mostrando el código de la función **`Burbuja`** y el procedimiento paso a paso.

Solución:

En el software:

In[ ] :=

```
Burbuja[{9, -7, 5, 4, 10, -8, 10, 1}, code->True, steps->True,
ascendente->False]
```

Out[ ] :=

{{10, 10, 9, 5, 4, 1, -7, -8}, Burbuja[L_List]:=Module[{Datos=L}, For[i=Length[Datos], i=1, For[j=1, j<i, If[Datos[[j]]<Datos[[j+1]], aux=Datos[[j]]; Datos[[j]]=Datos[[j+1]]; Datos[[j+1]]=aux]; j++]; i− =1]; Datos]}
Iteración 1:
{9, -7, 5, 4, 10, -8, 10, 1}
{9, -7, 5, 4, 10, -8, 10, 1}
{9, 5, -7, 4, 10, -8, 10, 1}
⋮
{9, 5, 4, 10, -7, 10, 1, -8}
Iteración 2:
⋮
{10, 10, 9, 5, 4, 1, -7, -8}

Explicación en video

2) **Seleccion**: **ordena** una **lista** de datos de forma **ascendente** o **descendente** utilizando el "algoritmo de selección". Las opciones "**code->True**" y "**steps->True**" le permiten al usuario obtener el **código** de programación interno y las ejecuciones **paso a paso**, respectivamente. Además, "**ascendente->True**" ordena la lista de manera ascendente y "**ascendente->False**" de forma descendente.

Sintaxis: **Seleccion[L]**, o bien, **Seleccion[L, code->Valor, steps->Valor, ascendente->Valor]**.

Ejemplo 5.3

Ordene la lista de datos {9, -7, 5, 4, 10, -8, 10, 1} mediante el *algoritmo de selección*. Muestre el código de la función y el procedimiento paso a paso.

Solución:

In[ ] :=

```
Seleccion[{9, -7, 5, 4, 10, -8, 10, 1}, code->True, steps->True]
```

Out[ ] :=

{{-8, -7, 1, 4, 5, 9, 10, 10}, Seleccion[L_List]:=Module[{Datos=L}, For[i=Length[Datos], i=1, max=Datos[[1]]; pos=1; For[j=1, j<=i, If[max<Datos[[j]], max=Datos[[j]]; pos=j]; j++]; aux=Datos[[i]]; Datos[[i]]=Datos[[pos]]; Datos[[pos]]=aux; i− =1]; Datos]}
Iteración 1:
{9, -7, 5, 4, 10, -8, 10, 1}
{9, -7, 5, 4, 1, -8, 10, 10}
Iteración 2:
{9, -7, 5, 4, 1, -8, 10}
{9, -7, 5, 4, 1, -8, 10}
Iteración 3:
{9, -7, 5, 4, 1, -8}
⋮
Iteración 7:
{-8, -7}
{-8, -7}
Iteración 8:
{-8}
{-8}
{-8, -7, 1, 4, 5, 9, 10, 10}

Ejemplo 5.4

Ordene de forma descendente {9, -7, 5, 4, 10, -8, 10, 1} mostrando el código de la función **Seleccion** y el procedimiento iteración por iteración.

Solución:

En **Seleccion** se emplea la opción "**ascendente->False**":

In[ ] :=

```
Seleccion[{9, -7, 5, 4, 10, -8, 10, 1}, code->True, steps->True,
ascendente->False]
```

Out[ ] :=

{{10, 10, 9, 5, 4, 1, -7, -8}, Seleccion[L_List]:=Module[{Datos=L}, For[i=Length[Datos], i=1, max=Datos[[1]]; pos=1; For[j=1, j<=i, If[max<Datos[[j]], max=Datos[[j]]; pos=j]; j++]; aux=Datos[[i]]; Datos[[i]]=Datos[[pos]]; Datos[[pos]]=aux; i− =1]; Datos]}

Iteración 1:

{9, -7, 5, 4, 10, -8, 10, 1}

{9, -7, 5, 4, 10, 1, 10, -8}

Iteración 2:

⋮

{10, 10, 9, 5, 4, 1, -7, -8}

Explicación en video

3) **Insercion**: **ordena** una **lista** de datos de forma **ascendente** o **descendente** utilizando el "algoritmo de inserción". Las opciones "**code -> True**" y "**steps -> True**" le facilitan al usuario visualizar el **código** de programación interno de la instrucción y observar las ejecuciones **paso a paso**, respectivamente. En adición, "**ascendente -> True**" ordena de forma ascendente y "**ascendente -> False**" de manera descendente.

Sintaxis: **Insercion[L]**, o bien, **Insercion[L, code->Valor, steps->Valor, ascendente->Valor]**

Ejemplo 5.5

Ordene la lista de datos {9, -7, 5, 4, 10, -8, 10, 1} mediante el *algoritmo de inserción*. Muestre el código de la función y el procedimiento completo.

Solución:

In[ ] :=

```
Insercion[{9, -7, 5, 4, 10, -8, 10, 1}, code->True, steps->True]
```

Out[ ] :=

{{-8, -7, 1, 4, 5, 9, 10, 10}, Insercion[L_List]:=Module[{Datos=L}, For[i=1, i<=Length[Datos], eo=Datos[[i]]; v=i; For[j=i-1, j=1, If[eo<Datos[[j]], Datos=Delete[Datos, v]; Datos=Insert[Datos, eo, j]; v−−,Break[]]; j−−]; i++]; Datos]}

Iteración 1:

{9, -7, 5, 4, 10, -8, 10, 1}

{9, -7, 5, 4, 10, -8, 10, 1}

Iteración 2:

{9, -7, 5, 4, 10, -8, 10, 1}

{-7, 9, 5, 4, 10, -8, 10, 1}
Iteración 3:
{-7, 9, 5, 4, 10, -8, 10, 1}
⋮
Iteración 7:
{-8, -7, 4, 5, 9, 10, 10, 1}
{-8, -7, 4, 5, 9, 10, 10, 1}
Iteración 8:
{-8, -7, 4, 5, 9, 10, 10, 1}
{-8, -7, 1, 4, 5, 9, 10, 10}
{-8, -7, 1, 4, 5, 9, 10, 10}

Ejemplo 5.6

Ordene de forma descendente la lista {9, -7, 5, 4, 10, -8, 10, 1} mostrando el código de la función **`Insercion`** y el procedimiento paso a paso.

Solución:

En el software:

In[ ] :=
```
Insercion[{9, -7, 5, 4, 10, -8, 10, 1}, code->True, steps->True,
ascendente->False]
```

Out[ ] :=
{{10, 10, 9, 5, 4, 1, -7, -8}, Insercion[L_List]:=Module[{Datos=L}, For[i=1, i<=Length[Datos], eo=Datos[[i]]; v=i; For[j=i-1, j=1, If[eo<Datos[[j]], Datos=Delete[Datos, v]; Datos=Insert[Datos, eo, j]; v−−,Break[]]; j−−]; i++]; Datos]}
Iteración 1:
{9, -7, 5, 4, 10, -8, 10, 1}
{9, -7, 5, 4, 10, -8, 10, 1}
Iteración 2:
⋮
{10, 10, 9, 5, 4, 1, -7, -8}

Explicación en video

4) **PruebaADA2**: analiza la **eficiencia** de **dos algoritmos** que resuelven el mismo problema, ejecutando un **experimento** con "k" invocaciones y retornando el número de veces en el que cada método se comportó **más eficiente**, en términos de los tiempos de salida registrados por el software *Mathematica*.

Sintaxis: **`PruebaADA2[{Metodo1, Metodo2}, k, inicio]`**, o bien, **`PruebaADA2[{Metodo1,`**

Metodo2}, k, inicio, lista->True]. "**lista->True**" indica que los métodos reciben como parámetros una **lista** (internamente las listas son generadas de forma **seudoaleatoria**). "inicio" define el valor inicial de evaluación. Se asume el **dominio** como el conjunto de los **números naturales** o un **subconjunto** de él.

Ejemplo 5.7

Dados los métodos **Sumatoria1** y **Sumatoria2** determine de forma experimental cuál es más eficiente.

```
Sumatoria1[n_] := Module[{}, s = Sum[(j - 3) 7^j, {j, 2, n - 2}]]
Sumatoria2[n_] := Module[{suma = -49}, For[j = 3, j <= n - 2, suma =
suma + (j - 3) 7^j; j++]; suma]
```

Solución:

En *Mathematica* se deben ejecutar primero, las funciones **Sumatoria1** y **Sumatoria2** expuestas en el enunciado, luego al utilizar el comando **PruebaADA2**:

In[ ] :=
```
PruebaADA2[{Sumatoria1, Sumatoria2}, 2000, 4]
```

Out[ ] :=
```
El primer algoritmo fue mejor: 458
El segundo algoritmo fue mejor: 333
Se comportaron igual: 1209
```

En el experimento realizado, se gestionan pruebas de ejecución iniciando en cuatro y hasta $n = 2000$. Los resultados dependen de las características locales del ordenador que se utilice, por lo que se aclara al lector, que al correr la misma instrucción se pueden obtener otros valores de rendimiento. Los resultados parecen estar a favor de considerar al método **Sumatoria1** más rápido.

Ejemplo 5.8

Compare en tiempo de ejecución los métodos **Seleccion** e **Insercion** creados en *VilCretas* para ordenar una lista de datos.

Solución:

En *Mathematica* se debe recurrir a la opción "**lista->True**" del comando **PruebaADA2**:

In[ ] :=
```
PruebaADA2[{Seleccion, Insercion}, 100, 10, lista->True]
```

Out[ ] :=
```
El primer algoritmo fue mejor: 22
El segundo algoritmo fue mejor: 18
Se comportaron igual: 60
```

N De acuerdo con distintas listas seudoaleatorias de longitud diez y hasta cien, generadas de manera automática y ordenadas por los métodos, los resultados son relativamente similares. Teóricamente lo anterior tiene sentido, pues ambos algoritmos poseen un orden asintótico $O\left(n^2\right)$.

Explicación en video

5) **PruebaADA3**: analiza la **eficiencia** de **tres algoritmos** que resuelven el mismo problema, ejecutando un **experimento** con "k" invocaciones y retornando el número de veces en el que cada método se comportó **más eficiente**, en términos de los tiempos de salida registrados por el software *Mathematica*.

Sintaxis: `PruebaADA3[{Metodo1, Metodo2, Metodo3}, k, inicio]`, o bien,

```
PruebaADA3[{Metodo1, Metodo2, Metodo3}, k, inicio, lista->True]
```

"`lista->True`" indica que los métodos reciben como parámetros una **lista** (internamente las listas son generadas de forma **seudoaleatoria**). "inicio" define el valor inicial de evaluación. Se asume el **dominio** como el conjunto de los **números naturales** o un **subconjunto** de él.

Ejemplo 5.9

Considerando los métodos **Sumatoria1**, **Sumatoria2** y **Sumatoria3** expuestos a continuación, determine de forma experimental cuál es más eficiente.

```
Sumatoria1[n_] := Module[{}, s = Sum[(j - 3) 7^j, {j, 2, n - 2}]]
Sumatoria2[n_] := Module[{suma = -49}, For[j = 3, j <= n - 2, suma =
suma + (j - 3) 7^j; j++]; suma]
Sumatoria3[n_] := If[n == 4, -49, Sumatoria3[n - 1] + 7^(-2 + n) (-5 + n)]
```

Solución:

Al declarar primero las tres funciones en el software *Mathematica*, se procede así:

In[ ] :=
```
PruebaADA3[{Sumatoria1, Sumatoria2, Sumatoria3}, 1000, 4]
```

Out[ ] :=
El primer algoritmo fue mejor: 0
El segundo algoritmo fue mejor: 0
El tercer algoritmo fue mejor: 0
El primer y segundo algoritmo fueron mejores: 130

El primer y tercer algoritmo fueron mejores: 112
El segundo y tercer algoritmo fueron mejores: 67
Se comportaron igual: 691

Los resultados apoyan la idea de considerar a **`Sumatoria1`** y **`Sumatoria2`** como los procedimientos más veloces en tiempo de ejecución. Lo anterior resulta consistente dado que **`Sumatoria3`** constituye un programa basado en una relación de recurrencia, lo cual lo caracteriza como un proceso más lento.

Ejemplo 5.10

Compare los métodos de *VilCretas* **`Seleccion`**, **`Insercion`** y **`Burbuja`** para ordenar una lista de datos.

Solución:

En el programa se corre:

In[ ] :=

```
PruebaADA3[{Insercion, Burbuja, Seleccion}, 100, 10, lista->True]
```

Out[ ] :=

El primer algoritmo fue mejor: 7
El segundo algoritmo fue mejor: 6
El tercer algoritmo fue mejor: 10
El primer y segundo algoritmo fueron mejores: 8
El primer y tercer algoritmo fueron mejores: 26
El segundo y tercer algoritmo fueron mejores: 8
Se comportaron igual: 35

En general, los tres se comportan de manera similar dado que tienen el mismo orden notacional $O\left(n^2\right)$.

Explicación en video

6) **CDFGraficaNA**: recibe **dos funciones** donde una se encuentra en el orden **notacional asintótico** de la otra (**O grande u omega**), representando **visualmente** la notación.

Sintaxis: **`CDFGraficaNA[L, cmin, cmax, graph]`** donde "L" es un **vector** que contiene a las funciones (en términos de "n"), "cmin" (real mayor que cero) y "cmax" (entero mayor o igual a 10) corresponden a los valores **mínimo** y **máximo** que es posible dar respectivamente, a la **constante** "c" de la **definición** de **notación asintótica** y "graph" (entero mayor o igual a 10) es un número que representa la **variación probable** sobre el **eje x** y el **eje y**. La instrucción genera una **animación**.

Ejemplo 5.11

Represente gráficamente: $\ln(n!) = \theta(n\ln(n))$.

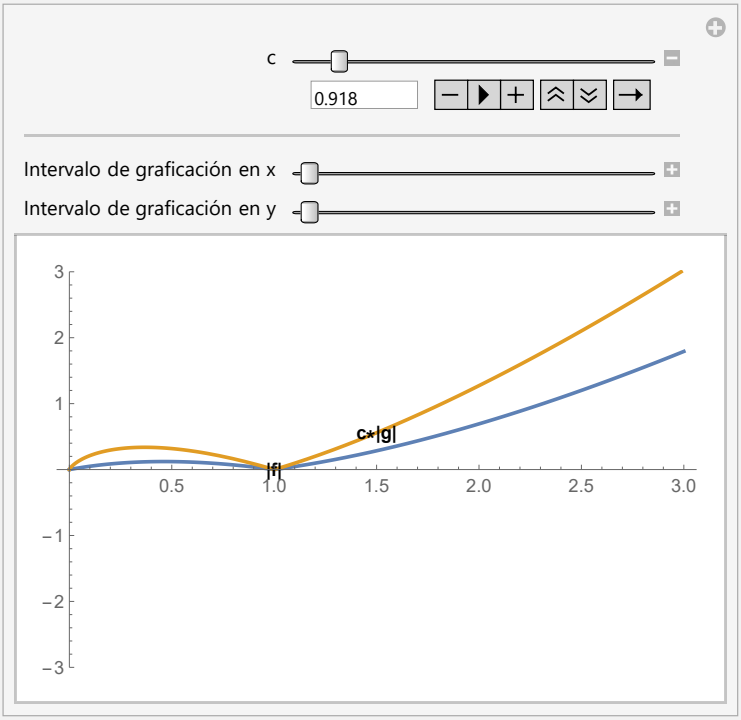
Solución:

Al recurrir a la instrucción **`CDFGraficaNA`**, se obtiene:

In[ ] :=

```
CDFGraficaNA[{Log[n!], n Log[n]}, 0.001, 10, 1000]
```

Out[ ] :=



N Se aclara al lector que el comando **`Log[n]`** corresponde a $\ln(n)$ en *Mathematica*. En la instrucción **`CDFGraficaNA`** el valor 0.001 constituye el incremento desde 0.001 hasta 10 de la constante positiva "c" en la definición de notación asintótica O u Ω. Por otra parte, 1000 es el valor máximo de graficación sobre el eje x y el eje y. Al variar el controlador "c" en la animación, se puede comprobar cómo la gráfica de $c\,|n\ln(n)|$ en algunas ocasiones está por encima y en otras por debajo, de la gráfica de $|\ln(n!)|$. Lo anterior es una verificación visual de la notación asintótica θ.

Ejemplo 5.12

Compruebe gráficamente: $6^n = O(n!)$

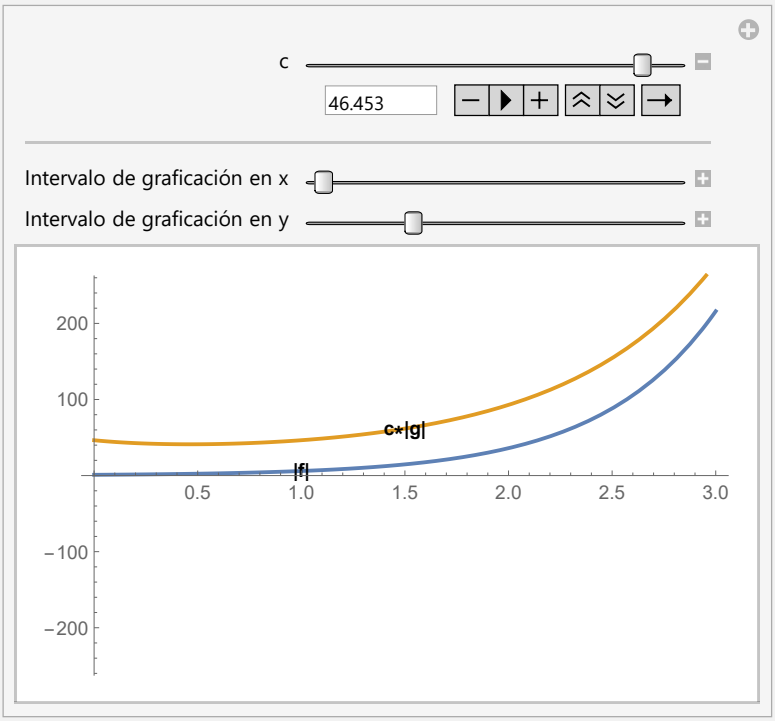
Solución:

En el software:

In[ ] :=

```
CDFGraficaNA[{6^n, n!}, 0.001, 50, 1000]
```

Out[ ] :=



En c =46.453 se satisface la notación asintótica "O grande" dado que la gráfica de la función $c\,|n!|$ está por encima de la gráfica de $|6^n|$.

N En este ejemplo, un valor máximo para la constante "c" de diez (como el asumido en el ejercicio anterior) no resulta ser conveniente con el objetivo de realizar la verificación correspondiente, por lo que se tomó el extremo 50.

Explicación en video

7) **CDFGraficaNOG**: representa **gráficamente** la relación **asintótica** "O grande" de las funciones: **ln(ln(n))**, **ln(n)**, **n** (**función identidad**), **nln(n)**, **n^2**, **n^3** y **2^n**. La instrucción muestra una **animación** donde el usuario puede cambiar los intervalos de graficación sobre cada eje coordenado.

Sintaxis: `CDFGraficaNOG[graph]`, "graph" es un número entero mayor o igual a cuarenta que representa la **variación probable** sobre el **eje x** y el **eje y**.

Ejemplo 5.13

Represente gráficamente la relación asintótica O de las funciones: $\ln(\ln(n))$, $\ln(n)$, n, $n\ln(n)$, n^2, n^3 y 2^n.

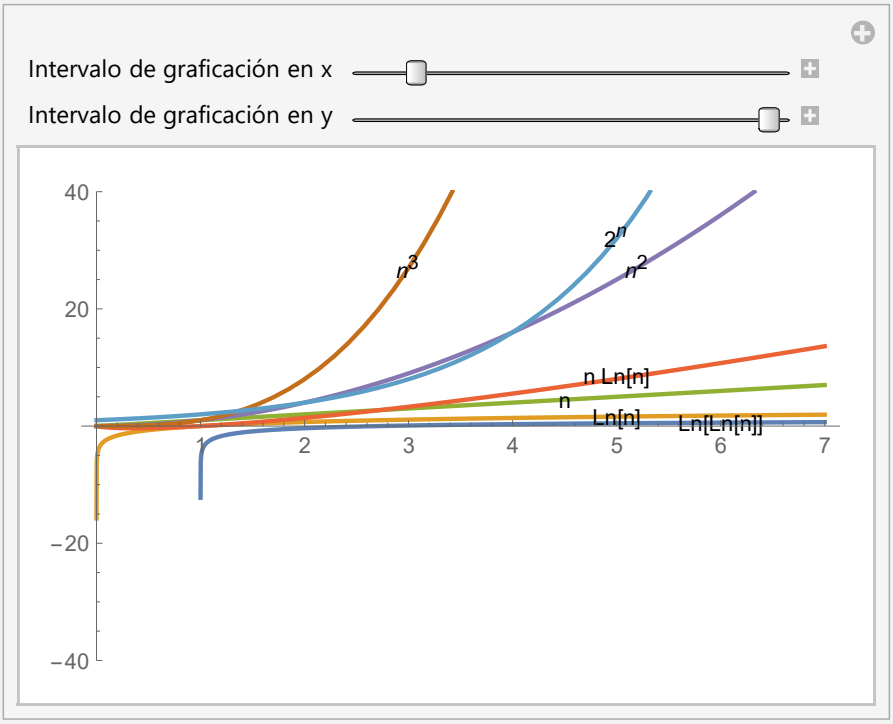
Solución:

En *Mathematica*:

In[ ] :=

```
CDFGraficaNOG[40]
```

Out[ ] :=



N El valor 40 define el intervalo de graficación máximo sobre el eje x y el eje y. La figura hace notar que los mejores órdenes "O grande" se dan en las funciones $\ln(n)$ y $\ln(\ln(n))$ al encontrarse por debajo de la función identidad.

Ejemplo 5.14

En la gráfica mostrada en el ejemplo anterior, la función 2^n aparece por debajo de n^3, sin embargo, $n^3 = O(2^n)$. Muestre esta notación asintótica con el comando **CDFGraficaNOG**.

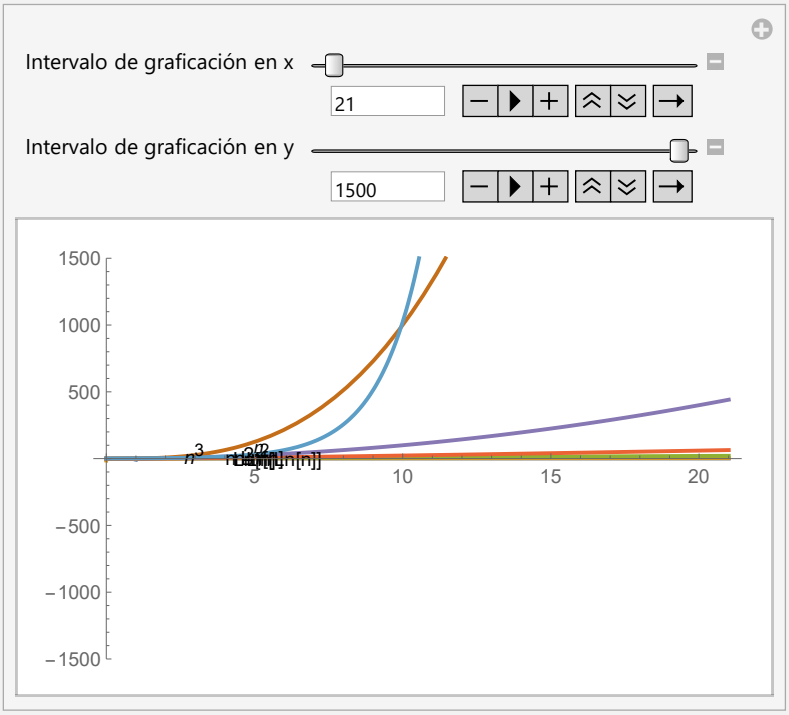
Solución:

Al cambiar el extremo máximo del intervalo de graficación a 1500, se representa la relación notacional requerida:

In[ ] :=

```
CDFGraficaNOG[1500]
```

Out[ ] :=



Explicación en video

8) **CDFGraficaNAP**: recibe **dos funciones** donde una se encuentra en el orden **notacional asintótico** de la otra (**O grande u omega**), dependiendo de un **parámetro** "j". El comando genera una **animación** con distintos valores enteros del parámetro, representando **visualmente** la notación.

Sintaxis: `CDFGraficaNAP[L, cmin, cmax, graph, jmax]`, "L" es un **vector** que contiene las funciones (en términos de "n" y "j"), "cmin" (real mayor que cero) y "cmax" (entero mayor o igual a 10) constituyen los valores **mínimo** y **máximo** que es posible dar respectivamente, a la **constante** "c" de la **definición** de **notación asintótica**, "graph" (entero mayor o igual a 10) es un número que representa la **variación probable** sobre el **eje x** y el **eje y** y finalmente, "jmax" es un entero mayor o igual a diez que define la **variación** del **parámetro** "j" en la **animación**. **No corre** funciones racionales.

Ejemplo 5.15

Verifique: $\sum_{i=0}^{j} n^i = \theta\left(n^j\right)$ con $j = 1,2,\ldots,20$.

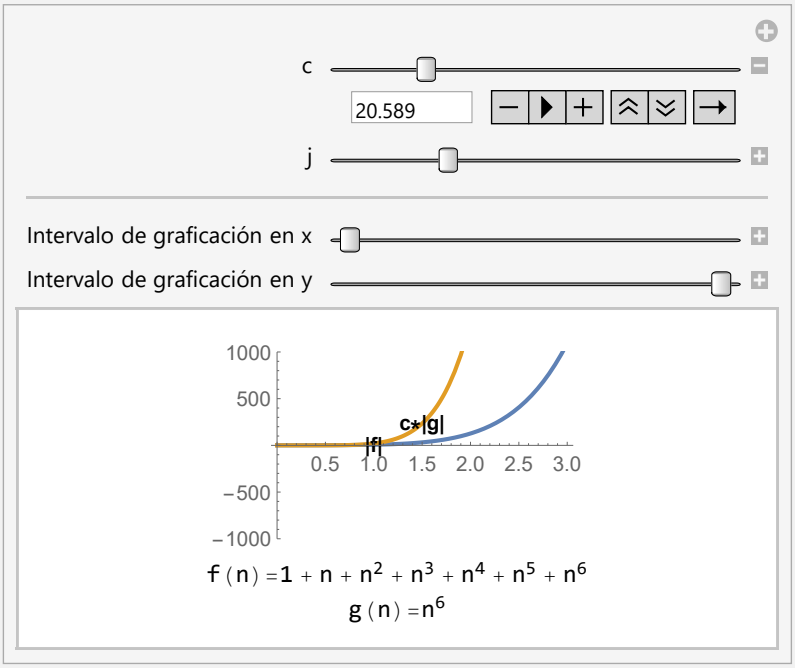
Solución:

Al emplear la instrucción **CDFGraficaNAP**:

In[ ] :=

```
CDFGraficaNAP[{Sum[n^i, {i, 0, j}], n^j}, 0.001, 100, 1000, 20]
```

Out[ ] :=



N El comando **Sum** forma parte del programa *Mathematica* y se utiliza para construir una sumatoria. En este caso, al jugar con los controladores de "c" y "j" se verifica visualmente la notación asintótica θ.

Ejemplo 5.16

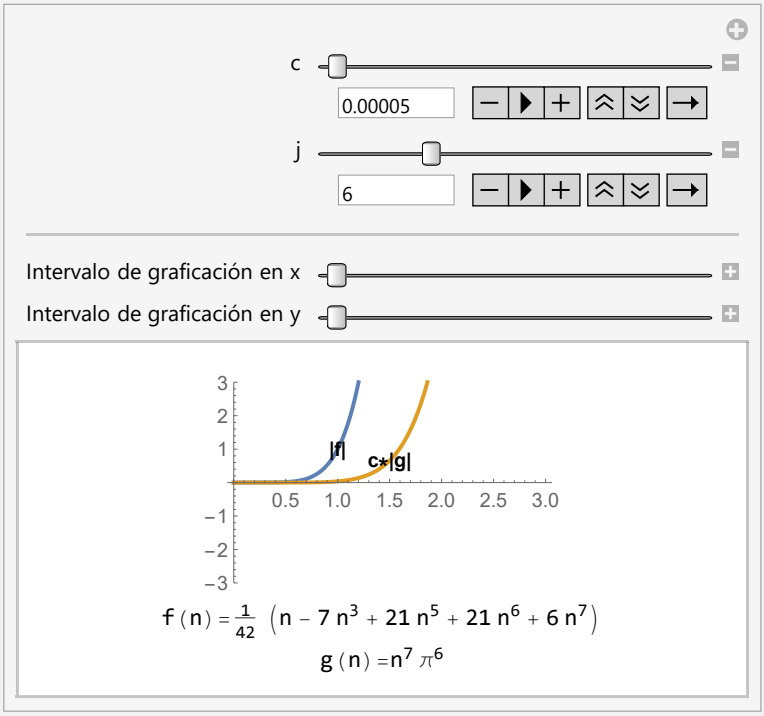
Compruebe gráficamente: $\sum_{i=1}^{n} i^j = \theta\left(\pi^j n^{j+1}\right)$ con $j = 1,2,\ldots,20$.

Solución:

In[ ] :=

```
CDFGraficaNAP[{Sum[i^j, {i, 1, n}], Pi^j n^(j + 1)}, 0.00001, 100,
1000, 20]
```

Out[ ] :=



En este ejemplo fue necesario disminuir los incrementos en el controlador "c" a 0.00001, con la intención de visualizar la notación asintótica θ.

Explicación en video

9) **CompLimit**: **compara** en el límite **dos funciones** para determinar si la primera es "theta", "O grande" u "Omega" de la segunda.

Sintaxis: **`CompLimit[L]`** con "L" un **vector** que contiene las funciones (en términos de "n" y "j"). Si alguna incluye el parámetro "j" el comando genera una **animación** haciendo variar "j" de uno a mil y mostrando el comportamiento asintótico en cada caso. La opción "**`jvalor->Valor`**" permite **cambiar** la variación por defecto mil, a cualquier otra, mayor o igual a **diez**.

Ejemplo 5.17

Demuestre mediante el uso de software y recurriendo al teorema del límite: $\log_3(n) + c = O(\ln(n))$, con c una constante real positiva.

Solución:

En *Mathematica* el comando **`CompLimit`** de *VilCretas* resuelve lo solicitado:

```
In[ ]:=
CompLimit[{Log[3, n] + c, Log[n]}]
```

Out[ ] :=

$$\frac{1}{\log(3)} \to \frac{c\log(3)+\log(n)}{\log(3)} = \Theta(\log(n)),\ \text{Notación theta}$$

La salida mostrada indica:

$$\lim_{n\to\infty} \frac{\log_3(n)+c}{\ln(n)} = \frac{1}{\ln(3)}$$

Por lo que se satisface la notación asintótica θ.

Ejemplo 5.18

Analice por comparación en el límite para cuáles valores de "j" se satisface: $f(n) = \theta(g(n))$, $f(n) = O(g(n))$ y $f(n) = \Omega(g(n))$, con $f(n) = \sum_{i=1}^{n}(i^7 + c)$ y $g(n) = 2n^j + c$, siendo c una constante real positiva.

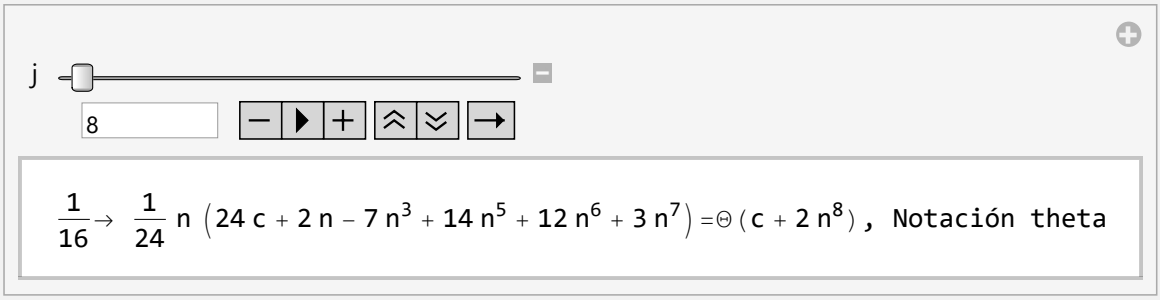
Solución:

Al utilizar el comando **CompLimit** se obtiene:

In[ ] :=

```
CompLimit[{Sum[i^7 + c, {i, 1, n}], 2 n^j + c}]
```

Out[ ] :=



Se infiere de este análisis de sensibilidad que para $j \leq 7$ la notación es Ω, para $j = 8$ la notación es θ y para $j \geq 9$ la notación asintótica presente es O.

N Por defecto el controlador "j" varía hasta mil. En este ejercicio dicho valor es desmesurado, por lo que se podría modificar haciéndolo variar hasta diez, de la siguiente manera:

In[ ] :=

```
CompLimit[{Sum[i^7 + c, {i, 1, n}], 2 n^j + c}, jvalor -> 10]
```

Out[ ] :=

j

10

$$0 \to \frac{1}{24} n \left(24 c + 2 n - 7 n^3 + 14 n^5 + 12 n^6 + 3 n^7\right) = O\left(c + 2 n^{10}\right), \text{ Notación O grande}$$

Explicación en video

10) **CalculaSuma**: ejercicio para **analizar un algoritmo** brindando la posibilidad de observar el **código** de programación con esta finalidad.

Sintaxis: `CalculaSuma[n]`, o bien, `CalculaSuma[n, code -> True]`, "`code -> True`" muestra el código de la función `CalculaSuma`.

Ejemplo 5.19

Devuelva el resultado de `CalculaSuma[800]`.

Solución:

En el software:

In[ ] :=
```
CalculaSuma[800]
```

Out[ ] :=
1280000

Ejemplo 5.20

Retorne el resultado de la función `CalculaSuma` en $n = 800$, mostrando además, el código de programación que la caracteriza.

Solución:

Al emplear la opción "`code -> True`":

In[ ] :=
```
CalculaSuma[800, code -> True]
```

Out[ ] :=
{1280000, CalculaSuma[n_]:=Module[{p=0}, For[i=1, i<=n, For[j=1, j<=n, p=p+2; j++]; i++]; Return[p]]}

Explicación en video

11) **CalculaProducto**: ejercicio para **analizar un algoritmo** brindando la posibilidad de observar el **código** de programación con esta finalidad.

Sintaxis: `CalculaProducto[n]`, o bien, `CalculaProducto[n, code -> True]`, "`code -> True`" despliega el código interno de la función.

Ejemplo 5.21

Halle la salida de `CalculaProducto[800]`.

Solución:

In[ ] :=
```
CalculaProducto[800]
```
Out[ ] :=
2187

Ejemplo 5.22

Determine el resultado de `CalculaProducto[800]`, mostrando además, el código de programación de la instrucción.

Solución:

En `CalculaProducto` se añade la opción "`code -> True`":

In[ ] :=
```
CalculaProducto[800, code -> True]
```
Out[ ] :=
{2187, CalculaProducto[n_]:=Module[{p=1}, While[p<=n, p=p*3]; Return[p]]}

Explicación en video

12) **CalculaOtroProducto**: ejercicio para **analizar un algoritmo** pudiendo visualizar su **código** de programación con este objetivo.

Sintaxis: `CalculaOtroProducto[n]`, o bien, `CalculaOtroProducto[n, code->True]`, "`code->True`" muestra el código interno de la función.

Ejemplo 5.23

Encuentre el *Out*[ ] de `CalculaOtroProducto[800]`.

Solución:

In[ ] :=
```
CalculaOtroProducto[800]
```
Out[ ] :=
2187

Ejemplo 5.24

Retorne el resultado de la función `CalculaOtroProducto` en $n = 800$, mostrando además, su código de programación.

Solución:

In[ ] :=
```
CalculaOtroProducto[800, code->True]
```
Out[ ] :=
{2187, CalculaOtroProducto[n_]:=Module[{p=1}, For[i=1, i<=n, While[p<=i, p=p*3]; i++]; Return[p]]}

Explicación en video

Aporte pedagógico

El **análisis de algoritmos** mediante el uso de **notaciones asintóticas** es una tarea docente muy complicada al emplear recursos netamente tradicionales. Los comandos de *VilCretas*: **`CDFGraficaNA`**, **`CDFGraficaNOG`** y **`CDFGraficaNAP`** proporcionan herramientas de **representación automática**, colaborando desde un punto de vista didáctico, en la explicación de la definición de las notaciones "theta", "O grande" u "Omega" o eventualmente, en la resolución de ejercicios particulares.

Aporte de investigación

Una instrucción muy interesante que permite realizar **análisis de sensibilidad** la constituye **`CompLimit`**. Este comando podría ser empleado para presentar a la población estudiantil, ejemplos donde se tenga la necesidad de realizar **inferencias** ante problemas que **no necesariamente** tienen una respuesta única.

Ejercicios

Resuelva los siguientes ejercicios utilizando como apoyo el paquete *VilCretas*.

1) Mediante el comando **Burbuja** ordene de forma descendente la lista $L = \{-9, 8, -100, 45, 9.3, 5, 10\}$. Muestre el código de programación.

2) Repita el ejercicio 1 usando las instrucciones **Seleccion** e **Insercion**.

3) Encuentre una notación asintótica para los métodos implementados en *VilCretas*: **Burbuja**, **Seleccion** e **Insercion**. Sugerencia: emplee la opción "**code -> True**".

4) Analice la complejidad de los algoritmos que caracterizan a las funciones de *VilCretas*: **CalculaSuma**, **CalculaProducto**, **CalculaOtroProducto**. Sugerencia: emplee la opción "**code -> True**" de los comandos.

5) Represente gráficamente la notación hallada en el ejercicio 3. Sugerencia: use la instrucción **CDFGraficaNA**.

6) Construya tres programas distintos que calculen cada una de las siguientes sumatorias:

   a) $\sum_{i=3}^{n-6} \frac{i-9}{i^2-8}$

   b) $\sum_{i=5}^{n+2} 3^{i-1} \ln(i)$

   Compare experimentalmente cuál de ellos es más eficiente en tiempo de ejecución. Sugerencia: los programas pueden tener una lógica recursiva, iterativa y a través del uso de la instrucción **Sum** propia del software *Mathematica*. Recurra además, a los comandos de *VilCretas*: **PrueADA2** y **PruebaADA3**.

7) La función dada a continuación es una recursividad de cola para calcular el factorial de un número natural o cero:

```
FactorialCola[n_] := Module[{OFactorialCola}, OFactorialCola[on_, m_] :=
If[on == 1, m, OFactorialCola[on - 1, on*m]]; OFactorialCola[n, 1]]
```

   Compare utilizando la sentencia **PrueADA2**, la eficiencia de **FactorialCola** con respecto al comando **Factoriales** de *VilCretas* ¿Cuál es más eficiente en tiempo de ejecución?

8) Verifique gráficamente las siguientes notaciones asintóticas, usando según corresponda, las instrucciones **CDFGraficaNA** o **CDFGraficaNAP**:

   a) $4n^3 - 3n + 2 = \Omega(n \ln n)$

   b) $n^{2n} = \Omega((2n)!)$

   c) $\ln n = \Theta(\log_2 n)$

   d) $\frac{n^2 + \ln(n)(n+1)}{n+n^2} = O(\ln n)$

e) $\sum_{i=1}^{n} \frac{1}{i} = \Theta(\ln n)$

f) $n^3 + 2n^2 + 2n + 1 = O(n^j)$ con j un entero positivo

g) $\frac{n^3}{1+2n^j} = O(n^j)$ con j un entero positivo ¿Qué ocurre cuando $j \in \{1,2,3\}$?

9) Conjeture para cuáles valores enteros positivos de j se satisfacen las notaciones asintóticas $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$ y $f(n) = \Theta(g(n))$, donde:

a) $f(n) = n^3 + 2n^2 + 2n + 1$ y $g(n) = n^j$

b) $f(n) = \frac{n^3}{1+2n^j}$ y $g(n) = n^j$

Se sugiere el empleo de **`CompLimit`**.

Capítulo 6

Relaciones binarias con *VilCretas*

El tema de relaciones binarias usualmente es abordado con un enfoque tradicional en cursos de matemática discreta. *VilCretas* pretende incorporar el uso de **treinta y seis** comandos con la intención de complementar la clase tradicional con una metodología asistida por computadora. Las instrucciones compartidas en este capítulo, permiten emprender la definición, representaciones y operaciones en relaciones binarias mediante el uso del software *Mathematica*, además, de la clasificación típica en relaciones de equivalencia y de orden parcial. Se presenta al lector un conjunto de interesantes herramientas mediante las cuales se promueve un aprendizaje interactivo en el contenido de las relaciones binarias.

1) **PC**: calcula el **producto cartesiano** entre dos conjuntos dados distintos de vacío.

Sintaxis: `PC[A, B]`, con "A" y "B" los conjuntos correspondientes.

Ejemplo 6.1

Halle el producto cartesiano entre {a, b, c} y {1, 2, 9, -7}.

Solución:

Al utilizar el comando **PC**:

In[ ] :=
```
PC[{a, b, c}, {1, 2, 9, -7}]
```

Out[ ] :=
{{a, 1}, {a, 2}, {a, 9}, {a, -7}, {b, 1}, {b, 2}, {b, 9}, {b, -7}, {c, 1}, {c, 2}, {c, 9}, {c, -7}}

N El lector debe observar cómo en *Mathematica* se representa un par ordenado mediante el uso de llaves y no de paréntesis redondos.

Ejemplo 6.2

Encuentre $A \times B$ con $A = \{2,4,\ldots,60\}$ y $B = \{1,3,\ldots,59\}$.

Solución:

En el software:

In[ ] :=
```
PC[Table[2i, {i, 1, 30}], Table[2i - 1, {i, 1, 30}]]
```

Out[ ] :=
{{2, 1}, {2, 3}, {2, 5}, {2, 7}, {2, 9}, {2, 11}, {2, 13}, {2, 15}, {2, 17}, {2, 19}, {2, 21}, {2, 23}, {2, 25}, {2, 27}, ..., {60, 47}, {60, 49}, {60, 51}, {60, 53}, {60, 55}, {60, 57}, {60, 59}}

N No se muestra el *Out* [] completo pues posee 900 pares ordenados. **`Table`** es un comando de *Mathematica* que automatiza la creación de un conjunto de datos. Por ejemplo en: **`Table[2i, {i, 1, 30}]`**, **`2i`** es la forma de los elementos (números pares) y **`{i, 1, 30}`** define la variación del parámetro **`i`** de uno a treinta.

Explicación en video

2) **GraficaPC**: **grafica** en el plano cartesiano el **producto cruz** entre **dos intervalos** del conjunto de los números reales recibidos como parámetros. Si alguno de los extremos es infinito, la instrucción grafica desde 100 (+ infinito) o -100 (- infinito) según corresponda.

Sintaxis: **`GraficaPC[{a, b}, {c, d}]`** siendo **{a, b}** el intervalo sobre el eje de las **abscisas** y **{c, d}** el intervalo sobre el eje de las **ordenadas**.

Ejemplo 6.3

Grafique: $[-9,30] \times [0,6]$.

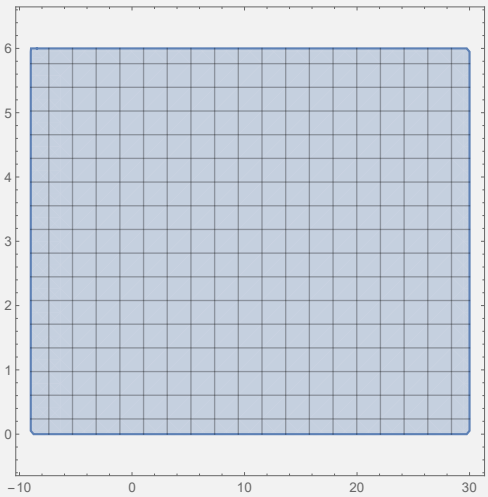
Solución:

En *Mathematica*:

In[ ] :=

```
GraficaPC[{-9, 30}, {0, 6}]
```

Out[ ] :=



Ejemplo 6.4

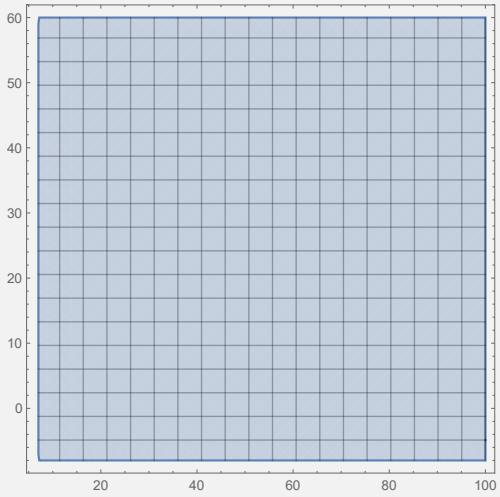
Represente en el plano cartesiano: $[7,+\infty[\times [-8,60]$.

Solución:

In[ ] :=

```
GraficaPC[{7, ∞}, {-8, 60}]
```

Out[ ] :=



Explicación en video

3) **RelBin**: **retorna** de **forma explícita** los **elementos** de una **relación binaria finita**, dada la **condición** que la define y los **conjuntos** sobre los cuáles se **construye**.

Sintaxis: `RelBin[Condicion, A, B]` con "Condicion" un "string" que contiene la condición de creación de la relación binaria (en función de "a" y "b" haciendo referencia al par ordenado "(a, b)") y, "A" y "B" los conjuntos que la **determinan**.

Ejemplo 6.5

Halle de forma explícita los pares ordenados de la relación binaria: $aRb \Leftrightarrow a^2 \geq b$ con $a, b \in A = \{1,3,5,\ldots,19\}$.

Solución:

Al emplear el comando **RelBin**:

In[ ] :=

```
A = Table[2i - 1, {i, 1, 10}];
RelBin["a^2>=b", A, A]
```

Out[ ] :=

{{1, 1}, {3, 1}, {3, 3}, {3, 5}, {3, 7}, {3, 9}, {5, 1}, {5, 3}, {5, 5}, {5, 7}, {5, 9}, {5, 11}, {5, 13}, {5, 15}, {5, 17}, {5, 19}, {7, 1}, {7, 3}, {7, 5}, {7, 7}, {7, 9}, {7, 11}, {7, 13}, {7, 15}, {7, 17}, {7, 19}, {9, 1}, {9, 3}, {9, 5}, {9, 7}, {9, 9}, {9, 11}, {9, 13}, {9, 15}, {9, 17}, {9, 19}, {11, 1}, {11, 3}, {11, 5}, {11, 7}, {11, 9}, {11, 11}, {11, 13}, {11, 15}, {11, 17}, {11, 19}, {13, 1}, {13, 3}, {13, 5}, {13, 7}, {13, 9}, {13, 11}, {13, 13}, {13, 15}, {13, 17}, {13, 19}, {15, 1}, {15, 3}, {15, 5}, {15, 7}, {15, 9}, {15, 11}, {15, 13}, {15, 15}, {15, 17}, {15, 19}, {17, 1}, {17, 3}, {17, 5}, {17, 7}, {17, 9}, {17, 11}, {17, 13}, {17, 15}, {17, 17}, {17, 19}, {19, 1}, {19, 3}, {19, 5}, {19, 7}, {19, 9}, {19, 11}, {19, 13}, {19, 15}, {19, 17}, {19, 19}}

N Es importante mencionar que el "**;**" después del **Table**, se ha colocado en *Mathematica* para crear el conjunto **A** en la sesión de trabajo, sin mostrarlo como salida.

Ejemplo 6.6

Encuentre los pares ordenados de una relación binaria R definida sobre $A = \{2,4,6,\ldots,100\}$ donde: $aRb \Leftrightarrow \log_b a$ es un número entero.

Solución:

En *Mathematica*:

In[ ] :=

```
A = Table[2i, {i, 1, 50}];
RelBin["IntegerQ[Log[b,a]]", A, A]
```

Out[ ] :=

{{2, 2}, {4, 2}, {4, 4}, {6, 6}, {8, 2}, {8, 8}, {10, 10}, {12, 12}, {14, 14}, {16, 2}, {16, 4}, {16, 16}, {18, 18}, {20, 20}, {22, 22}, {24, 24}, {26, 26}, {28, 28}, {30, 30}, {32, 2}, {32, 32}, {34, 34}, {36, 6}, {36, 36}, {38, 38}, {40, 40}, {42, 42}, {44, 44}, {46, 46}, {48, 48}, {50, 50}, {52, 52}, {54, 54}, {56, 56}, {58, 58}, {60, 60}, {62, 62}, {64, 2}, {64, 4}, {64, 8}, {64, 64}, {66, 66}, {68, 68}, {70, 70}, {72, 72}, {74, 74}, {76, 76}, {78, 78}, {80, 80}, {82, 82}, {84, 84}, {86, 86}, {88, 88}, {90, 90}, {92, 92}, {94, 94}, {96, 96}, {98, 98}, {100, 10}, {100, 100}}

N **IntegerQ** es una función booleana del software *Mathematica* que retorna "**True**" si su argumento es un número entero y "**False**", en caso contrario.

Explicación en video

4) **GraficaRelBin**: **grafica** los **pares ordenados** de una **relación binaria** definida por una **ecuación**, o bien, una **inecuación**.

Sintaxis: `GraficaRelBin[Condicion, xmax, ymax]`, siendo "Condicion" un "string" que contiene la **ecuación** o **inecuación** respectiva (en función de "a" y "b" haciendo referencia al par ordenado "(a, b)") y, "xmax" y "ymax" los **extremos máximos** de **graficación** sobre el eje "x" y el eje "y", respectivamente. Por defecto, el **valor mínimo** de **graficación** corresponde a -10. El usuario puede cambiar este número al incluir las opciones "`xmin->Valor`" o "`ymin->Valor`".

Ejemplo 6.7

Grafique los pares ordenados de la relación binaria: $aRb \Leftrightarrow \frac{a^2}{25} + \frac{b^2}{4} \geq 8$.

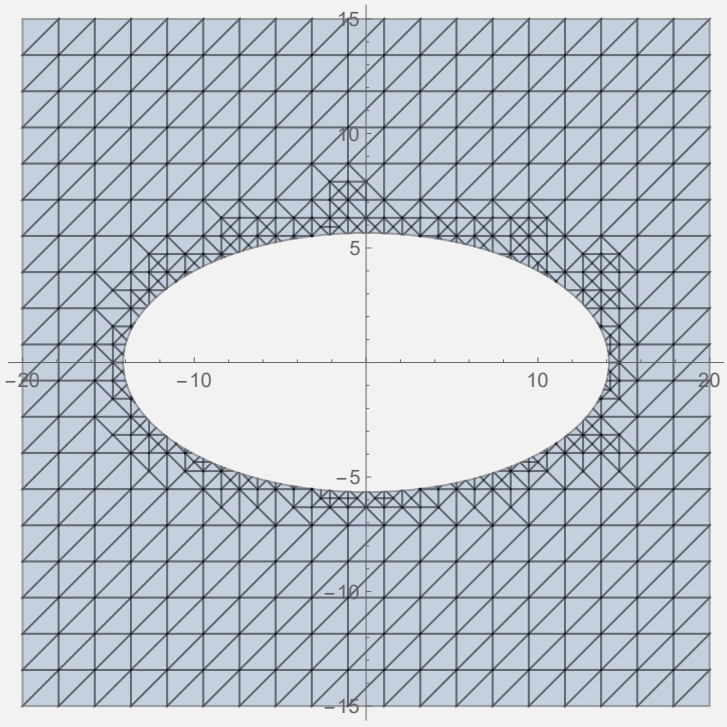
Solución:

Al utililizar el comando **`GraficaRelBin`** y sus opciones "xmin" y "ymin" se obtiene:

In[ ] :=

```
GraficaRelBin[a^2/25+b^2/4>=8, 20, 15, xmin->-20, ymin->-15]
```

Out[ ] :=



Ejemplo 6.8

Grafique en el plano cartesiano la relación binaria definida por: $aRb \Leftrightarrow \frac{(a-9)^2}{25} + \frac{b}{4} = 8$.

Solución:

In[ ] :=

```
GraficaRelBin[(a-9)^2/25+b/4==8, 30, 50]
```

Out[ ] :=

Explicación en video

5) **GraficaRelBinPares**: **grafica** en el **plano cartesiano** los **pares ordenados** de una **relación binaria** dada de forma **explícita**.

Sintaxis: `GraficaRelBinPares[R]`, siendo "R" un conjunto de **pares ordenados** que definen la **relación**.

Ejemplo 6.9

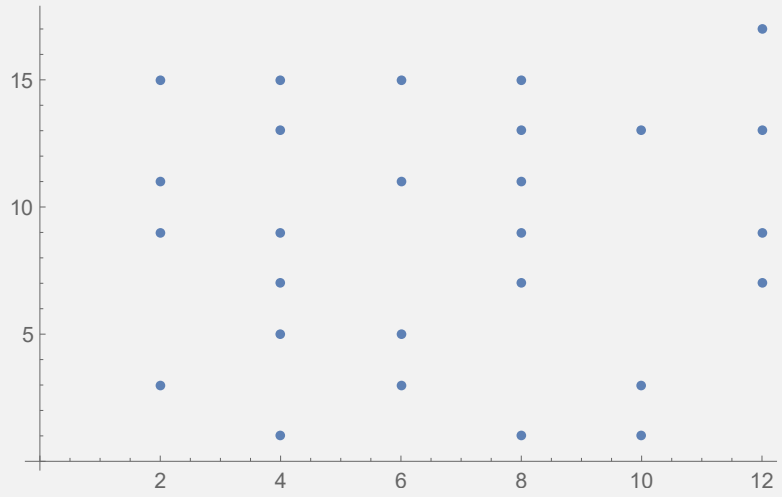
Grafique la relación binaria $R = \{(2, 3), (2, 9), (2, 11), (2, 15), (4, 1), (4, 5), (4, 7), (4, 9), (4, 13), (4, 15), (6, 3), (6, 5), (6, 11), (6, 15), (8, 1), (8, 7), (8, 9), (8, 11), (8, 13), (8, 15), (10, 1), (10, 3), (10, 13), (12, 7), (12, 9), (12, 13), (12, 17)\}$.

Solución:

In[ ] :=

```
R = {{2, 3}, {2, 9}, {2, 11}, {2, 15}, {4, 1}, {4, 5}, {4, 7}, {4,
9}, {4, 13}, {4, 15}, {6, 3}, {6, 5}, {6, 11}, {6, 15}, {8, 1}, {8,
7}, {8, 9}, {8, 11}, {8, 13}, {8, 15}, {10, 1}, {10, 3}, {10, 13},
{12, 7}, {12, 9}, {12, 13}, {12, 17}};
GraficaRelBinPares[R]
```

Out[ ] :=



Ejemplo 6.10

Represente en el plano cartesiano la relación binaria $R = \{(2, 10), (8, 8), (10, 8), (10, 10), (12, 12), (14, 12), (14, 14), (16, 12), (16, 14), (16, 16), (18, 12), (18, 14), (18, 16), (18, 18), (20, 12), (20, 14), (20, 16), (20, 18), (20, 20), (22, 12), (22, 14), (22, 16), (22, 18), (22, 20), (22, 22), (24, 12), (24, 14), (24, 16), (24, 18), (24, 20), (24, 22), (24, 24), (26, 12), (26, 14), (26, 16), (26, 18), (26, 20), (26, 22), (26, 24), (26, 26), (28, 12), (28, 14), (28, 16), (28, 18), (28, 20), (28, 22), (28, 24), (28, 26), (28, 28), (30, 12), (30, 14), (30, 16), (30, 18), (30, 20), (30, 22), (30, 24), (30, 26), (30, 28), (30, 30), (32, 12), (32, 14), (32, 16), (32, 18), (32, 20), (32, 22), (32, 24), (32, 26), (32, 28), (32, 30), (32, 32), (34, 12), (34, 14), (34, 16), (34, 18), (34, 20), (34, 22), (34, 24), (34, 26), (34, 28), (34, 30), (34, 32), (34, 34), (36, 12), (36, 14), (36, 16), (36, 18), (36, 20), (36, 22), (36, 24), (36, 26), (36, 28), (36, 30), (36, 32), (36, 34), (36, 36), (38, 12), (38, 14), (38, 16), (38, 18), (38, 20), (38, 22), (38, 24), (38, 26), (38, 28), (38, 30), (38, 32), (38, 34), (38, 36), (38, 38), (40, 2), (40, 4), (40, 12), (40, 14), (40, 16), (40, 18), (40, 20), (40, 22), (40, 24), (40, 26), (40, 28), (40, 30), (40, 32), (40, 34), (40, 36), (40, 38), (40, 40)\}$.

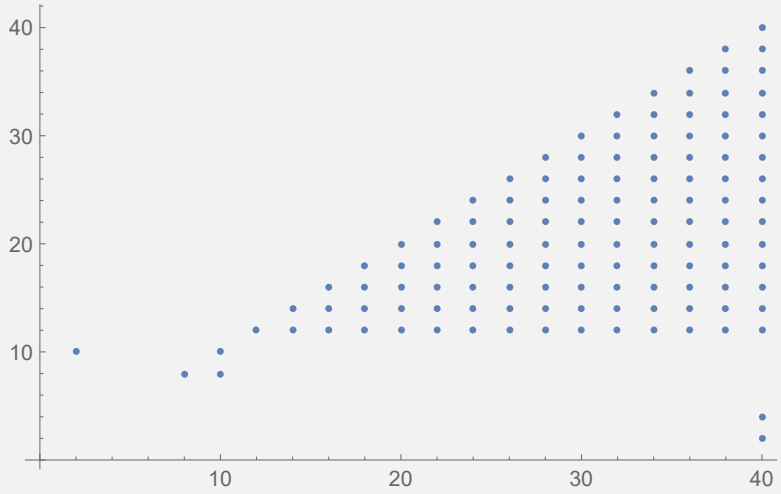
Solución:

In[ ] :=

```
R = {{2, 10}, {8, 8}, {10, 8}, {10, 10}, {12, 12}, {14, 12}, {14,
14}, {16, 12}, {16, 14}, {16, 16}, {18, 12}, {18, 14}, {18, 16}, {18,
18}, {20, 12}, {20, 14}, {20, 16}, {20, 18}, {20, 20}, {22, 12}, {22,
14}, {22, 16}, {22, 18}, {22, 20}, {22, 22}, {24, 12}, {24, 14}, {24,
16}, {24, 18}, {24, 20}, {24, 22}, {24, 24}, {26, 12}, {26, 14}, {26,
16}, {26, 18}, {26, 20}, {26, 22}, {26, 24}, {26, 26}, {28, 12}, {28,
14}, {28, 16}, {28, 18}, {28, 20}, {28, 22}, {28, 24}, {28, 26}, {28,
28}, {30, 12}, {30, 14}, {30, 16}, {30, 18}, {30, 20}, {30, 22}, {30,
24}, {30, 26}, {30, 28}, {30, 30}, {32, 12}, {32, 14}, {32, 16}, {32,
18}, {32, 20}, {32, 22}, {32, 24}, {32, 26}, {32, 28}, {32, 30}, {32,
32}, {34, 12}, {34, 14}, {34, 16}, {34, 18}, {34, 20}, {34, 22}, {34,
24}, {34, 26}, {34, 28}, {34, 30}, {34, 32}, {34, 34}, {36, 12}, {36,
14}, {36, 16}, {36, 18}, {36, 20}, {36, 22}, {36, 24}, {36, 26}, {36,
28}, {36, 30}, {36, 32}, {36, 34}, {36, 36}, {38, 12}, {38, 14}, {38,
```

```
16}, {38, 18}, {38, 20}, {38, 22}, {38, 24}, {38, 26}, {38, 28}, {38,
30}, {38, 32}, {38, 34}, {38, 36}, {38, 38}, {40, 2}, {40, 4}, {40,
12}, {40, 14}, {40, 16}, {40, 18}, {40, 20}, {40, 22}, {40, 24}, {40,
26}, {40, 28}, {40, 30}, {40, 32}, {40, 34}, {40, 36}, {40, 38}, {40,
40}};
GraficaRelBinPares[R]
```

Out[ ] :=



Explicación en video

6) **ElementRelBinQ**: función boolena que **determina** si un **par ordenado** "{a, b}" es un **elemento** de una **relación binaria** dada de manera **explícita**, o bien, mediante una **ecuación** o **inecuación**.

Sintaxis: `ElementRelBinQ[R, {a, b}]`, si la relación binaria "R" está dada de forma **explícita**, o,

```
ElementRelBinQ[R, {a, b}, expalgebra->True]
```

en caso de que la relación binaria dependa de una **ecuación** o **inecuación**. En "`expalgebra -> True`", "R" debe ser un "string" que contiene la expresión algebraica.

Ejemplo 6.11

Determine a través del uso de software, si el par $(3,4)$ pertemece a la relación binaria: $aRb \Leftrightarrow$ el máximo común divisor entre a y b es igual a uno, definida sobre $A = \{1,3,5,7\}$ y $B = \{2,4,6,8\}$.

Solución:

En *Mathematica*:

In[ ] :=

```
A = {1, 3, 5, 7};
B = {2, 4, 6, 8};
ElementRelBinQ[RelBin["GCD[a, b]==1", A, B], {3, 4}]
```

Out[ ] :=
True

N **RelBin** devuelve todos los pares ordenados de la relación R, además, el comando **GCD** es propio del software y encuentra el máximo común divisor.

Ejemplo 6.12

Sea la relación binaria dada por: $aRb \Leftrightarrow \frac{(x-9)^2}{25} + \frac{y}{4} \geq 8$ ¿El par ordenado $(4,1)$ pertenece a R?

Solución:

Como la relación binaria se define mediante una inecuación, se emplea la opción "**expalgebra -> True**" de **ElementRelBinQ**. Luego:

In[ ] :=
```
ElementRelBinQ[(x-9)^2/25+y/4>=8, {4, 1}, expalgebra -> True]
```

Out[ ] :=
False

Explicación en video

7) **MatrizRelBin**: construye la **matriz** asociada a una **relación binaria** definida sobre dos conjuntos finitos "A" y "B" distintos de vacío.

Sintaxis: **MatrizRelBin[R, A, B]** con "R" la relación binaria en su **forma explícita**. El comando dota al usuario de la **opción** "**table -> False**" para devolver la **matriz sin forma de tabla**.

Ejemplo 6.13

Represente por medio de una matriz la relación binaria: $aRb \Leftrightarrow a \geq b$, con $a, b \in A = \{1,2,3,\ldots,20\}$.

Solución:

Al utilizar el comando **MatrizRelBin**:

In[ ] :=
```
A = Table[i, {i, 1, 20}];
```

```
MatrizRelBin[RelBin["a>=b", A, A], A, A]
```

Out[ ] :=

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}$$

(N) La instrucción **RelBin** permite en este ejemplo construir explícitamente la relación R.

Ejemplo 6.14

Genere la matriz asociada a la relación binaria: $aRb \Leftrightarrow \log_b a$ es un número entero, con $a, b \in A = \{2,4,6,\ldots,100\}$.

Solución:

En *Mathematica*:

In[ ] :=

```
A = Table[2i, {i, 1, 50}];
MatrizRelBin[RelBin["IntegerQ[Log[b,a]]", A, A], A, A]
```

La salida no se muestra pues la matriz es de tamaño 50 por 50.

Explicación en video

8) **RelBinMatriz**: retorna los **pares ordenados** de una **relación binaria** definida sobre dos conjuntos finitos "A" y "B" distintos de vacío, dada su **matriz de reprensentación**.

Sintaxis: `RelBinMatriz[M, A, B]` siendo "M" la matriz de la relación binaria.

Ejemplo 6.15

Considere la siguiente matriz:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Si representa una relación binaria R definida sobre $A = \{3,5,7,\ldots,21\}$, encuentre mediante el uso de software, los pares ordenados que conforman a R.

Solución:

Al emplear el comando **`RelBinMatriz`** se obtiene:

In[ ] :=

```
A = Table[2i + 1, {i, 1, 10}];
```

```
RelBinMatriz[ (matriz identidad 10×10) , A, A]
```

Out[ ] :=

{{3, 3}, {5, 5}, {7, 7}, {9, 9}, {11, 11}, {13, 13}, {15, 15}, {17, 17}, {19, 19}, {21, 21}}

N Una matriz en *Mathematica* se puede crear en el menú: *Insertar/ Tabla/Matriz /Nuevo*.

Ejemplo 6.16

Sea la relación binaria R definida sobre $A = \{2,4,6,\ldots,40\}$, representada por la matriz:

$$\begin{pmatrix}
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0\\
1&1&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

Encuentre explícitamente a R.

Solución:

En el software:

In[ ] :=

```
A = Table[2i, {i, 1, 20}];
RelBinMatriz[
```

$$\begin{pmatrix}
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0\\
1&1&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

```
, A,
A]
```

Out[ ] :=

{{2, 10}, {8, 8}, {10, 8}, {10, 10}, {12, 12}, {14, 12}, {14, 14}, {16, 12}, {16, 14}, {16, 16}, {18, 12}, {18, 14}, {18, 16}, {18, 18}, {20, 12}, {20, 14}, {20, 16}, {20, 18}, {20, 20}, {22, 12}, {22, 14}, {22, 16}, {22, 18}, {22, 20}, {22, 22}, {24, 12}, {24, 14}, {24, 16}, {24, 18}, {24, 20}, {24, 22}, {24, 24}, {26, 12}, {26, 14}, {26, 16}, {26, 18}, {26, 20}, {26, 22}, {26, 24}, {26, 26}, {28, 12}, {28, 14}, {28, 16}, {28, 18}, {28, 20}, {28, 22}, {28, 24}, {28, 26}, {28, 28}, {30, 12}, {30, 14}, {30, 16}, {30, 18}, {30, 20}, {30, 22}, {30, 24}, {30, 26}, {30, 28}, {30, 30}, {32, 12}, {32, 14}, {32, 16}, {32, 18}, {32, 20}, {32, 22}, {32, 24}, {32, 26}, {32, 28}, {32, 30}, {32, 32}, {34, 12}, {34, 14}, {34, 16}, {34, 18}, {34, 20}, {34, 22}, {34, 24}, {34, 26}, {34, 28}, {34, 30}, {34, 32}, {34, 34}, {36, 12}, {36, 14}, {36, 16}, {36, 18}, {36, 20}, {36, 22}, {36, 24}, {36, 26}, {36, 28}, {36, 30}, {36, 32}, {36, 34}, {36, 36}, {38, 12}, {38, 14}, {38, 16}, {38, 18}, {38, 20}, {38, 22}, {38, 24}, {38, 26}, {38, 28}, {38, 30}, {38, 32}, {38, 34}, {38, 36}, {38, 38}, {40, 2}, {40, 4}, {40, 12}, {40, 14}, {40, 16}, {40, 18}, {40, 20}, {40, 22}, {40, 24}, {40, 26}, {40, 28}, {40, 30}, {40, 32}, {40, 34}, {40, 36}, {40, 38}, {40, 40}}

En este ejercicio el gráfico de R posee 126 pares ordenados.

Explicación en video

9) **GrafoRelBin**: **representa** mediante un **grafo** una **relación binaria** sobre un conjunto finito no vacío "A", recibida como parámetro a través de los pares ordenados que la conforman.

Sintaxis: `GrafoRelBin[R, A]` con "R" la relación binaria **explícita**. La instrucción **retorna** un **digrafo** solamente en los casos en los que la relación **no es simétrica**.

Ejemplo 6.17

Sea $R = \{(1, 5), (2, 4), (3, 3), (3, 4), (3, 5), (4, 2), (4, 3), (4, 4), (4, 5), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5)\}$. Determine un grafo que represente la relación binaria dada.

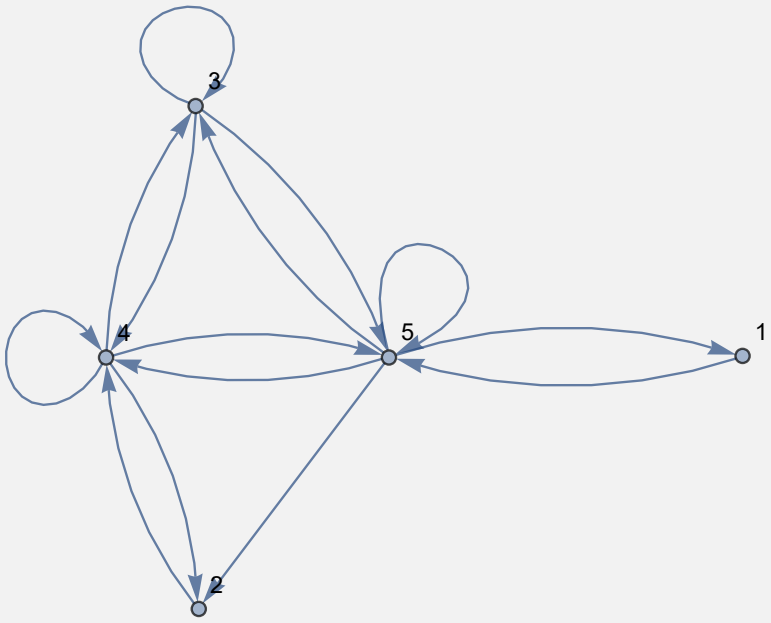
Solución:

La instrucción **`GrafoRelBin`** resuelve lo solicitado. Al observar los pares ordenados del enunciado, se infiere que R está definida sobre $A = \{1,2,3,4,5\}$:

In[ ] :=

```
A = {1, 2, 3, 4, 5};
GrafoRelBin[{{1, 5}, {2, 4}, {3, 3}, {3, 4}, {3, 5}, {4, 2}, {4, 3},
{4, 4}, {4, 5}, {5, 1}, {5, 2}, {5, 3}, {5, 4}, {5, 5}}, A]
```

Out[ ] :=



Como la relación no es simétrica la instrucción genera un digrafo.

Ejemplo 6.18

Considere la relación binaria: $aRb \Leftrightarrow a^2 \geq (b-a)^2$ con $a, b \in A = \{1,3,5,\ldots,19\}$. Construya un grafo que represente a R.

Solución:

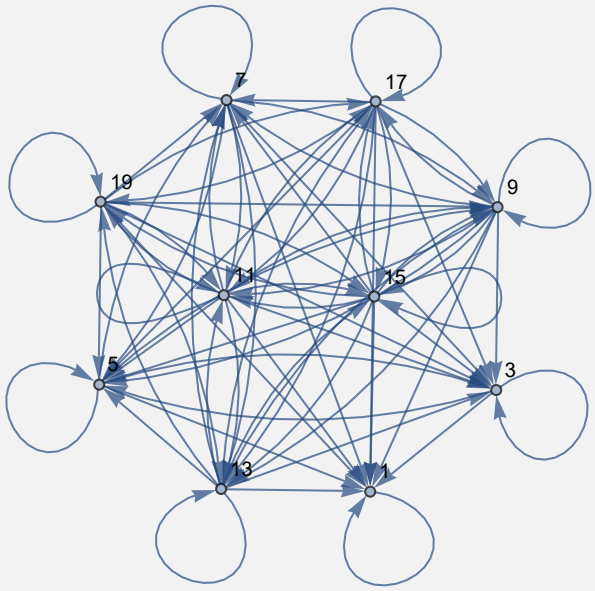
En el software:

In[ ] :=

```
A = Table[2i - 1, {i, 1, 10}];
RelBin["a^2>=(b-a)^2", A, A]
GrafoRelBin[RelBin["a^2>=(b-a)^2", A, A], A]
```

Out[ ] :=

{{1, 1}, {3, 1}, {3, 3}, {3, 5}, {5, 1}, {5, 3}, {5, 5}, {5, 7}, {5, 9}, {7, 1}, {7, 3}, {7, 5}, {7, 7}, {7, 9}, {7, 11}, {7, 13}, {9, 1}, {9, 3}, {9, 5}, {9, 7}, {9, 9}, {9, 11}, {9, 13}, {9, 15}, {9, 17}, {11, 1}, {11, 3}, {11, 5}, {11, 7}, {11, 9}, {11, 11}, {11, 13}, {11, 15}, {11, 17}, {11, 19}, {13, 1}, {13, 3}, {13, 5}, {13, 7}, {13, 9}, {13, 11}, {13, 13}, {13, 15}, {13, 17}, {13, 19}, {15, 1}, {15, 3}, {15, 5}, {15, 7}, {15, 9}, {15, 11}, {15, 13}, {15, 15}, {15, 17}, {15, 19}, {17, 1}, {17, 3}, {17, 5}, {17, 7}, {17, 9}, {17, 11}, {17, 13}, {17, 15}, {17, 17}, {17, 19}, {19, 1}, {19, 3}, {19, 5}, {19, 7}, {19, 9}, {19, 11}, {19, 13}, {19, 15}, {19, 17}, {19, 19}}



N En este ejemplo, además del uso de **`GrafoRelBin`**, se añadió antes la instrucción **`RelBin`** para mostrar explícitamente los pares ordenados que determinan la relación binaria R.

Explicación en video

10) **RelBinOperaciones**: recibe como parámetros los **pares ordenados** que constituyen a **dos relaciones binarias** y los **conjuntos** "A" y "B" sobre los que se definen. El comando **retorna** los **resultados** de las **operaciones** "unión", "intersección", "relación complementaria", "relación inversa" y "composición".

Sintaxis: **`RelBinOperaciones[R1, R2, A, B]`** siendo "R1" y "R2" las relaciones correspondientes.

Ejemplo 6.19

Sean $R_1 = \{(1, 1), (1, 2), (2, 1), (2, 2), (2, 1), (3, 1), (4, 12), (30, 8)\}$ y $R_2 = \{(a,b) \mid a + b \leq 10\}$ definida sobre $A = \{1,2,3,\ldots,30\}$. Encuentre las relaciones: $R_1 \cup R_2$, $R_1 \cap R_2$, $\overline{R_1}$, $\overline{R_2}$, R_1^{-1}, R_2^{-1}, $R_1 o R_2$ y $R_2 o R_1$.

Solución:

Al utilizar el comando **RelBinOperaciones**:

In[ ] :=

```
A = Table[i, {i, 1, 30}];
R1 = {{1, 1}, {1, 2}, {2, 1}, {2, 2}, {2, 1}, {3, 1}, {4, 12}, {30,
8}};
R2 = RelBin["a+b<=10", A, A]
RelBinOperaciones[R1, R2, A, A]
```

Out[ ] :=

{{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 7}, {1, 8}, {1, 9}, {2, 1}, {2, 2}, {2, 3}, {2, 4}, {2, 5}, {2, 6}, {2, 7}, {2, 8}, {3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5}, {3, 6}, {3, 7}, {4, 1}, {4, 2}, {4, 3}, {4, 4}, {4, 5}, {4, 6}, {5, 1}, {5, 2}, {5, 3}, {5, 4}, {5, 5}, {6, 1}, {6, 2}, {6, 3}, {6, 4}, {7, 1}, {7, 2}, {7, 3}, {8, 1}, {8, 2}, {9, 1}}
La unión corresponde a: {{1, 1}, {1, 2}, {1, 3}, ..., {9, 1}, {30, 8}}
La intersección corresponde a: {{1, 1}, {1, 2}, {2, 1}, {2, 2}, {3, 1}}
La relación complementaria de la primera es: {{1, 3}, {1, 4}, {1, 5}, ..., {30, 29}, {30, 30}}
La relación complementaria de la segunda es: {{1, 10}, {1, 11}, {1, 12}, ..., {30, 29}, {30, 30}}
La relación inversa de la primera es: {{1, 1}, {1, 2}, {1, 3}, {2, 1}, {2, 2}, {8, 30}, {12, 4}}
La relación inversa de la segunda es: {{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 7}, {1, 8}, {1, 9}, {2, 1}, {2, 2}, {2, 3}, {2, 4}, {2, 5}, {2, 6}, {2, 7}, {2, 8}, {3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5}, {3, 6}, {3, 7}, {4, 1}, {4, 2}, {4, 3}, {4, 4}, {4, 5}, {4, 6}, {5, 1}, {5, 2}, {5, 3}, {5, 4}, {5, 5}, {6, 1}, {6, 2}, {6, 3}, {6, 4}, {7, 1}, {7, 2}, {7, 3}, {8, 1}, {8, 2}, {9, 1}}
La relación composición de la primera con respecto a la segunda es: {{1, 1}, {1, 2}, {1, 12}, {2, 1}, {2, 2}, {2, 12}, {3, 1}, {3, 2}, {3, 12}, {4, 1}, {4, 2}, {4, 12}, {5, 1}, {5, 2}, {5, 12}, {6, 1}, {6, 2}, {6, 12}, {7, 1}, {7, 2}, {8, 1}, {8, 2}, {9, 1}, {9, 2}}
La relación composición de la segunda con respecto a la primera es: {{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 7}, {1, 8}, {1, 9}, {2, 1}, {2, 2}, {2, 3}, {2, 4}, {2, 5}, {2, 6}, {2, 7}, {2, 8}, {2, 9}, {3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5}, {3, 6}, {3, 7}, {3, 8}, {3, 9}, {30, 1}, {30, 2}}

No se muestra el *Out* [ ] completo por su tamaño.

Ejemplo 6.20

Determine: $R_1 \cup R_2$, $R_1 \cap R_2$, $\overline{R_1}$, $\overline{R_2}$, R_1^{-1}, R_2^{-1}, $R_1 o R_2$ y $R_2 o R_1$, con $R_1 = \{(a,b) \mid a - b \leq -10\}$ y $R_2 = \{(a,b) \mid a + b \geq 30\}$ definidas sobre $A = \{1,2,3,\ldots,20\}$.

Solución:

En *Mathematica*:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
R1 = RelBin["a-b<=-10", A, A]
R2 = RelBin["a+b>=30", A, A]
RelBinOperaciones[R1, R2, A, A]
```

Out[ ] :=

{{1, 11}, {1, 12}, {1, 13}, {1, 14}, {1, 15}, {1, 16}, {1, 17}, {1, 18}, {1, 19}, {1, 20}, {2, 12}, {2, 13}, {2, 14}, {2, 15}, {2, 16}, {2, 17}, {2, 18}, {2, 19}, {2, 20}, {3, 13}, {3, 14}, {3, 15}, {3, 16}, {3, 17}, {3, 18}, {3, 19}, {3, 20},

{4, 14}, {4, 15}, {4, 16}, {4, 17}, {4, 18}, {4, 19}, {4, 20}, {5, 15}, {5, 16}, {5, 17}, {5, 18}, {5, 19}, {5, 20}, {6, 16}, {6, 17}, {6, 18}, {6, 19}, {6, 20}, {7, 17}, {7, 18}, {7, 19}, {7, 20}, {8, 18}, {8, 19}, {8, 20}, {9, 19}, {9, 20}, {10, 20}}
{{10, 20}, {11, 19}, {11, 20}, {12, 18}, {12, 19}, {12, 20}, {13, 17}, {13, 18}, {13, 19}, {13, 20}, {14, 16}, {14, 17}, {14, 18}, {14, 19}, {14, 20}, {15, 15}, {15, 16}, {15, 17}, {15, 18}, {15, 19}, {15, 20}, {16, 14}, {16, 15}, {16, 16}, {16, 17}, {16, 18}, {16, 19}, {16, 20}, {17, 13}, {17, 14}, {17, 15}, {17, 16}, {17, 17}, {17, 18}, {17, 19}, {17, 20}, {18, 12}, {18, 13}, {18, 14}, {18, 15}, {18, 16}, {18, 17}, {18, 18}, {18, 19}, {18, 20}, {19, 11}, {19, 12}, {19, 13}, {19, 14}, {19, 15}, {19, 16}, {19, 17}, {19, 18}, {19, 19}, {19, 20}, {20, 10}, {20, 11}, {20, 12}, {20, 13}, {20, 14}, {20, 15}, {20, 16}, {20, 17}, {20, 18}, {20, 19}, {20, 20}}
La unión corresponde a: { {1, 11}, {1, 12}, {1, 13}, ..., {20, 19}, {20, 20}}
La intersección corresponde a: {{10, 20}}
La relación complementaria de la primera es: {{1, 1}, {1, 2}, {1, 3}, {1, 4}, ..., {20, 19}, {20, 20}}
La relación complementaria de la segunda es: {{1, 1}, {1, 2}, {1, 3}, ..., {20, 8}, {20, 9}}
La relación inversa de la primera es: {{11, 1}, {12, 1}, {12, 2}, {13, 1}, {13, 2}, {13, 3}, {14, 1}, {14, 2}, {14, 3}, {14, 4}, {15, 1}, {15, 2}, {15, 3}, {15, 4}, {15, 5}, {16, 1}, {16, 2}, {16, 3}, {16, 4}, {16, 5}, {16, 6}, {17, 1}, {17, 2}, {17, 3}, {17, 4}, {17, 5}, {17, 6}, {17, 7}, {18, 1}, {18, 2}, {18, 3}, {18, 4}, {18, 5}, {18, 6}, {18, 7}, {18, 8}, {19, 1}, {19, 2}, {19, 3}, {19, 4}, {19, 5}, {19, 6}, {19, 7}, {19, 8}, {19, 9}, {20, 1}, {20, 2}, {20, 3}, {20, 4}, {20, 5}, {20, 6}, {20, 7}, {20, 8}, {20, 9}, {20, 10}}
La relación inversa de la segunda es: {{10, 20}, {11, 19}, {11, 20}, {12, 18}, {12, 19}, {12, 20}, {13, 17}, {13, 18}, {13, 19}, {13, 20}, {14, 16}, {14, 17}, {14, 18}, {14, 19}, {14, 20}, {15, 15}, {15, 16}, {15, 17}, {15, 18}, {15, 19}, {15, 20}, {16, 14}, {16, 15}, {16, 16}, {16, 17}, {16, 18}, {16, 19}, {16, 20}, {17, 13}, {17, 14}, {17, 15}, {17, 16}, {17, 17}, {17, 18}, {17, 19}, {17, 20}, {18, 12}, {18, 13}, {18, 14}, {18, 15}, {18, 16}, {18, 17}, {18, 18}, {18, 19}, {18, 20}, {19, 11}, {19, 12}, {19, 13}, {19, 14}, {19, 15}, {19, 16}, {19, 17}, {19, 18}, {19, 19}, {19, 20}, {20, 10}, {20, 11}, {20, 12}, {20, 13}, {20, 14}, {20, 15}, {20, 16}, {20, 17}, {20, 18}, {20, 19}, {20, 20}}
La relación composición de la primera con respecto a la segunda es: {{20, 20}}
La relación composición de la segunda con respecto a la primera es: {{1, 10}, {1, 11}, {1, 12}, {1, 13}, {1, 14}, {1, 15}, {1, 16}, {1, 17}, {1, 18}, {1, 19}, {1, 20}, {2, 10}, {2, 11}, {2, 12}, {2, 13}, {2, 14}, {2, 15}, {2, 16}, {2, 17}, {2, 18}, {2, 19}, {2, 20}, {3, 10}, {3, 11}, {3, 12}, {3, 13}, {3, 14}, {3, 15}, {3, 16}, {3, 17}, {3, 18}, {3, 19}, {3, 20}, {4, 10}, {4, 11}, {4, 12}, {4, 13}, {4, 14}, {4, 15}, {4, 16}, {4, 17}, {4, 18}, {4, 19}, {4, 20}, {5, 10}, {5, 11}, {5, 12}, {5, 13}, {5, 14}, {5, 15}, {5, 16}, {5, 17}, {5, 18}, {5, 19}, {5, 20}, {6, 10}, {6, 11}, {6, 12}, {6, 13}, {6, 14}, {6, 15}, {6, 16}, {6, 17}, {6, 18}, {6, 19}, {6, 20}, {7, 10}, {7, 11}, {7, 12}, {7, 13}, {7, 14}, {7, 15}, {7, 16}, {7, 17}, {7, 18}, {7, 19}, {7, 20}, {8, 10}, {8, 11}, {8, 12}, {8, 13}, {8, 14}, {8, 15}, {8, 16}, {8, 17}, {8, 18}, {8, 19}, {8, 20}, {9, 10}, {9, 11}, {9, 12}, {9, 13}, {9, 14}, {9, 15}, {9, 16}, {9, 17}, {9, 18}, {9, 19}, {9, 20}, {10, 10}, {10, 11}, {10, 12}, {10, 13}, {10, 14}, {10, 15}, {10, 16}, {10, 17}, {10, 18}, {10, 19}, {10, 20}}

Explicación en video

11) **RelBinUnion**: calcula la **unión** entre **dos relaciones binarias** brindando la opción de **visualizar** el procedimiento por **definición**, o bien, usando **matrices booleanas**.

Sintaxis: `RelBinUnion[R1, R2, A, B]`, "R1", "R2" son las relaciones y "A", "B" los conjuntos

sobre los cuales están definidas. **RelBinUnion[R1, R2, A, B, steps -> True]** muestra la operación **paso a paso**.

Ejemplo 6.21

Halle $R_1 \cup R_2$ donde $R_1 = \{(1, 1), (1, 2), (2, 1), (2, 2), (2, 1), (3, 1), (4, 12), (30, 8)\}$ y $R_2 = \{(a,b) \mid a + b \leq 10\}$ definida sobre $A = \{1,2,3,\ldots,30\}$.

Solución:

RelBinUnion retorna:

In[ ] :=

```
A = Table[i, {i, 1, 30}];
R1 = {{1, 1}, {1, 2}, {2, 1}, {2, 2}, {2, 1}, {3, 1}, {4, 12}, {30,
8}};
R2 = RelBin["a+b<=10", A, A];
RelBinUnion[R1, R2, A, A]
```

Out[ ] :=

{{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 6}, {1, 7}, {1, 8}, {1, 9}, {2, 1}, {2, 2}, {2, 3}, {2, 4}, {2, 5}, {2, 6}, {2, 7}, {2, 8}, {3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5}, {3, 6}, {3, 7}, {4, 1}, {4, 2}, {4, 3}, {4, 4}, {4, 5}, {4, 6}, {4, 12}, {5, 1}, {5, 2}, {5, 3}, {5, 4}, {5, 5}, {6, 1}, {6, 2}, {6, 3}, {6, 4}, {7, 1}, {7, 2}, {7, 3}, {8, 1}, {8, 2}, {9, 1}, {30, 8}}

Ejemplo 6.22

Encuentre paso a paso $R_1 \cup R_2$ con $R_1 = \{(a,b) \mid a - b \leq -10\}$ y $R_2 = \{(a,b) \mid a + b \geq 30\}$ definidas sobre $A = \{1,2,3,\ldots,20\}$.

Solución:

Para mostrar el procedimiento por definición y usando matrices booleanas, paso a paso, se utiliza la opción "**steps -> True**" del comando:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
R1 = RelBin["a-b<=-10", A, A];
R2 = RelBin["a+b>=30", A, A];
RelBinUnion[R1, R2, A, A, steps -> True]
```

Out[ ] :=

Por definición:

Union = {{1, 11}, {1, 12}, {1, 13}, ..., {10, 20}} ∪ {{10, 20}, {11, 19}, {11, 20}, ..., {20, 19}, {20, 20}} = {{1, 11}, {1, 12}, {1, 13}, ..., {20, 19}, {20, 20}}

Por matrices booleanas:

Matriz de la primera relación: $$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

Matriz de la segunda relación: $$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} \vee$$

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}$$

$$=\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación unión corresponde a: {{1, 11}, {1, 12}, {1, 13}, ..., {20, 19}, {20, 20}}

Explicación en video

12) **RelBinInter**: encuentra la **intersección** entre **dos relaciones binarias** recibidas como parámetros, dando la opción al usuario de **visualizar** el procedimiento por **definición**, o bien, usando **matrices booleanas**.

Sintaxis: `RelBinInter[R1, R2, A, B]`, "R1", "R2" son las relaciones y "A", "B" los conjuntos sobre los cuales están definidas. `RelBinInter[R1, R2, A, B, steps->True]` muestra la operación **paso a paso**.

Ejemplo 6.23

Determine $R_1 \cap R_2$ donde $R_1 = \{(1, 1), (1, 2), (2, 1), (2, 2), (2, 1), (3, 1), (4, 12), (30, 8)\}$ y $R_2 = \{(a,b) \mid a+b \leq 10\}$ definida sobre $A = \{1,2,3,\ldots,30\}$.

Solución:

En el software:

In[ ] :=

```
A = Table[i, {i, 1, 30}];
```

```
R1 = {{1, 1}, {1, 2}, {2, 1}, {2, 2}, {2, 1}, {3, 1}, {4, 12}, {30,
8}};
R2 = RelBin[“a+b<=10”, A, A];
RelBinInter[R1, R2, A, A]
```

Out[ ] :=

```
{{1, 1}, {1, 2}, {2, 1}, {2, 2}, {3, 1}}
```

Ejemplo 6.24

Halle paso a paso $R_1 \cap R_2$ con $R_1 = \{(a,b) \mid a - b \leq -10\}$ y $R_2 = \{(a,b) \mid a + b \geq 30\}$ definidas sobre $A = \{1,2,3,\ldots,20\}$.

Solución:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
R1 = RelBin[“a-b<=-10”, A, A];
R2 = RelBin[“a+b>=30”, A, A];
RelBinInter[R1, R2, A, A, steps -> True]
```

Out[ ] :=

Por definición:

Intersection = {{1, 11}, {1, 12}, {1, 13}, ..., {10, 20}} ∩ {{10, 20}, {11, 19}, {11, 20}, ..., {20, 19}, {20, 20}} = {{10, 20}}

Por matrices booleanas:

⋮

$$= \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación intersección corresponde a: {{10, 20}}

Ⓝ No se muestra toda la salida por su tamaño, sin embargo, el *Out* [ ] arrojado por *Mathematica* es muy similar al presentado por la instrucción `RelBinUnion`.

Explicación en video

13) **RelBinComplement**: determina la **relación complementaria** de **una relación binaria**.

Sintaxis: `RelBinComplement[R, A, B]`, o bien, `RelBinComplement[R, A, B, steps -> True]` si se desea el resultado **paso a paso**. "R" constituye un conjunto de pares ordenados que forman la relación y "A", "B" los conjuntos donde se encuentra definida.

Ejemplo 6.25

Sea la relación binaria $R = \{(a,b) \mid a+b \leq 10\}$ sobre $A = \{1,2,3,\ldots,30\}$. Determine $\overline{R}$.

Solución:

El comando **`RelBinComplement`** facilita el resultado:

In[ ] :=

```
A = Table[i, {i, 1, 30}];
R = RelBin["a+b<=10", A, A];
RelBinComplement[R, A, A]
```

Out[ ] :=

{{1, 10}, {1, 11}, {1, 12}, ..., {30, 29}, {30, 30}}

La salida es considerablemente grande por lo que no se muestra en detalle.

Ejemplo 6.26

Resuelva paso a paso $\overline{R}$ con $R = \{(a,b) \mid a-b \leq -10\}$ definida sobre $A = \{1,2,3,\ldots,20\}$.

Solución:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
R = RelBin["a-b<=-10", A, A];
RelBinComplement[R, A, A, steps -> True]
```

Out[ ] :=

Por definición:

Complement = ¬ {{1, 11}, {1, 12}, {1, 13}, ..., {20, 19}, {20, 20}}

Matriz de la relación:

⋮

$$
= \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
$$

Tomando como base la matriz anterior, la relación complementaria corresponde a: {{1, 1}, {1, 2}, {1, 3}, ..., {20, 19}, {20, 20}}

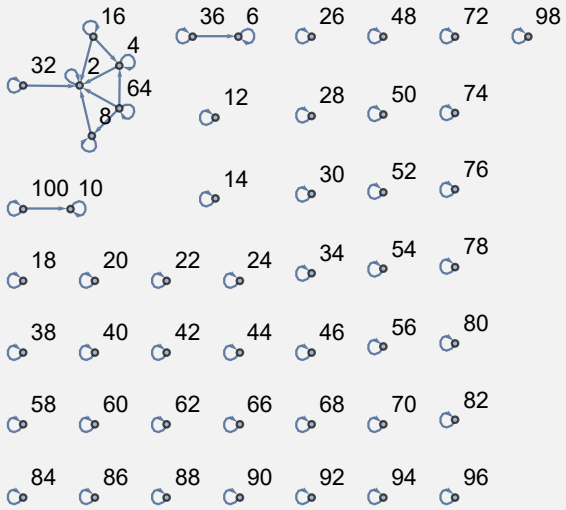
Explicación en video

14) **RelBinGrafo**: retorna los **elementos** de una **relación binaria** presentada a través de un **grafo**.

Sintaxis: `RelBinGrafo[Graph]` siendo “Graph” el grafo correspondiente.

Ejemplo 6.27

Dado el grafo:



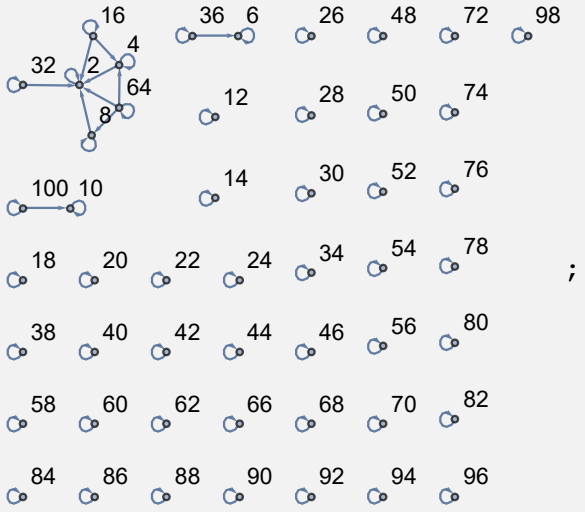
Encuentre explícitamente los pares ordenados de la relación binaria que representa.

Solución:

En *Mathematica*:

In[ ] :=

```
G =                                                    ;
```




```
RelBinGrafo[G]
```

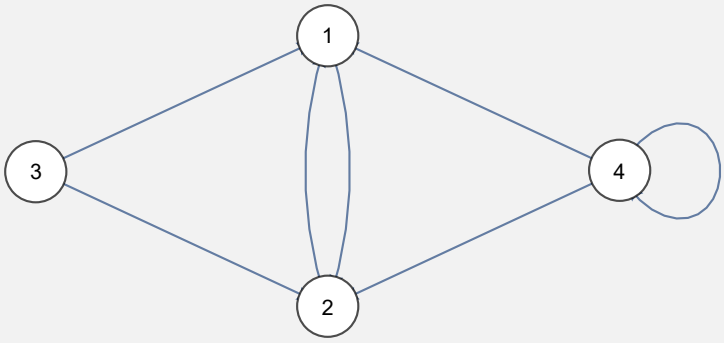
Out[ ] :=

{{2, 2}, {4, 2}, {4, 4}, {6, 6}, {8, 2}, {8, 8}, {10, 10}, {12, 12}, {14, 14}, {16, 2}, {16, 4}, {16, 16}, {18, 18}, {20, 20}, {22, 22}, {24, 24}, {26, 26}, {28, 28}, {30, 30}, {32, 2}, {32, 32}, {34, 34}, {36, 6}, {36, 36}, {38, 38}, {40, 40}, {42, 42}, {44, 44}, {46, 46}, {48, 48}, {50, 50}, {52, 52}, {54, 54}, {56, 56}, {58, 58}, {60, 60}, {62, 62}, {64, 2}, {64, 4}, {64, 8}, {64, 64}, {66, 66}, {68, 68}, {70, 70}, {72, 72}, {74, 74}, {76, 76}, {78, 78}, {80, 80}, {82, 82}, {84, 84}, {86, 86}, {88, 88}, {90, 90}, {92, 92}, {94, 94}, {96, 96}, {98, 98}, {100, 10}, {100, 100}}

Se asume que la imagen correspondiente al grafo, se posee directamente en el software. Esto permite almacenarla en la variable **G**.

Ejemplo 6.28

Determine los pares ordenados de una relación binaria representada por:

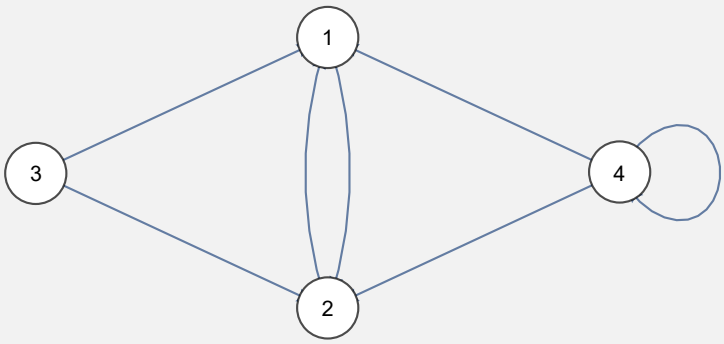


Solución:

In[ ] :=

```
G = Graph[{1 -> 2, 2 -> 1, 3 -> 1, 3 -> 2, 4 -> 1, 4 -> 2, 4 -> 4},
VertexShapeFunction -> ({White, EdgeForm[Black] , Disk[#, .1], Black,
Text[#2, #1]} &)]
RelBinGrafo[G]
```

Out[ ] :=



{{1, 2}, {2, 1}, {3, 1}, {3, 2}, {4, 1}, {4, 2}, {4, 4}}

N **Graph** es un comando del software *Mathematica* que se emplea para crear grafos. En este ejemplo, se ha utilizado para almacenar en la variable **G** el grafo de interés.

Explicación en video

15) **RelBinInver**: determina la **relación binaria inversa** de otra recibida como parámetro.

Sintaxis: `RelBinInver[R, A, B]`, o bien, `RelBinInver[R, A, B, steps->True]` si el usuario pretende **visualizar** el resultado **paso a paso**, por **definición** o mediante el procedimiento de **matriz booleana**. Los argumentos "R", "A" y "B" constituyen la relación binaria dada mediante sus pares ordenados y los conjuntos en los cuales está definida, respectivamente.

Ejemplo 6.29

Halle a través de un prodecimiento completo R^{-1} con $R = \{(a,b) \mid a - b \leq -10\}$ definida sobre $A = \{1,2,3,\ldots,20\}$.

Solución:

Al emplear **RelBinInver**:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
R = RelBin["a-b<=-10", A, A];
RelBinInver[R, A, A, steps->True]
```

Out[ ] :=

Por definición:

Inverse = ({{1, 11}, {1, 12}, {1, 13}, {1, 14}, {1, 15}, {1, 16}, {1, 17}, {1, 18}, {1, 19}, {1, 20}, {2, 12}, {2, 13}, {2, 14}, {2, 15}, {2, 16}, {2, 17}, {2, 18}, {2, 19}, {2, 20}, {3, 13}, {3, 14}, {3, 15}, {3, 16}, {3, 17}, {3, 18}, {3, 19}, {3, 20}, {4, 14}, {4, 15}, {4, 16}, {4, 17}, {4, 18}, {4, 19}, {4, 20}, {5, 15}, {5, 16}, {5, 17}, {5, 18}, {5, 19}, {5, 20}, {6, 16}, {6, 17}, {6, 18}, {6, 19}, {6, 20}, {7, 17}, {7, 18}, {7, 19}, {7, 20}, {8, 18}, {8, 19}, {8, 20}, {9, 19}, {9, 20}, {10, 20}})^-1 = {{11, 1}, {12, 1}, {12, 2}, {13, 1}, {13, 2}, {13, 3}, {14, 1}, {14, 2}, {14, 3}, {14, 4}, {15, 1}, {15, 2}, {15, 3}, {15, 4}, {15, 5}, {16, 1}, {16, 2}, {16, 3}, {16, 4}, {16, 5}, {16, 6}, {17, 1}, {17, 2}, {17, 3}, {17, 4}, {17, 5}, {17, 6}, {17, 7}, {18, 1}, {18, 2}, {18, 3}, {18, 4}, {18, 5}, {18, 6}, {18, 7}, {18, 8}, {19, 1}, {19, 2}, {19, 3}, {19, 4}, {19, 5}, {19, 6}, {19, 7}, {19, 8}, {19, 9}, {20, 1}, {20, 2}, {20, 3}, {20, 4}, {20, 5}, {20, 6}, {20, 7}, {20, 8}, {20, 9}, {20, 10}}

Por matriz booleana:

Matriz de la relación: $$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

$$(\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix})^t$$

$$= \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación inversa corresponde a: {{11, 1}, {12, 1}, {12, 2}, {13, 1}, {13, 2}, {13, 3}, {14, 1}, {14, 2}, {14, 3}, {14, 4}, {15, 1}, {15, 2}, {15, 3}, {15, 4}, {15, 5}, {16, 1}, {16, 2}, {16, 3}, {16, 4}, {16, 5}, {16, 6}, {17, 1}, {17, 2}, {17, 3}, {17, 4}, {17, 5}, {17, 6}, {17, 7}, {18, 1}, {18, 2}, {18, 3}, {18, 4}, {18, 5}, {18, 6}, {18, 7}, {18, 8}, {19, 1}, {19, 2}, {19, 3}, {19, 4}, {19, 5}, {19, 6}, {19, 7}, {19, 8}, {19, 9}, {20, 1}, {20, 2}, {20, 3}, {20, 4}, {20, 5}, {20, 6}, {20, 7}, {20, 8}, {20, 9}, {20, 10}}

Ejemplo 6.30

Encuentre paso a paso R^{-1} con $R = \{(a,b) \mid a+b \geq 30\}$ definida sobre $A = \{1,2,3,\ldots,20\}$.

Solución:

En el software:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
R = RelBin[“a+b>=30”, A, A];
RelBinInver[R, A, A, steps->True]
```

Out[ ] :=

Por definición:

Inverse = ({{10, 20}, {11, 19}, {11, 20}, ...

Por matriz booleana:

⋮

$$= \begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación inversa corresponde a: {{10, 20}, {11, 19}, {11, 20}, {12, 18}, {12, 19}, {12, 20}, {13, 17}, {13, 18}, {13, 19}, {13, 20}, {14, 16}, {14, 17}, {14, 18}, {14, 19}, {14, 20}, {15, 15}, {15, 16}, {15, 17}, {15, 18}, {15, 19}, {15, 20}, {16, 14}, {16, 15}, {16, 16}, {16, 17}, {16, 18}, {16, 19}, {16, 20}, {17, 13}, {17, 14}, {17, 15}, {17, 16}, {17, 17}, {17, 18}, {17, 19}, {17, 20}, {18, 12}, {18, 13}, {18, 14}, {18, 15}, {18, 16}, {18, 17}, {18, 18}, {18, 19}, {18, 20}, {19, 11}, {19, 12}, {19, 13}, {19, 14}, {19, 15}, {19, 16}, {19, 17}, {19, 18}, {19, 19}, {19, 20}, {20, 10}, {20, 11}, {20, 12}, {20, 13}, {20, 14}, {20, 15}, {20, 16}, {20, 17}, {20, 18}, {20, 19}, {20, 20}}

Ⓝ El lector puede observar que la relación y su inversa son iguales pues *R* es simétrica.

Explicación en video

16) **RelBinComp**: retorna la **composición** entre **dos relaciones binarias**.

Sintaxis: `RelBinComp[R1, R2, A, B, C]`, o bien, `RelBinComp[R1, R2, A, B, C, steps -> True]` para observar el procedimiento por **definición** o mediante el uso de **matrices booleanas**, siendo "R1" y "R2" dos relaciones binarias dadas de manera explícita por sus pares ordenados, tales que: "R2: A -> B" y "R1: B -> C".

Ejemplo 6.31

Halle la relación $R_1 o R_2$ con $R_1 = \{(1, 4), (90, r), (100, t), (2, w), (3, q), (3, e), (45, 8), (2, 1)\}$ y $R_2 = \{(a,b) \mid a+b \leq 10\}$ definida sobre $A = \{1,2,3,\ldots,30\}$ y $B = \{1,90,100,2,3,45,6\}$.

Solución:

En este ejercicio se infiere un tercer conjunto $C = \{4,6,8,1,r,t,e,w,q\}$, luego:

In[ ] :=

```
A = Table[i, {i, 1, 30}];
B = {1, 90, 100, 2, 3, 45, 6};
c = {4, 6, 8, 1, r, t, e, w, q};
R1 = {{1, 4}, {90, r}, {100, t}, {2, w}, {3, q}, {3, e}, {45, 8}, {2,
1}};
R2 = RelBin["a+b<=10", A, B];
RelBinComp[R1, R2, A, B, c]
```

Out[ ] :=

{{1, 1}, {1, 4}, {1, e}, {1, q}, {1, w}, {2, 1}, {2, 4}, {2, e}, {2, q}, {2, w}, {3, 1}, {3, 4}, {3, e}, {3, q}, {3, w}, {4, 1}, {4, 4}, {4, e}, {4, q}, {4, w}, {5, 1}, {5, 4}, {5, e}, {5, q}, {5, w}, {6, 1}, {6, 4}, {6, e}, {6, q}, {6, w}, {7, 1}, {7, 4}, {7, e}, {7, q}, {7, w}, {8, 1}, {8, 4}, {8, w}, {9, 4}}

N En el código de *Mathematica* anterior, el conjunto C se declaró mediante una variable minúscula y no mayúscula, pues **C** es un comando propio del software, por lo que no es posible utilizarlo como nombre o identificador.

Ejemplo 6.32

Determine paso a paso la relación binaria $R_2 o R_1$ siendo $R_1 = \{(a,b) \mid a-b \leq -10\}$ sobre $A = \{1,2,3,\ldots,20\}$ y $B = \{2,4,6,\ldots,40\}$ y, $R_2 = \{(a,b) \mid a+b \geq 30\}$ definida en B y $C = \{1,3,5,\ldots,39\}$.

Solución:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
B = Table[2i, {i, 1, 20}];
c = Table[2i - 1, {i, 1, 20}];
R1 = RelBin["a-b<=-10", A, B];
R2 = RelBin["a+b>=30", B, c];
RelBinComp[R2, R1, A, B, c, steps -> True]
```

Out[ ] :=

Por definición:

{{{1,12}, {12,19}} -> {1,19}, {{1,12}, {12,21}} -> {1,21}, {{1,12}, {12,23}} -> {1,23}, ..., {{20,40}, {40,37}} -> {20,37}, {{20,40}, {40,39}} -> {20,39}}

Composition = {{2, 29}, {2, 31}, {2, 33}, ..., {40, 37}, {40, 39}} o {{1, 12}, {1, 14}, {1, 16}, ..., {20, 38}, {20,

40}} = {{1, 1}, {1, 3}, {1, 5}, ..., {20, 37}, {20, 39}}

Por matrices booleanas:

Matriz de la primera relación: . . .

⋮

$$\begin{pmatrix}
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1
\end{pmatrix} \odot$$

$$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

$$= \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación composición corresponde a: {{1, 1}, {1, 3}, {1, 5}, ..., {20, 37}, {20, 39}}

Explicación en video

17) **MBooleanQ**: muestra "`True`" si al recibir una **matriz**, ésta es **boolena** y "`False`" en caso contrario.

Sintaxis: `MBooleanQ[M]`, con "M" la matriz.

Ejemplo 6.33

Mediante el uso de software determine si la matriz dada es o no booleana:

$$\begin{pmatrix}
0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 2 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1
\end{pmatrix}$$

Solución:

Al utilizar `MBooleanQ`:

In[ ] :=

$$\texttt{Matriz} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 2 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix};$$

```
MBooleanQ[Matriz]
```

Out[ ] :=

False

N "False" indica que la matriz no es booleana. Naturalmente esto ocurre pues no todas sus entradas son unos o ceros.

Ejemplo 6.34

¿Es booleana la matriz dada a continuación?

$$\begin{pmatrix} \square & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Solución:

In[ ] :=

$$\texttt{Matriz} = \begin{pmatrix} \square & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix};$$

```
MBooleanQ[Matriz]
```

Out[ ] :=

False

N Al tener la matriz una entrada vacía, la instrucción **`MBooleanQ`** devuelve "False".

Explicación en video

18) **UnionBooleana**: calcula la **unión booleana** entre **dos matrices boolenas**.

Sintaxis: **`UnionBooleana[M1, M2]`** siendo "M1" y "M2" las matrices booleanas correspondientes.

Ejemplo 6.35

Halle $M_1 \vee M_2$ con:

$$M_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} \text{ y } M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Solución:

En *Mathematica*:

In[ ] :=

$$\mathbf{M1 =} \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix};$$

$$\mathbf{M2 =} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix};$$

```
UnionBooleana[M1, M2]
```

Out[ ] :=

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Ejemplo 6.36

Encuentre $M_1 \vee M_2$ donde:

$$M_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ y } M_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Solución:

In[ ] :=

```
M1 = {{1, 0, 1, 0, 1, 1, 0}, {1, 0, 0, 0, 1, 0, 0}, {1, 0, 0, 0, 1, 1, 1}, {0, 0, 1, 1, 1, 0, 1}, {0, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 0, 0, 0}};
M2 = {{0, 0, 0, 0, 0, 0, 0}, {1, 0, 1, 0, 1, 0, 0}, {0, 0, 0, 0, 1, 1, 1}, {1, 0, 1, 1, 1, 0, 1}, {0, 0, 0, 0, 1, 0, 0}, {1, 0, 1, 0, 0, 0, 1}};
UnionBooleana[M1, M2]
```

Out[ ] :=

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Explicación en video

19) **InterseccionBooleana**: determina la **intersección booleana** entre **dos matrices boolenas**.

Sintaxis: `InterseccionBooleana[M1, M2]`, con "M1" y "M2" las matrices booleanas correspondientes.

Ejemplo 6.37

Determine $M_1 \wedge M_2$ con:

$$M_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} \text{ y } M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Solución:

En el software:

In[ ] :=

```
M1 = {{1, 0, 1, 0, 1}, {1, 0, 0, 0, 1}, {1, 0, 0, 0, 1}, {0, 0, 1, 1, 1}, {0, 1, 0, 0, 1}};
M2 = {{1, 0, 0, 0, 0}, {0, 1, 0, 0, 0}, {0, 0, 1, 0, 0}, {0, 0, 0, 1, 0}, {0, 0, 0, 0, 1}};
InterseccionBooleana[M1, M2]
```

Out[ ] :=

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Ejemplo 6.38

Halle $M_1 \wedge M_2$ donde:

$$M_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ y } M_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Solución:

In[ ] :=

$$\mathbf{M1} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix};$$

$$\mathbf{M2} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix};$$

```
InterseccionBooleana[M1, M2]
```

Out[ ] :=

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Explicación en video

20) **ComplementoBooleano**: calcula el **complemento booleano** sobre una **matriz booleana**.

Sintaxis: `ComplementoBooleano[M]` siendo "M" la matriz booleana correspondiente.

Ejemplo 6.39

Determine mediante el uso del software *Mathematica* el complemento booleano de:

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Solución:

Al utilizar el comando **`ComplementoBooleano`** se obtiene:

In[ ] :=

```
Matriz =
```
$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix};$$

```
ComplementoBooleano[Matriz]
```

Out[ ] :=

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Ejemplo 6.40

Calcule el complemento booleano de la siguiente matriz:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Solución:

En el software:

In[ ] :=

```
Matriz =
```
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix};$$

```
ComplementoBooleano[Matriz]
```

Out[ ] :=

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Explicación en video

21) **ProductoBooleano**: realiza la **multiplicación booleana**.

Sintaxis: `ProductoBooleano[M1, M2]` siendo "M1" y "M2" las matrices booleanas correspondientes.

Ejemplo 6.41

Halle $M_1 \odot M_2$, con:

$$M_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ y } M_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Solución:

Al emplear la instrucción **`ProductoBooleano`**:

In[ ] :=

$$\texttt{M1 =} \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix};$$

$$\texttt{M2 =} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix};$$

```
ProductoBooleano[M1, M2]
```

Out[ ] :=

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Ejemplo 6.42

Determine $M_1 \odot M_2$ y $M_2 \odot M_1$, siendo:

$$M_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \text{ y } M_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

¿Es conmutativa la operación producto booleano?

Solución:

In[ ] :=

$$\mathbf{M1 =} \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{;}$$

$$\mathbf{M2 =} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \mathbf{;}$$

```
ProductoBooleano[M1, M2]
ProductoBooleano[M2, M1]
```

Out[ ] :=

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

N El estudiante puede comprobar cómo $M_1 \odot M_2 \neq M_2 \odot M_1$, de donde se concluye la no conmutatividad.

Explicación en video

22) **CDFOperacionesMBooleanas**: realiza las operaciones de **unión**, **intersección**, **complemento** y **producto booleano** sobre **dos matrices booleanas**.

Sintaxis: `CDFOperacionesMBooleanas[M1, M2]`, con "M1" y "M2" las matrices booleanas correspondientes. Los resultados se **muestran** en una **animación** donde el usuario puede **cambiar** las matrices y **recalcular** las operaciones.

Ejemplo 6.43

Realice las operaciones booleanas unión, intersección, complemento y producto booleano sobre:

$$M_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} \text{ y } M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Solución:

Al recurrir al comando `CDFOperacionesMBooleanas`:

In[ ] :=

$$\texttt{CDFOperacionesMBooleanas[}\begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}\texttt{,}\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}\texttt{]}$$

Out[ ] :=

Matrices booleanas

A = {{1, 0, 1, 0, 1},
{1, 0, 0, 0, 1}, {1, 0, 0, 0, 1},
{0, 0, 1, 1, 1}, {0, 1, 0, 0, 1}}

B = {{1, 0, 0, 0, 0},
{0, 1, 0, 0, 0}, {0, 0, 1, 0, 0},
{0, 0, 0, 1, 0}, {0, 0, 0, 0, 1}}

Operaciones con las matrices booleanas:

$$A=\begin{pmatrix}1&0&1&0&1\\1&0&0&0&1\\1&0&0&0&1\\0&0&1&1&1\\0&1&0&0&1\end{pmatrix},\ B=\begin{pmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{pmatrix}:$$

$$A\vee B=\begin{pmatrix}1&0&1&0&1\\1&1&0&0&1\\1&0&1&0&1\\0&0&1&1&1\\0&1&0&0&1\end{pmatrix}$$

$$A\wedge B=\begin{pmatrix}1&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&1&0\\0&0&0&0&1\end{pmatrix}$$

$$\neg A=\begin{pmatrix}0&1&0&1&0\\0&1&1&1&0\\0&1&1&1&0\\1&1&0&0&0\\1&0&1&1&0\end{pmatrix}$$

$$\neg B=\begin{pmatrix}0&1&1&1&1\\1&0&1&1&1\\1&1&0&1&1\\1&1&1&0&1\\1&1&1&1&0\end{pmatrix}$$

$$A\odot B=\begin{pmatrix}1&0&1&0&1\\1&0&0&0&1\\1&0&0&0&1\\0&0&1&1&1\\0&1&0&0&1\end{pmatrix}$$

$$B\odot A=\begin{pmatrix}1&0&1&0&1\\1&0&0&0&1\\1&0&0&0&1\\0&0&1&1&1\\0&1&0&0&1\end{pmatrix}$$

Ejemplo 6.44

Calcule las operaciones booleanas unión, intersección, complemento y producto booleano dadas las siguientes matrices:

$$M_1=\begin{pmatrix}1&0&1&0&1&1&0\\1&0&0&0&1&0&0\\1&0&0&0&1&1&1\\0&0&1&1&1&0&1\\0&1&1&0&1&1&0\\1&1&1&0&0&0&0\end{pmatrix}\ \text{y}\ M_2=\begin{pmatrix}0&0&0&0&0&0&0\\1&0&1&0&1&0&0\\0&0&0&0&1&1&1\\1&0&1&1&1&0&1\\0&0&0&0&1&0&0\\0&0&1&1&0&1&0\\1&0&1&0&0&0&1\end{pmatrix}$$

Solución:

En *Mathematica*:

In[ ] :=

$$\mathbf{M1} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix};$$

$$\mathbf{M2} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix};$$

```
CDFOperacionesMBooleanas[M1, M2]
```

Out[ ] :=

Matrices booleanas

A = {{1, 0, 1, 0, 1, 1, 0}, {1, 0, 0, 0, 1, 0, 0},
{1, 0, 0, 0, 1, 1, 1}, {0, 0, 1, 1, 1, 0, 1},
{0, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 0, 0, 0}}

B = {{0, 0, 0, 0, 0, 0, 0},
{1, 0, 1, 0, 1, 0, 0}, {0, 0, 0, 0, 1, 1, 1},
{1, 0, 1, 1, 1, 0, 1}, {0, 0, 0, 0, 1, 0, 0},
{0, 0, 1, 1, 0, 1, 0}, {1, 0, 1, 0, 0, 0, 1}}

Operaciones con las matrices booleanas:

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix};$$

A∨B = No existe

A∧B = No existe

$$\neg A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\neg B = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$A \odot B = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

B⊙A = No existe

N $B \odot A$ no existe al no corresponder la cantidad de columnas de la matriz M_2 con la cantidad de filas de M_1.

Explicación en video

23) **MRelBinUnion**: calcula la **unión** entre **dos relaciones binarias** dadas mediante sus **matrices de representación**, **retornando** la **matriz booleana unión** y la **relación binaria** de **forma explícita**.

Sintaxis: `MRelBinUnion[M1, M2, A, B]`, con "M1" y "M2" las matrices de las relaciones y, "A" y "B" los conjuntos sobre los que se encuentran definidas.

Ejemplo 6.45

Considere las siguientes relaciones binarias: $R_1 = \{(a,b) \mid a - b \leq -10\}$ $R_2 = \{(a,b) \mid a + b \geq 30\}$ definidas sobre $A = \{1,2,3,\ldots,20\}$. Mediante el uso de matrices booleanas halle $R_1 \cup R_2$.

Solución:

La instrucción **`MRelBinUnion`** resuelve lo solicitado en este ejemplo, sin embargo, para ello, primero es necesario determinar de forma explícita los pares ordenados de cada relación a través del uso del comando **`RelBin`** y posteriormente, encontrar las matrices de representación al emplear la instrucción **`MatrizRelBin`**. En *Mathematica* se procede así:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
M1 = MatrizRelBin[RelBin["a-b<=-10", A, A], A, A, table->False];
M2 = MatrizRelBin[RelBin["a+b>=30", A, A], A, A, table->False];
MRelBinUnion[M1, M2, A, A]
```

Out[ ] :=

$$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación unión corresponde a: {{1, 11}, {1, 12}, {1, 13}, {1, 14}, {1, 15}, {1, 16}, {1, 17}, {1, 18}, {1, 19}, {1, 20}, {2, 12}, {2, 13}, {2, 14}, {2, 15}, {2, 16}, {2, 17}, {2, 18}, {2, 19}, {2, 20}, {3, 13}, {3, 14}, {3, 15}, {3, 16}, {3, 17}, {3, 18}, {3, 19}, {3, 20}, {4, 14}, {4, 15}, {4, 16}, {4, 17}, {4, 18}, {4, 19}, {4, 20}, {5, 15}, {5, 16}, {5, 17}, {5, 18}, {5, 19}, {5, 20}, {6, 16}, {6, 17}, {6, 18}, {6, 19}, {6, 20}, {7, 17}, {7, 18}, {7, 19}, {7, 20}, {8, 18}, {8, 19}, {8, 20}, {9, 19}, {9, 20}, {10, 20}, {11, 19}, {11, 20}, {12, 18}, {12, 19}, {12, 20}, {13, 17}, {13, 18}, {13, 19}, {13, 20}, {14, 16}, {14, 17}, {14, 18}, {14, 19}, {14, 20}, {15, 15}, {15, 16}, {15, 17}, {15, 18}, {15, 19}, {15, 20}, {16, 14}, {16, 15}, {16, 16}, {16, 17}, {16, 18}, {16,

19}, {16, 20}, {17, 13}, {17, 14}, {17, 15}, {17, 16}, {17, 17}, {17, 18}, {17, 19}, {17, 20}, {18, 12}, {18, 13}, {18, 14}, {18, 15}, {18, 16}, {18, 17}, {18, 18}, {18, 19}, {18, 20}, {19, 11}, {19, 12}, {19, 13}, {19, 14}, {19, 15}, {19, 16}, {19, 17}, {19, 18}, {19, 19}, {19, 20}, {20, 10}, {20, 11}, {20, 12}, {20, 13}, {20, 14}, {20, 15}, {20, 16}, {20, 17}, {20, 18}, {20, 19}, {20, 20}}

N El uso de la opción "**table->False**" es imprescindible dentro de **MatrizRelBin** con el objetivo de retornar una estructura que pueda ser procesada luego por **MRelBinUnion**. Sin su empleo el comando **MRelBinUnion** no funciona.

Ejemplo 6.46

Sean dadas las siguientes relaciones binarias:

$$R_1 = \{(a,b) \mid a - b \leq -10\}$$
$$R_2 = \{(a,b) \mid a + b \geq 30\}$$

definidas sobre $A = \{1,3,5,\ldots,39\}$ y $B = \{2,4,6,\ldots,40\}$. Usando matrices booleanas encuentre $R_1 \cup R_2$.

Solución:

De manera similar al ejemplo anterior, se tiene en el software:

In[ ] :=

```
A = Table[2i - 1, {i, 1, 20}];
B = Table[2i, {i, 1, 20}];
M1 = MatrizRelBin[RelBin["a-b<=-10", A, B], A, B, table->False];
M2 = MatrizRelBin[RelBin["a+b>=30", A, B], A, B, table->False];
MRelBinUnion[M1, M2, A, B]
```

Out[ ] :=

$$\begin{pmatrix}
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación unión corresponde a: {{1, 12}, {1, 14}, {1, 16}, ..., {39, 38}, {39, 40}}

No se muestra el resultado completo por sus dimensiones.

Explicación en video

24) **MRelBinInter**: calcula la **intersección** entre **dos relaciones binarias** recibidas a través de sus **matrices de representación**, **mostrando** la **matriz booleana intersección** y los **pares ordenados** de dicha **relación**.

Sintaxis: `MRelBinInter[M1, M2, A, B]` siendo "M1" y "M2" las matrices de las relaciones y, "A" y "B" los conjuntos sobre los que se encuentran definidas.

Ejemplo 6.47

Sean las matrices M_1 y M_2 de dos relaciones binarias R_1 y R_2 respectivamente, definidas sobre el conjunto $A = \{1,2,3,\ldots,20\}$. Calcule $R_1 \cap R_2$, siendo:

$$M_1 = \left(\begin{array}{cccccccccccccccccccc}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right)$$

$$M_2 = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Solución:

En *Mathematica*:

In[ ] :=

$$\mathbf{M1} = \begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix};$$

$$\mathbf{M2} = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix};$$

```
A = Table[i, {i, 1, 20}];
```

```
MRelBinInter[M1, M2, A, A]
```

Out[ ] :=

$$\begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación intersección corresponde a: {{1, 1}, {1, 2}, {2, 1}, {2, 2}, {3, 1}}

Ejemplo 6.48

Considere las siguientes relaciones binarias:

$$R_1 = \{(a,b) \mid a - b \leq -10\}$$
$$R_2 = \{(a,b) \mid a + b \geq 30\}$$

definidas sobre $A = \{1,3,5,\ldots,39\}$ y $B = \{2,4,6,\ldots,40\}$. Utilizando matrices booleanas halle $R_1 \cap R_2$.

Solución:

In[ ] :=

```
A = Table[2i - 1, {i, 1, 20}];
B = Table[2i, {i, 1, 20}];
M1 = MatrizRelBin[RelBin["a-b<=-10", A, B], A, B, table->False];
M2 = MatrizRelBin[RelBin["a+b>=30", A, B], A, B, table->False];
MRelBinInter[M1, M2, A, B]
```

Out[ ] :=

$$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación intersección corresponde a: {{1, 30}, {1, 32}, {1, 34}, {1, 36}, {1, 38}, {1, 40}, {3, 28}, {3, 30}, {3, 32}, {3, 34}, {3, 36}, {3, 38}, {3, 40}, {5, 26}, {5, 28}, {5, 30}, {5, 32}, {5, 34}, {5, 36}, {5, 38}, {5, 40}, {7, 24}, {7, 26}, {7, 28}, {7, 30}, {7, 32}, {7, 34}, {7, 36}, {7, 38}, {7, 40}, {9, 22}, {9, 24}, {9, 26}, {9, 28}, {9, 30}, {9, 32}, {9, 34}, {9, 36}, {9, 38}, {9, 40}, {11, 22}, {11, 24}, {11, 26}, {11, 28}, {11, 30}, {11, 32}, {11, 34}, {11, 36}, {11, 38}, {11, 40}, {13, 24}, {13, 26}, {13, 28}, {13, 30}, {13, 32}, {13, 34}, {13, 36}, {13, 38}, {13, 40}, {15, 26}, {15, 28}, {15, 30}, {15, 32}, {15, 34}, {15, 36}, {15, 38}, {15, 40}, {17, 28}, {17, 30}, {17, 32}, {17, 34}, {17, 36}, {17, 38}, {17, 40}, {19, 30}, {19, 32}, {19, 34}, {19, 36}, {19, 38}, {19, 40}, {21, 32}, {21, 34}, {21, 36}, {21, 38}, {21, 40}, {23, 34}, {23, 36}, {23, 38}, {23, 40}, {25, 36}, {25, 38}, {25, 40}, {27, 38}, {27, 40}, {29, 40}}

Explicación en video

25) **MRelBinComplement**: determina la **relación complemento** de **una relación binaria** recibida por medio de su **matriz de representación**.

Sintaxis: `MRelBinComplement[M, A, B]`, con "M" la matriz de la relación y, "A" y "B" los conjuntos sobre los que se encuentra definida. La instrucción muestra la **matriz booleana complementaria** y la **relación complemento** de **manera explícita**.

Ejemplo 6.49

Sean dadas dos relaciones binarias:

$$R_1 = \{(a,b) \mid a - b \leq -10\}$$
$$R_2 = \{(a,b) \mid a + b \geq 30\}$$

definidas sobre $A = \{1,2,3,\ldots,20\}$. Mediante el uso de matrices booleanas halle $\overline{R_1}$ y $\overline{R_2}$.

Solución:

En el software:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
M1 = MatrizRelBin[RelBin[“a-b<=-10”, A, A], A, A, table->False];
M2 = MatrizRelBin[RelBin[“a+b>=30”, A, A], A, A, table->False];
MRelBinComplement[M1, A, A]
MRelBinComplement[M2, A, A]
```

Out[ ] :=

$$\begin{pmatrix}
1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación complementaria corresponde a: {{1, 1}, {1, 2}, {1, 3}, ..., {20, 19}, {20, 20}}

$$\begin{pmatrix}
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación complementaria corresponde a: {{1, 1}, {1, 2}, {1, 3}, ..., {20, 8}, {20, 9}}

Se omite el resultado en detalle por su tamaño.

Ejemplo 6.50

Sean dadas las siguientes relaciones binarias:

$$R_1 = \{(a,b) \mid a - b \leq -10\}$$
$$R_2 = \{(a,b) \mid a + b \geq 30\}$$

definidas sobre $A = \{1,3,5,\ldots,39\}$ y $B = \{2,4,6,\ldots,40\}$. Utilizando matrices booleanas encuentre los gráficos de $\overline{R_1}$ y $\overline{R_2}$.

Solución:

In[ ] :=

```
A = Table[2i - 1, {i, 1, 20}];
B = Table[2i, {i, 1, 20}];
M1 = MatrizRelBin[RelBin["a-b<=-10", A, B], A, B, table->False];
M2 = MatrizRelBin[RelBin["a+b>=30", A, B], A, B, table->False];
MRelBinComplement[M1, A, B]
MRelBinComplement[M2, A, B]
```

Out[ ] :=

$$\begin{pmatrix}
1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación complementaria corresponde a: {{1, 2}, {1, 4}, {1, 6}, ..., {39, 38}, {39, 40}}

$$\begin{pmatrix}
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación complementaria corresponde a: {{1, 2}, {1, 4}, {1, 6}, ..., {25, 4}, {27, 2}}

La salida no se muestra por completo dadas sus dimensiones.

Explicación en video

26) **MRelBinInver**: encuentra la **relación binaria inversa** de otra recibida a través de una **matriz booleana**. El comando **retorna** la **matriz booleana** de la **relación inversa** y sus correspondientes **pares ordenados**.

Sintaxis: `MRelBinInver[M, A, B]`, con "M" la matriz de la relación y, "A" y "B" los conjuntos sobre los que se encuentra definida.

Ejemplo 6.51

Considere dos relaciones binarias:

$$R_1 = \{(a,b) \mid a - b \leq -10\}$$
$$R_2 = \{(a,b) \mid a + b \geq 30\}$$

definidas sobre $A = \{1,2,3,\ldots,20\}$. Emplee matrices booleanas para determinar los gráficos de R_1^{-1} y R_2^{-1}.

Solución:

En *Mathematica*:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
M1 = MatrizRelBin[RelBin[“a-b<=-10”, A, A], A, A, table->False];
M2 = MatrizRelBin[RelBin[“a+b>=30”, A, A], A, A, table->False];
MRelBinInver[M1, A, A]
MRelBinInver[M2, A, A]
```

Out[ ] :=

$$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación inversa corresponde a: {{11, 1}, {12, 1}, {12, 2}, {13, 1}, {13, 2}, {13, 3}, {14, 1}, {14, 2}, {14, 3}, {14, 4}, {15, 1}, {15, 2}, {15, 3}, {15, 4}, {15, 5}, {16, 1}, {16, 2}, {16, 3}, {16, 4}, {16, 5}, {16, 6}, {17, 1}, {17, 2}, {17, 3}, {17, 4}, {17, 5}, {17, 6}, {17, 7}, {18, 1}, {18, 2}, {18, 3}, {18, 4}, {18, 5}, {18, 6}, {18, 7}, {18, 8}, {19, 1}, {19, 2}, {19, 3}, {19, 4}, {19, 5}, {19, 6}, {19, 7}, {19, 8}, {19, 9}, {20, 1}, {20, 2}, {20, 3}, {20, 4}, {20, 5}, {20, 6}, {20, 7}, {20, 8}, {20, 9}, {20, 10}}

$$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación inversa corresponde a: {{10, 20}, {11, 19}, {11, 20}, {12, 18}, {12, 19}, {12, 20}, {13, 17}, {13, 18}, {13, 19}, {13, 20}, {14, 16}, {14, 17}, {14, 18}, {14, 19}, {14, 20}, {15, 15}, {15, 16}, {15, 17}, {15, 18}, {15, 19}, {15, 20}, {16, 14}, {16, 15}, {16, 16}, {16, 17}, {16, 18}, {16,

19}, {16, 20}, {17, 13}, {17, 14}, {17, 15}, {17, 16}, {17, 17}, {17, 18}, {17, 19}, {17, 20}, {18, 12}, {18, 13}, {18, 14}, {18, 15}, {18, 16}, {18, 17}, {18, 18}, {18, 19}, {18, 20}, {19, 11}, {19, 12}, {19, 13}, {19, 14}, {19, 15}, {19, 16}, {19, 17}, {19, 18}, {19, 19}, {19, 20}, {20, 10}, {20, 11}, {20, 12}, {20, 13}, {20, 14}, {20, 15}, {20, 16}, {20, 17}, {20, 18}, {20, 19}, {20, 20}}

Ejemplo 6.52

Sean dadas las siguientes relaciones binarias:

$$R_1 = \{(a,b) \mid a - b \leq -10\}$$
$$R_2 = \{(a,b) \mid a + b \geq 30\}$$

definidas sobre $A = \{1,3,5,\ldots,39\}$ y $B = \{2,4,6,\ldots,40\}$. Mediante el uso de matrices booleanas halle los gráficos de R_1^{-1} y R_2^{-1}.

Solución:

En el software:

In[ ] :=

```
A = Table[2i - 1, {i, 1, 20}];
B = Table[2i, {i, 1, 20}];
M1 = MatrizRelBin[RelBin["a-b<=-10", A, B], A, B, table->False];
M2 = MatrizRelBin[RelBin["a+b>=30", A, B], A, B, table->False];
MRelBinInver[M1, A, B]
MRelBinInver[M2, A, B]
```

Out[ ] :=

$$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación inversa corresponde a: {{12, 1}, {14, 1}, {14, 3}, ..., {40, 27}, {40, 29}}

$$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación inversa corresponde a: {{2, 29}, {2, 31}, {2, 33}, ..., {40, 37}, {40, 39}}

El *Out* [ ] generado es muy grande, por lo que no se muestra en su totalidad.

Explicación en video

27) **MRelBinComp**: **retorna** la **composición** si existe, entre **dos relaciones binarias** recibidas mediante sus **matrices de representación**. La instrucción **muestra** la **matriz booleana** de la **relación composición** y sus **pares ordenados** de **forma explícita**.

Sintaxis: **`MRelBinComp[M1, M2, A, B, C]`** siendo "M1" y "M2" las matrices de las relaciones "R1" y "R2", respectivamente, y, "R2: A -> B" y "R1: B -> C". La composición calculada corresponde a "R1 o R2".

Ejemplo 6.53

Sean dadas dos relaciones binarias $R_1 : B \to C$ y $R_2 : A \to B$ a través de sus matrices de repre-

sentación, con $A = \{1,2,3,\ldots,30\}$, $B = \{1,90,100,2,3,45,6\}$, $C = \{4,6,8,1,r,t,e,w,q\}$ y:

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ y } M_2 = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Determine con ayuda de *Mathematica* $R_1 o R_2$.

Solución:

Al utilizar el comando **`MRelBinComp`** se obtiene:

In[ ] :=

```
A = Table[i, {i, 1, 30}];
B = {1, 90, 100, 2, 3, 45, 6};
c = {4, 6, 8, 1, r, t, e, w, q};
```

$$\texttt{M1 =} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix};$$

$$M2 = \begin{pmatrix}
1 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix};$$

```
MRelBinComp[M1, M2, A, B, c]
```

Out[ ] :=

$$\begin{pmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación composición de la primera con respecto a la segunda, corresponde a: {{1, 1}, {1, 4}, {1, e}, {1, q}, {1, w}, {2, 1}, {2, 4}, {2, e}, {2, q}, {2, w}, {3, 1}, {3, 4}, {3, e}, {3, q}, {3, w}, {4, 1}, {4, 4}, {4, e}, {4, q}, {4, w}, {5, 1}, {5, 4}, {5, e}, {5, q}, {5, w}, {6, 1}, {6, 4}, {6, e}, {6, q}, {6, w}, {7, 1}, {7, 4}, {7, e}, {7, q}, {7, w}, {8, 1}, {8, 4}, {8, w}, {9, 4}}

N ¡Se recuerda al lector! que la variable **C** no puede ser empleada pues es una instrucción nativa de *Mathematica*, por lo que en este ejemplo se tomó una **c** (minúscula).

Ejemplo 6.54

Considere las siguientes relaciones binarias:

$$R_1 = \{(a,b) \mid a - b \leq -10\}$$
$$R_2 = \{(a,b) \mid a + b \geq 30\}$$

$R_1 : A \to B$ y $R_2 : B \to C$ con $A = \{1,2,3,\ldots,20\}$, $B = \{2,4,6,\ldots,40\}$ y $C = \{1,3,5,\ldots,39\}$. Mediante el uso de matrices booleanas encuentre $R_2 o R_1$.

Solución:

En el software:

```
In[ ] :=
A = Table[i, {i, 1, 20}];
B = Table[2i, {i, 1, 20}];
c = Table[2i - 1, {i, 1, 20}];
R1 = RelBin["a-b<=-10", A, B];
M1 = MatrizRelBin[R1, A, B, table->False];
R2 = RelBin["a+b>=30", B, c];
M2 = MatrizRelBin[R2, B, c, table->False];
MRelBinComp[M2, M1, A, B, c]
```

Out[ ] :=

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}$$

Tomando como base la matriz anterior, la relación composición de la primera con respecto a la segunda, corresponde a: {{1, 1}, {1, 3}, {1, 5}, ..., {20, 37}, {20, 39}}

La salida no se muestra completamente por su tamaño.

Explicación en video

28) **TipoRelacion**: **clasifica** una relación binaria en **reflexiva**, **simétrica**, **antisimétrica**, **transitiva**, de **equivalencia** y de **orden parcial**, dados sus **pares ordenados** de **forma explícita**. En caso de **no cumplir** alguna de las propiedades **retorna** un **contraejemplo**.

Sintaxis: `TipoRelacion[R, A]`, con "R" la relación binaria definida sobre el conjunto finito no vacío "A".

Ejemplo 6.55

Sea la relación $R = \{(a,b) \mid a \geq b\}$ siendo $A = \{1,2,3,\ldots,20\}$. Clasifique a R en reflexiva, simétrica, antisimétrica, transitiva, de equivalencia o de orden parcial.

Solución:

En *Mathematica*:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
R = RelBin["a>=b", A, A];
TipoRelacion[R, A]
```

Out[ ] :=

```
La relación es reflexiva
La relación no es simétrica, un contraejemplo es: {2, 1} está en la relación pero {1, 2} no pertenece
La relación es antisimétrica
La relación es transitiva
La relación es de orden parcial
```

N Es interesante observar cómo el comando devuelve un contraejemplo, si cualquiera de las propiedades reflexiva, simétrica, antisimétrica y transitiva no se satisface, en este caso, se ejecutó para la propiedad simétrica.

Ejemplo 6.56

Considere a $R = \{(a,b) \mid a^2 \geq b\}$ definida sobre $A = \{1,3,5,\ldots,19\}$. Clasifique a R como una relación reflexiva, simétrica, antisimétrica, transitiva, de equivalencia o de orden parcial.

Solución:

In[ ] :=

```
A = Table[2i - 1, {i, 1, 10}];
```

```
R = RelBin["a^2>=b", A, A];
TipoRelacion[R, A]
```

Out[ ] :=

```
La relación es reflexiva
La relación no es simétrica, un contraejemplo es: {3, 1} está en la relación pero {1, 3} no pertenece
La relación no es antisimétrica, un contraejemplo es: {3, 5} y {5, 3} están en la relación
La relación no es transitiva, un contraejemplo es: {3, 5}, {5, 11} están en la relación, sin embargo, {3, 11}
no pertenece
```

Explicación en video

29) **TipoMRelacion**: **clasifica** una relación binaria en **reflexiva**, **simétrica**, **antisimétrica**, **transitiva**, de **equivalencia** y de **orden parcial**, dada de **manera explícita**. Para ello el comando utiliza las **propiedades** de la **matriz booleana** que **representa la relación**, justificando el **por qué** en cada caso.

Sintaxis: **TipoMRelacion[R, A]**, con "R" la relación binaria definida sobre el conjunto finito no vacío "A".

Ejemplo 6.57

Sea la relación binaria $R = \left\{(a,b) \mid a^2 \geq b\right\}$ con $A = \{1,3,5,\ldots,19\}$. Clasifique a R como reflexiva, simétrica, antisimétrica, transitiva, de equivalencia o de orden parcial, utilizando para ello propiedades de matrices booleanas.

Solución:

En *Mathematica*:

In[ ] :=

```
A = Table[2i - 1, {i, 1, 10}];
R = RelBin["a^2>=b", A, A];
TipoMRelacion[R, A]
```

Out[ ] :=

Una matriz que representa la relación binaria es M =

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

La relación es reflexiva, en la matriz M todos los elementos de la diagonal principal son iguales a uno:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

La relación no es simétrica, M $\neq$ M^t, por ejemplo las entradas {1, 2} y {2, 1} son distintas en la matriz M:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$M^t = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

La relación no es antisimétrica, por ejemplo en las entradas {2, 3} y {3, 2} de la matriz M, hay un uno:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

La relación no es transitiva, (M$\odot$M)$\vee$M $\neq$ M, por ejemplo en la entrada {2, 6} de (M$\odot$M)$\vee$M y M, los valores son distintos:

$$(M\odot M)\vee M = \begin{pmatrix} 1&0&0&0&0&0&0&0&0&0\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1 \end{pmatrix}$$

$$M = \begin{pmatrix} 1&0&0&0&0&0&0&0&0&0\\ 1&1&1&1&1&0&0&0&0&0\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1 \end{pmatrix}$$

$$M\odot M = \begin{pmatrix} 1&0&0&0&0&0&0&0&0&0\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1 \end{pmatrix}$$

$$(M\odot M)\vee M = \begin{pmatrix} 1&0&0&0&0&0&0&0&0&0\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1&1&1&1 \end{pmatrix}$$

Ejemplo 6.58

Considere la relación $R = \{(a,b) \mid \log_b a \text{ es un número entero}\}$ definida sobre $A = \{2,4,6,\ldots,100\}$. Clasifique a R como reflexiva, simétrica, antisimétrica, transitiva, de equivalencia o de orden parcial, empleando propiedades de matrices booleanas.

Solución:

Al utilizar el comando **`TipoMRelacion`**:

In[ ] :=

```
A = Table[2i, {i, 1, 50}];
R = RelBin["IntegerQ[Log[b,a]]", A, A];
TipoMRelacion[R, A]
```

Out[ ] :=

Una matriz que representa la relación binaria es M = $\begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$

La relación es reflexiva, en la matriz M todos los elementos de la diagonal principal son iguales a uno:

$\begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$

La relación no es simétrica, $M \neq M^t$, por ejemplo las entradas {1, 2} y {2, 1} son distintas en la matriz M:

$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$

$M^t = \begin{pmatrix} 1 & 1 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$

La relación es antisimétrica

La relación es transitiva

$M \odot M = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$

$(M \odot M) \vee M = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$

La relación es de orden parcial

Al estar procesando matrices 50 por 50, no se muestran en el *Out* [ ], dadas sus dimensiones.

Explicación en video

30) **RelacionEquivalenciaQ**: retorna "**`True`**" si la **relación binaria** recibida como **pares ordenados** es de **equivalencia** o "**`False`**", en caso contrario.

Sintaxis: `RelacionEquivalenciaQ[R, A]`, con "R" la relación binaria definida sobre el conjunto finito no vacío "A".

Ejemplo 6.59

Determine si la relación binaria R definida sobre $A = \{1,2,3,4\}$ es de equivalencia:

$$R = \{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3),(4,4)\}$$

Solución:

En *Mathematica*:

In[ ] :=

```
A = {1, 2, 3, 4};
R = {{1, 1}, {1, 2}, {1, 3}, {2, 1}, {2, 2}, {2, 3}, {3, 1}, {3, 2},
{3, 3}, {4, 4}};
RelacionEquivalenciaQ[R, A]
```

Out[ ] :=

True

N "True" indica que R sí es una relación de equivalencia.

Ejemplo 6.60

¿Es R una relación de equivalencia? con: $aRb \Leftrightarrow a + b \geq 6$ definida sobre $A = \{1,2,3,4,5\}$.

Solución:

In[ ] :=

```
A = {1, 2, 3, 4, 5};
R = RelBin["a+b>=6", A, A];
RelacionEquivalenciaQ[R, A]
```

Out[ ] :=

False

Se concluye que R no es de equivalencia.

Explicación en video

31) **RelacionOrdenParcialQ**: retorna "`True`" si la **relación binaria** recibida como parámetro de **manera explícita** es de **orden parcial** o "`False`", en caso contrario.

Sintaxis: `RelacionOrdenParcialQ[R, A]`, con "R" la relación binaria definida sobre el conjunto finito no vacío "A".

Ejemplo 6.61

Sea $aRb \Leftrightarrow \log_b a$ es un número entero, definida sobre $A = \{2,4,6,\ldots,100\}$, ¿R es de orden parcial?

Solución:

En el software:

In[ ] :=
```
A = Table[2i, {i, 1, 50}];
R = RelBin["IntegerQ[Log[b,a]]", A, A];
RelacionOrdenParcialQ[R, A]
```

Out[ ] :=
True

Ejemplo 6.62

Determine si la siguiente relación binaria es de orden parcial: $aRb \Leftrightarrow b^3 \geq a$ siendo $A = \{1,3,5,\ldots,99\}$.

Solución:

In[ ] :=
```
A = Table[2i - 1, {i, 1, 50}];
R = RelBin["b^3>=a", A, A];
RelacionOrdenParcialQ[R, A]
```

Out[ ] :=
False

Explicación en video

32) **ClasesEquivalencia**: retorna **todas** las **clases de equivalencia** de una **relación de equivalencia** definida sobre un conjunto "A" finito distinto de vacío. La relación se recibe a través de sus **pares ordenados** de **forma explícita**.

Sintaxis: `ClasesEquivalencia[R, A]`, con "R" la relación binaria definida sobre "A".

Ejemplo 6.63

Halle todas las clases de equivalencia de $R = \{(3, 3), (3, 9), (3, a), (5, 5), (5, b), (5, p), (9, 3), (9, 9), (9, a), (a, 3), (a, 9), (a, a), (b, 5), (b, b), (b, p), (n, n), (n, y), (p, 5), (p, b), (p, p), (t, t), (y, n), (y, y)\}$.

Solución:

Al utilizar la instrucción **`ClasesEquivalencia`** se tiene:

In[ ] :=

```
A = {a, b, n, 3, 5, t, y, 9, p};
R = {{3, 3}, {3, 9}, {3, a}, {5, 5}, {5, b}, {5, p}, {9, 3}, {9, 9},
{9, a}, {a, 3}, {a, 9}, {a, a}, {b, 5}, {b, b}, {b, p}, {n, n}, {n,
y}, {p, 5}, {p, b}, {p, p}, {t, t}, {y, n}, {y, y}};
ClasesEquivalencia[R, A]
```

Out[ ] :=

```
[a]={a, 3, 9}
[b]={b, 5, p}
[n]={n, y}
[3]={a, 3, 9}
[5]={b, 5, p}
[t]={t}
[y]={n, y}
[9]={a, 3, 9}
[p]={b, 5, p}
El conjunto de clases de equivalencia distintas es:
{{a, 3, 9}, {b, 5, p}, {n, y}, {t}}
```

El último conjunto retornado por **`ClasesEquivalencia`**, como el lector recordará, es una partición de **`A`**.

Ejemplo 6.64

Construya un código en *Mathematica* que genere una partición seudoaleatoria sobre A, constituido este conjunto por números enteros seudoaleatorios de uno a cien y de cardinalidad máxima quince. Además, se debe encontrar la relación R de equivalencia definida por la partición y mediante el uso del comando **`ClasesEquivalencia`** se retorne todas las clases de equivalencia de R.

Solución:

En el software se podría proceder de la siguiente manera:

In[ ] :=

```
Quiet[<<Combinatorica`]
A = DeleteDuplicates[RandomInteger[{1, 100}, 15]]
Particion = RandomSetPartition[A]
R = {};
```

```
For[i = 1, i <= Length[Particion], R = Sort[Union[R,
PC[Particion[[i]], Particion[[i]]]]]; i++]
R
ClasesEquivalencia[R, A]
```

Out[ ] :=

{99, 40, 1, 100, 19, 84, 81, 37, 74, 24, 70, 98}

{{99, 100, 74}, {40, 19, 84, 81, 37, 24}, {1}, {70}, {98}}

{{1, 1}, {19, 19}, {19, 24}, {19, 37}, {19, 40}, {19, 81}, {19, 84}, {24, 19}, {24, 24}, {24, 37}, {24, 40}, {24, 81}, {24, 84}, {37, 19}, {37, 24}, {37, 37}, {37, 40}, {37, 81}, {37, 84}, {40, 19}, {40, 24}, {40, 37}, {40, 40}, {40, 81}, {40, 84}, {70, 70}, {74, 74}, {74, 99}, {74, 100}, {81, 19}, {81, 24}, {81, 37}, {81, 40}, {81, 81}, {81, 84}, {84, 19}, {84, 24}, {84, 37}, {84, 40}, {84, 81}, {84, 84}, {98, 98}, {99, 74}, {99, 99}, {99, 100}, {100, 74}, {100, 99}, {100, 100}}

[99]={99, 100, 74}

[40]={40, 19, 84, 81, 37, 24}

[1]={1}

[100]={99, 100, 74}

[19]={40, 19, 84, 81, 37, 24}

[84]={40, 19, 84, 81, 37, 24}

[81]={40, 19, 84, 81, 37, 24}

[37]={40, 19, 84, 81, 37, 24}

[74]={99, 100, 74}

[24]={40, 19, 84, 81, 37, 24}

[70]={70}

[98]={98}

El conjunto de clases de equivalencia distintas es:

{{99, 100, 74}, {40, 19, 84, 81, 37, 24}, {1}, {70}, {98}}

N **Combinatorica** es un paquete propio del software *Mathematica*, en este ejemplo se ha empleado para construir una partición seudoaleatoria mediante el uso del comando **RandomSetPartition**. La solución mostrada es dinámica, es decir, sin cambiar absolutamente nada en el código del *In*[ ], cada vez que se ejecuta, mostrará otro resultado.

Explicación en video

33) **ParticionReEquivalencia**: retorna una **partición** sobre un conjunto finito no vacío "A", **construida** a través de una **relación de equivalencia** "R" sobre "A". La relación se ingresa de **manera explícita** mediante **pares ordenados**.

Sintaxis: `ParticionReEquivalencia[R, A]`.

Ejemplo 6.65

Tomando como base el código expuesto en el ejemplo anterior, utilice el comando **ParticionReEquivalencia** sobre la relación de equivalencia **R** devuelta.

Solución:

En el programa:

In[ ] :=

```
Quiet[<<Combinatorica`]
A = DeleteDuplicates[RandomInteger[{1, 100}, 15]]
Particion = RandomSetPartition[A]
R = {};
For[i = 1, i <= Length[Particion], R = Sort[Union[R,
PC[Particion[[i]], Particion[[i]]]]]; i++]
R
ParticionReEquivalencia[R, A]
```

Out[ ] :=

{80, 56, 88, 37, 8, 9, 57, 75, 60, 52, 20, 4, 28, 43}
{{80, 4, 28, 43}, {56, 88, 57, 20}, {37}, {8}, {9, 60}, {75, 52}}
{{4, 4}, {4, 28}, {4, 43}, {4, 80}, {8, 8}, {9, 9}, {9, 60}, {20, 20}, {20, 56}, {20, 57}, {20, 88}, {28, 4}, {28, 28}, {28, 43}, {28, 80}, {37, 37}, {43, 4}, {43, 28}, {43, 43}, {43, 80}, {52, 52}, {52, 75}, {56, 20}, {56, 56}, {56, 57}, {56, 88}, {57, 20}, {57, 56}, {57, 57}, {57, 88}, {60, 9}, {60, 60}, {75, 52}, {75, 75}, {80, 4}, {80, 28}, {80, 43}, {80, 80}, {88, 20}, {88, 56}, {88, 57}, {88, 88}}
{{80, 4, 28, 43}, {8}, {9, 60}, {20, 56, 57, 88}, {37}, {52, 75}}

Nuevamente en este caso, la salida es dinámica. Cada vez que se ejecuta el $In[]$, el resultado arrojado será distinto. En el ejemplo se observa lógicamente, la coincidencia entre el contenido de la variable **Particion** con la salida mostrada por la instrucción **ParticionReEquivalencia**.

Ejemplo 6.66

Sea la relación de equivalencia: $aRb \Leftrightarrow a \equiv r \ mod\ 3 \wedge b \equiv r \ mod\ 3$ sobre $A = \{1,2,3,\ldots,100\}$. Encuentre la partición del conjunto A que es posible formar a través de R.

Solución:

En *Mathematica*:

In[ ] :=

```
A = Table[i, {i, 1, 100}];
R = RelBin["Mod[a,3]==Mod[b,3]", A, A]
ParticionReEquivalencia[R, A]
```

Out[ ] :=

```
{{1, 1}, {1, 4}, {1, 7}, {1, 10}, {1, 13}, {1, 16}, {1, 19}, {1, 22}, {1, 25}, {1, 28}, {1, 31},
 {1, 34}, {1, 37}, {1, 40}, ··· 3306 ···, {100, 61}, {100, 64}, {100, 67}, {100, 70}, {100, 73},
 {100, 76}, {100, 79}, {100, 82}, {100, 85}, {100, 88}, {100, 91}, {100, 94}, {100, 97}, {100, 100}}
```

salida grande | Mostrar menos | Mostrar más | Mostrar salida completa | Establecer límite de tamaño...

{{1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49, 52, 55, 58, 61, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91, 94, 97, 100}, {2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47, 50, 53, 56, 59, 62, 65, 68, 71, 74, 77, 80, 83, 86, 89, 92, 95, 98}, {3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99}}

N Al contener **R** una cantidad de pares ordenados significativamente grande, el software *Mathematica* no muestra por defecto todos los pares en su totalidad. Si el usuario lo desea puede presionar el botón "Mostrar salida completa". Es importante analizar en este ejercicio su nivel de complejidad, si se resolviera sin la ayuda de software.

Explicación en video

34) **ReEquivalenciaParticion**: **construye** una **relación de equivalencia**, dada una **partición** "P" definida sobre un conjunto finito no vacío "A".

Sintaxis: `ReEquivalenciaParticion[P, A]`.

Ejemplo 6.67

Considere la partición $P = \{\{a,b,n\},\{3,5,t\},\{y,9,p\}\}$ sobre el conjunto $A = \{a,b,n,3,5,t,y,9,p\}$. Halle la relación de equivalencia definida por P.

Solución:

Al emplear la instrucción **ReEquivalenciaParticion** se obtiene:

In[ ] :=
```
A = {a, b, n, 3, 5, t, y, 9, p};
ReEquivalenciaParticion[{{a, b, n}, {3, 5, t}, {y, 9, p}}, A]
```

Out[ ] :=

{{3, 3}, {3, 5}, {3, t}, {5, 3}, {5, 5}, {5, t}, {9, 9}, {9, p}, {9, y}, {a, a}, {a, b}, {a, n}, {b, a}, {b, b}, {b, n}, {n, a}, {n, b}, {n, n}, {p, 9}, {p, p}, {p, y}, {t, 3}, {t, 5}, {t, t}, {y, 9}, {y, p}, {y, y}}

Ejemplo 6.68

Sea $A = \{1,2,3,\ldots,20\}$ y $P = \{\{1,2\},\{3,4\},\{5,6\},\{7,8\},\{9,10\},\{11,12\},\{13,14\},\{15,16\},\{17,18\},\{19,20\}\}$ una partición sobre A. Determine la relación de equivalencia implícita en P.

Solución:

En el software:

In[ ] :=

```
A = Table[i, {i, 1, 20}];
P = {{1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10}, {11, 12}, {13, 14},
{15, 16}, {17, 18}, {19, 20}};
ReEquivalenciaParticion[P, A]
```

Out[ ] :=

{{1, 1}, {1, 2}, {2, 1}, {2, 2}, {3, 3}, {3, 4}, {4, 3}, {4, 4}, {5, 5}, {5, 6}, {6, 5}, {6, 6}, {7, 7}, {7, 8}, {8, 7}, {8, 8}, {9, 9}, {9, 10}, {10, 9}, {10, 10}, {11, 11}, {11, 12}, {12, 11}, {12, 12}, {13, 13}, {13, 14}, {14, 13}, {14, 14}, {15, 15}, {15, 16}, {16, 15}, {16, 16}, {17, 17}, {17, 18}, {18, 17}, {18, 18}, {19, 19}, {19, 20}, {20, 19}, {20, 20}}

Explicación en video

35) **AllReEquivalencia**: **construye todas** las **relaciones de equivalencia posibles** sobre un conjunto finito no vacío "A" recibido como parámetro, mediante **todas** las **particiones existentes** en "A".

Sintaxis: `AllReEquivalencia[A]`.

Ejemplo 6.69

Halle todas las relaciones de equivalencia posibles sobre $A = \{1,2,a,b\}$.

Solución:

In[ ] :=

```
A = {1, 2, a, b};
AllReEquivalencia[A]
```

Out[ ] :=

{{1 -> {{1, 1}, {1, 2}, {1, a}, {1, b}, {2, 1}, {2, 2}, {2, a}, {2, b}, {a, 1}, {a, 2}, {a, a}, {a, b}, {b, 1}, {b, 2}, {b, a}, {b, b}}}, {2 -> {{1, 1}, {2, 2}, {2, a}, {2, b}, {a, 2}, {a, a}, {a, b}, {b, 2}, {b, a}, {b, b}}}, {3 -> {{1, 1}, {1, 2}, {2, 1}, {2, 2}, {a, a}, {a, b}, {b, a}, {b, b}}}, {4 -> {{1, 1}, {1, a}, {1, b}, {2, 2}, {a, 1}, {a, a}, {a, b}, {b, 1}, {b, a}, {b, b}}}, {5 -> {{1, 1}, {1, 2}, {1, a}, {2, 1}, {2, 2}, {2, a}, {a, 1}, {a, 2}, {a, a}, {b, b}}}, {6 -> {{1, 1}, {1, b}, {2, 2}, {2, a}, {a, 2}, {a, a}, {b, 1}, {b, b}}}, {7 -> {{1, 1}, {1, 2}, {1, b}, {2, 1}, {2, 2}, {2, b}, {a, a}, {b, 1}, {b, 2}, {b, b}}}, {8 -> {{1, 1}, {1, a}, {2, 2}, {2, b}, {a, 1}, {a, a}, {b, 2}, {b, b}}}, {9 -> {{1, 1}, {2, 2}, {a, a}, {a, b}, {b, a}, {b, b}}}, {10 -> {{1, 1}, {2, 2}, {2, a}, {a, 2}, {a, a}, {b, b}}}, {11 -> {{1, 1}, {2, 2}, {2, b}, {a, a}, {b, 2}, {b, b}}},

{12 -> {{1, 1}, {1, 2}, {2, 1}, {2, 2}, {a, a}, {b, b}}}, {13 -> {{1, 1}, {1, a}, {2, 2}, {a, 1}, {a, a}, {b, b}}}, {14 -> {{1, 1}, {1, b}, {2, 2}, {a, a}, {b, 1}, {b, b}}}, {15 -> {{1, 1}, {2, 2}, {a, a}, {b, b}}}}

El comando retorna quince relaciones de equivalencia distintas.

Ejemplo 6.70

Construya de forma seudoaleatoria un conjunto A constituido como máximo por cinco números enteros de uno a cien y encuentre todas las posibles relaciones de equivalencia sobre A.

Solución:

En *Mathematica*:

In[ ] :=

```
A = DeleteDuplicates[RandomInteger[{1, 100}, 5]]
AllReEquivalencia[A]
```

Out[ ] :=

{87, 92, 18, 95, 44}

{{1 -> {{18, 18}, {18, 44}, {18, 87}, {18, 92}, {18, 95}, {44, 18}, {44, 44}, {44, 87}, {44, 92}, {44, 95}, {87, 18}, {87, 44}, {87, 87}, {87, 92}, {87, 95}, {92, 18}, {92, 44}, {92, 87}, {92, 92}, {92, 95}, {95, 18}, {95, 44}, {95, 87}, {95, 92}, {95, 95}}}, {2 -> {{18, 18}, {18, 44}, {18, 92}, {18, 95}, {44, 18}, {44, 44}, {44, 92}, {44, 95}, {87, 87}, {92, 18}, {92, 44}, {92, 92}, {92, 95}, {95, 18}, {95, 44}, {95, 92}, {95, 95}}}, {3 -> {{18, 18}, {18, 44}, {18, 95}, {44, 18}, {44, 44}, {44, 95}, {87, 87}, {87, 92}, {92, 87}, {92, 92}, {95, 18}, {95, 44}, {95, 95}}}, ..., {51 -> {{18, 18}, {44, 44}, {44, 87}, {87, 44}, {87, 87}, {92, 92}, {95, 95}}}, {52 -> {{18, 18}, {44, 44}, {87, 87}, {92, 92}, {95, 95}}}}

N La instrucción **`AllReEquivalencia`** encontró cincuenta y dos relaciones de equivalencia distintas, no se muestran en detalle por su tamaño. Además, se aclara al lector que **`DeleteDuplicates`** elimina elementos duplicados (si los tuviera) en el conjunto seudoaleatorio retornado por **`RandomInteger`**.

Explicación en video

36) **RandomReEquivalencia**: **genera** de manera **seudoaleatoria** "n" **relaciones de equivalencia distintas** sobre un conjunto finito no vacío "A". Si "n" es **mayor o igual** que la cantidad de relaciones de equivalencia **existentes**, muestra **todas** las **relaciones de equivalencia posibles**.

Sintaxis: **`RandomReEquivalencia[A, n]`**.

Ejemplo 6.71

Sea A un conjunto seudoaleatorio de cardinalidad máxima cinco, formado por números enteros de uno a cien. Determine de manera seudoaleatoria diez relaciones de equivalencia distintas sobre A.

Solución:

En el software:

In[ ] :=

```
A = DeleteDuplicates[RandomInteger[{1, 100}, 5]]
RandomReEquivalencia[A, 10]
```

Out[ ] :=

{4, 46, 100, 73, 5}

{{1 -> {{4, 4}, {4, 100}, {5, 5}, {46, 46}, {46, 73}, {73, 46}, {73, 73}, {100, 4}, {100, 100}}}, {2 -> {{4, 4}, {4, 5}, {4, 100}, {5, 4}, {5, 5}, {5, 100}, {46, 46}, {46, 73}, {73, 46}, {73, 73}, {100, 4}, {100, 5}, {100, 100}}}, {3 -> {{4, 4}, {4, 73}, {5, 5}, {5, 46}, {46, 5}, {46, 46}, {73, 4}, {73, 73}, {100, 100}}}, {4 -> {{4, 4}, {4, 5}, {5, 4}, {5, 5}, {46, 46}, {46, 73}, {73, 46}, {73, 73}, {100, 100}}}, {5 -> {{4, 4}, {4, 46}, {5, 5}, {5, 100}, {46, 4}, {46, 46}, {73, 73}, {100, 5}, {100, 100}}}, {6 -> {{4, 4}, {4, 5}, {5, 4}, {5, 5}, {46, 46}, {73, 73}, {100, 100}}}, {7 -> {{4, 4}, {4, 5}, {5, 4}, {5, 5}, {46, 46}, {46, 73}, {46, 100}, {73, 46}, {73, 73}, {73, 100}, {100, 46}, {100, 73}, {100, 100}}}, {8 -> {{4, 4}, {5, 5}, {5, 46}, {46, 5}, {46, 46}, {73, 73}, {73, 100}, {100, 73}, {100, 100}}}, {9 -> {{4, 4}, {5, 5}, {46, 46}, {73, 73}, {73, 100}, {100, 73}, {100, 100}}}, {10 -> {{4, 4}, {4, 73}, {4, 100}, {5, 5}, {46, 46}, {73, 4}, {73, 73}, {73, 100}, {100, 4}, {100, 73}, {100, 100}}}}

N El *Out* [] anterior cambia cada vez que el usuario ejecuta el *In* [].

Ejemplo 6.72

Considere el siguiente código de *Mathematica*:

```
RandomAlfabeto[n_] := If[n <= 26, Module[{}, L = RandomInteger[{1, 26}, n];
While[Length[DeleteDuplicates[L]] != Length[L], L = RandomInteger[{1, 26},
n]]; For[i = 1, i <= Length[L], L[[i]] = Alphabet[][[L[[i]]]]; i++]; L]]
```

La función **RandomAlfabeto[n]** brinda una lista seudoaleatoria con "n" elementos distintos del abecedario. Utilice **RandomAlfabeto** para construir un conjunto A con seis elementos y halle veinte relaciones de equivalencia distintas definidas sobre A.

Solución:

In[ ] :=

```
RandomAlfabeto[n_] := If[n <= 26, Module[{}, L = RandomInteger[{1,
26}, n]; While[Length[DeleteDuplicates[L]] != Length[L], L =
RandomInteger[{1, 26}, n]]; For[i = 1, i <= Length[L], L[[i]] =
Alphabet[][[L[[i]]]]; i++]; L]]
A = RandomAlfabeto[6]
RandomReEquivalencia[A, 20]
```

Out[ ] :=
{l, c, p, k, b, n}
{{1 -> {{b, b}, {b, k}, {b, p}, {c, c}, {k, b}, {k, k}, {k, p}, {l, l}, {l, n}, {n, l}, {n, n}, {p, b}, {p, k}, {p, p}}}, {2 -> {{b, b}, {b, n}, {c, c}, {c, l}, {c, p}, {k, k}, {l, c}, {l, l}, {l, p}, {n, b}, {n, n}, {p, c}, {p, l}, {p, p}}}, {3 -> {{b, b}, {c, c}, {c, p}, {k, k}, {k, l}, {k, n}, {l, k}, {l, l}, {l, n}, {n, k}, {n, l}, {n, n}, {p, c}, {p, p}}}, ..., {19 -> {{b, b}, {b, p}, {c, c}, {c, l}, {c, n}, {k, k}, {l, c}, {l, l}, {l, n}, {n, c}, {n, l}, {n, n}, {p, b}, {p, p}}}, {20 -> {{b, b}, {c, c}, {c, n}, {k, k}, {l, l}, {n, c}, {n, n}, {p, p}}}}

N Como ha ocurrido con otros ejemplos, en este caso, la salida es dinámica por lo que si el alumno corre el mismo código, no necesariamente obtendrá el resultado anterior.

Explicación en video

Aporte pedagógico

En esta sección se han compartido muchos comandos mediante los cuáles, el estudiante tiene la posibilidad de **verificar** por **sí mismo** el resultado de ejercicios de muy distinta naturaleza, donde se asume su resolución previa, utilizando lápiz y papel. Este tipo de instrucciones se conciben **esenciales** en *VilCretas* para proporcionar herramientas de **autoconsulta** y **autoaprendizaje**. Algunas de ellas son: **`RelBinOperaciones`**, **`RelBinUnion`**, **`RelBinInter`**, **`RelBinComplement`**, **`RelBinInver`**, **`RelBinComp`**, **`CDFOperacionesMBooleanas`**, **`MRelBinUnion`**, **`MRelBinInter`**, **`MRelBinComplement`**, **`MRelBinInver`**, **`MRelBinComp`**, entre otras.

Aporte de investigación

Una instrucción muy interesante con un claro potencial para proponer ejercicios de **conjetura** la constituye **`RelBin`**. Mediante este comando es posible **estudiar** de **manera explícita** distintas **relaciones** construidas de forma automática bajo el principio de contener las condiciones que la definen. Un aspecto esencial de estas condiciones es que en ellas se puede integrar **cualquier comando** nativo de *Mathematica*, facilitando la **exploración** de relaciones binarias **no convencionales**.

Ejercicios

Resuelva los siguientes ejercicios utilizando como apoyo el paquete *VilCretas*.

1) Calcule el producto cartesiano dado y encuentre su cardinalidad:

   a) $\{1,2,3,\ldots,100\} \times \{4,6,8,\ldots,200\}$

   b) $\{2,4,8,\ldots,1048576\} \times \{9,27,81,\ldots,14348907\}$

Sugerencia: para la cardinalidad emplee **Length**.

2) Represente en el plano cartesiano $[-10,+\infty[\times [-5,1000]$, a través de la instrucción **GraficaPC**.

3) Mediante el uso del comando **RelBin** determine el gráfico de las siguientes relaciones binarias:

   a) $aR_1b \Leftrightarrow a$ y b son números palíndromos, con $a,b \in$ {11, 13, 17, 19, 21, 22, 23, 29, 32, 51, 72, 83, 89, 97, 113, 121, 127, 222, 312, 723}. Se sugiere el uso del comando **PalindromeQ** del software *Mathematica*.

   b) $aR_2b \Leftrightarrow a^2 - b^2$ es múltiplo de 5, con $a,b \in$ {1, 2, 3, 4, 5}. Utilice la instrucción **Mod** de *Mathematica*.

   c) $aR_3b \Leftrightarrow$ el residuo de la división $(a-3) \div 3$ es igual al residuo $(b-3) \div 3$, definida sobre $A =$ {1, 2, 3, 4, 5}.

   d) $aR_4b \Leftrightarrow a - b \in A$, con $A = \{0,1,3,4\}$, definida sobre A. Use el comando **MemberQ** del software *Mathematica*.

   e) $aR_5b \Leftrightarrow a - b + \sqrt[3]{2} \geq 10$, con $a,b \in$ {1, 2, 3, ..., 50}.

   f) $aR_6b \Leftrightarrow a + b$ es un número impar, $a,b \in$ {1, 2, 3, ..., 50}.

   g) $aR_7b \Leftrightarrow \dfrac{a^2}{36} - \dfrac{b^2}{8} \leq -\dfrac{1}{2}$, $a,b \in$ {1, 2, 3, ..., 100}.

   h) $aR_8b \Leftrightarrow \dfrac{a^2}{36} + \dfrac{b^2}{8} \geq -\dfrac{1}{2}$, $a,b \in$ {1, 2, 3, ..., 100}.

   i) $aR_9b \Leftrightarrow$ el máximo común divisor entre a y b es igual a 9, sobre $A =$ {1, 3, 5, ..., 99} y $B =$ {2, 4 , 6, ..., 100} ¿Cuál es el valor mínimo del máximo común divisor para que la relación sea distinta de vacío?

   j) $aR_{10}b \Leftrightarrow$ el mínimo común múltiplo entre a y b es igual a 300, sobre $A =$ {1, 3, 5, ..., 99} y $B =$ {2, 4 , 6, ..., 100} ¿Cuál es el valor máximo del mínimo común múltiplo para que la relación sea distinta de vacío?

4) Grafique utilizando la instrucción **GraficaRelBin** del paquete *VilCretas*, las relaciones R_7 y R_8, suponiendo ambas definidas sobre el conjunto de los números reales.

5) Grafique haciendo uso del comando **GraficaRelBinPares** las relaciones binarias expuestas en el ejercicio 3.

6) Determine si **RandomChoice[PC[A, B]]** es un elemento de cada una de las relaciones del ejercicio 3. Sugerencia: recurra a la sentencia **ElementRelBinQ**.

7) Obtenga una matriz que represente cada una de las relaciones del ejercicio 3.

8) Encuentre un grafo que represente la relación R_j con $j \in$ {1, 2, 3, 4, 5, 6, 7, 8}.

9) Halle el gráfico de la relación binaria definida sobre $A =$ {1, 2, 3, ..., 10}, representada por la matriz **RandomInteger[{0, 1}, {10, 10}]**.

10) Resuelva por definición y utilizando matrices booleanas:

   a) $R_2 \cup R_3$ y $R_2 \cap R_3$

b) $\left(\overline{R_3} \cap (R_2)^{-1}\right) \cup R_3$

c) $\left(\overline{\overline{R_2} \cap (R_3)^{-1}}\right)^{-1} o\,(R_3)^{-1}$

d) $((R_5 o R_6)\, o R_5)^{-1} \cup \left(\left(\overline{R_6 o R_5}\right) \cap (R_5)^{-1}\right)$

11) Realice todas las operaciones booleanas entre dos matrices seudoaleatorias generadas por: **`RandomInteger[{0, 1}, {10, 10}]`**. Sugerencia: utilice la instrucción **`CDFOperacionesMBooleanas`**.

12) Clasifique como reflexiva, simétrica, antisimétrica, transitiva, de equivalencia y de orden parcial, la relación binaria R_j con $j \in \{1, 2, 3, 4, 5, 6, 7, 8\}$. Resuelva por definición y usando matrices booleanas, a través de los comandos **`TipoRelacion`** y **`TipoMRelacion`**.

13) Determine utilizando la instrucción **`ClasesEquivalencia`** las clases de equivalencia de las relaciones R_2 y R_3.

14) Sea una partición P obtenida por **`RandomSetPartition[A]`** con $A = \{1, 2, 3, \ldots, 10\}$. Construya a través de P empleando el comando **`ReEquivalenciaParticion`**, una relación de equivalencia sobre A.

15) Retorne todas las relaciones de equivalencia posibles sobre el conjunto $A = \{1, 2, 3, \ldots, 10\}$ ¿Cuántas relaciones de equivalencia distintas existen?

Capítulo 7

Teoría de grafos con *VilCretas*

En este capítulo se compartirán con el lector **ochenta y ocho** instrucciones para tratar con un enfoque asistido por computadora la teoría de grafos. Los distintos comandos diseñados permiten al alumno estudiar el contenido en todas sus componentes: definiciones, propiedades, teoremas y algoritmos clásicos, enfocándose en facilitar herramientas orientadas hacia la visualización, construcción autónoma de conceptos y evidencia de resultados. *VilCretas* incorpora además, instrucciones para la generación de grafos con datos reales aprovechando las características de datos geográficos del software *Mathematica*. Se insta al educando a emprender el análisis de la teoría de grafos desde un punto de vista computacional.

1) **AristasWolframSystemToCombinatorica**: recibe las **aristas** de un grafo creado en el "Wolfram System" y las **convierte** a la forma que toman en un grafo generado con el paquete "Combinatorica" (**lista de pares ordenados**).

Sintaxis: `AristasWolframSystemToCombinatorica[Aristas]`.

Ejemplo 7.1

Transforme el conjunto de aristas dado a pares ordenados de acuerdo con la notación empleada por el software *Mathematica*:

- $\{a \bullet\!\!-\!\!\bullet\, b, 2 \bullet\!\!-\!\!\bullet\, 4, 4 \bullet\!\!-\!\!\bullet\, f, b \bullet\!\!-\!\!\bullet\, a\}$
- $\{a \bullet\!\!-\!\!> b, 2 \bullet\!\!-\!\!> 4, 4 \bullet\!\!-\!\!> f, b \bullet\!\!-\!\!> a\}$
- $\{a -> b, 2 -> 4, 4 -> f, b -> a\}$
- $\{a \bullet\!\!-\!\!> b, 2 \bullet\!\!-\!\!\bullet\, 4, 4 -> f, b \bullet\!\!-\!\!\bullet\, a\}$

Solución:

In[ ] :=

```
AristasWolframSystemToCombinatorica[{a <-> b, 2 <-> 4, 4 <-> f,
b <-> a}]
AristasWolframSystemToCombinatorica[{a •-> b, 2 •-> 4, 4 •-> f,
b •-> a}]
AristasWolframSystemToCombinatorica[{a -> b, 2 -> 4, 4 -> f,
b -> a}]
AristasWolframSystemToCombinatorica[{a •-> b, 2 <-> 4, 4 -> f,
b <-> a}]
```

Out[ ] :=

```
{{a, b}, {2, 4}, {4, f}, {b, a}}
{{a, b}, {2, 4}, {4, f}, {b, a}}
{{a, b}, {2, 4}, {4, f}, {b, a}}
```

{{a, b}, {2, 4}, {4, f}, {b, a}}

N El símbolo $\bullet\!\!-\!\!>$ puede generarse en *Mathematica* mediante el uso del comando **`DirectedEdge`**.

Ejemplo 7.2

Convierta todas las aristas de un grafo completo de orden diez creado en el "Wolfram System" de *Mathematica*.

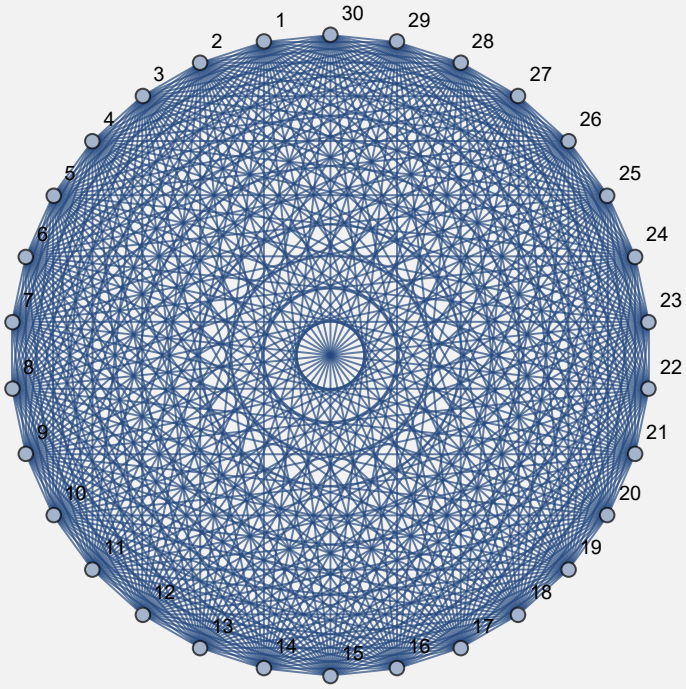
Solución:

En el software se cuenta con el comando **`CompleteGraph`** para generar un grafo completo, luego:

In[ ] :=

```
grafo = System`CompleteGraph[30, VertexLabels -> "Name",
ImagePadding -> 10]
AristasWolframSystemToCombinatorica[EdgeList[grafo]]
```

Out[ ] :=



{{1, 2}, {1, 3}, {1, 4}, ..., {28, 30}, {29, 30}}

N Por sus dimensiones el *Out*[] no se muestra en detalle. **`EdgeList`** es un comando de *Mathematica* que regresa la lista de todas las aristas de un grafo. Por otra parte, el estudiante debe tener claro que la instrucción **`AristasWolframSystemToCombinatorica`** resulta de utilidad, para pasar de un grafo creado en el "Wolfram System" a otro, en el entorno del paquete "Combinatorica".

Explicación en video

2) **AristasCombinatoricaToWolframSystem**: recibe los **lados** de un grafo construido mediante el paquete "Combinatorica" y **retorna** el conjunto como las aristas de un grafo **creado** en el "Wolfram System".

Sintaxis: **`AristasCombinatoricaToWolframSystem[Aristas]`**, o bien,

```
AristasCombinatoricaToWolframSystem[Aristas, dirigido->True]
```

si se desea obtener los **lados** de un **digrafo**.

Ejemplo 7.3

Convierta los siguientes conjuntos de aristas al "Wolfram System":

- {{a, b}, {2, 4}, {4, f}, {b, a}}
- {{a, b}, {2, 4}, {4, f}, {b, a}} como lados dirigidos

Solución:

En el software:

```
In[ ]:=
AristasCombinatoricaToWolframSystem[{{a, b}, {2, 4}, {4, f}, {b, a}}]
AristasCombinatoricaToWolframSystem[{{a, b}, {2, 4}, {4, f}, {b, a}},
dirigido -> True]
Out[ ]:=
{a ●—● b, 2 ●—● 4, 4 ●—● f, b ●—● a}
{a ●—> b, 2 ●—> 4, 4 ●—> f, b ●—> a}
```

N La opción "**dirigido->True**" permite crear con **AristasCombinatorica-ToWolframSystem** lados dirigidos.

Ejemplo 7.4

Transforme el conjunto "aristas"como lados de un grafo creado en el "Wolfram System" de *Mathematica*, considerando aristas no dirigidas y dirigidas, con: aristas = {{1, 13}, {1, 25}, {1, 40}, {2, 15}, {2, 42}, {2, 46}, {3, 42}, {4, 9}, {4, 29}, {4, 41}, {5, 34}, {5, 43}, {6, 13}, {6, 14}, {6, 49}, {7, 12}, {9, 38}, {9, 42}, {9, 49}, {10, 29}, {11, 49}, {12, 19}, {12, 20}, {12, 21}, {12, 33}, {13, 18}, {13, 20}, {14, 36}, {15, 24}, {15, 38}, {15, 45}, {17, 31}, {21, 37}, {22, 28}, {22, 46}, {23, 28}, {26, 34}, {28, 36}, {28, 43}, {29, 43}, {32, 43}, {33, 43}, {33, 47}, {34, 45}, {34, 46}, {37, 38}, {41, 44}, {43, 50}, {44, 50}, {45, 50}}.

Solución:

In[ ] :=

```
aristas = {{1, 13}, {1, 25}, {1, 40}, {2, 15}, {2, 42}, {2, 46}, {3,
42}, {4, 9}, {4, 29}, {4, 41}, {5, 34}, {5, 43}, {6, 13}, {6, 14},
{6, 49}, {7, 12}, {9, 38}, {9, 42}, {9, 49}, {10, 29}, {11, 49}, {12,
19}, {12, 20}, {12, 21}, {12, 33}, {13, 18}, {13, 20}, {14, 36}, {15,
24}, {15, 38}, {15, 45}, {17, 31}, {21, 37}, {22, 28}, {22, 46}, {23,
28}, {26, 34}, {28, 36}, {28, 43}, {29, 43}, {32, 43}, {33, 43}, {33,
47}, {34, 45}, {34, 46}, {37, 38}, {41, 44}, {43, 50}, {44, 50}, {45,
50}};
AristasCombinatoricaToWolframSystem[aristas]
AristasCombinatoricaToWolframSystem[aristas, dirigido->True]
```

Out[ ] :=

{1 ●—● 13, 1 ●—● 25, 1 ●—● 40, 2 ●—● 15, 2 ●—● 42, 2 ●—● 46, 3 ●—● 42, 4 ●—● 9, 4 ●—● 29, 4 ●—● 41, 5 ●—● 34, 5 ●—● 43, 6 ●—● 13, 6 ●—● 14, 6 ●—● 49, 7 ●—● 12, 9 ●—● 38, 9 ●—● 42, 9 ●—● 49, 10 ●—● 29, 11 ●—● 49, 12 ●—● 19, 12 ●—● 20, 12 ●—● 21, 12 ●—● 33, 13 ●—● 18, 13 ●—● 20, 14 ●—● 36, 15 ●—● 24, 15 ●—● 38, 15 ●—● 45, 17 ●—● 31, 21 ●—● 37, 22 ●—● 28, 22 ●—● 46, 23 ●—● 28, 26 ●—● 34, 28 ●—● 36, 28 ●—● 43, 29 ●—● 43, 32 ●—● 43, 33 ●—● 43, 33 ●—● 47, 34 ●—● 45, 34 ●—● 46, 37 ●—● 38, 41 ●—● 44, 43 ●—● 50, 44 ●—● 50, 45 ●—● 50}

{1 ●—> 13, 1 ●—> 25, 1 ●—> 40, 2 ●—> 15, 2 ●—> 42, 2 ●—> 46, 3 ●—> 42, 4 ●—> 9, 4 ●—> 29, 4 ●—> 41, 5 ●—> 34, 5 ●—> 43, 6 ●—> 13, 6 ●—> 14, 6 ●—> 49, 7 ●—> 12, 9 ●—> 38, 9 ●—> 42, 9 ●—> 49, 10 ●—> 29, 11 ●—> 49, 12 ●—> 19, 12 ●—> 20, 12 ●—> 21, 12 ●—> 33, 13 ●—> 18, 13 ●—> 20, 14 ●—> 36, 15 ●—> 24, 15 ●—> 38, 15 ●—> 45, 17 ●—> 31, 21 ●—> 37, 22 ●—> 28, 22 ●—> 46, 23 ●—> 28, 26 ●—> 34, 28 ●—> 36, 28 ●—> 43, 29 ●—> 43, 32 ●—> 43, 33 ●—> 43, 33 ●—> 47, 34 ●—> 45, 34 ●—> 46, 37 ●—> 38, 41 ●—> 44, 43 ●—> 50, 44 ●—> 50, 45 ●—> 50}

Explicación en video

3) **AristasMixtasQ**: devuelve "**`True`**" si un **grafo creado** en el "Wolfram System" de *Mathematica*, posee **aristas dirigidas y no dirigidas** y, "**`False`**" en caso contrario.

Sintaxis: **`AristasMixtasQ[G]`**.

Ejemplo 7.5

Considere un grafo cuyos lados vienen dados por el conjunto: $\{a \bullet\!\!-\!\!> b, c \bullet\!\!-\!\!> d\}$. Utilice el comando **`AristasMixtasQ`** para determinar si posee aristas mixtas.

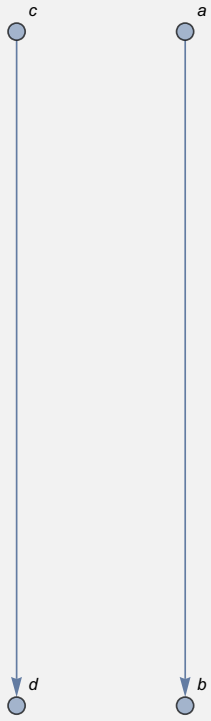
Solución:

En el software se empleará la instrucción de *VilCretas* denominada **`Grafo`**, para crear el grafo de interés:

In[ ] :=

```
grafo = Grafo[{{a, b}, {c, d}}, dirigido->True]
AristasMixtasQ[grafo]
```

Out[ ] :=

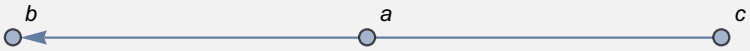


False

El grafo del ejemplo, solo tiene lados dirigidos, por lo que no presenta aristas mixtas.

Ejemplo 7.6

Determine si el siguiente grafo posee aristas mixtas mediante el uso del software *Mathematica*.



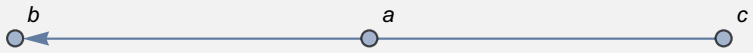
Solución:

Para crear el grafo se utilizará el comando **Graph** nativo de *Mathematica* (también se pudo haber empleado la instrucción **Grafo**):

In[ ] :=

```
grafo = System`Graph[{a -> b, a <-> c}, VertexLabels -> "Name",
ImagePadding -> 10]
AristasMixtasQ[grafo]
```

Out[ ] :=



True

N La respuesta es "**True**" pues el grafo posee un lado dirigido y otro no dirigido. **VertexLabels** añade etiquetas a los vértices del grafo creado con **Graph** y **ImagePadding** deja un espacio de contorno con la finalidad de evitar cortes en las etiquetas de los nodos.

Explicación en video

4) **AristasMixtasWolframSystemToCombinatorica**: recibe las **aristas** de un grafo con **lados mixtos** (**dirigidos y no dirigidos**) creado en el "Wolfram System" y las **convierte** a la forma que toman en un grafo generado con el paquete "Combinatorica" (**lista de pares ordenados**).

Sintaxis: **AristasMixtasWolframSystemToCombinatorica[Aristas]**, la salida de la instrucción **representa** los **lados** de un **grafo dirigido**.

Ejemplo 7.7

Convierta la lista de aristas mixtas $\{a->b, a\bullet\!\!-\!\!\bullet c, a\bullet\!\!-\!\!\bullet c, a\bullet\!\!-\!\!\bullet c, b\bullet\!\!-\!\!\bullet e, b\bullet\!\!-\!\!\bullet e, b->e\}$ a pares ordenados.

Solución:

En *Mathematica*:

In[ ] :=

```
AristasMixtasWolframSystemToCombinatorica[{a -> b, a <-> c, a <->
c, a <-> c, b <-> e, b <-> e, b -> e}]
```

Out[ ] :=

{{a, b}, {b, e}, {a, c}, {a, c}, {a, c}, {b, e}, {b, e}, {c, a}, {c, a}, {c, a}, {e, b}, {e, b}}

Ⓝ Se observa que por cada arista no dirigida se añaden a la salida dos más, una en cada dirección. Por ejemplo, la arista $a \bullet\!-\!\bullet\, c$ produce en el $Out[\,]$ dos pares ordenados: {a, c} y {c, a}.

Ejemplo 7.8

Dadas las aristas mixtas $\{a->b, a \bullet\!-\!\bullet\, c, b \bullet\!-\!\bullet\, e, h->e, f \bullet\!-\!\bullet\, g\}$, mediante uso de software pase la lista a pares ordenados.

Solución:

In[ ] :=

```
AristasMixtasWolframSystemToCombinatorica[{a -> b, a <-> c, b <->
e, h -> e, f <-> g}]
```

Out[ ] :=

{{a, b}, {h, e}, {a, c}, {b, e}, {f, g}, {c, a}, {e, b}, {g, f}}

Explicación en video

5) **Grafo**: construye un **grafo** a través del "Wolfram System" dadas sus **aristas** como una **matriz** de **pares ordenados**. El comando presenta **siete opciones**: "**dirigido -> True**", "**vertices -> Lista**", "**dimensions3d -> True**", "**pesos -> Lista**", "**mostrarpesos -> True**", "**shape -> True**" y "**padding -> Valor**". "dirigido" crea un **digrafo**, "vertices" añade **nuevos vértices** mediante una lista de nodos, "dimensions3d" presenta el grafo en **tercera dimensión**, "pesos" genera un **grafo ponderado**, "mostrarpesos" **muestra** los **pesos** sobre cada una de sus aristas, "shape" coloca los nodos del grafo en **discos**, o bien, **esferas** dependiendo de la dimensión de graficación y "**padding -> Valor**" añade un **espacio** de **contorno** especificado en "Valor". Si se muestran los pesos de un grafo con aristas múltiples, por limitaciones en el software *Mathematica*, toma la **misma ponderación** para cada **lado repetido**.

Sintaxis: **Grafo[Aristas]**, o bien,

```
Grafo[Aristas, dirigido->True, vertices->Lista, dimensions3d->True,
pesos->Lista, mostrarpesos->True, shape->True, padding->Valor]
```

En esta última invocación, es posible **prescindir** de cualquiera de las opciones y "Valor" es un **entero mayor o igual a diez**.

Ejemplo 7.9

Construya el grafo que representa la relación binaria: $aRb \Leftrightarrow a - b \geq 4$, con $a, b \in A = \{2,4,6,\ldots,20\}$, añadiendo adicionalmente los nodos aislados a y b.

Solución:

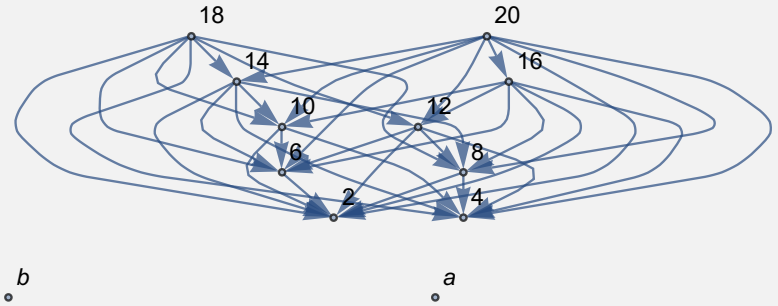
En este ejemplo se debe emplear la opción "**vertices -> Lista**" del comando **Grafo**:

In[ ] :=

```
A = Table[2i, {i, 1, 10}];
aristas = RelBin["a-b>=4", A, A]
grafo = Grafo[aristas, vertices -> {a, b}, dirigido -> True]
VertexList[grafo]
VertexInDegree[grafo]
VertexOutDegree[grafo]
```

Out[ ] :=

{{6, 2}, {8, 2}, {8, 4}, {10, 2}, {10, 4}, {10, 6}, {12, 2}, {12, 4}, {12, 6}, {12, 8}, {14, 2}, {14, 4}, {14, 6}, {14, 8}, {14, 10}, {16, 2}, {16, 4}, {16, 6}, {16, 8}, {16, 10}, {16, 12}, {18, 2}, {18, 4}, {18, 6}, {18, 8}, {18, 10}, {18, 12}, {18, 14}, {20, 2}, {20, 4}, {20, 6}, {20, 8}, {20, 10}, {20, 12}, {20, 14}, {20, 16}}



{a, b, 6, 2, 8, 4, 10, 12, 14, 16, 18, 20}

{0, 0, 6, 8, 5, 7, 4, 3, 2, 1, 0, 0}

{0, 0, 1, 0, 2, 0, 3, 4, 5, 6, 7, 8}

N Para comprobar la creación correcta del grafo, se recurrió al empleo de las instrucciones: **VertexList**, **VertexInDegree**, **VertexOutDegree**. La primera encuentra la lista de vértices del grafo y las otras dos, calculan las valencias internas y externas respectivamente, de sus vértices.

Ejemplo 7.10

Genere un grafo seudoaletorio con diez nodos y diez lados, para ello utilice el comando de *VilCretas* **GrafoRandom**. Empleando sus vértices y aristas, construya otro grafo: 3D y dirigido.

Solución:

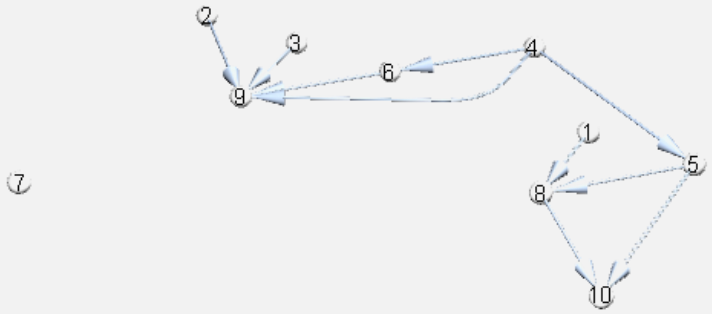
En el software:

In[ ] :=

```
grafo = GrafoRandom[10, 10];
aristas = AristasWolframSystemToCombinatorica[EdgeList[grafo]]
Grafo[aristas, vertices->VertexList[grafo], dirigido->True,
dimensions3d->True, shape->True]
```

Out[ ] :=

{{1, 8}, {2, 9}, {3, 9}, {4, 5}, {4, 6}, {4, 9}, {5, 8}, {5, 10}, {6, 9}, {8, 10}}



N La opción "**shape->True**" se agregó con la intención de colocar los nodos del grafo 3D en esferas. Por otra parte, cada vez que se ejecuta el *In*[] la salida mostrada será distinta.

Explicación en video

6) **GrafoC**: construye un **grafo** con el paquete "Combinatorica" de *Mathematica*, dadas sus **aristas** como una **matriz** de **pares ordenados**. El comando presenta **cuatro opciones**: "**dirigido->True**", "**vertices->n**", "**pesos->Lista**" y "**mostrarpesos->True**". "dirigido" genera un **digrafo**, "vértices" añade "n" **nodos aislados** en posiciones **seudoaleatorias**, "pesos" crea un **grafo ponderado** y "mostrarpesos" **muestra** los **pesos** sobre cada uno de los lados del grafo. Si se muestran los pesos de un grafo con aristas múltiples, por limitaciones en el software *Mathematica*, toma el **mismo valor** para cada **lado repetido**.

Sintaxis: **GrafoC[Aristas]**, o bien, **GrafoC[Aristas, dirigido->True, vertices->n, pesos->Lista, mostrarpesos->True]**, pudiendo **prescindir** de cualquiera de las opciones. Por defecto, el grafo creado se almacena en una **variable llamada** "G".

Ejemplo 7.11

Genere un grafo no dirigido con el paquete "Combinatorica", que posea cinco vértices aislados y cuyos lados vienen dados por: aristas = {{1, 2}, {1, 5}, {1, 6}, {2, 3}, {2, 7}, {3, 4}, {3, 8}, {4, 5}, {4, 9}, {5, 10}, {6, 11}, {6, 12}, {7, 11}, {7, 15}, {8, 14}, {8, 15}, {9, 13}, {9, 14}, {10, 12}, {10, 13}, {11, 16}, {12, 17}, {13, 18}, {14, 19}, {15, 20}, {16, 17}, {16, 20}, {17, 18}, {18, 19}, {19, 20}}.

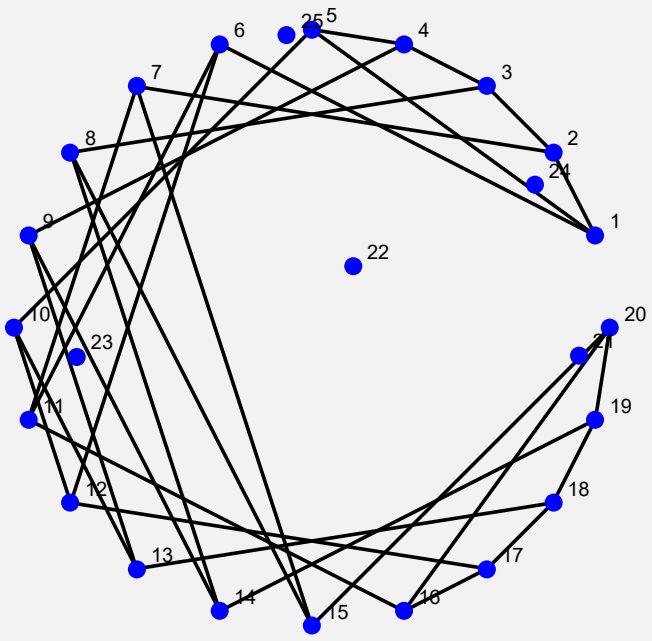
Solución:

Al utilizar **GrafoC**:

In[ ] :=

```
aristas = {{1, 2}, {1, 5}, {1, 6}, {2, 3}, {2, 7}, {3, 4}, {3, 8},
{4, 5}, {4, 9}, {5, 10}, {6, 11}, {6, 12}, {7, 11}, {7, 15}, {8, 14},
{8, 15}, {9, 13}, {9, 14}, {10, 12}, {10, 13}, {11, 16}, {12, 17},
{13, 18}, {14, 19}, {15, 20}, {16, 17}, {16, 20}, {17, 18}, {18, 19},
{19, 20}};
GrafoC[aristas, vertices -> 5]
```

Out[ ] :=



Ejemplo 7.12

Mediante el uso del paquete "Combinatorica", construya un grafo poderado con pesos iguales a uno y con: aristas = {{1, 2}, {1, 3}, {2, 3}, {4, 5}, {4, 6}, {5, 6}, {1, 4}, {1, 6}, {2, 4}, {2, 5}, {3, 5}, {3, 6}}.

Solución:

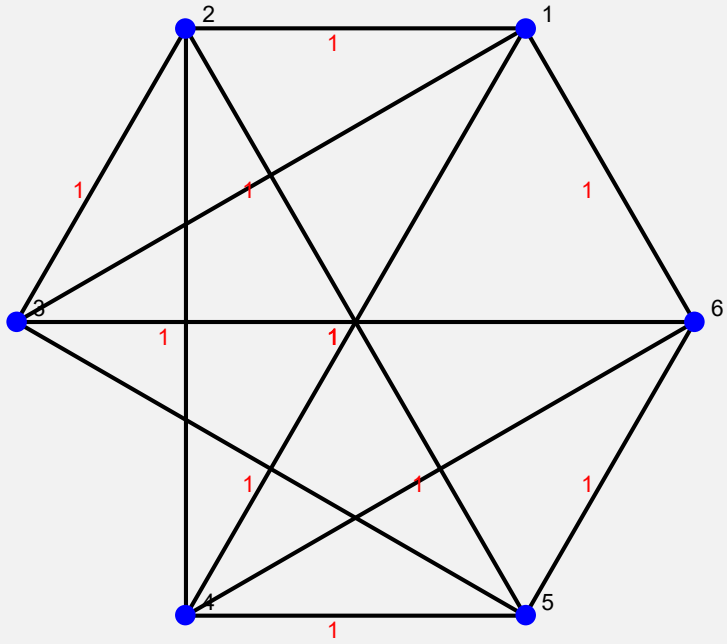
En *Mathematica*:

In[ ] :=

```
aristas = {{1, 2}, {1, 3}, {2, 3}, {4, 5}, {4, 6}, {5, 6}, {1, 4},
```

```
{1, 6}, {2, 4}, {2, 5}, {3, 5}, {3, 6}};
GrafoC[aristas, mostrarpesos -> True]
```

Out[ ] :=



N Al incluir la opción "**mostrarpesos -> True**" automáticamente se crea un grafo ponderado con pesos iguales a uno, en todos los lados del grafo.

Explicación en video

7) **GrafoQ**: retorna "**True**" si al recibir como parámetro un **grafo**, éste se ha **creado** en el **ambiente** provisto por "Wolfram System" y "**False**", si el argumento **no es un grafo**, o bien, es un **grafo construido** mediante el **paquete** "Combinatorica".

Sintaxis: `GrafoQ[G]`.

Ejemplo 7.13

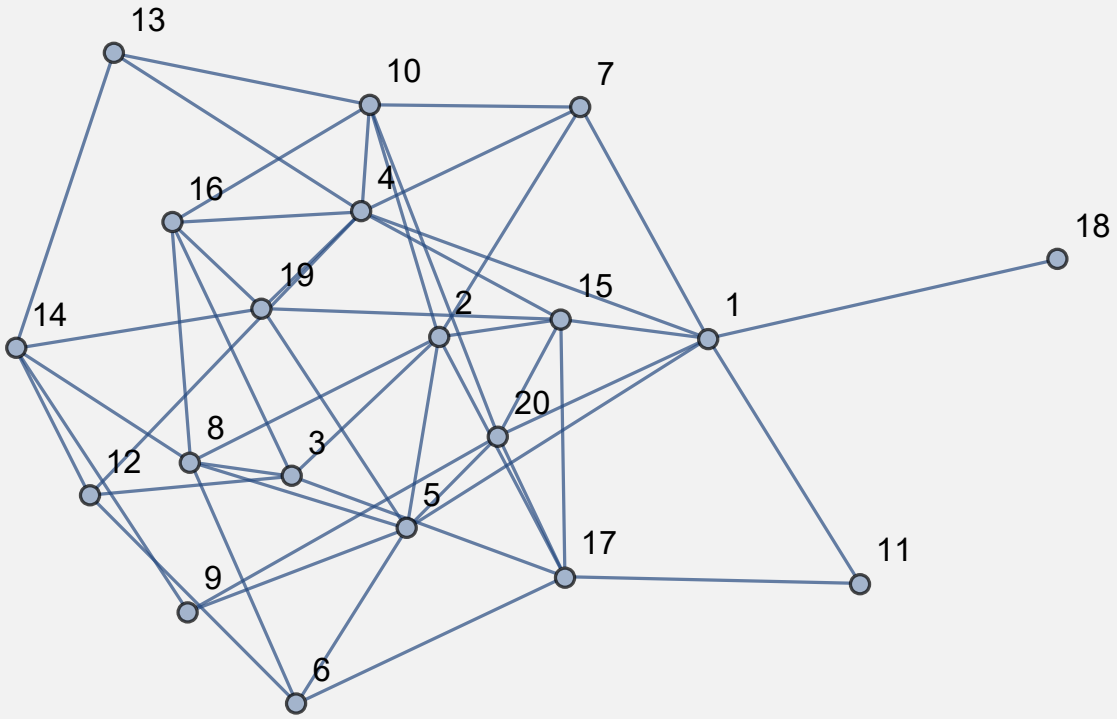
Determine con ayuda de software si un grafo retornado con la instrucción **GrafoRandom** con veinte vértices y cincuenta lados, es un grafo del "Wolfram System".

Solución:

In[ ] :=

```
grafo = GrafoRandom[20, 50]
GrafoQ[grafo]
```

Out[ ] :=



True

La respuesta indica que el contenido de la variable **grafo** se creó en el "Wolfram System" de *Mathematica*.

Ejemplo 7.14

Determine el valor de verdad arrojado por el comando **GrafoQ** al crear un grafo mediante **DodecahedralGraph** (instrucción del paquete "Combinatorica").

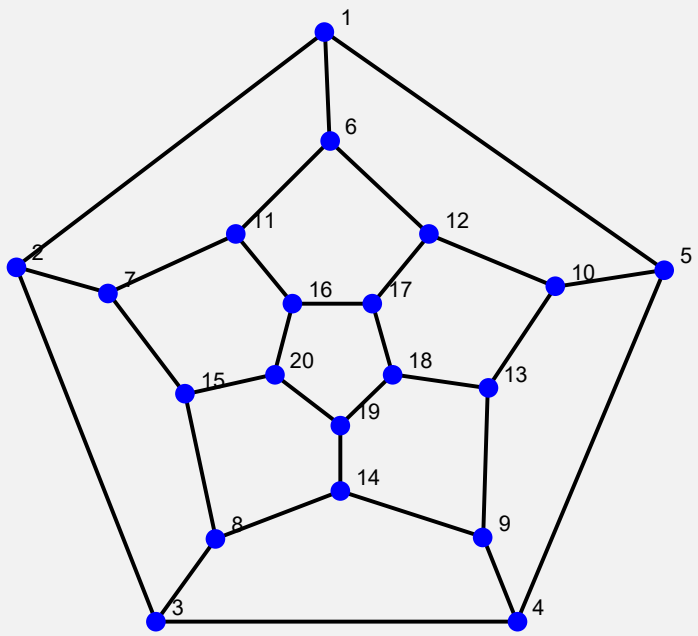
Solución:

En "Combinatorica" la creación de un grafo a través de **DodecahedralGraph** se realiza mediante el uso de **SetGraphOptions** y **ShowGraph**, tal y como se muestra a continuación:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
GrafoQ[grafo]
```

Out[ ] :=



False

Explicación en video

8) **GrafoCQ**: retorna "`True`" si el argumento recibido es un **grafo creado** con el **paquete** "Combinatorica", o bien, "`False`" en caso contrario. Es decir, devuelve "`False`" si el parámetro **no es un grafo**, o se ha **construido** en el "Wolfram System".

Sintaxis: `GrafoCQ[G]`.

Ejemplo 7.15

Utilice el comando **`GrafoC`** para crear el grafo que representa la siguiente relación binaria: $aRb \Leftrightarrow a > b+2$ definida sobre $A = \{1,2,3,4,5\}$. Verifique empleando **`GrafoCQ`** que el grafo se construyó mediante el uso del paquete "Combinatorica".

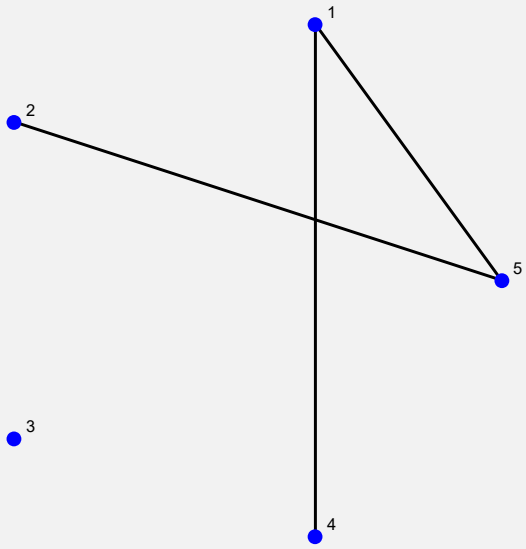
Solución:

In[ ] :=

```
A = {1, 2, 3, 4, 5};
aristas = RelBin["a>b+2", A, A]
GrafoC[aristas]
GrafoCQ[G]
```

Out[ ] :=

{{4, 1}, {5, 1}, {5, 2}}



True

Ejemplo 7.16

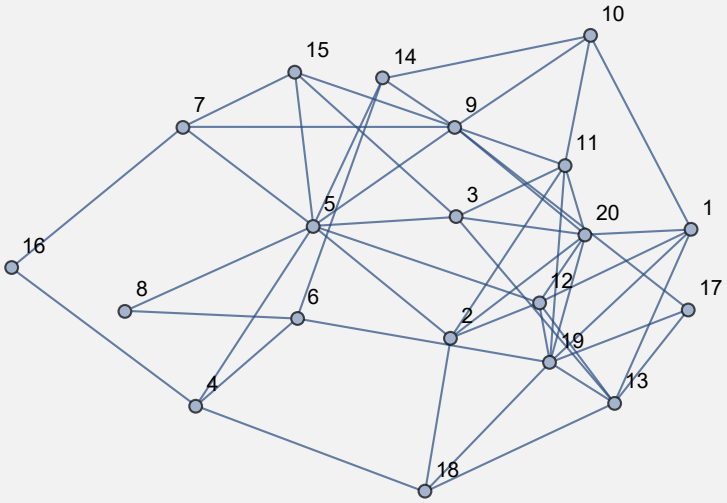
Responda mediante el uso de software: ¿**GrafoRandom[20, 50]** es un grafo creado con "Combinatorica"?

Solución:

In[ ] :=

```
grafo = GrafoRandom[20, 50]
GrafoCQ[grafo]
```

Out[ ] :=



False

Se concluye que **grafo** no está construido a través del paquete "Combinatorica".

Explicación en video

9) **PesosAristas**: **retorna** los **pesos** arista por arista de un grafo "G" **ponderado**. El parámetro recibido pudo haber sido **creado** en el "Wolfram System", o bien, a través del **paquete** "Combinatorica".

Sintaxis: `PesosAristas[G]`.

Ejemplo 7.17

Genere un grafo con pesos seudoaleatorios reales de uno a diez y retorne las ponderaciones arista por arista. El grafo representa la siguiente relación binaria: $aRb \Leftrightarrow a - b \geq 2$ con $A = \{1,2,3,\ldots,30\}$.

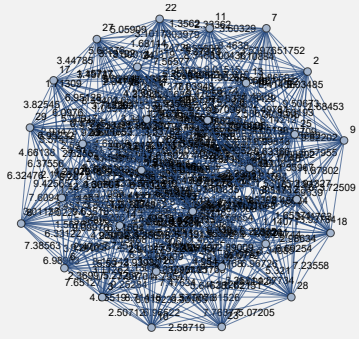
Solución:

En *Mathematica*:

In[ ] :=

```
A = Table[i, {i, 1, 30}];
aristas = RelBin["a-b>=2", A, A];
grafo = Grafo[aristas, pesos -> RandomReal[{1, 10}, Length[aristas]],
mostrarpesos -> True]
PesosAristas[grafo]
```

Out[ ] :=



{{3 ●—● 1, 9.25284}, {4 ●—● 1, 9.98185}, {4 ●—● 2, 8.56794}, ..., {30 ●—● 27, 5.93506}, {30 ●—● 28, 1.30241}}

La salida por su tamaño no se muestra en detalle. El ejemplo es seudoaleatorio, produciendo pesos distintos al ejecutar el *In*[ ] cada vez.

Ejemplo 7.18

Sea $A = \{1,2,3,4,5\}$ construya un grafo dirigido en el ambiente provisto por el paquete "Combinatorica" con pesos seudoaleatorios reales de uno a diez, siendo sus aristas el conjunto $A \times A$.

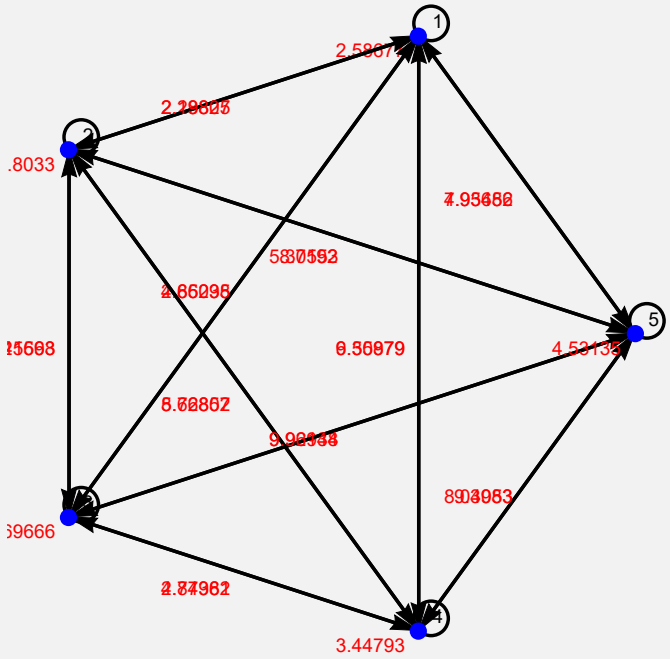
Solución:

En el software:

In[ ] :=

```
Quiet[<<Combinatorica`]
A = {1, 2, 3, 4, 5};
aristas = PC[A, A];
GrafoC[aristas, dirigido -> True, pesos -> RandomReal[{1, 10},
Length[aristas]], mostrarpesos -> True]
Edges[G]
PesosAristas[G]
```

Out[ ] :=



{{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {2, 1}, {2, 2}, {2, 3}, {2, 4}, {2, 5}, {3, 1}, {3, 2}, {3, 3}, {3, 4}, {3, 5}, {4, 1}, {4, 2}, {4, 3}, {4, 4}, {4, 5}, {5, 1}, {5, 2}, {5, 3}, {5, 4}, {5, 5}}

{{{1, 1}, 2.58677}, {{1, 2}, 2.28827}, {{1, 3}, 4.66238}, {{1, 4}, 6.35879}, {{1, 5}, 4.93452}, {{2, 1}, 2.19805}, {{2, 2}, 5.8033}, {{2, 3}, 7.25693}, {{2, 4}, 8.62857}, {{2, 5}, 8.7192}, {{3, 1}, 2.85095}, {{3, 2}, 3.41568}, {{3, 3}, 8.69666}, {{3, 4}, 2.77382}, {{3, 5}, 9.92938}, {{4, 1}, 9.50979}, {{4, 2}, 5.76802}, {{4, 3}, 4.84961}, {{4, 4}, 3.44793}, {{4, 5}, 9.3983}, {{5, 1}, 7.95686}, {{5, 2}, 5.30553}, {{5, 3}, 9.96144}, {{5, 4}, 8.04053}, {{5, 5}, 4.53135}}

Se recuerda al lector el funcionamiento del comando **`GrafoC`** que almacena por defecto el grafo en una variable denominada **`G`**, razón por la cuál al invocar **`PesosAristas`** se hace sobre **`G`**. Además, se aclara que el comando **`Edges`** es propio de "Combinatorica" y devuelve todos los lados de un grafo (es similar a **`EdgeList`** en el "Wolfram System").

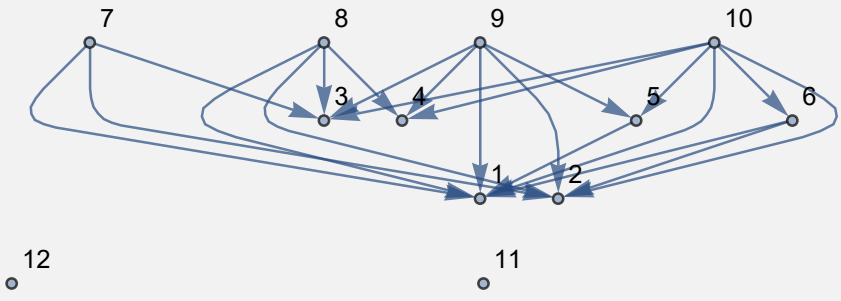
Explicación en video

10) **GraphToCombinatorica**: **convierte** un **grafo** "G" **construido** en el **ambiente** provisto por "Wolfram System" de *Mathematica*, a otro **equivalente** a través del uso del **paquete** "Combinatorica". La instrucción presenta la **opción** "**`mostrarpesos -> True`**" que **muestra** un **grafo ponderado** con los **pesos** sobre cada uno de sus **lados**.

Sintaxis: **`GraphToCombinatorica[G]`**, o bien, **`GraphToCombinatorica[G, mostrarpesos -> True]`**. Por la forma en cómo funciona el paquete "Combinatorica", los vértices del grafo "G" deben ser **números naturales consecutivos**, **iniciando** en **uno**. En caso contrario, el comando **no corre**. Por defecto, el nuevo grafo se almacena en una **variable denominada** "G".

Ejemplo 7.19

Convierta el grafo dado a continuación, al ambiente del paquete "Combinatorica".



Solución:

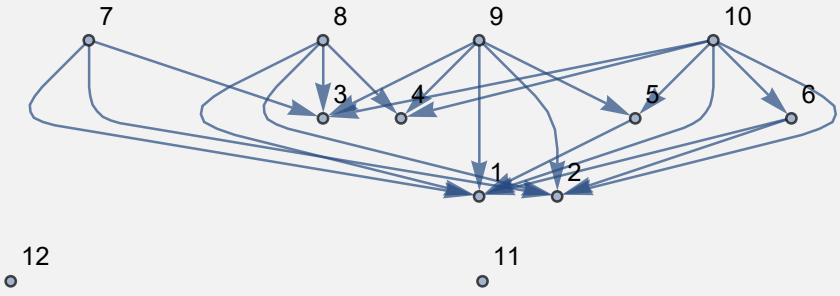
En el software:

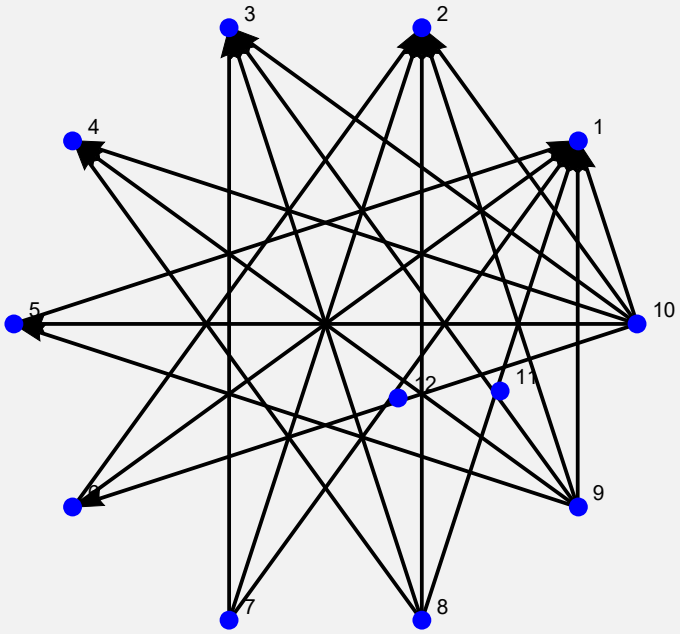
In[ ] :=

```
aristas = {{5, 1}, {6, 1}, {6, 2}, {7, 1}, {7, 2}, {7, 3}, {8, 1},
{8, 2}, {8, 3}, {8, 4}, {9, 1}, {9, 2}, {9, 3}, {9, 4}, {9, 5}, {10,
1}, {10, 2}, {10, 3}, {10, 4}, {10, 5}, {10, 6}};
grafo = Grafo[aristas, dirigido -> True, vertices -> {11, 12}]
```

```
GraphToCombinatorica[grafo]
```

Out[ ] :=





Ejemplo 7.20

Convierta un grafo 3D dirigido, generado por las aristas devueltas en **GrafoRandom[10, 10]**, con pesos seudoaleatorios enteros de uno a diez, a otro creado a través de "Combinatorica".

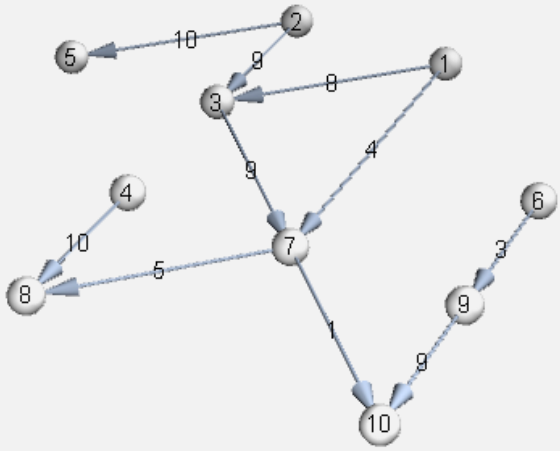
Solución:

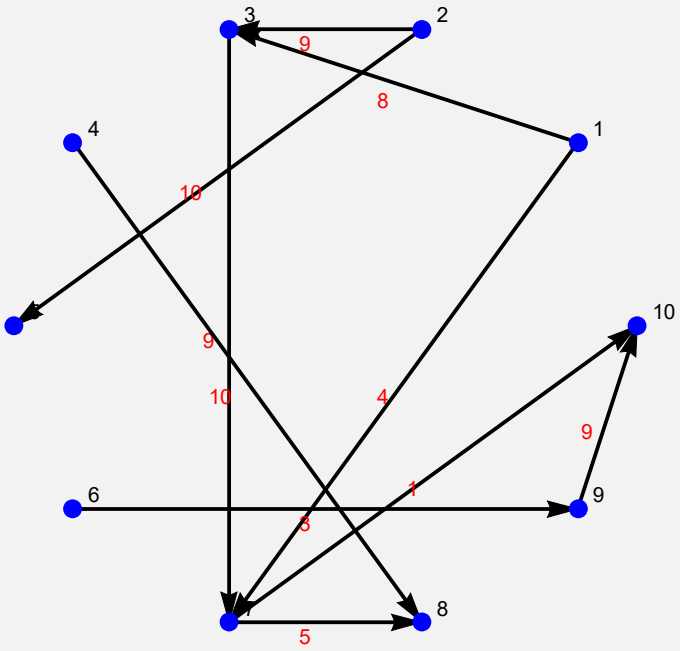
In[ ] :=

```
grafo = GrafoRandom[10, 10];
aristas = AristasWolframSystemToCombinatorica[EdgeList[grafo]]
grafo = Grafo[aristas, dirigido -> True, vertices -> VertexList[grafo],
dimensions3d -> True, pesos -> RandomInteger[{1, 10}, Length[aristas]],
mostrarpesos -> True, shape -> True]
GraphToCombinatorica[grafo, mostrarpesos -> True]
```

Out[ ] :=

{{1, 3}, {1, 7}, {2, 3}, {2, 5}, {3, 7}, {4, 8}, {6, 9}, {7, 8}, {7, 10}, {9, 10}}





N La opción "**shape -> True**" se utilizó de forma complementaria en este ejemplo, para colocar los nodos del grafo en esferas.

Explicación en video

11) **CombinatoricaToGraph**: **convierte** un **grafo** "G" **creado** a través del uso del **paquete** "Combinatorica" (**sin aristas mixtas**: dirigidas y no dirigidas) a otro **equivalente** en el **ambiente** provisto por "Wolfram System" de *Mathematica*. La instrucción presenta **tres opciones**: "**dimensions3d->True**", "**mostrarpesos->True**" y "**shape->True**". "dimensions3d" **muestra** el grafo en **tercera dimensión**, "mostrarpesos" genera un **grafo** con los **pesos** sobre cada una de sus **aristas** y "shape" coloca los vértices en **discos**, o bien, **esferas** dependiendo de la dimensión de graficación.

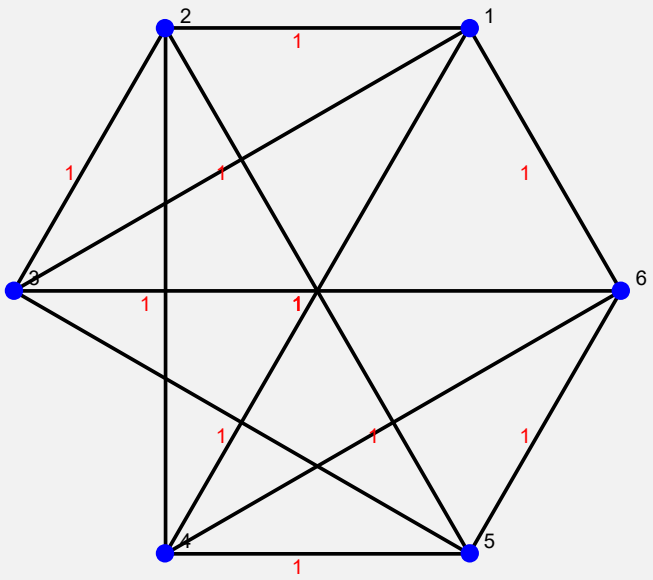
Sintaxis: `CombinatoricaToGraph[G]`, o,

```
CombinatoricaToGraph[G, dimensions3d->True, mostrarpesos->True,
shape->True]
```

En esta última invocación, es posible **prescindir** de cualquiera de las opciones.

Ejemplo 7.21

Convierta el grafo dado al "Wolfram System" de *Mathematica*.



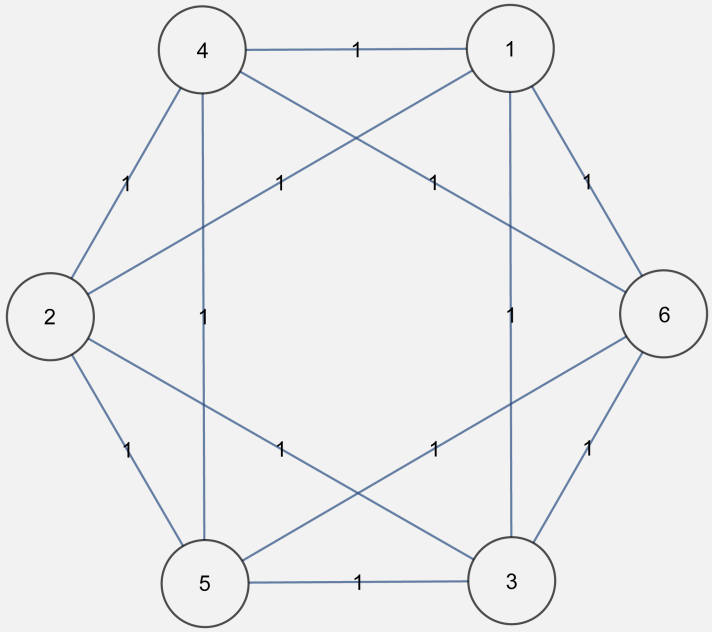
Solución:

Al usar la instrucción **CombinatoricaToGraph**:

In[ ] :=

```
aristas = {{1, 2}, {1, 3}, {2, 3}, {4, 5}, {4, 6}, {5, 6}, {1, 4},
{1, 6}, {2, 4}, {2, 5}, {3, 5}, {3, 6}};
GrafoC[aristas, mostrarpesos -> True];
CombinatoricaToGraph[G, mostrarpesos -> True, shape -> True]
```

Out[ ] :=



N En este ejercicio, también se recurrió a la opción "**shape -> True**" para colocar los nodos del grafo en discos (al no ser 3D).

Ejemplo 7.22

Considere un grafo con: aristas = {{18, 20}, {17, 20}, {15, 20}, {13, 20}, {12, 20}, {7, 20}, {6, 20}, {5, 20}, {3, 20}, {16, 19}, {11, 19}, {10, 19}, {9, 19}, {18, 19}, {17, 19}, {13, 19}, {7, 19}, {5, 19}, {4, 14}, {2, 14}, {1, 14}, {14, 16}, {9, 14}, {3, 14}, {14, 18}, {14, 17}, {5, 14}, {8, 15}, {8, 12}, {8, 11}, {6, 8}, {4, 8}, {8, 16}, {3, 8}, {8, 17}, {5, 8}, {2, 10}, {1, 10}, {10, 15}, {10, 11}, {7, 10}, {4, 10}, {3, 10}, {10, 17}, {9, 13}, {6, 13}, {2, 13}, {1, 13}, {13, 16}, {7, 13}, {4, 13}, {12, 15}, {9, 12}, {2, 12}, {1, 12}, {12, 16}, {7, 12}, {3, 12}, {15, 18}, {9, 18}, {4, 18}, {2, 18}, {1, 18}, {17, 18}, {11, 15}, {9, 11}, {4, 11}, {3, 11}, {2, 11}, {11, 17}, {5, 6}, {6, 16}, {6, 15}, {4, 6}, {3, 6}, {2, 6}, {1, 7}, {7, 9}, {7, 15}, {4, 7}, {5, 17}, {5, 9}, {2, 5}, {5, 15}, {3, 16}, {16, 17}, {9, 16}, {1, 4}, {1, 3}, {1, 2}}. Por medio de **GrafoC** construya el grafo con pesos seudoaleatorios enteros de uno a diez y transfórmelo a otro, 3D, del "Wolfram System".

Solución:

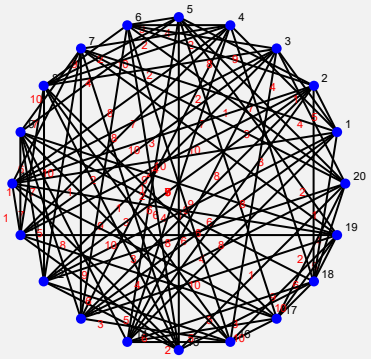
En *Mathematica*:

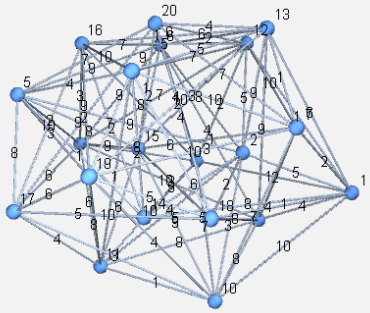
In[ ] :=

```
aristas = {{18, 20}, {17, 20}, {15, 20}, {13, 20}, {12, 20}, {7, 20},
{6, 20}, {5, 20}, {3, 20}, {16, 19}, {11, 19}, {10, 19}, {9, 19},
{18, 19}, {17, 19}, {13, 19}, {7, 19}, {5, 19}, {4, 14}, {2, 14},
```

```
{1, 14}, {14, 16}, {9, 14}, {3, 14}, {14, 18}, {14, 17}, {5, 14}, {8,
15}, {8, 12}, {8, 11}, {6, 8}, {4, 8}, {8, 16}, {3, 8}, {8, 17}, {5,
8}, {2, 10}, {1, 10}, {10, 15}, {10, 11}, {7, 10}, {4, 10}, {3, 10},
{10, 17}, {9, 13}, {6, 13}, {2, 13}, {1, 13}, {13, 16}, {7, 13}, {4,
13}, {12, 15}, {9, 12}, {2, 12}, {1, 12}, {12, 16}, {7, 12}, {3, 12},
{15, 18}, {9, 18}, {4, 18}, {2, 18}, {1, 18}, {17, 18}, {11, 15}, {9,
11}, {4, 11}, {3, 11}, {2, 11}, {11, 17}, {5, 6}, {6, 16}, {6, 15},
{4, 6}, {3, 6}, {2, 6}, {1, 7}, {7, 9}, {7, 15}, {4, 7}, {5, 17}, {5,
9}, {2, 5}, {5, 15}, {3, 16}, {16, 17}, {9, 16}, {1, 4}, {1, 3}, {1,
2}};
GrafoC[aristas, pesos -> RandomInteger[{1, 10}, Length[aristas]],
mostrarpesos -> True]
CombinatoricaToGraph[G, dimensions3d -> True, mostrarpesos -> True]
```

Out[ ] :=





Explicación en video

12) **GeneraRutas**: **retorna** una **cantidad máxima** de "n" **rutas** de un nodo "a" a un nodo "b" sobre un grafo "G", en caso de lograr encontrar los caminos. El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, a través del uso del **paquete** "Combinatorica" (**sin aristas mixtas**: dirigidas y no dirigidas).

Sintaxis: `GeneraRutas[G, a, b, n]`.

Ejemplo 7.23

Obtenga tres rutas distintas sobre el grafo dodecaedro del vértice 1 al nodo 2.

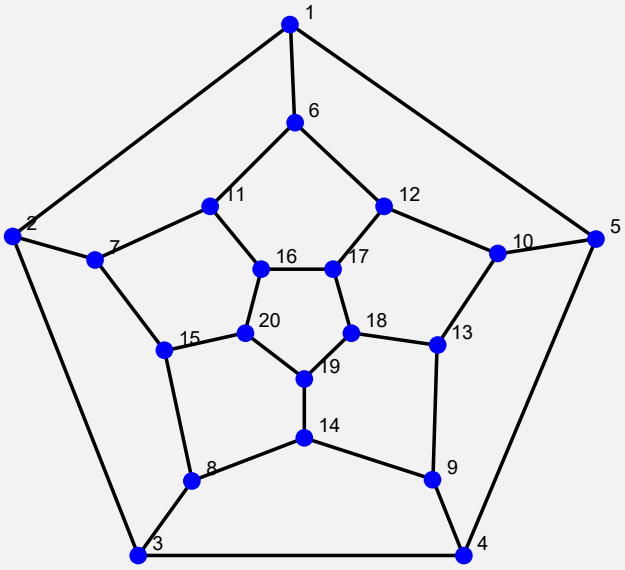
Solución:

El grafo dodecaedro se puede construir mediante el uso del comando **`DodecahedralGraph`**, luego:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
GeneraRutas[grafo, 1, 2, 3]
```

Out[ ] :=



{{{1, 2}}, {{1, 6}, {6, 11}, {11, 7}, {7, 2}}, {{1, 5}, {5, 4}, {4, 3}, {3, 2}}}

Las tres rutas devueltas son:

- {{1, 2}}
- {{1, 6}, {6, 11}, {11, 7}, {7, 2}}
- {{1, 5}, {5, 4}, {4, 3}, {3, 2}}

Ejemplo 7.24

Encuentre cinco rutas sobre un grafo obtenido al ejecutar **`GrafoRandom[20, 50]`**, del vértice 2 al nodo 4.

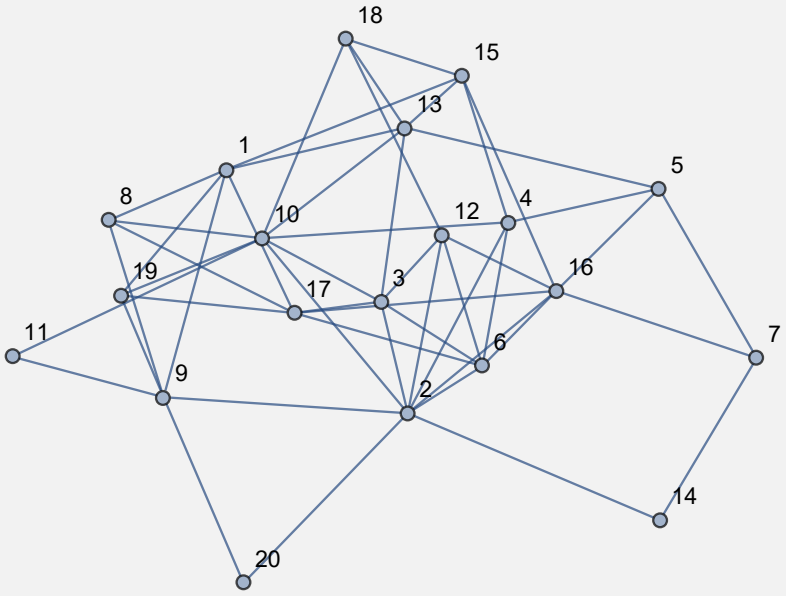
Solución:

En *Mathematica*:

In[ ] :=

```
grafo = GrafoRandom[20, 50]
GeneraRutas[grafo, 2, 4, 5]
```

Out[ ] :=



{{{2, 4}}, {{2, 10}, {10, 4}}, {{2, 6}, {6, 4}}, {{2, 16}, {16, 15}, {15, 4}}, {{2, 14}, {14, 7}, {7, 5}, {5, 4}}}

Las cinco trayectorias corresponden a:

- {{2, 4}}
- {{2, 10}, {10, 4}}
- {{2, 6}, {6, 4}}
- {{2, 16}, {16, 15}, {15, 4}}
- {{2, 14}, {14, 7}, {7, 5}, {5, 4}}

Explicación en video

13) **GeneraRutasGraphSimple**: **retorna** una **cantidad máxima** de "n" **rutas** de un nodo "a" a un nodo "b" sobre un grafo "G" **simple** (**sin aristas múltiples**, **ni lazos**), en caso de lograr encontrar los caminos. El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, a través del uso del **paquete** "Combinatorica" (**sin aristas mixtas**: dirigidas y no dirigidas).

Sintaxis: `GeneraRutasGraphSimple[G, a, b, n]`.

Ejemplo 7.25

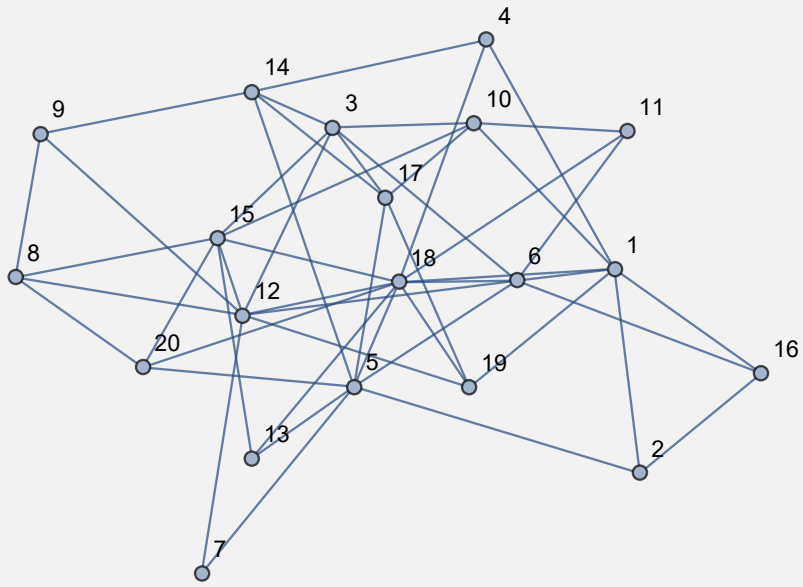
A través del uso del comando **GeneraRutasGraphSimple**, determine veinte trayectorias del vértice 2 al 4, construyendo un grafo devuelto por **GrafoRandom[20, 50]**.

Solución:

In[ ] :=

```
grafo = GrafoRandom[20, 50]
GeneraRutasGraphSimple[grafo, 2, 4, 20]
```

Out[ ] :=



{{{2, 1}, {1, 4}}, {{2, 1}, {1, 6}, {6, 3}, {3, 10}, {10, 11}, {11, 18}, {18, 4}}, {{2, 1}, {1, 6}, {6, 3}, {3, 10}, {10, 11}, {11, 18}, {18, 5}, {5, 14}, {14, 4}}, ..., {{2, 1}, {1, 6}, {6, 3}, {3, 10}, {10, 11}, {11, 18}, {18, 5}, {5, 13}, {13, 15}, {15, 8}, {8, 9}, {9, 12}, {12, 19}, {19, 17}, {17, 14}, {14, 4}}, {{2, 1}, {1, 6}, {6, 3}, {3, 10}, {10, 11}, {11, 18}, {18, 5}, {5, 13}, {13, 15}, {15, 20}, {20, 8}, {8, 9}, {9, 12}, {12, 19}, {19, 17}, {17, 14}, {14, 4}}}

El *Out* [] posee un tamaño considerable, por lo que se omite la salida completa.

Ejemplo 7.26

Utilizando los comandos del paquete "Combinatorica": **LineGraph[CompleteGraph[5]]**, **CirculantGraph[21, RandomKSubset[Range[10], 3]]** y **DeBruijnGraph[2, 5]**, determine si es posible cinco rutas en cada caso, del nodo 1 al 7.

Solución:

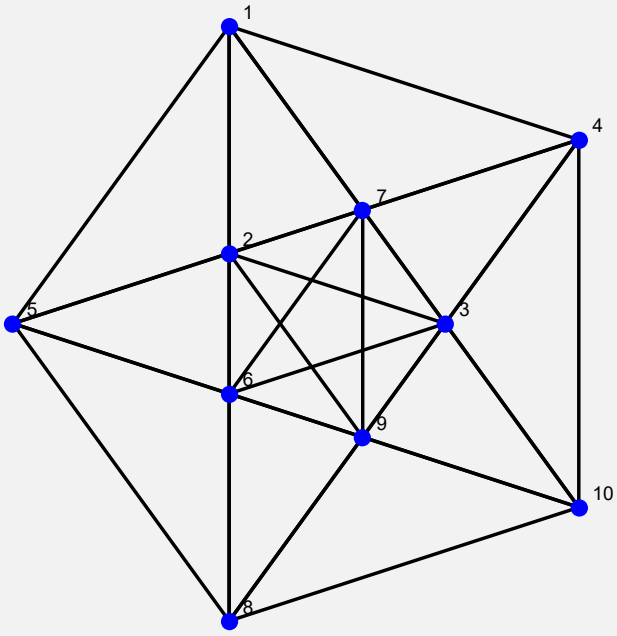
En el software:

In[ ] :=

```
Quiet[<<Combinatorica`]
(* Grafo G1 *)
ShowGraph[G1 = SetGraphOptions[LineGraph[CompleteGraph[5]],
```

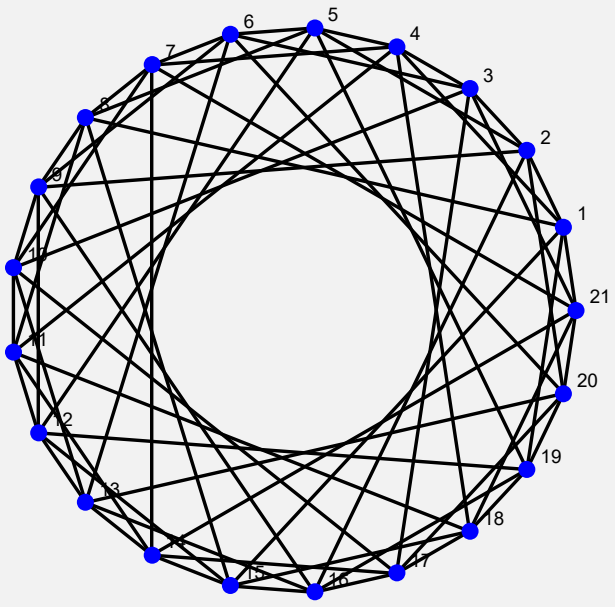
```
VertexColor -> Blue, EdgeColor -> Black], VertexLabel -> True,
PlotRange -> 0.1]
Length[GeneraRutasGraphSimple[G1, 1, 7, 5]]
GeneraRutasGraphSimple[G1, 1, 7, 5]
(* Grafo G2 *)
ShowGraph[G2 = SetGraphOptions[CirculantGraph[21,
RandomKSubset[Range[10], 3]], VertexColor -> Blue, EdgeColor -> Black],
VertexLabel -> True, PlotRange -> 0.1]
Length[GeneraRutasGraphSimple[G2, 1, 7, 5]]
GeneraRutasGraphSimple[G2, 1, 7, 5]
(* Grafo G3 *)
ShowGraph[G3 = SetGraphOptions[DeBruijnGraph[2, 5],
VertexColor -> Blue, EdgeColor -> Black], VertexLabel -> True,
PlotRange -> 0.1]
Length[GeneraRutasGraphSimple[G3, 1, 7, 5]]
GeneraRutasGraphSimple[G3, 1, 7, 5]
```

Out[ ] :=



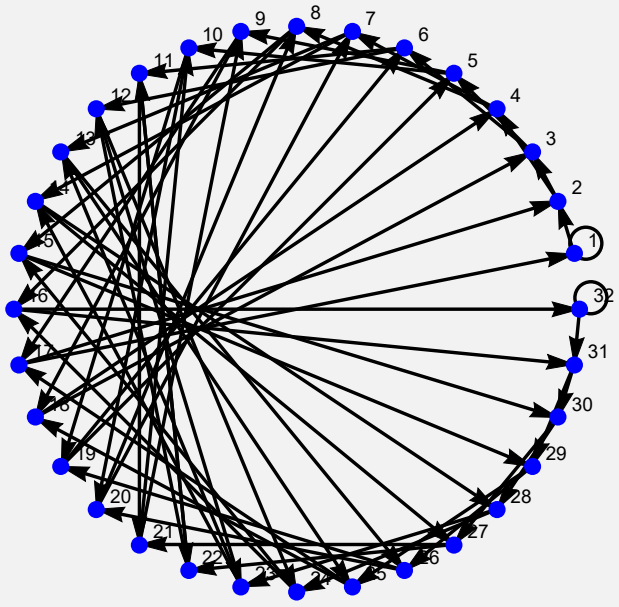
5

{{{1, 2}, {2, 3}, {3, 4}, {4, 7}}, {{1, 2}, {2, 3}, {3, 4}, {4, 9}, {9, 5}, {5, 7}}, {{1, 2}, {2, 3}, {3, 4}, {4, 9}, {9, 5}, {5, 6}, {6, 7}}, {{1, 2}, {2, 3}, {3, 4}, {4, 9}, {9, 5}, {5, 6}, {6, 10}, {10, 7}}, {{1, 2}, {2, 3}, {3, 4}, {4, 9}, {9, 5}, {5, 6}, {6, 8}, {8, 10}, {10, 7}}}



5

{{{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6}, {6, 7}}, {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6}, {6, 9}, {9, 8}, {8, 7}}, {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6}, {6, 9}, {9, 8}, {8, 11}, {11, 10}, {10, 7}}, {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6}, {6, 9}, {9, 8}, {8, 11}, {11, 10}, {10, 13}, {13, 12}, {12, 15}, {15, 14}, {14, 7}}, {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6}, {6, 9}, {9, 8}, {8, 11}, {11, 10}, {10, 13}, {13, 12}, {12, 15}, {15, 14}, {14, 17}, {17, 16}, {16, 19}, {19, 18}, {18, 21}, {21, 7}}}



0

N En *Mathematica* si se desea incluir comentarios en una entrada se usa **(* Comentario *)**, lo cuál se ha hecho en este ejemplo para etiquetar cada grafo. Es importante señalar también, con respecto al último grafo del *In*[], el por qué **Length[GeneraRutasGraphSimple[G3, 1, 7, 5]]** retorna 0. Lo anterior ocurre, pues el comando **GeneraRutasGraphSimple[G3, 1, 7, 5]** no corre, al ser **G3** una variable que contiene un grafo no simple.

Explicación en video

14) **AnimarGrafo**: **construye** una **animación** sobre un **grafo** "G" y una **lista** "L" de **aristas**, remarcando **paso a paso** las aristas **sobre** el **grafo** "G" en el **orden** de "L". El comando es capaz de procesar a "G", tanto si éste ha sido **creado** en el "Wolfram System" de *Mathematica*, como también, si ha sido **generado** con el **paquete** "Combinatorica" (**sin aristas mixtas**: dirigidas y no dirigidas). Brinda la **opción** "**padding -> Valor**" que **añade** un **espacio** de **contorno** especificado en "Valor" a grafos **creados** en el "Wolfram System" (por **defecto** es igual a **diez**).

Sintaxis: **AnimarGrafo[G, L]**, o bien, **AnimarGrafo[G, L, padding -> Valor]**, "L" es un conjunto de **pares ordenados**.

Ejemplo 7.27

Represente en una animación el recorrido de una ruta seleccionada de cinco trayectorias devueltas por el comando **GeneraRutasGraphSimple** del vértice 1 al 20, sobre el grafo dodecaedro.

Solución:

En este ejemplo se escogerá de forma seudoaleatoria una de las cinco rutas construidas por **GeneraRutasGraphSimple**, con la intención de utilizarla dentro del comando **AnimarGrafo**:

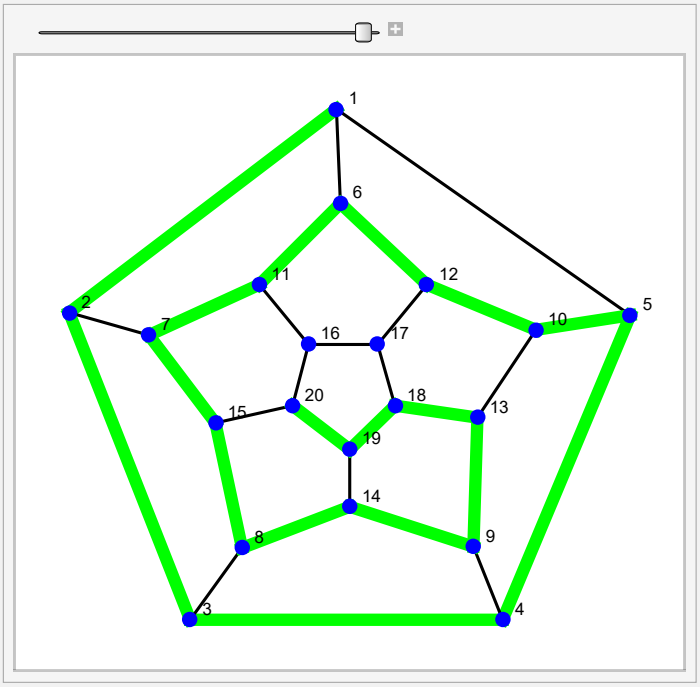
In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor -> Blue, EdgeColor -> Black], VertexLabel -> True,
PlotRange -> 0.1];
L = GeneraRutasGraphSimple[grafo, 1, 20, 5];
route = L[[RandomInteger[{1, Length[L]}]]]
AnimarGrafo[grafo, route]
```

Out[ ] :=

{{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 10}, {10, 12}, {12, 6}, {6, 11}, {11, 7}, {7, 15}, {15, 8}, {8, 14}, {14, 9}, {9, 13},

{13, 18}, {18, 19}, {19, 20}}



N El deslizador de la parte superior de la ventana permite visualizar el recorrido arista por arista.

Ejemplo 7.28

Anime una de las rutas de cinco caminos, obtenidos por la instrucción **`GeneraRutasGraphSimple`** del nodo 2 al nodo 4, en un grafo construido a través de **`GrafoRandom[20, 50]`**.

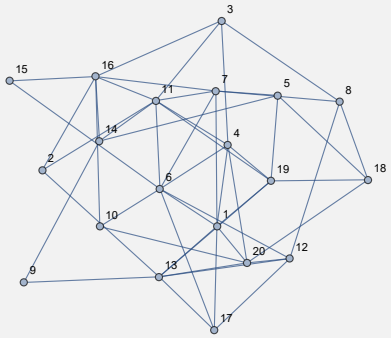
Solución:

En *Mathematica*, como la salida es seudoaleatoria, se debe primero crear el grafo:

In[ ] :=

```
grafo = GrafoRandom[20, 50]
```

Out[ ] :=



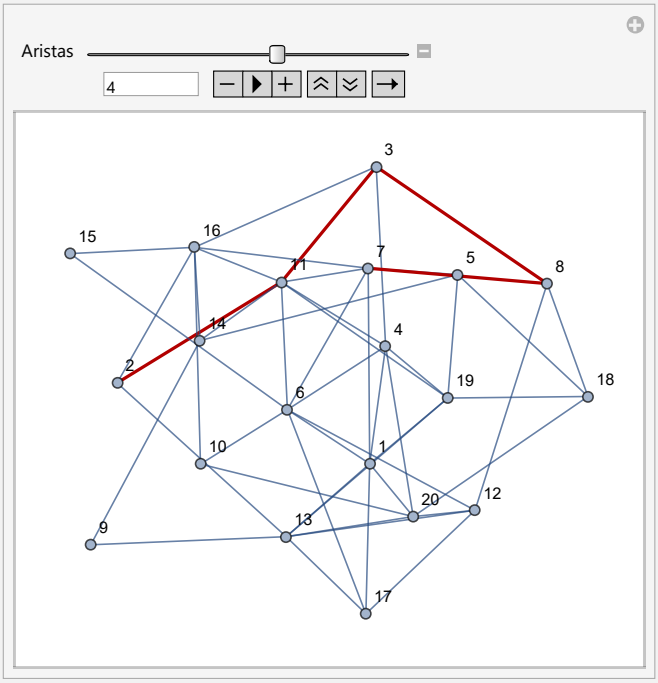
Posteriormente, al estar almacenado en la variable **grafo** se procede de la siguiente manera:

In[ ] :=

```
L = GeneraRutasGraphSimple[grafo, 2, 4, 5];
route = L[[RandomInteger[{1, Length[L]}]]]
AnimarGrafo[grafo, route]
```

Out[ ] :=

{{2, 11}, {11, 3}, {3, 8}, {8, 7}, {7, 1}, {1, 4}}



Explicación en video

15) **AnimarGrafoWithCombinatorica**: **construye** una **animación** en el **ambiente** provisto por "Combinatorica" sobre un **grafo** "G" y una **lista** "L" de **aristas**, remarcando **paso a paso** las aristas **sobre** el **grafo** "G" en el **orden** de "L". El comando es capaz de procesar a "G", tanto si éste ha sido **creado** en el "Wolfram System" de *Mathematica* (debe haber **consistencia** en el **grafo** para poder ser **convertido** a "Combinatorica"), como también, si ha sido **generado** con el **paquete** "Combinatorica" (**sin aristas mixtas**: dirigidas y no dirigidas).

Sintaxis: **`AnimarGrafoWithCombinatorica[G, L]`**, "L" es un conjunto de **pares ordenados**.

Ejemplo 7.29

Mediante el comando **AnimarGrafoWithCombinatorica** genere una animación que contenga el recorrido de una ruta seleccionada de cinco caminos devueltos por **GeneraRutasGraphSimple** del vértice 1 al 20, sobre el grafo dodecaedro.

Solución:

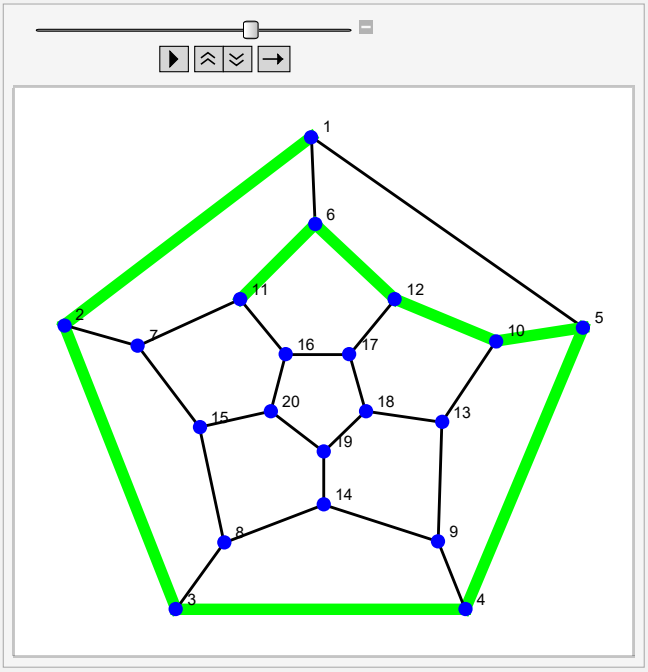
En el software:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1];
L = GeneraRutasGraphSimple[grafo, 1, 20, 5];
route = L[[RandomInteger[{1, Length[L]}]]]
AnimarGrafoWithCombinatorica[grafo, route]
```

Out[ ] :=

{{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 10}, {10, 12}, {12, 6}, {6, 11}, {11, 7}, {7, 15}, {15, 20}}



Ejemplo 7.30

Construya un grafo a través de la invocación **GrafoRandom[20, 50]**. Posteriormente, anime utilizando el comando **AnimarGrafoWithCombinatorica**, una de cinco rutas obtenidas por **GeneraRutasGraphSimple** del nodo 2 al nodo 4.

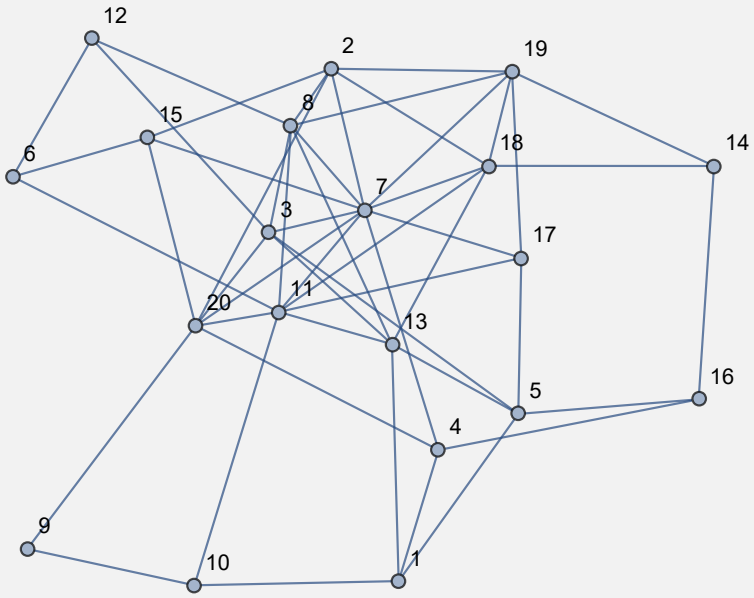
Solución:

En *Mathematica*, se crea en primera instancia el grafo:

In[ ] :=

```
grafo = GrafoRandom[20, 50]
```

Out[ ] :=



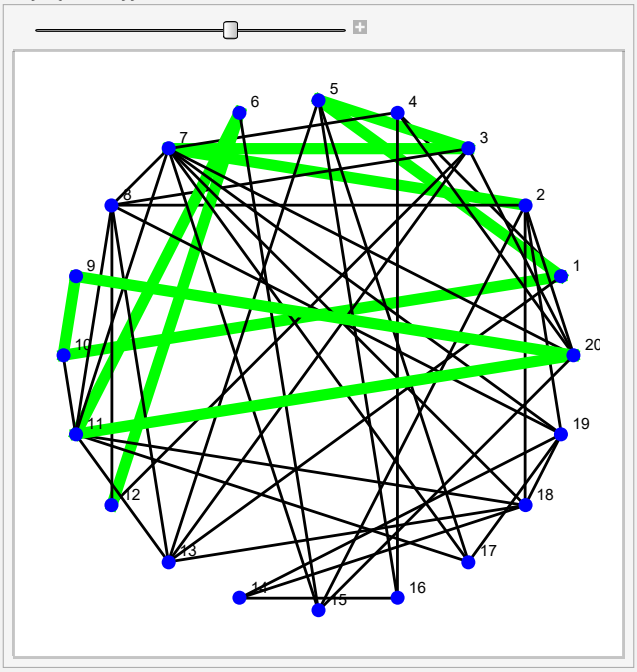
Luego:

In[ ] :=

```
L = GeneraRutasGraphSimple[grafo, 2, 4, 5];
route = L[[RandomInteger[{1, Length[L]}]]]
AnimarGrafoWithCombinatorica[grafo, route]
```

Out[ ] :=

{{2, 7}, {7, 3}, {3, 5}, {5, 1}, {1, 10}, {10, 9}, {9, 20}, {20, 11}, {11, 6}, {6, 12}, {12, 8}, {8, 19}, {19, 14}, {14, 16}, {16, 4}}



Explicación en video

16) **GrafoM**: **crea** un **grafo** por medio del "Wolfram System" generado por su **matriz de adyacencia** de acuerdo con un **conjunto** "Nodos", que representa el **orden** de los vértices del grafo, a través del cual se **construyó** la **matriz de adyacencia** "M" recibida como parámetro. La instrucción facilita **dos opciones**: "`dimensions3d->True`" y "`shape->True`". "dimensions3d" muestra el grafo en **tercera dimensión** y "shape" coloca los vértices del grafo en **discos**, o bien, **esferas** dependiendo de la dimensión de graficación.

Sintaxis: `GrafoM[M, Nodos]`, o bien, `GrafoM[M, Nodos, dimensions3d->True, shape->True]`, en esta última invocación pudiendo **prescindir** de cualquiera de las opciones.

Ejemplo 7.31

Construya dos grafos 3D cuyas matrices de adyacencia corresponden a:

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ y } \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

donde los vértices en orden, vienen dados por $\{1, a, 2, b, 3, c, 4\}$.

Solución:

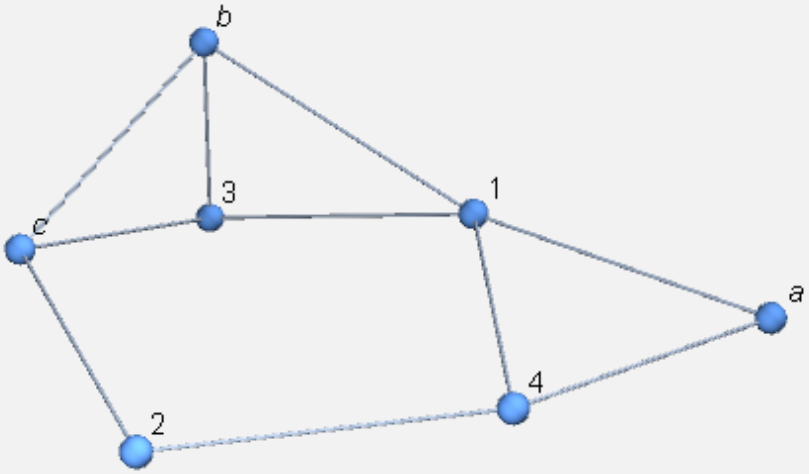
Al emplear la instrucción `GrafoM` se obtiene:

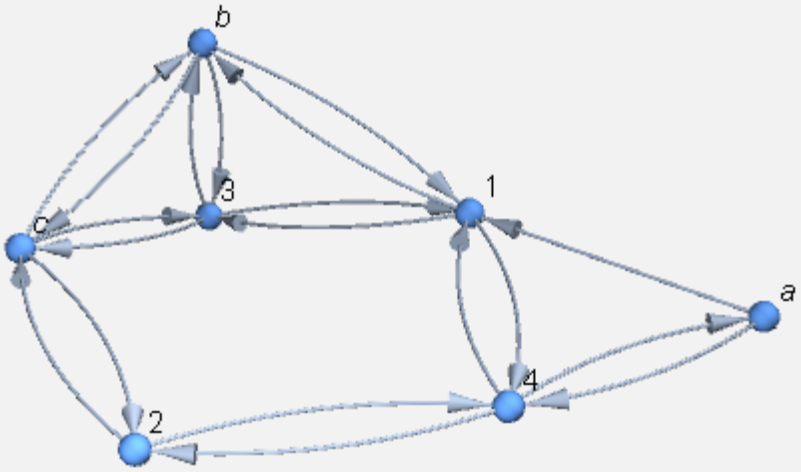
In[ ] :=

```
A = {1, a, 2, b, 3, c, 4};
Matriz = {{0, 1, 0, 1, 1, 0, 1}, {1, 0, 0, 0, 0, 0, 1}, {0, 0, 0, 0, 0, 1, 1}, {1, 0, 0, 0, 1, 1, 0}, {1, 0, 0, 1, 0, 1, 0}, {0, 0, 1, 1, 1, 0, 0}, {1, 1, 1, 0, 0, 0, 0}};
GrafoM[Matriz, A, dimensions3d->True]
```

```
Matriz = {{0, 0, 0, 1, 1, 0, 1}, {1, 0, 0, 0, 0, 0, 1}, {0, 0, 0, 0, 0, 1, 1}, {1, 0, 0, 0, 1, 1, 0}, {1, 0, 0, 1, 0, 1, 0}, {0, 0, 1, 1, 1, 0, 0}, {1, 1, 1, 0, 0, 0, 0}};
GrafoM[Matriz, A, dimensions3d -> True]
```

Out[ ] :=





N El segundo grafo es dirigido pues la matriz de adyacencia de la cuál procede no es simétrica.

Ejemplo 7.32

Genere el grafo representado por la matriz de adyacencia:

$$\begin{pmatrix} 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&1&0&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&1&0&1&0&0&0&0&0&0\\ 0&0&1&1&1&1&0&0&0&0&0&0\\ 0&0&1&1&1&1&1&0&0&0&0&0\\ 0&0&1&1&1&1&1&1&0&0&0&0\\ 0&0&1&1&1&1&1&1&1&0&0&0\\ 0&0&1&1&1&1&1&1&1&1&0&0 \end{pmatrix}$$

con respecto al conjunto de nodos $\{a,b,6,2,8,4,10,12,14,16,18,20\}$. Coloque los vértices del grafo en discos.

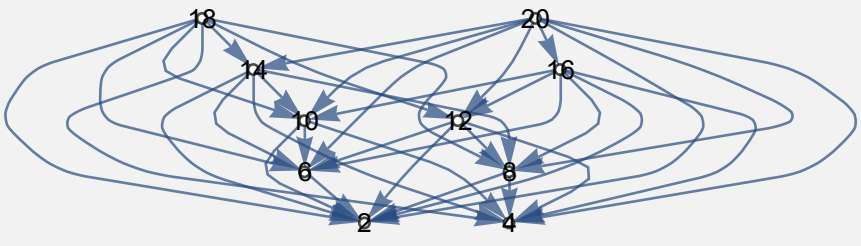
Solución:

En *Mathematica*:

In[ ] :=

```
A = {a, b, 6, 2, 8, 4, 10, 12, 14, 16, 18, 20};
Matriz = ( 0 0 0 0 0 0 0 0 0 0 0 0
           0 0 0 0 0 0 0 0 0 0 0 0
           0 0 0 1 0 0 0 0 0 0 0 0
           0 0 0 0 0 0 0 0 0 0 0 0
           0 0 0 1 0 1 0 0 0 0 0 0
           0 0 0 0 0 0 0 0 0 0 0 0
           0 0 1 1 0 1 0 0 0 0 0 0
           0 0 1 1 1 1 0 0 0 0 0 0
           0 0 1 1 1 1 1 0 0 0 0 0
           0 0 1 1 1 1 1 1 0 0 0 0
           0 0 1 1 1 1 1 1 1 0 0 0
           0 0 1 1 1 1 1 1 1 1 0 0 );
GrafoM[Matriz, A, shape -> True]
```

Out[ ] :=



Explicación en video

17) **MGrafo**: recibe un **grafo** "G" y sus **vértices** en un **conjunto** "Nodos", **retornando** la correspondiente **matriz de adyacencia** de acuerdo con el **orden** de "Nodos". La instrucción presenta la **opción** "`table -> True`", la cual permite **mostrar** la **matriz** con un **formato de tabla**, añadiendo **encabezados** en filas y columnas con los **vértices** contenidos en el **orden** del conjunto "Nodos".

Sintaxis: `MGrafo[G, Nodos]`, o, `MGrafo[G, Nodos, table -> True]`. Si el grafo se **generó** con "Combinatorica" no acepta **aristas mixtas** (dirigidas y no dirigidas).

Ejemplo 7.33

Encuentre la matriz de adyacencia del grafo dodecaedro considerando el orden en sus vértices {14, 4, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 2, 15, 16, 17, 18, 19, 20}. Retorne además, la matriz en formato de tabla. Resuelva este mismo problema tomando los lados del grafo como dirigidos.

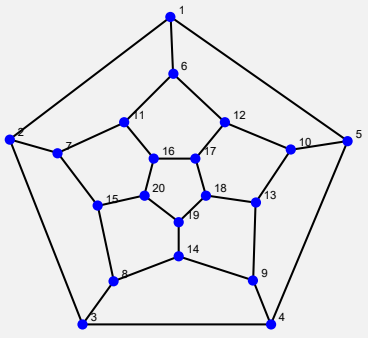
Solución:

En el software:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor -> Blue, EdgeColor -> Black], VertexLabel -> True,
PlotRange -> 0.1]
MGrafo[grafo, {14, 4, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 2, 15, 16,
17, 18, 19, 20}]
MGrafo[grafo, {14, 4, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 2, 15, 16,
17, 18, 19, 20}, table -> True]
```

Out[ ] :=



$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0
\end{pmatrix}$$

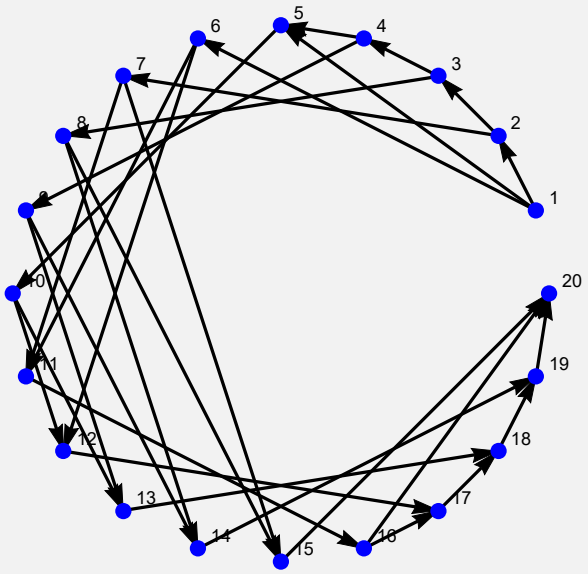
	14	4	1	3	5	6	7	8	9	10	11	12	13	2	15	16	17	18	19	20
14	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0
4	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
3	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0
5	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0
8	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
9	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
10	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
11	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0
12	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0
13	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0
2	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1
16	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1
17	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0
19	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0

El comando **FromOrderedPairs** del paquete "Combinatorica" muestra un grafo con aristas dirigidas, de donde para resolver la segunda parte del ejercicio se tiene:

In[ ] :=

```
ShowGraph[grafo = SetGraphOptions[FromOrderedPairs[Edges[Dodecahedral-
Graph]], VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
MGrafo[grafo, {14, 4, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 2, 15, 16,
17, 18, 19, 20}]
MGrafo[grafo, {14, 4, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 2, 15, 16,
17, 18, 19, 20}, table->True]
```

Out[ ] :=



$$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0
\end{pmatrix}$$

	14	4	1	3	5	6	7	8	9	10	11	12	13	2	15	16	17	18	19	20
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
4	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
3	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
2	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ejemplo 7.34

Construya un grafo al invocar **`GrafoRandom[10, 10]`**. Retorne una matriz de adyacencia (como matriz y en formato de tabla) considerando alguna de las permutaciones obtenidas al ordenar el conjunto de vértices devuelto por **`VertexList`**. Resuelva el mismo problema tomando las aristas de este grafo y generándo otro, dirigido. Finalmente, duplique las aristas de **`GrafoRandom[10, 10]`** y encuentre su matriz de adyacencia.

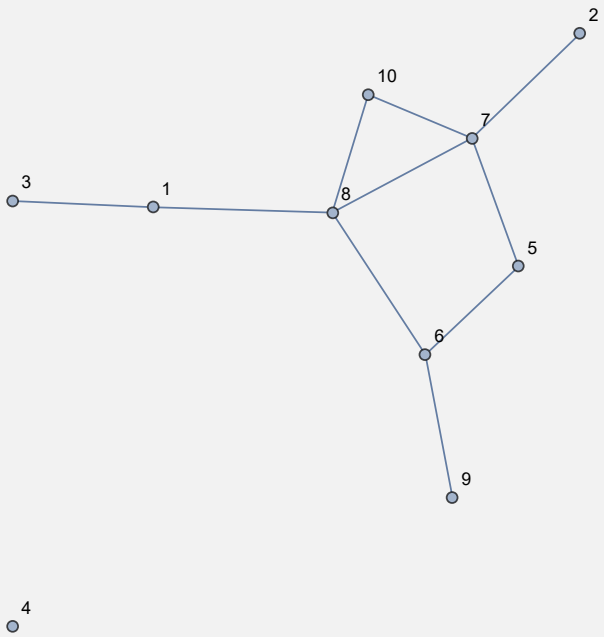
Solución:

La primera parte de este ejercicio se resolvería así:

In[ ] :=

```
grafo = GrafoRandom[10, 10]
nodos = RandomChoice[Permutations[VertexList[grafo]]]
MGrafo[grafo, nodos]
MGrafo[grafo, nodos, table->True]
```

Out[ ] :=



{5, 7, 10, 1, 6, 4, 3, 9, 2, 8}

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	5	7	10	1	6	4	3	9	2	8
5	0	1	0	0	1	0	0	0	0	0
7	1	0	1	0	0	0	0	0	1	1
10	0	1	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	0	0	1
6	1	0	0	0	0	0	0	1	0	1
4	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
9	0	0	0	0	1	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0
8	0	1	1	1	1	0	0	0	0	0

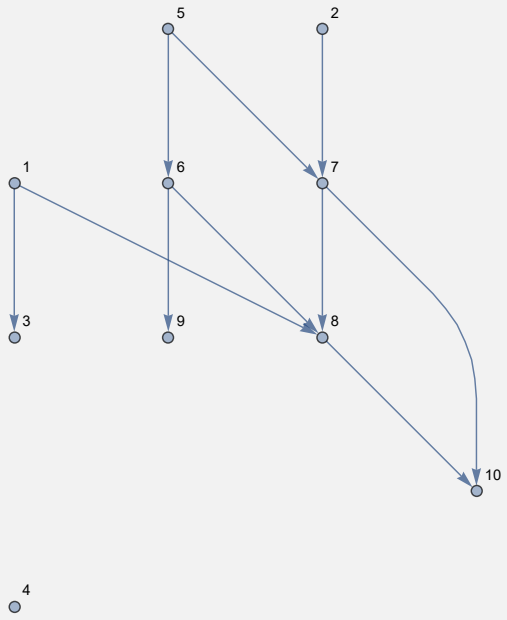
N **RandomChoice** es una instrucción de *Mathematica* que selecciona de una lista un elemento de forma seudoaleatoria.

Para la segunda parte, es posible proceder como se detalla:

In[ ] :=

```
grafo = Grafo[AristasWolframSystemToCombinatorica[EdgeList[grafo]],
dirigido -> True, vertices -> VertexList[grafo]]
MGrafo[grafo, nodos]
MGrafo[grafo, nodos, table -> True]
```

Out[ ] :=



$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

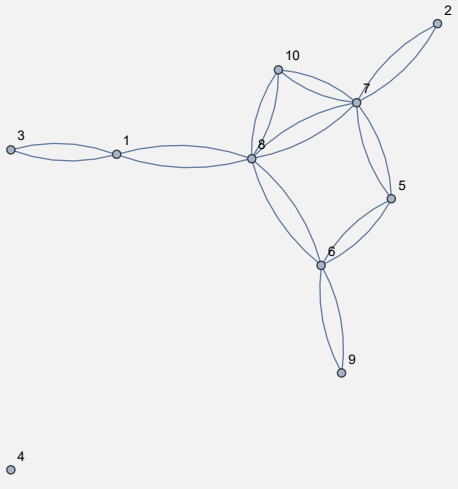
	5	7	10	1	6	4	3	9	2	8
5	0	1	0	0	1	0	0	0	0	0
7	0	0	1	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0	1
6	0	0	0	0	0	0	0	1	0	1
4	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0
8	0	0	1	0	0	0	0	0	0	0

Con relación a la tercera parte, se tendría:

In[ ] :=

```
grafo = Grafo[Join[AristasWolframSystemToCombinatorica[EdgeList[grafo]],
AristasWolframSystemToCombinatorica[EdgeList[grafo]]],
vertices -> VertexList[grafo]]
MGrafo[grafo, nodos]
MGrafo[grafo, nodos, table -> True]
```

Out[ ] :=



$$\begin{pmatrix} 0 & 2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 2 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	5	7	10	1	6	4	3	9	2	8
5	0	2	0	0	2	0	0	0	0	0
7	2	0	2	0	0	0	0	0	2	2
10	0	2	0	0	0	0	0	0	0	2
1	0	0	0	0	0	0	2	0	0	2
6	2	0	0	0	0	0	0	2	0	2
4	0	0	0	0	0	0	0	0	0	0
3	0	0	0	2	0	0	0	0	0	0
9	0	0	0	0	2	0	0	0	0	0
2	0	2	0	0	0	0	0	0	0	0
8	0	2	2	2	2	0	0	0	0	0

N **`Join`** ejecuta la operación de unión entre la lista retornada por **`AristasWolframSystemToCombinatorica[EdgeList[grafo]]`** con ella misma, duplicando el conjunto de lados del grafo.

Explicación en video

18) **GrafoRandom**: **construye** si es posible, un **grafo seudoaleatorio no dirigido** con "n" **vértices** y un **mínimo** de "m" **aristas**. Por defecto, el comando lo hace en el **ambiente** provisto por "Wolfram System" de *Mathematica*, sin embargo, brinda la **opción** "**`combinatorica->True`**" creando el grafo por medio del **paquete** "Combinatorica" (en este caso, el grafo queda almacenado en una **variable denominada** "G"). Además, la instrucción integra la **propiedad** "**`simple->Valor`**" donde "Valor" toma un contenido **booleano**, "**`True`**" genera un **grafo simple** (sin lazos y lados múltiples) y "**`False`**", realiza lo **contrario**.

Sintaxis: **`GrafoRandom[n, m]`**, o bien, **`GrafoRandom[n, m, combinatorica->True, simple->Valor]`**, pudiendo **prescindir** de cualquiera de sus opciones. Inicialmente "**`simple->True`**".

Ejemplo 7.35

Genere un grafo seudoaleatorio con 5 nodos y 6 aristas. Construya además, otro grafo seudoaleatorio no simple con los mismos valores.

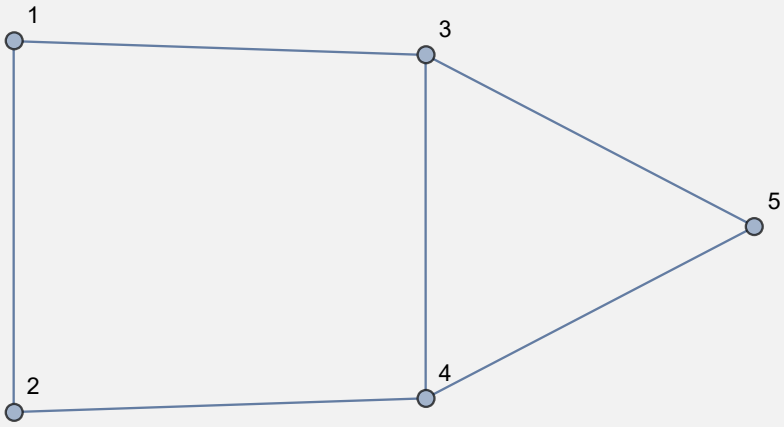
Solución:

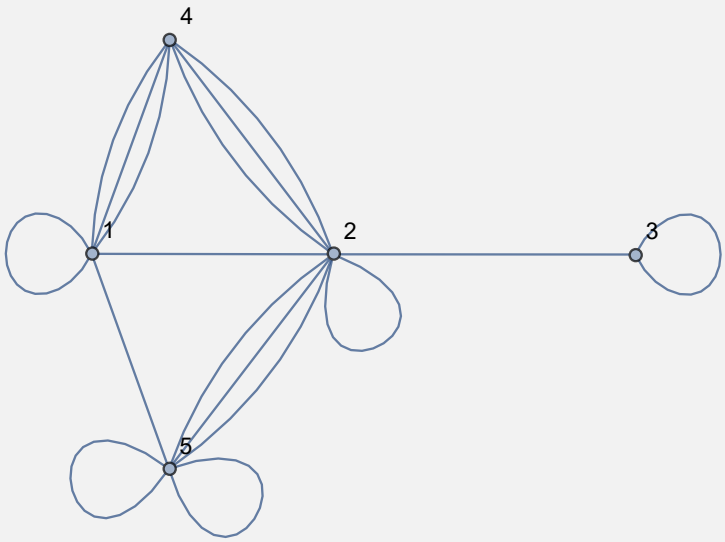
En *Mathematica*:

In[ ] :=

```
GrafoRandom[5, 6]
GrafoRandom[5, 6, simple->False]
```

Out[ ] :=





Ejemplo 7.36

Empleando la instrucción **GrafoRandom** construya grafos seudoaletorios en "Combinatorica", no simples con 10 vértices y un mínimo de 9 aristas y, 10 nodos y un mínimo de 20 lados.

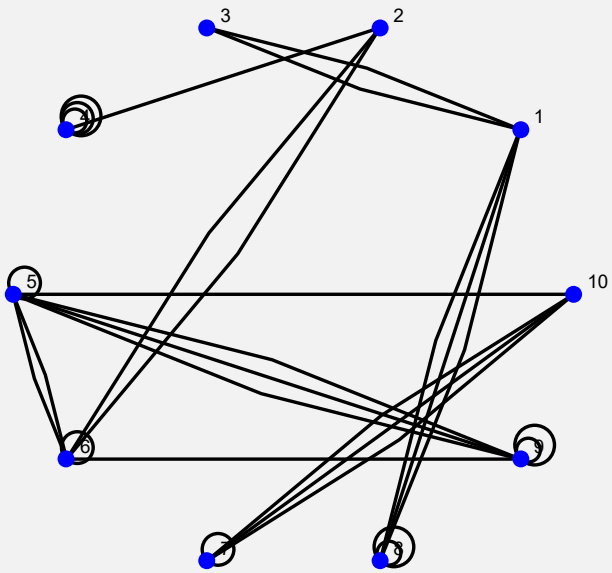
Solución:

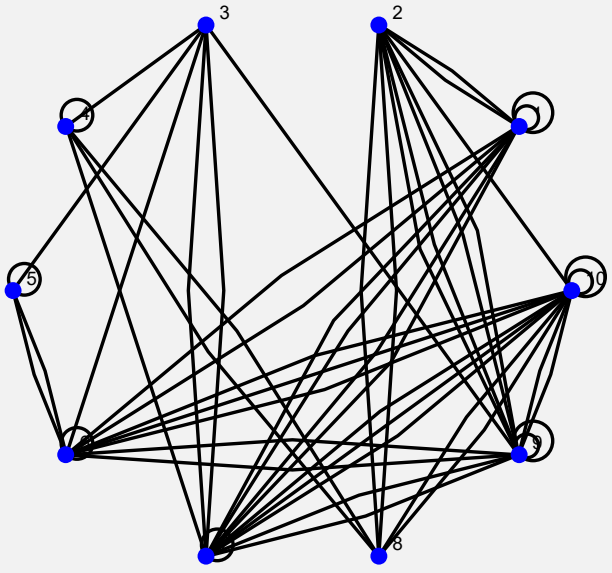
En el software:

In[ ] :=

```
GrafoRandom[10, 9, combinatorica -> True, simple -> False]
GrafoRandom[10, 20, combinatorica -> True, simple -> False]
```

Out[ ] :=





Explicación en video

19) **MGrafos**: **genera** las **matrices de adyacencia** de "k" **grafos** (simples o no) **seudoaleatorios no dirigidos** con "n" **vértices** y "n-1" **aristas**. El objetivo de este comando es **deducir** de los ejemplos arrojados por el software, las **propiedades** de una matriz de adyacencia de un **grafo no dirigido**.

Sintaxis: `MGrafos[k, n]`, o bien, `MGrafos[k, n, simple->False]` si se desea que los grafos **no sean simples**. Por defecto, la matriz se muestra en cada caso con un **formato de tabla**.

Ejemplo 7.37

Muestre la matriz de adyacencia de cinco grafos seudoaleatorios con 7 vértices y 6 aristas.

Solución:

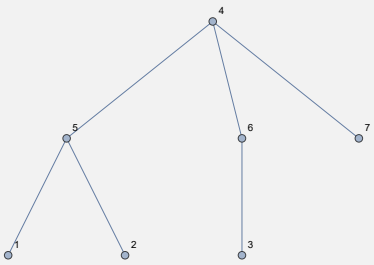
En *Mathematica*:

In[ ] :=

```
MGrafos[5, 7]
```

Out[ ] :=

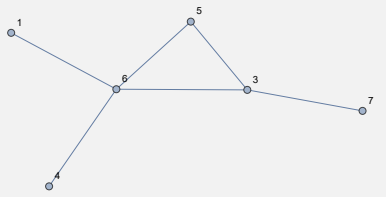
{



,

	1	2	3	4	5	6	7
1	0	0	0	0	1	0	0
2	0	0	0	0	1	0	0
3	0	0	0	0	0	1	0
4	0	0	0	0	1	1	1
5	1	1	0	1	0	0	0
6	0	0	1	1	0	0	0
7	0	0	0	1	0	0	0

,

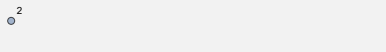


,

	1	2	3	4	5	6	7
1	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0
3	0	0	0	0	1	1	1
4	0	0	0	0	0	1	0
5	0	0	1	0	0	1	0
6	1	0	1	1	1	0	0
7	0	0	1	0	0	0	0

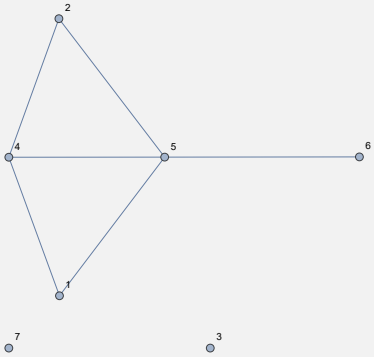
,

,



	1	2	3	4	5	6	7
1	0	0	0	0	1	1	0
2	0	0	0	1	0	1	0
3	0	0	0	0	1	0	1
4	0	1	0	0	0	0	0
5	1	0	1	0	0	0	0
6	1	1	0	0	0	0	0
7	0	0	1	0	0	0	0

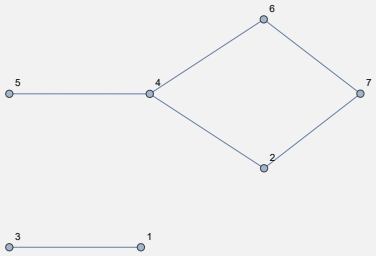
,



,

	1	2	3	4	5	6	7
1	0	0	0	1	1	0	0
2	0	0	0	1	1	0	0
3	0	0	0	0	0	0	0
4	1	1	0	0	1	0	0
5	1	1	0	1	0	1	0
6	0	0	0	0	1	0	0
7	0	0	0	0	0	0	0

,



	1	2	3	4	5	6	7
1	0	0	1	0	0	0	0
2	0	0	0	1	0	0	1
3	1	0	0	0	0	0	0
4	0	1	0	0	1	1	0
5	0	0	0	1	0	0	0
6	0	0	0	1	0	0	1
7	0	1	0	0	0	1	0

Ejemplo 7.38

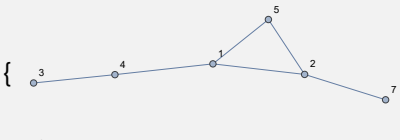
Obtenga la matriz de adyacencia de siete grafos seudoaleatorios con 7 vértices y 6 aristas.

Solución:

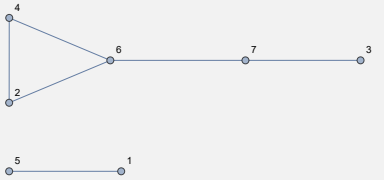
In[ ] :=

```
MGrafos[7, 7]
```

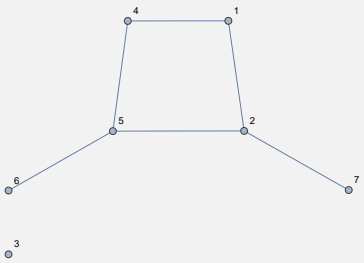
Out[ ] :=



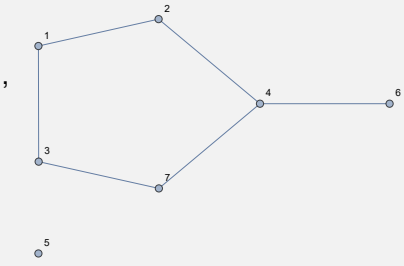
	1	2	3	4	5	6	7
1	0	1	0	1	1	0	0
2	1	0	0	0	1	0	1
3	0	0	0	1	0	0	0
4	1	0	1	0	0	0	0
5	1	1	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	1	0	0	0	0	0



	1	2	3	4	5	6	7
1	0	0	0	0	1	0	0
2	0	0	0	1	0	1	0
3	0	0	0	0	0	0	1
4	0	1	0	0	0	1	0
5	1	0	0	0	0	0	0
6	0	1	0	1	0	0	1
7	0	0	1	0	0	1	0



	1	2	3	4	5	6	7
1	0	1	0	1	0	0	0
2	1	0	0	0	1	0	1
3	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0
5	0	1	0	1	0	1	0
6	0	0	0	0	1	0	0
7	0	1	0	0	0	0	0



	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0
2	1	0	0	1	0	0	0
3	1	0	0	0	0	0	1
4	0	1	0	0	0	1	1
5	0	0	0	0	0	0	0
6	0	0	0	1	0	0	0
7	0	0	1	1	0	0	0

}

No se muestra la salida completa por sus dimensiones.

Explicación en video

20) **GrafoMP**: **construye** un **grafo** por medio del "Wolfram System" generado por su **matriz de adyacencia de pesos** de acuerdo con un **conjunto** "Nodos", que representa el **orden** de los vértices del grafo, a través del cual se **construyó** la **matriz** "M" recibida como parámetro. El comando presenta **tres opciones**: "`dimensions3d->True`", "`mostrarpesos->True`" y "`shape->True`". "dimensions3d" genera el grafo en **tercera dimensión**, "mostrarpesos" **muestra** el grafo con los **pesos** en sus aristas y "shape" coloca los vértices en **discos**, o bien, **esferas** dependiendo de la dimensión de graficación.

Sintaxis: `GrafoMP[M, Nodos]`, o bien, `GrafoMP[M, Nodos, dimensions3d->True, mostrarpesos->True, shape->True]`, pudiendo **prescindir** de cualquiera de las opciones. **No admite** pesos iguales a **cero**.

Ejemplo 7.39

Construya un grafo 3D con la siguiente matriz de adyacencia de pesos:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 3 & 1 & 0 \\ 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 4 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

en el orden de sus vértices $\{c,a,b,e,f,g\}$. Emplee la opción "`mostrarpesos->True`" para imprimir las ponderaciones sobre cada uno de sus lados.

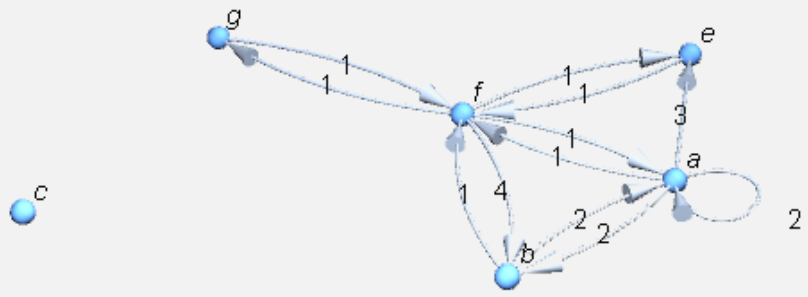
Solución:

Al utilizar **GrafoMP**:

In[ ] :=

```
A = {c, a, b, e, f, g};
Matriz = {{0, 0, 0, 0, 0, 0}, {0, 2, 2, 3, 1, 0}, {0, 2, 0, 0, 1, 0}, {0, 0, 0, 0, 1, 0}, {0, 1, 4, 1, 0, 1}, {0, 0, 0, 0, 1, 0}};
GrafoMP[Matriz, A, mostrarpesos->True, dimensions3d->True]
```

Out[ ] :=



Ejemplo 7.40

Considere las siguientes matrices de adyacencia de pesos de dos grafos distintos:

$$\begin{pmatrix} 0&1&0&1&1&0&1\\ 1&0&0&0&0&0&1\\ 0&0&0&0&0&1&1\\ 1&0&0&0&1&1&0\\ 1&0&0&1&0&1&0\\ 0&0&1&1&1&0&0\\ 1&1&1&0&0&0&0 \end{pmatrix} \text{ y } \begin{pmatrix} 0&0&0&1&2&0&3\\ 4&0&0&0&0&0&5\\ 0&0&0&0&0&1&1\\ 1&0&0&0&1&1&0\\ 1&0&0&1&0&1&0\\ 0&0&1&8&1&0&0\\ 1&1&20&0&0&0&0 \end{pmatrix}$$

Construya los grafos en tercera dimensión con los pesos sobre sus aristas, en el orden de sus vértices $\{1,a,2,b,3,c,4\}$.

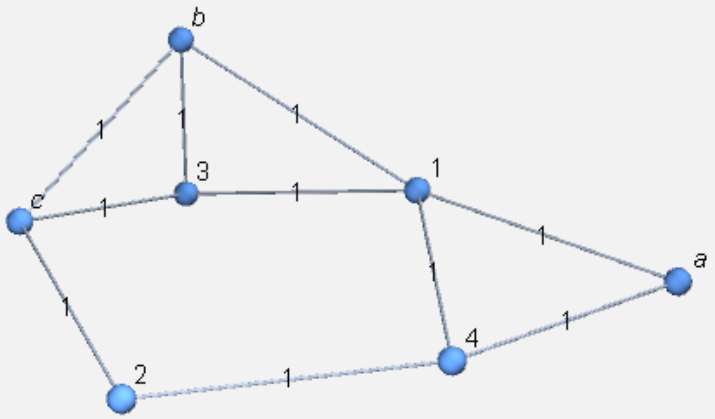
Solución:

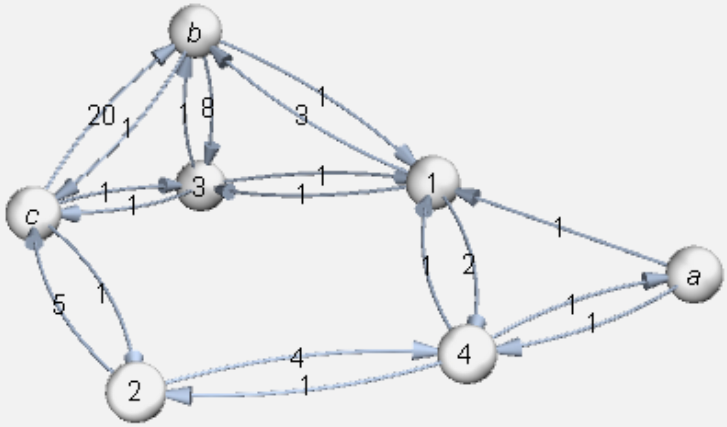
En *Mathematica*:

In[ ] :=

```
A = {1, a, 2, b, 3, c, 4};
Matriz = {{0,1,0,1,1,0,1},{1,0,0,0,0,0,1},{0,0,0,0,0,1,1},{1,0,0,0,1,1,0},{1,0,0,1,0,1,0},{0,0,1,1,1,0,0},{1,1,1,0,0,0,0}};
GrafoMP[Matriz, A, dimensions3d->True, mostrarpesos->True]
Matriz = {{0,0,0,1,2,0,3},{4,0,0,0,0,0,5},{0,0,0,0,0,1,1},{1,0,0,0,1,1,0},{1,0,0,1,0,1,0},{0,0,1,8,1,0,0},{1,1,20,0,0,0,0}};
GrafoMP[Matriz, A, dimensions3d->True, shape->True,
mostrarpesos->True]
```

Out[ ] :=





N El segundo grafo es dirigido pues la matriz de adyacencia de pesos no es simétrica. En él se recurrió a la opción "**`shape->True`**" para contener cada nodo en esferas.

Explicación en video

21) **MPGrafo**: recibe un **grafo ponderado** "G" y sus **vértices** en un **conjunto** "Nodos", **devolviendo** la correspondiente **matriz de adyacencia de pesos** de acuerdo con el **orden** de "Nodos". El comando brinda la **opción** "**`table->True`**", la cual permite **mostrar** la **matriz** con un **formato de tabla**, añadiendo **encabezados** en filas y columnas con los **vértices** contenidos en el **orden** del conjunto "Nodos".

Sintaxis: **`MPGrafo[G, Nodos]`**, o, **`MPGrafo[G, Nodos, table->True]`**. Por defecto, si el grafo posee **aristas múltiples**, por cada una en la entrada correspondiente de la matriz, se muestra la **suma de los pesos**. Si el grafo se generó con "Combinatorica" no acepta **aristas mixtas** (dirigidas y no dirigidas).

Ejemplo 7.41

Determine una matriz de adyacencia de pesos (como matriz y en formato de tabla) sobre un grafo bipartito completo de orden 5×7, asignando a cada una de sus aristas un peso real seudoaleatorio de uno a diez.

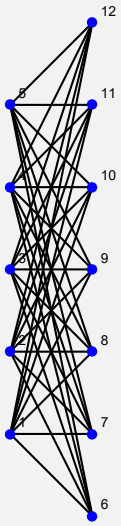
Solución:

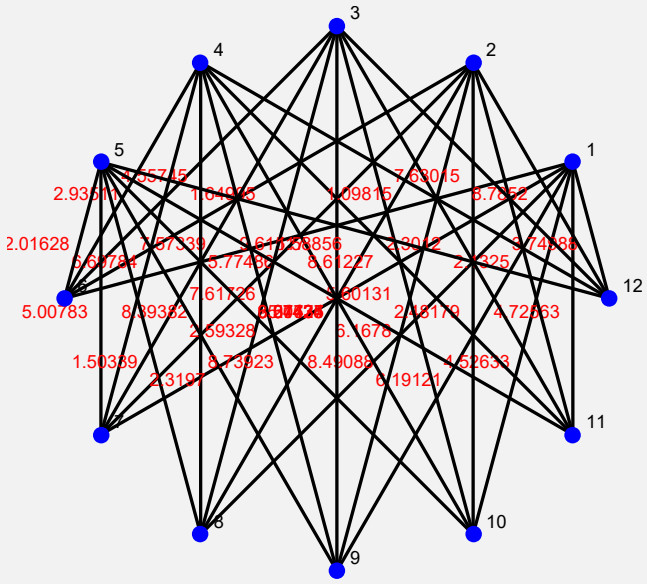
El grafo $K_{5,7}$ se puede construir en "Combinatorica" mediante **CompleteKPartiteGraph**, luego:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[CompleteKPartiteGraph[5,
7], VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
GrafoC[Edges[grafo], pesos->RandomReal[{1, 10}, M[grafo]],
mostrarpesos->True]
nodos = Reverse[Table[i, {i, V[G]}]]
MPGrafo[G, nodos]
MPGrafo[G, nodos, table->True]
```

Out[ ] :=





{12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1}

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.58856 & 1.09815 & 7.63015 & 8.7852 & 3.74988 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 5.7425 & 8.61227 & 2.3012 & 2.1325 & 4.72563 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 8.73923 & 3.37435 & 5.60131 & 2.48179 & 4.52633 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.3197 & 2.59328 & 6.54774 & 6.1678 & 6.19121 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.50339 & 8.39382 & 7.61726 & 3.00434 & 8.49088 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 5.00783 & 6.69784 & 7.57339 & 5.77486 & 9.21627 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 2.01628 & 2.93511 & 4.55745 & 1.64995 & 9.61517 \\
1.58856 & 5.7425 & 8.73923 & 2.3197 & 1.50339 & 5.00783 & 2.01628 & 0 & 0 & 0 & 0 & 0 \\
1.09815 & 8.61227 & 3.37435 & 2.59328 & 8.39382 & 6.69784 & 2.93511 & 0 & 0 & 0 & 0 & 0 \\
7.63015 & 2.3012 & 5.60131 & 6.54774 & 7.61726 & 7.57339 & 4.55745 & 0 & 0 & 0 & 0 & 0 \\
8.7852 & 2.1325 & 2.48179 & 6.1678 & 3.00434 & 5.77486 & 1.64995 & 0 & 0 & 0 & 0 & 0 \\
3.74988 & 4.72563 & 4.52633 & 6.19121 & 8.49088 & 9.21627 & 9.61517 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

	12	11	10	9	8	7	6	5	4	3	2	1
12	0	0	0	0	0	0	0	1.58856	1.09815	7.63015	8.7852	3.74988
11	0	0	0	0	0	0	0	5.7425	8.61227	2.3012	2.1325	4.72563
10	0	0	0	0	0	0	0	8.73923	3.37435	5.60131	2.48179	4.52633
9	0	0	0	0	0	0	0	2.3197	2.59328	6.54774	6.1678	6.19121
8	0	0	0	0	0	0	0	1.50339	8.39382	7.61726	3.00434	8.49088
7	0	0	0	0	0	0	0	5.00783	6.69784	7.57339	5.77486	9.21627
6	0	0	0	0	0	0	0	2.01628	2.93511	4.55745	1.64995	9.61517
5	1.58856	5.7425	8.73923	2.3197	1.50339	5.00783	2.01628	0	0	0	0	0
4	1.09815	8.61227	3.37435	2.59328	8.39382	6.69784	2.93511	0	0	0	0	0
3	7.63015	2.3012	5.60131	6.54774	7.61726	7.57339	4.55745	0	0	0	0	0
2	8.7852	2.1325	2.48179	6.1678	3.00434	5.77486	1.64995	0	0	0	0	0
1	3.74988	4.72563	4.52633	6.19121	8.49088	9.21627	9.61517	0	0	0	0	0

N **`M`** cuenta la cantidad de lados y **`V`** el número de nodos sobre un grafo creado con el paquete "Combinatorica".

Ejemplo 7.42

Encuentre una matriz de adyacencia de pesos (como matriz y en formato de tabla) sobre un grafo devuelto por **`GrafoRandom[10, 10]`**, considerando alguna de las permutaciones obtenidas al ordenar el conjunto de vértices retornado a través de **`VertexList`** y asignando pesos seudoaleatorios enteros de uno a diez. Resuelva el mismo problema tomando los

lados de este grafo y generándo otro, dirigido. Finalmente, ejecute **GrafoRandom[3, 3, simple->False]** y encuentre su matriz de adyacencia de pesos, asignando un uno como ponderación sobre cada arista.

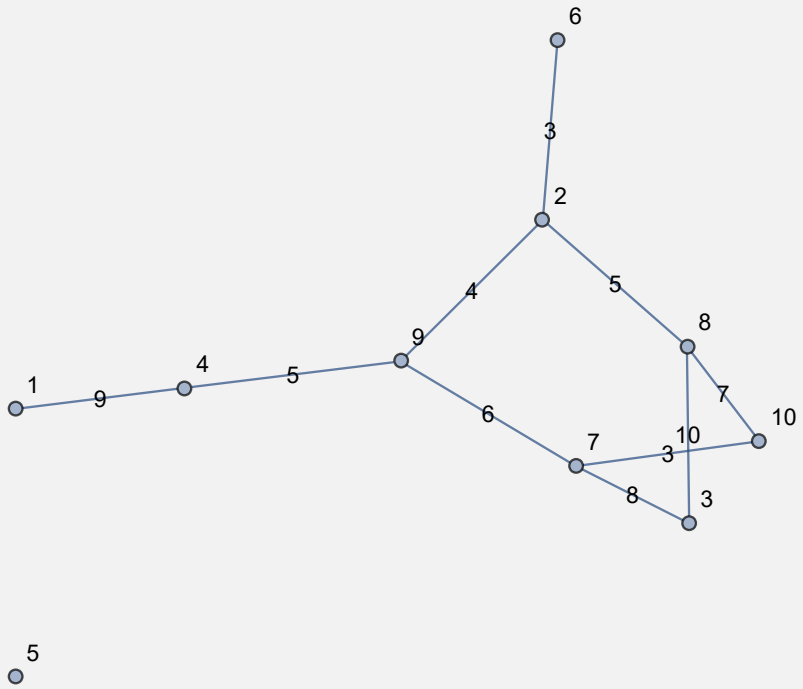
Solución:

En el software la primera y segunda parte del ejemplo, se podría resolver así:

In[ ]:=

```
G = GrafoRandom[10, 10];
grafo = Grafo[AristasWolframSystemToCombinatorica[EdgeList[G]],
vertices->VertexList[G], pesos->RandomInteger[{1, 10},
EdgeCount[G]], mostrarpesos->True]
nodos = RandomChoice[Permutations[VertexList[G]]]
MPGrafo[grafo, nodos]
MPGrafo[grafo, nodos, table->True]
grafo = Grafo[AristasWolframSystemToCombinatorica[EdgeList[G]],
dirigido->True, vertices->VertexList[G], pesos->RandomInteger[{1,
10}, EdgeCount[G]], mostrarpesos->True]
MPGrafo[grafo, nodos]
MPGrafo[grafo, nodos, table->True]
```

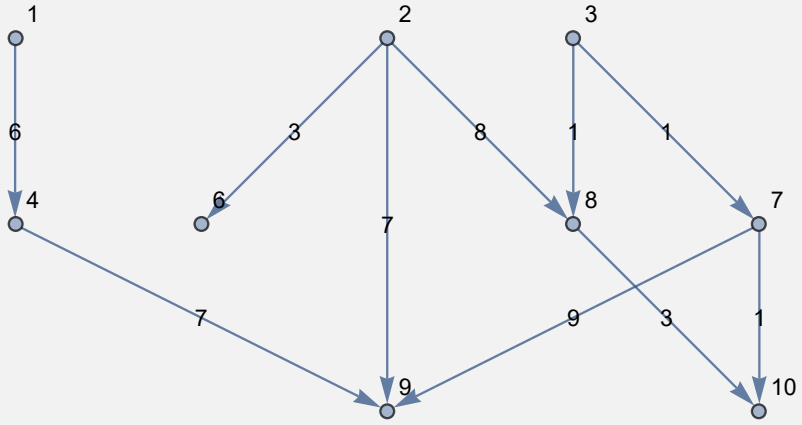
Out[ ]:=



{7, 3, 5, 9, 4, 1, 6, 2, 10, 8}

$$\begin{pmatrix} 0 & 8 & 0 & 6 & 0 & 0 & 0 & 0 & 3 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 5 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 3 & 0 & 0 & 5 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 10 & 0 & 0 & 0 & 0 & 0 & 5 & 7 & 0 \end{pmatrix}$$

	7	3	5	9	4	1	6	2	10	8
7	0	8	0	6	0	0	0	0	3	0
3	8	0	0	0	0	0	0	0	0	10
5	0	0	0	0	0	0	0	0	0	0
9	6	0	0	0	5	0	0	4	0	0
4	0	0	0	5	0	9	0	0	0	0
1	0	0	0	0	9	0	0	0	0	0
6	0	0	0	0	0	0	0	3	0	0
2	0	0	0	4	0	0	3	0	0	5
10	3	0	0	0	0	0	0	0	0	7
8	0	10	0	0	0	0	0	5	7	0



$$\begin{pmatrix} 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 & 3 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \end{pmatrix}$$

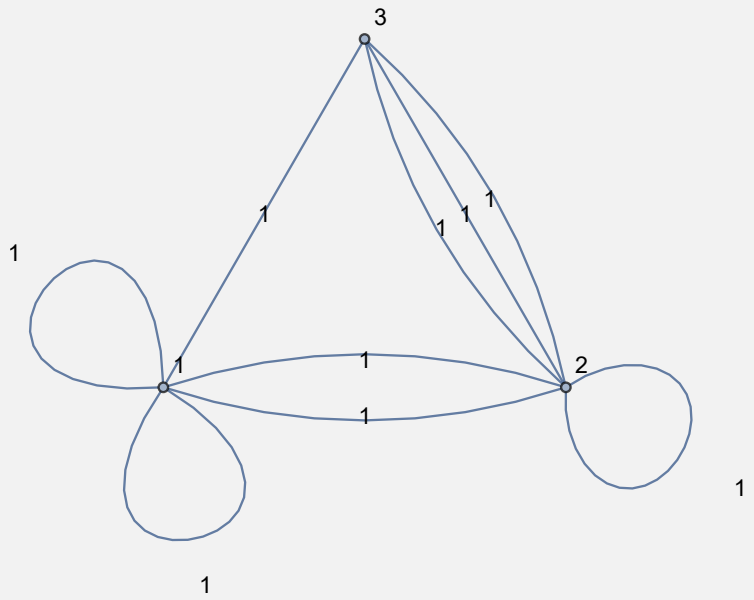
	7	3	5	9	4	1	6	2	10	8
7	0	0	0	9	0	0	0	0	1	0
3	1	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
4	0	0	0	7	0	0	0	0	0	0
1	0	0	0	0	6	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
2	0	0	0	7	0	0	3	0	0	8
10	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	3	0

Para dar respuesta a la tercera parte, se puede proceder como sigue:

In[ ] :=

```
G = GrafoRandom[3, 3, simple->False];
grafo = Grafo[AristasWolframSystemToCombinatorica[EdgeList[G]],
vertices->VertexList[G], pesos->Table[1, EdgeCount[G]],
mostrarpesos->True]
PesosAristas[grafo]
nodos = RandomChoice[Permutations[VertexList[G]]]
MPGrafo[grafo, nodos]
MPGrafo[grafo, nodos, table->True]
```

Out[ ] :=



{{1 ●—● 2, 1}, {1 ●—● 3, 1}, {2 ●—● 3, 1}, {2 ●—● 3, 1}, {1 ●—● 2, 1}, {2 ●—● 3, 1}, {1 ●—● 1, 1}, {2 ●—● 2, 1}, {1 ●—● 1, 1}}

{3, 2, 1}

$$\begin{pmatrix} 0 & 4 & 1 \\ 3 & 1 & 1 \\ 1 & 2 & 2 \end{pmatrix}$$

	3	2	1
3	0	4	1
2	3	1	1
1	1	2	2

Explicación en video

22) **GrafoMI**: **crea** un **grafo** por medio del "Wolfram System" generado por su **matriz de incidencia** de acuerdo con un **conjunto** "Nodos", que representa el **orden** de los vértices del grafo, a través del cual se **construyó** la **matriz de incidencia** "M" recibida como parámetro. La instrucción facilita **dos opciones**: "**dimensions3d->True**" y "**shape->True**". "dimensions3d" muestra el grafo en **tercera dimensión** y "shape" coloca los vértices en **discos**, o bien, **esferas** dependiendo de la dimensión de graficación.

Sintaxis: **GrafoMI[M, Nodos]**, o bien, **GrafoMI[M, Nodos, dimensions3d->True, shape->True]**, en esta última invocación pudiendo **prescindir** de cualquiera de las opciones.

Ejemplo 7.43

Represente los grafos cuyas matrices de incidencia son:

$$\begin{pmatrix} 1&1&1&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&1&0&0\\ 0&0&0&0&1&1&0&0&0&0\\ 0&0&0&1&0&0&0&0&1&1\\ 1&0&0&0&0&0&1&0&1&0\\ 0&0&0&0&0&1&1&0&0&1\\ 0&1&0&0&1&0&0&1&0&0 \end{pmatrix} \text{ y } \begin{pmatrix} 1&1&1&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&1&0&0\\ 0&0&0&0&1&1&0&0&0&0\\ 0&0&0&1&0&0&0&0&-1&1\\ 1&0&0&0&0&0&-1&0&1&0\\ 0&0&0&0&0&1&1&0&0&1\\ 0&1&0&0&1&0&0&1&0&0 \end{pmatrix}$$

con el conjunto de vértices ordenados $\{1,a,2,b,3,c,4\}$.

Solución:

Al utilizar la instrucción **GrafoMI**:

In[ ] :=

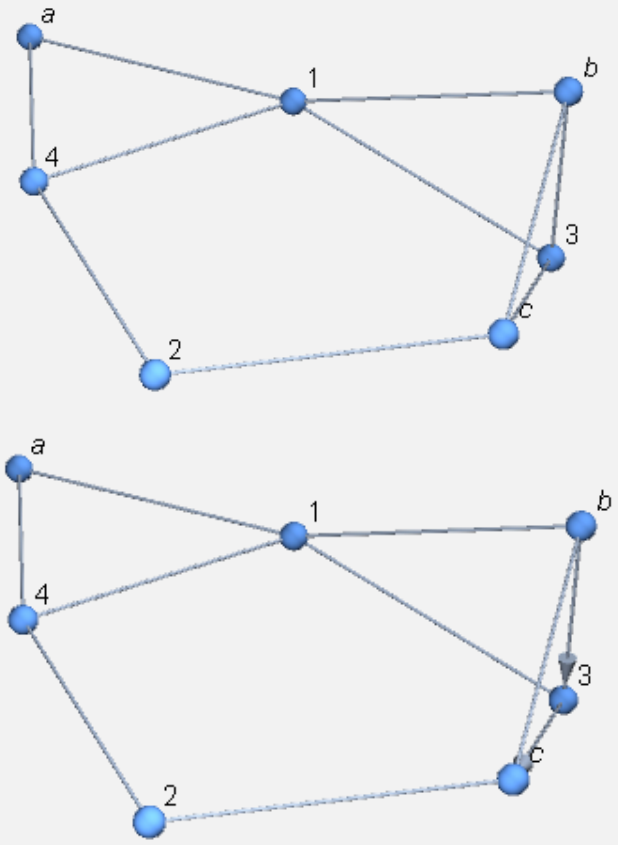
```
A = {1, a, 2, b, 3, c, 4};
Matriz = ( 1 1 1 1 0 0 0 0 0 0
           0 0 1 0 0 0 0 1 0 0
           0 0 0 0 1 1 0 0 0 0
           0 0 0 1 0 0 0 0 1 1
           1 0 0 0 0 0 1 0 1 0
           0 0 0 0 0 1 1 0 0 1
           0 1 0 0 1 0 0 1 0 0 );
GrafoMI[Matriz, A, dimensions3d->True]
```

Matriz =

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix};$$

```
GrafoMI[Matriz, A, dimensions3d->True]
```

Out[ ] :=



Ejemplo 7.44

Construya el grafo con matriz de incidencia:

$$\begin{pmatrix}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0
\end{pmatrix}$$

en el orden de los vértices $\{2,1,3,4,5,6,8,7,9,10,11,12,13,14,15,16,17,18,20,19\}$.

Solución:

In[ ] :=

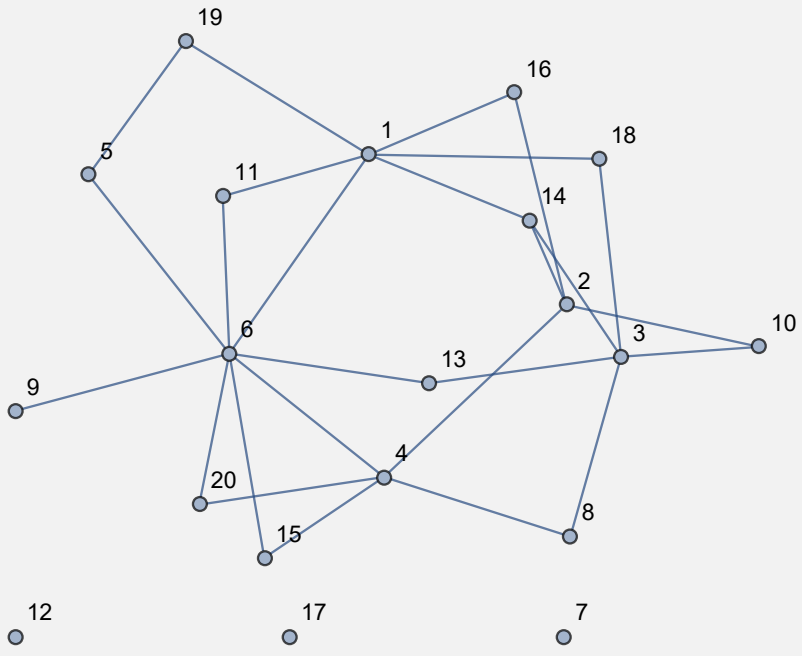
```
A = {2, 1, 3, 4, 5, 6, 8, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
20, 19};
```

Matriz =

$$\begin{pmatrix}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0
\end{pmatrix};$$

GrafoMI[Matriz, A]

Out[ ] :=



Explicación en video

23) **MIGrafo**: recibe un **grafo** "G" y sus **vértices** en un **conjunto** "Nodos", **retornando** la correspondiente **matriz de incidencia** de acuerdo con el **orden** de "Nodos". La instrucción presenta **dos opciones**: "`edgesgraph->L`" y "`table->True`". "L" contiene **todos** los **lados** del grafo, definiendo un **orden** específico a través del cual, se **mostrará** la **matriz de incidencia**. Si el usuario no utiliza esta opción, el software *Mathematica* escoge por **defecto** el orden de los lados. "table" permite **mostrar** la **matriz** con un **formato de tabla**, añadiendo **encabezados** en las filas con los **vértices** contenidos en el **orden** del conjunto "Nodos" y en las columnas con los **lados** del grafo de acuerdo al **orden** de la lista "L". Si no se emplea la opción "edgesgraph", las columnas de la tabla tendrán como etiquetas los **lados** del grafo en el **orden interno** que por oficio escogió el software.

Sintaxis: `MIGrafo[G, Nodos]`, o, `MIGrafo[G, Nodos, edgesgraph->L, table->True]`. Si el grafo tiene **aristas mixtas** (dirigidas y no dirigidas) al **cambiar** el **orden** de los **lados** se deben tomar como **dirigidos**, esta posibilidad no es válida si el grafo se **generó** con "Combinatorica" (no acepta **aristas mixtas**).

Ejemplo 7.45

Halle una matriz de incidencia del grafo dodecaedro tomando el orden en sus vértices {14, 4, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 2, 15, 16, 17, 18, 19, 20}. Encuentre además, la matriz en formato de tabla. Resuelva este mismo problema considerando las aristas del grafo como dirigidas.

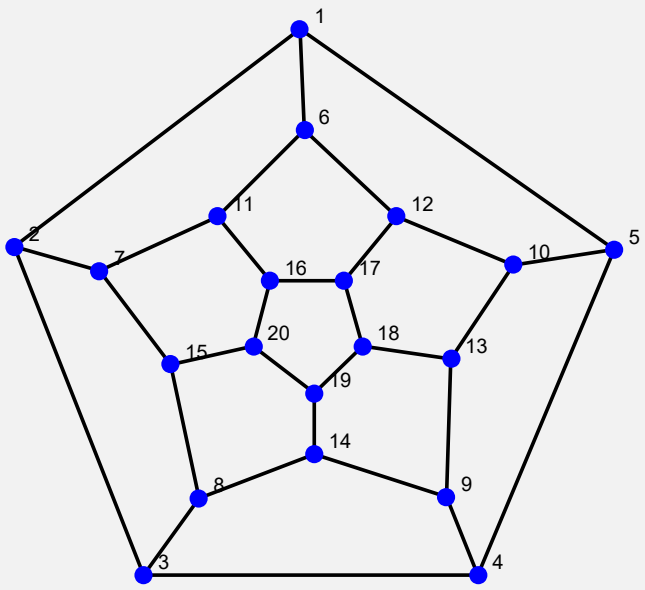
Solución:

En *Mathematica*:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
MIGrafo[grafo, {14, 4, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 2, 15,
16, 17, 18, 19, 20}]
MIGrafo[grafo, {14, 4, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 2, 15,
16, 17, 18, 19, 20}, table->True]
```

Out[ ] :=



$$\begin{pmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0\\
1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1
\end{pmatrix}$$

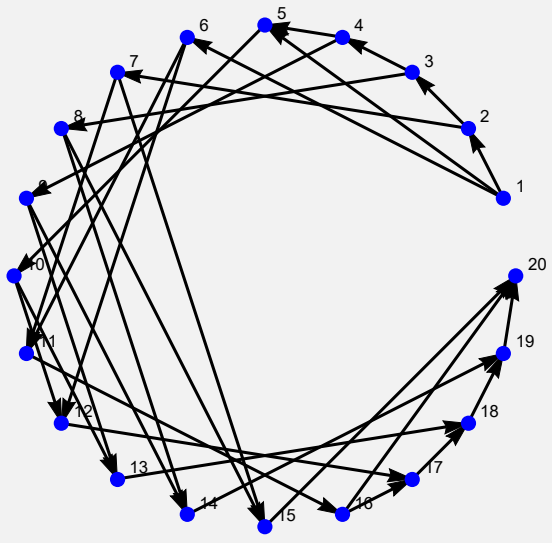
	1 ↔ 2	1 ↔ 5	1 ↔ 6	2 ↔ 3	2 ↔ 7	3 ↔ 4	3 ↔ 8	4 ↔ 5	4 ↔ 9	5 ↔ 10	6 ↔ 11	6 ↔ 12	7 ↔ 11	7 ↔ 15	
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	1	0	1	1	0	0	0	0	0	
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	1	0	1	1	0	0	0	0	0	0	0	
5	0	1	0	0	0	0	0	1	0	1	0	0	0	0	
6	0	0	1	0	0	0	0	0	0	0	1	1	0	0	
7	0	0	0	0	1	0	0	0	0	0	0	0	1	1	
8	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	1	0	1	0	•••
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	1	0	0	1	1	0	0	0	0	0	0	0	0	0	
15	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Si se asume los lados dirigidos:

In[ ] :=

```
ShowGraph[grafo = SetGraphOptions[FromOrderedPairs[Edges[Dodecahedral-
Graph]], VertexColor -> Blue, EdgeColor -> Black], VertexLabel -> True,
PlotRange -> 0.1]
MIGrafo[grafo, {14, 4, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 2, 15,
16, 17, 18, 19, 20}]
MIGrafo[grafo, {14, 4, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 2, 15,
16, 17, 18, 19, 20}, table -> True]
```

Out[ ] :=



$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
-1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \\
1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 &
\end{pmatrix}$$

	1 ↔ 2	1 ↔ 5	1 ↔ 6	2 ↔ 3	2 ↔ 7	3 ↔ 4	3 ↔ 8	4 ↔ 5	4 ↔ 9	5 ↔ 10	6 ↔ 11	6 ↔ 12	7 ↔ 11	7 ↔ 15	8 ↔ 14	8 ↔ 15	
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
4	0	0	0	0	0	1	0	-1	-1	0	0	0	0	0	0	0	
1	-1	-1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	1	0	-1	-1	0	0	0	0	0	0	0	0	0	
5	0	1	0	0	0	0	0	1	0	-1	0	0	0	0	0	0	
6	0	0	1	0	0	0	0	0	0	0	-1	-1	0	0	0	0	
7	0	0	0	0	1	0	0	0	0	0	0	0	-1	-1	0	0	
8	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-1	-1	
9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	•••
11	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	1	0	0	-1	-1	0	0	0	0	0	0	0	0	0	0	0	
15	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Las tablas y la última matriz no se muestran completas por su tamaño.

Ejemplo 7.46

Obtenga una matriz de incidencia de un grafo seudoaleatorio de orden 10×10, tanto simple como con lazos y/o aristas múltiples, al tomar alguna permutación sobre el conjunto generado por **`VertexList`**.

Solución:

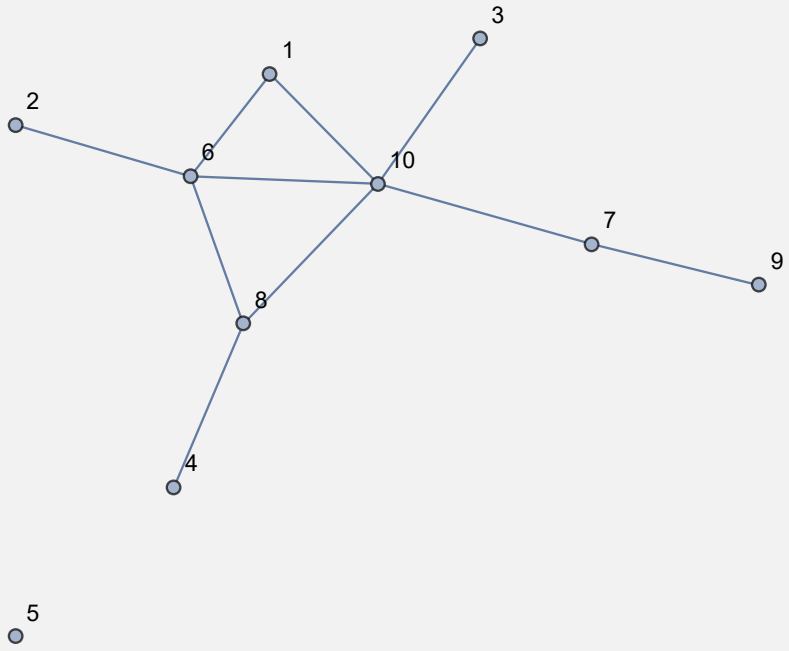
En el software:

In[ ] :=

```
grafo = GrafoRandom[10, 10]
nodos = RandomChoice[Permutations[VertexList[grafo]]]
MIGrafo[grafo, nodos]
```

```
MIGrafo[grafo, nodos, table->True]
grafo = GrafoRandom[10, 10, simple->False]
MIGrafo[grafo, nodos]
MIGrafo[grafo, nodos, table->True]
```

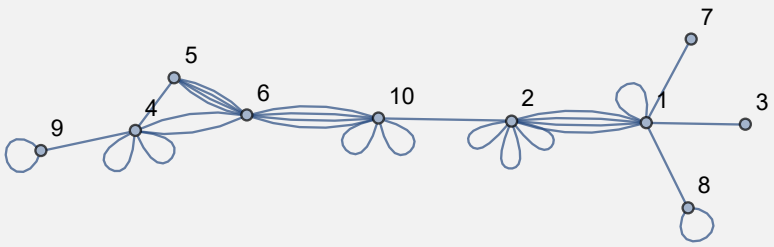
Out[ ]:=



{4, 6, 5, 3, 2, 1, 9, 7, 10, 8}

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

	1 ↔ 6	1 ↔ 10	2 ↔ 6	3 ↔ 10	4 ↔ 8	6 ↔ 8	6 ↔ 10	7 ↔ 9	7 ↔ 10	8 ↔ 10
4	0	0	0	0	1	0	0	0	0	0
6	1	0	1	0	0	1	1	0	0	0
5	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	1	0	0
7	0	0	0	0	0	0	0	1	1	0
10	0	1	0	1	0	0	1	0	1	1
8	0	0	0	0	1	1	0	0	0	1



$$\begin{pmatrix}
0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1\\
0&0&0&0&0&0&1&0&1&1&1&0&0&1&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&1&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&0&0&1&0&0&1&1&0\\
1&1&1&1&0&0&0&0&0&0&0&1&1&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&1&1&0&0&0&0&0&0&1&0&1&1&0&0&0&0&0&1&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0
\end{pmatrix}$$

	1 ↔ 2	1 ↔ 3	1 ↔ 7	1 ↔ 8	2 ↔ 10	4 ↔ 5	4 ↔ 6	4 ↔ 9	5 ↔ 6	6 ↔ 10	6 ↔ 10	
4	0	0	0	0	0	1	1	1	0	0	0	
6	0	0	0	0	0	0	1	0	1	1	1	
5	0	0	0	0	0	1	0	0	1	0	0	
3	0	1	0	0	0	0	0	0	0	0	0	
2	1	0	0	0	1	0	0	0	0	0	0	…
1	1	1	1	1	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	1	0	0	0	
7	0	0	1	0	0	0	0	0	0	0	0	
10	0	0	0	0	1	0	0	0	0	1	1	
8	0	0	0	1	0	0	0	0	0	0	0	

La última tabla omite algunas columnas dadas sus dimensiones.

Explicación en video

24) **ListaCountries**: **muestra** la **lista** de **todos** los **países** integrados en el software *Mathematica*, con la intención de que el usuario los tome en cuenta para su **uso** en **otros comandos**.

Sintaxis: `ListaCountries[]`.

Ejemplo 7.47

Genere la lista de todos los países disponibles en *Mathematica*.

Solución:

In[ ] :=

```
ListaCountries[]
```

Out[ ] :=

{ United States (country) , Jordan (country) , Algeria (country) , ..., Sint Maarten (country) , South Sudan (country) }

Ejemplo 7.48

Determine si "**USA**" y `Entity["Country", "UnitedStates"]` forman parte de `ListaCountries`.

Solución:

Se utilizará en este ejercicio el comando `MemberQ`. Esta instrucción booleana de *Mathematica* retorna "`True`" si recibiendo un conjunto y un elemento, el elemento pertenece al conjunto y "`False`", en caso contrario. Luego:

In[ ] :=

```
MemberQ[ListaCountries[], "USA"]
MemberQ[ListaCountries[], Entity["Country", "UnitedStates"]]
```

Out[ ] :=

False

True

N Se concluye que el string "**USA**" no pertenece a la lista de países. Por otra parte, se aclara al lector que la línea de código: `Entity["Country", "UnitedStates"]`, retorna en *Mathematica*:

United States

Explicación en video

25) **GrafoCountryRegions**: **construye** un **grafo** con las **regiones** o **provincias** de un **país** como **vértices**, de acuerdo con los **datos registrados** por la empresa *Wolfram Research*. De manera **automática** se genera un **grafo ponderado** cuyos pesos en las aristas corresponden a las **longitudes reales** entre cada par de regiones.

Sintaxis: `GrafoCountryRegions[Pais]`, con "Pais" un "string" que contiene el **nombre** del **país** correspondiente (sin tildes). La instrucción corre únicamente si el usuario se encuentra conectado a

Internet y el tiempo de ejecución en la respuesta, depende directamente de la **velocidad** en la networking disponible.

Ejemplo 7.49

Construya el grafo de provincias del país **Costa Rica** y retorne mediante el comando **`PesosAristas`** las distancias entre cada par de regiones.

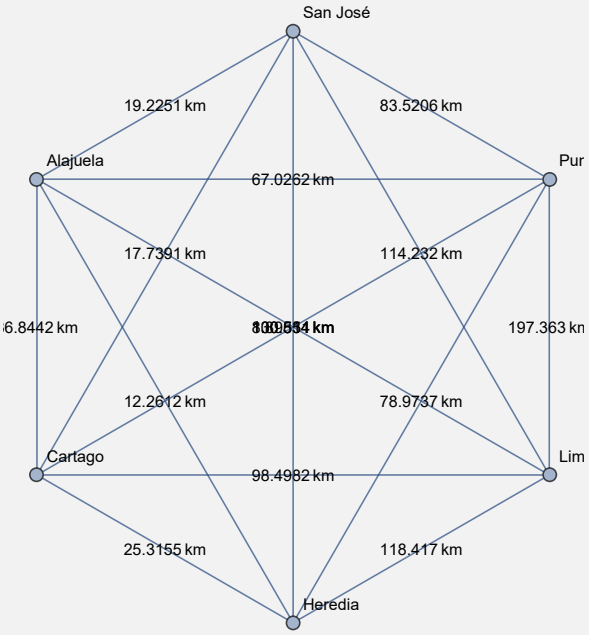
Solución:

En el software:

In[ ] :=

```
grafo = GrafoCountryRegions["Costa Rica"]
PesosAristas[grafo]
```

Out[ ] :=



{{Alajuela ↔ Cartago, 36.8442 km},
{Alajuela ↔ Heredia, 12.2612 km}, {Alajuela ↔ Limon, 130.511 km},
{Alajuela ↔ Puntarenas, 67.0262 km}, {Alajuela ↔ San José, 19.2251 km},
{Cartago ↔ Heredia, 25.3155 km}, {Cartago ↔ Limon, 98.4982 km},
{Cartago ↔ Puntarenas, 100.534 km}, {Cartago ↔ San José, 17.7391 km},
{Heredia ↔ Limon, 118.417 km}, {Heredia ↔ Puntarenas, 78.9737 km},
{Heredia ↔ San José, 8.89854 km}, {Limon ↔ Puntarenas, 197.363 km},
{Limon ↔ San José, 114.232 km}, {Puntarenas ↔ San José, 83.5206 km}}

N El alumno puede observar cómo la provincia de **Guanacaste** no aparece. Esto obedece a la no detección de esta región de acuerdo con los datos obtenidos a través de los servidores de la empresa *Wolfram Research*.

Ejemplo 7.50

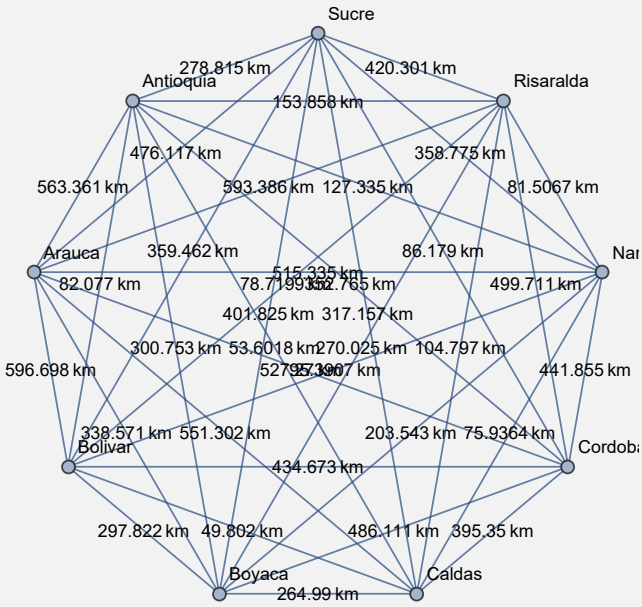
Genere el grafo de regiones de **Colombia**.

Solución:

In[ ] :=

```
GrafoCountryRegions["Colombia"]
```

Out[ ] :=



Explicación en video

26) **GrafoFronteraCountries**: **construye** un **grafo** con **todos** los **países fronterizos** a otro recibido como parámetro, de acuerdo con los **datos registrados** por la empresa *Wolfram Research*.

Sintaxis: **`GrafoFronteraCountries[Pais]`**, con "Pais" un "string" que contiene el **nombre** del **país** (sin tildes). El usuario debe estar conectado a **Internet** para el funcionamiento correcto de la instrucción y su tiempo de respuesta, depende directamente de la **velocidad** de conexión.

Ejemplo 7.51

Determine mediante un grafo los países fronterizos a **Francia**.

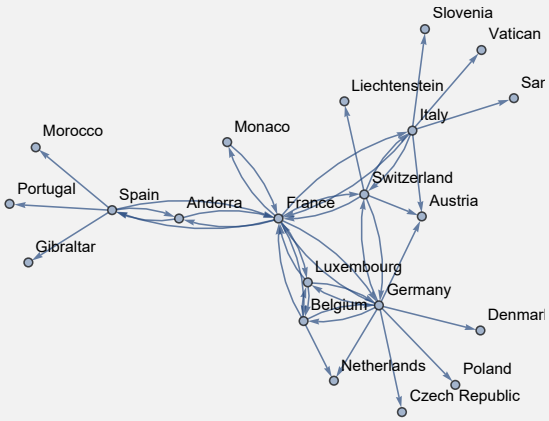
Solución:

Al utilizar **`GrafoFronteraCountries`**:

In[ ] :=

```
GrafoFronteraCountries["France"]
```

Out[ ] :=



N Es importante mencionar que **GrafoFronteraCountries** reconoce únicamente los países, tales y como son devueltos por **ListaCountries**. De allí que **Francia** se escribió en inglés.

Ejemplo 7.52

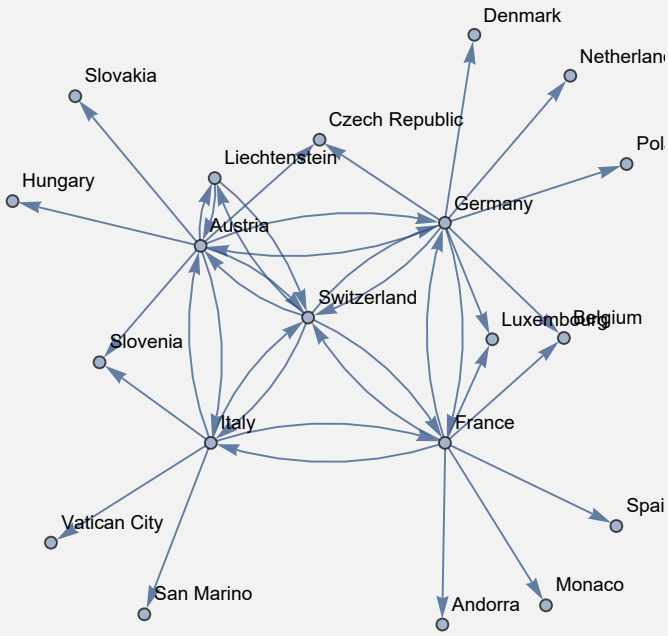
Represente mediante un grafo los países en la frontera de **Suiza**.

Solución:

In[ ] :=

```
GrafoFronteraCountries["Switzerland"]
```

Out[ ] :=



Explicación en video

27) **GrafoRandomConexo**: **construye** si es posible, un **grafo conexo seudoaleatorio no dirigido** con "n" **vértices** y un **mínimo** de "m" **aristas**. Por defecto, el comando lo hace en el **ambiente** provisto por "Wolfram System" de *Mathematica*, sin embargo, brinda la **opción** "`combinatorica -> True`" creando el grafo por medio del **paquete** "Combinatorica" (en este caso, el grafo queda almacenado en una **variable denominada** "G"). Además, la instrucción integra la **propiedad** "`simple -> Valor`" donde "Valor" toma un contenido **booleano**, "`True`" genera un **grafo simple** (sin lazos y lados múltiples) y "`False`", realiza lo **contrario**.

Sintaxis: `GrafoRandomConexo[n, m]`, o bien,

```
GrafoRandomConexo[n, m, combinatorica->True, simple->Valor]
```

pudiendo **prescindir** de cualquiera de sus opciones. Inicialmente "`simple -> True`".

Ejemplo 7.53

Construya un grafo seudoaleatorio conexo con 5 nodos y 6 aristas. Además, genere otro grafo seudoaleatorio no simple con los mismos valores.

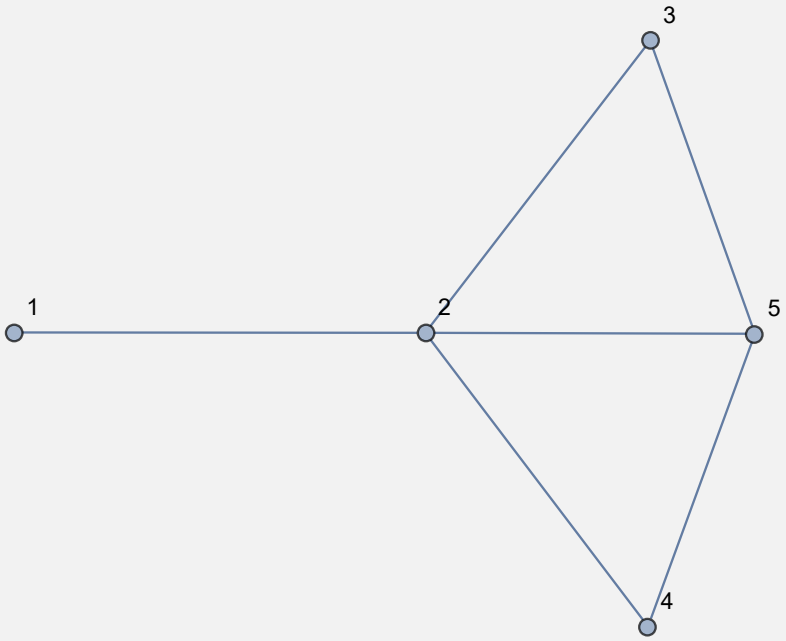
Solución:

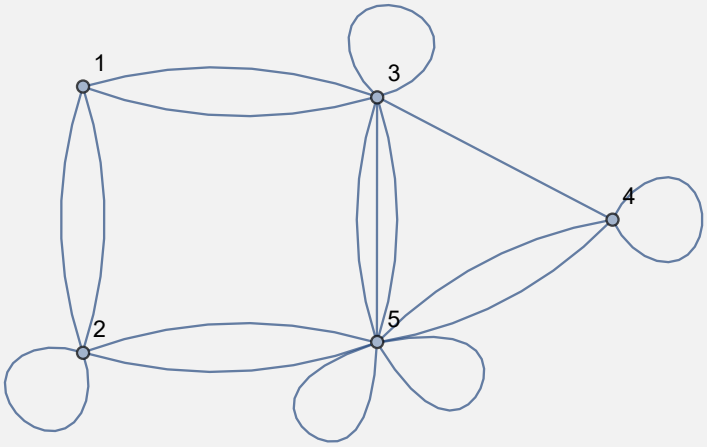
En *Mathematica*:

In[ ] :=

```
GrafoRandomConexo[5, 6]
GrafoRandomConexo[5, 6, simple->False]
```

Out[ ] :=





N **GrafoRandomConexo** ocasionalmente podría no arrojar ningún resultado. La instrucción realiza diez pruebas, si posterior a ellas no se encontró un grafo conexo, se detiene la búsqueda. Esto provoca que al volver a ejecutar el comando, algunas veces se logre hallar la salida satisfactoriamente. Se sugiere al estudiante ejecutar de forma repetida la instrucción si no muestra ningún grafo.

Ejemplo 7.54

Mediante el uso del paquete "Combinatorica", construya un grafo seudoaleatorio conexo no simple de orden 10×20.

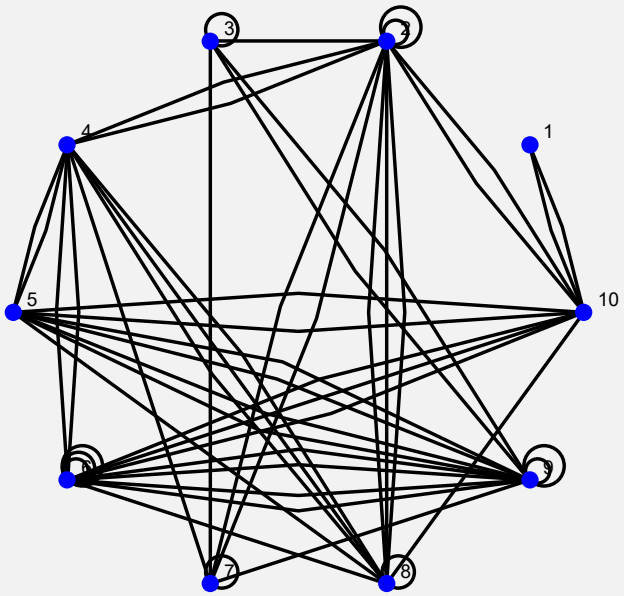
Solución:

En el software:

In[ ] :=

```
GrafoRandomConexo[10, 20, combinatorica->True, simple->False]
```

Out[ ] :=



Explicación en video

28) **CantAristas**: retorna el **número** de **lados** que contiene un **grafo** "G" **dirigido o no**, creado **con o sin** el paquete "Combinatorica".

Sintaxis: `CantAristas[G]` siendo "G" el grafo correspondiente.

Ejemplo 7.55

Utilizando la instrucción **`GrafoRandom`** genere un grafo seudoaleatorio no simple con 40 vértices y un mínimo de 30 aristas. Determine la cantidad de lados que posee este grafo.

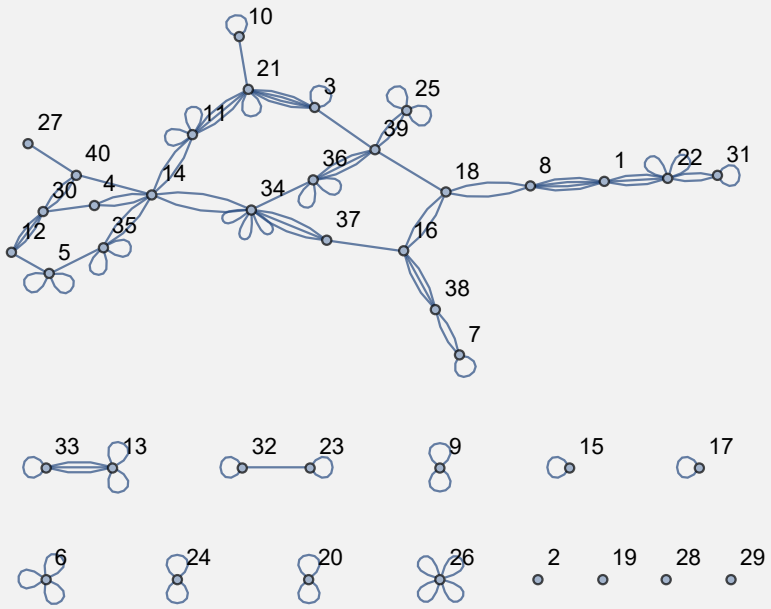
Solución:

Al emplear **CantAristas**:

In[ ] :=

```
grafo = GrafoRandom[40, 30, simple->False]
CantAristas[grafo]
```

Out[ ] :=



100

Ejemplo 7.56

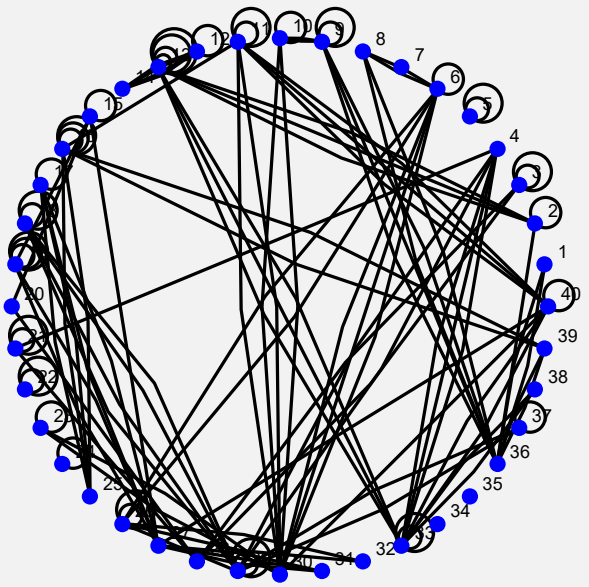
Cuente la cantidad de aristas de un grafo construido por medio de la invocación **GrafoRandom[40, 40, combinatorica->True, simple->False]**.

Solución:

In[ ] :=

```
GrafoRandom[40, 40, combinatorica->True, simple->False]
CantAristas[G]
```

Out[ ] :=



120

N Se recuerda al lector que al emplear la opción "**combinatorica->True**" en el comando **GrafoRandom**, por defecto el grafo queda almacenado en la variable **G**, por esta razón en este ejemplo, la instrucción **CantAristas** opera sobre **G**.

Explicación en video

29) **CantNodos**: retorna el **número** de **vértices** que contiene un **grafo** "G" **dirigido o no**, creado **con o sin** el paquete "Combinatorica".

Sintaxis: **CantNodos[G]** siendo "G" el grafo correspondiente.

Ejemplo 7.57

Halle la cantidad de nodos sobre un grafo mariposa de orden cuatro.

Solución:

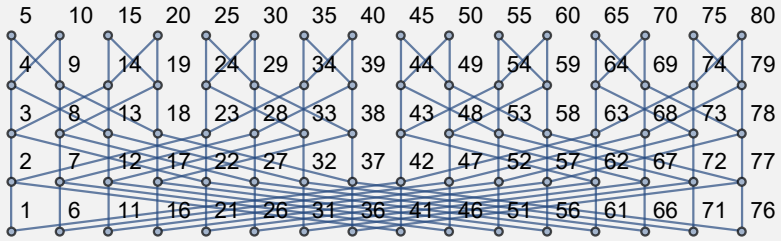
En *Mathematica* el grafo mariposa se puede construir al recurrir al comando **ButterflyGraph**, por lo que:

In[ ] :=

```
grafo = System`ButterflyGraph[4, VertexLabels -> "Name",
```

```
ImagePadding -> 10]
CantNodos[grafo]
```

Out[ ] :=



80

N El uso de **System`** antes del comando **ButterflyGraph** se realizó con la intención de evitar conflictos entre el "Wolfram System" de *Mathematica* y el paquete "Combinatorica". Si "Combinatorica" se ha levantado y se corre posteriormente **ButterflyGraph** sin el **System`** previo, ocurre un error.

Ejemplo 7.58

Encuentre la cantidad de vértices en el grafo tetraedro, generado por la instrucción de "Combinatorica" **TetrahedralGraph**.

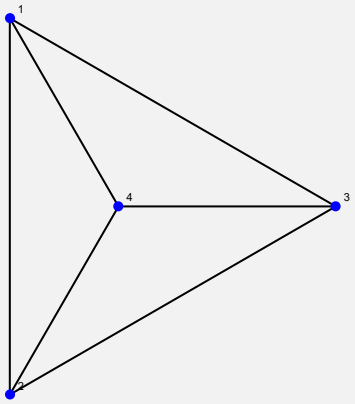
Solución:

En el software:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[TetrahedralGraph,
VertexColor -> Blue, EdgeColor -> Black], VertexLabel -> True,
PlotRange -> 0.1]
CantNodos[grafo]
```

Out[ ] :=



```
4
```

Explicación en video

30) **Valencias**: retorna las **valencias** o **grados** de todos los **vértices** de un **grafo** "G" **dirigido o no**, creado **con o sin** el paquete "Combinatorica".

Sintaxis: `Valencias[G]` siendo "G" el grafo correspondiente.

Ejemplo 7.59

Encuentre las valencias o grados de:

- Un grafo seudoaleatorio de orden 10×10.
- Un grafo seudoaleatorio no simple de orden 10×10.
- Un grafo dirigido con las mismas aristas del caso anterior.

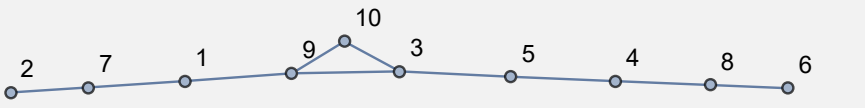
Solución:

En *Mathematica*:

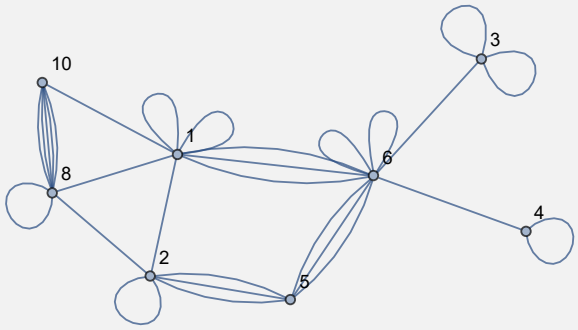
In[ ] :=

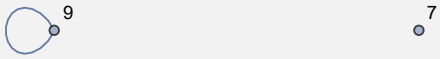
```
grafo = GrafoRandom[10, 10]
Valencias[grafo]
grafo = GrafoRandom[10, 10, simple->False]
Valencias[grafo]
grafo = Grafo[AristasWolframSystemToCombinatorica[EdgeList[grafo]],
dirigido->True, vertices->VertexList[grafo]]
Valencias[grafo]
```

Out[ ] :=

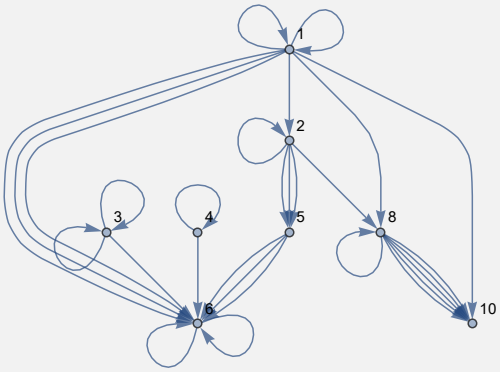


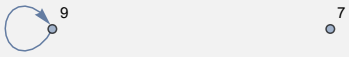
	1	2	3	4	5	6	7	8	9	10
Grado o valencia	2	1	3	2	2	1	2	2	3	2





	1	2	3	4	5	6	7	8	9	10
Grado o valencia	10	7	5	3	6	12	0	9	2	6





	1	2	3	4	5	6	7	8	9	10
Grado o valencia interna	2	2	2	1	3	10	0	3	1	6
Grado o valencia externa	8	5	3	2	3	2	0	6	1	0

N Como se observa en la última salida, cuando el grafo es dirigido **`Valencias`** lo detecta automáticamente, retornando una tabla con los grados internos y externos de cada vértice.

Ejemplo 7.60

A través del uso del paquete "Combinatorica" construya los grafos:

- `LineGraph[CompleteGraph[5]]`
- `CirculantGraph[5, RandomKSubset[Range[10], 3]]`
- `DeBruijnGraph[2, 3]`

Halle en cada caso los grados de sus respectivos nodos.

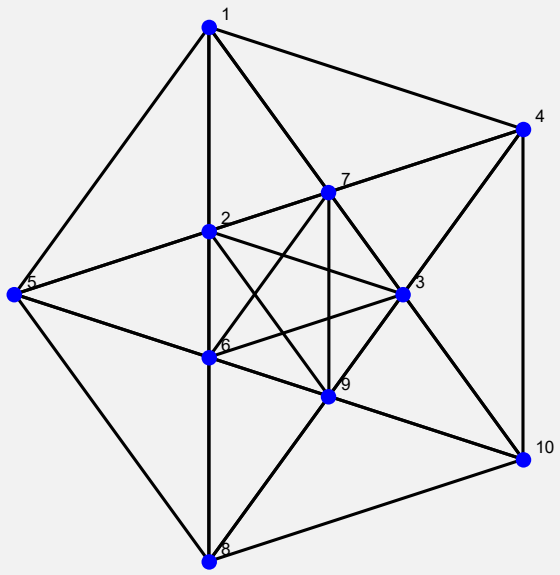
Solución:

En el software:

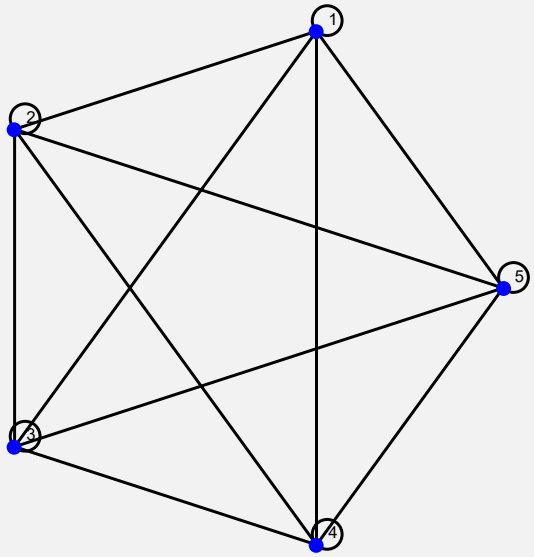
In[ ] :=

```
Quiet[<<Combinatorica`]
(* Grafo G1 *)
ShowGraph[G1 = SetGraphOptions[LineGraph[CompleteGraph[5]],
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
Valencias[G1]
(* Grafo G2 *)
ShowGraph[G2 = SetGraphOptions[CirculantGraph[5,
RandomKSubset[Range[10], 3]], VertexColor->Blue, EdgeColor->Black],
VertexLabel->True, PlotRange->0.1]
Valencias[G2]
(* Grafo G3 *)
ShowGraph[G3 = SetGraphOptions[DeBruijnGraph[2, 3],
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
Valencias[G3]
```

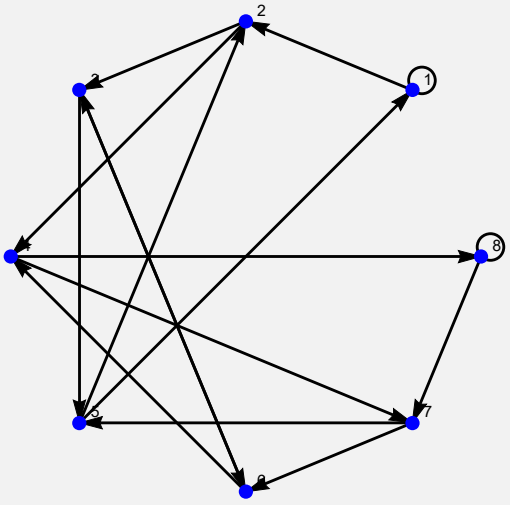
Out[ ] :=



	1	2	3	4	5	6	7	8	9	10
Grado o valencia	6	6	6	6	6	6	6	6	6	6



	1	2	3	4	5
Grado o valencia	6	6	6	6	6



	1	2	3	4	5	6	7	8
Grado o valencia interna	2	2	2	2	2	2	2	2
Grado o valencia externa	2	2	2	2	2	2	2	2

Explicación en video

31) **ElementosGrafo**: retorna sobre un **grafo** "G", la **lista de sus vértices** y sus correspondientes **valen-**

cias, la **lista de sus aristas** y, la **cantidad** de **nodos** y de **lados** de "G". El grafo pudo haber sido creado **con o sin** el paquete "Combinatorica", teniendo la posibilidad de ser **dirigido o no**.

Sintaxis: `ElementosGrafo[G]` siendo "G" el grafo correspondiente.

Ejemplo 7.61

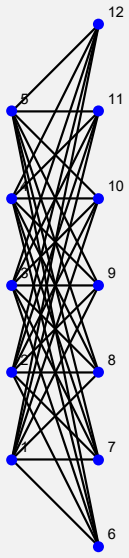
Ejecute el comando **`ElementosGrafo`** sobre $K_{5,7}$.

Solución:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[CompleteKPartiteGraph[5,
7], VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
ElementosGrafo[grafo]
```

Out[ ] :=



	1	2	3	4	5	6	7	8	9	10	11	12
Grado o valencia	7	7	7	7	7	5	5	5	5	5	5	5

La lista de aristas del grafo corresponde a: {1 ●—● 6, 1 ●—● 7, 1 ●—● 8, 1 ●—● 9, 1 ●—● 10, 1 ●—● 11, 1 ●—● 12, 2 ●—● 6, 2 ●—● 7, 2 ●—● 8, 2 ●—● 9, 2 ●—● 10, 2 ●—● 11, 2 ●—● 12, 3 ●—● 6, 3 ●—● 7, 3 ●—● 8, 3 ●—● 9, 3 ●—● 10, 3 ●—● 11, 3 ●—● 12, 4 ●—● 6, 4 ●—● 7, 4 ●—● 8, 4 ●—● 9, 4 ●—● 10, 4 ●—● 11, 4 ●—● 12, 5 ●—● 6, 5 ●—● 7, 5 ●—● 8, 5 ●—● 9, 5 ●—● 10, 5 ●—● 11, 5 ●—● 12}

La cantidad de vértices del grafo corresponde a: 12

La cantidad de aristas del grafo corresponde a: 35

Ejemplo 7.62

Considere los siguientes grafos:

- Un grafo seudoaleatorio de orden 10×10.
- Un grafo seudoaleatorio no simple de orden 10×10.
- Un grafo dirigido con las mismas aristas del caso anterior.

Sobre cada uno corra la instrucción **`ElementosGrafo`**.

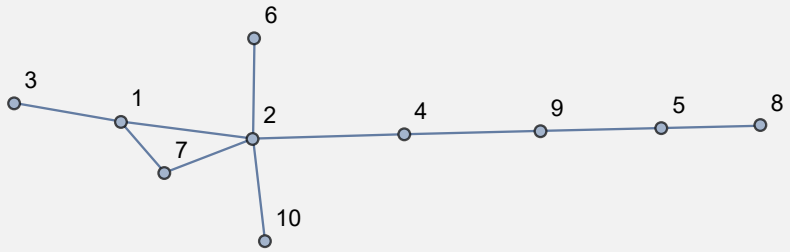
Solución:

En *Mathematica*:

In[ ] :=

```
grafo = GrafoRandom[10, 10]
ElementosGrafo[grafo]
grafo = GrafoRandom[10, 10, simple->False]
ElementosGrafo[grafo]
grafo = Grafo[AristasWolframSystemToCombinatorica[EdgeList[grafo]],
dirigido->True, vertices->VertexList[grafo]]
ElementosGrafo[grafo]
```

Out[ ] :=

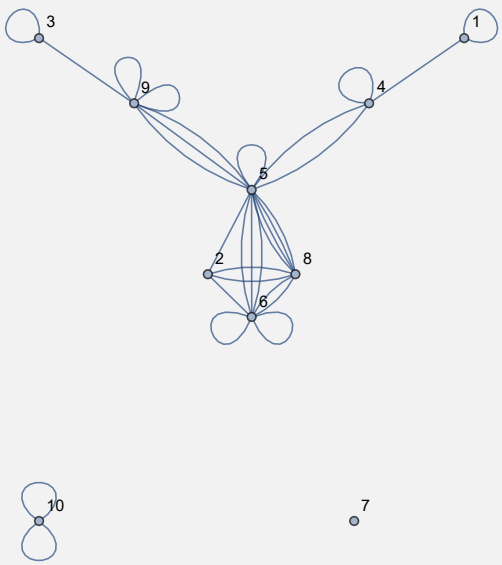


	1	2	3	4	5	6	7	8	9	10
Grado o valencia	3	5	1	2	2	1	2	1	2	1

La lista de aristas del grafo corresponde a: {1 ●—● 2, 1 ●—● 3, 1 ●—● 7, 2 ●—● 4, 2 ●—● 6, 2 ●—● 7, 2 ●—● 10, 4 ●—● 9, 5 ●—● 8, 5 ●—● 9}

La cantidad de vértices del grafo corresponde a: 10

La cantidad de aristas del grafo corresponde a: 10

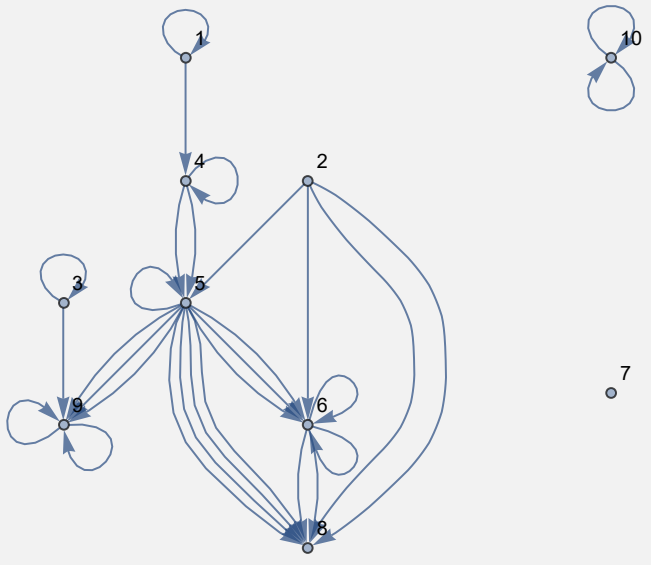


	1	2	3	4	5	6	7	8	9	10
Grado o valencia	3	4	3	5	15	10	0	8	8	4

La lista de aristas del grafo corresponde a: {1 ●—● 4, 2 ●—● 5, 2 ●—● 6, 2 ●—● 8, 3 ●—● 9, 4 ●—● 5, 5 ●—● 6, 5 ●—● 8, 5 ●—● 9, 6 ●—● 8, 5 ●—● 9, 5 ●—● 8, 5 ●—● 8, 5 ●—● 6, 2 ●—● 8, 4 ●—● 5, 6 ●—● 8, 5 ●—● 8, 5 ●—● 6, 5 ●—● 9, 10 ●—● 10, 9 ●—● 9, 1 ●—● 1, 3 ●—● 3, 9 ●—● 9, 10 ●—● 10, 5 ●—● 5, 6 ●—● 6, 6 ●—● 6, 4 ●—● 4}

La cantidad de vértices del grafo corresponde a: 10

La cantidad de aristas del grafo corresponde a: 30



	1	2	3	4	5	6	7	8	9	10
Grado o valencia interna	1	0	1	2	4	6	0	8	6	2
Grado o valencia externa	2	4	2	3	11	4	0	0	2	2

La lista de aristas del grafo corresponde a: {1●—> 4, 2 ●—> 5, 2 ●—> 6, 2 ●—> 8, 3 ●—> 9, 4 ●—> 5, 5 ●—> 6, 5 ●—> 8, 5 ●—> 9, 6 ●—> 8, 5 ●—> 9, 5 ●—> 8, 5 ●—> 8, 5 ●—> 6, 2 ●—> 8, 4 ●—> 5, 6 ●—> 8, 5 ●—> 8, 5 ●—> 6, 5 ●—> 9, 10 ●—> 10, 9 ●—> 9, 1 ●—> 1, 3 ●—> 3, 9 ●—> 9, 10 ●—> 10, 5 ●—>

5, 6 ●–> 6, 6 ●–> 6, 4 ●–> 4}
La cantidad de vértices del grafo corresponde a: 10
La cantidad de aristas del grafo corresponde a: 30

Explicación en video

32) **ResaltarRuta**: **resalta** sobre un **grafo** "G" una **ruta** obtenida de una **lista** "L" de **aristas**. El comando es capaz de procesar a "G", **tanto** si éste ha sido **generado** en el "Wolfram System" de *Mathematica*, como también, si ha sido creado con el **paquete** "Combinatorica" (**sin aristas mixtas**: dirigidas y no dirigidas).

Sintaxis: `ResaltarRuta[G, L]`, "L" es un conjunto de **pares ordenados**.

Ejemplo 7.63

Construya cinco rutas del nodo 1 al 20 en el grafo dodecaedro, seleccione una de ellas al azar y resalte el resultado sobre el grafo.

Solución:

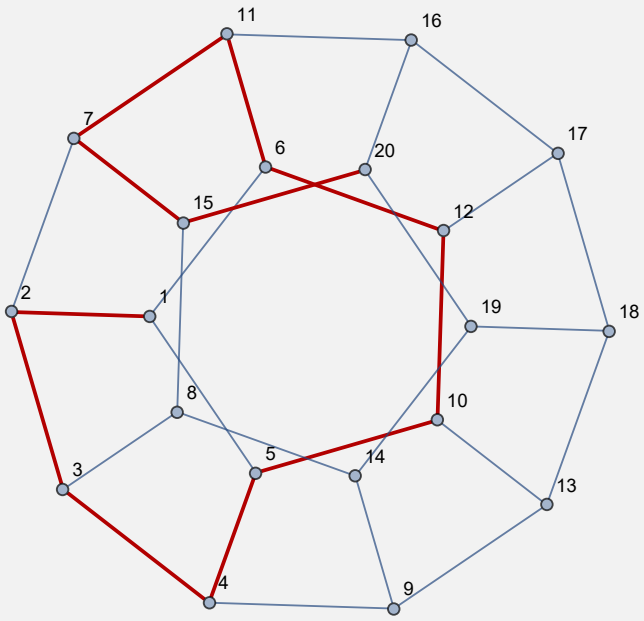
En *Mathematica*:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1];
L = GeneraRutasGraphSimple[grafo, 1, 20, 5];
route = L[[RandomInteger[{1, Length[L]}]]]
ResaltarRuta[grafo, route]
```

Out[ ] :=

{{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 10}, {10, 12}, {12, 6}, {6, 11}, {11, 7}, {7, 15}, {15, 20}}



N El grafo cambia de forma pues se creó con el paquete “Combinatorica”. El comando **`ResaltarRuta`** tiene este comportamiento por defecto.

Ejemplo 7.64

Obtenga de forma seudoaleatoria una ruta del vértice 2 al 4 sobre un grafo retornado por la instrucción **`GrafoRandom[20, 50]`**. Resalte visualmente la trayectoria devuelta.

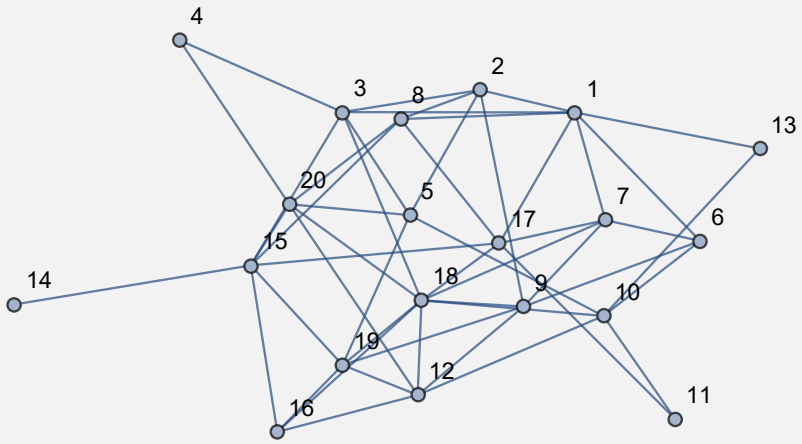
Solución:

En el software primero se debe almacenar el grafo seudoaleatorio en una variable:

In[ ] :=

```
grafo = GrafoRandom[20, 50]
```

Out[ ] :=



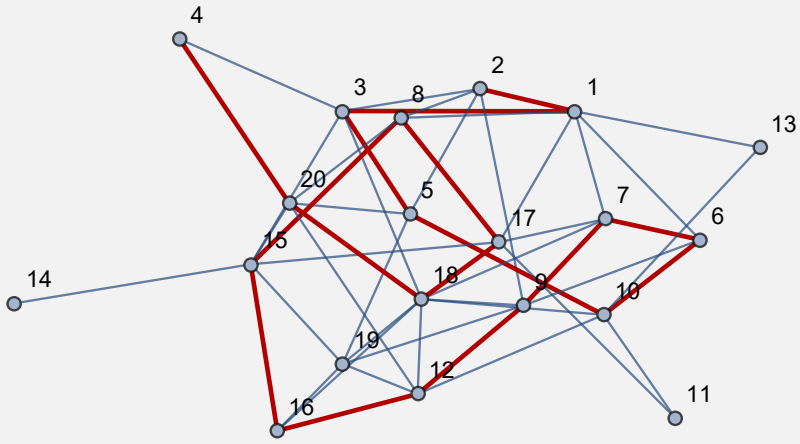
Luego:

In[ ] :=

```
L = GeneraRutasGraphSimple[grafo, 2, 4, 5];
route = L[[RandomInteger[{1, Length[L]}]]]
ResaltarRuta[grafo, route]
```

Out[ ] :=

{{2, 1}, {1, 3}, {3, 5}, {5, 10}, {10, 6}, {6, 7}, {7, 9}, {9, 12}, {12, 16}, {16, 15}, {15, 8}, {8, 17}, {17, 18}, {18, 20}, {20, 4}}



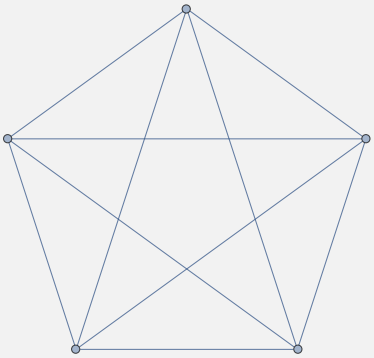
Explicación en video

33) **CDFPropertiesGrafo**: recibe los **vértices** y **aristas** de un **grafo**, además de **dos** de sus nodos, retornando **distintas propiedades**: los **grados** de cada vértice, la **cantidad** de **nodos**, la **cantidad** de **aristas**, si el grafo es **conexo o no**, la **matriz de adyacencia**, la **matriz de adyacencia de pesos**, la **matriz de incidencia**, un **circuito de Euler** si existe, un **circuito de Hamilton** si existe y una **ruta de longitud más corta**, **remarcándola** en el grafo original.

Sintaxis: `CDFPropertiesGrafo[Nodos, Aristas, Vertice1, Vertice2]`, con "Nodos" la lista de vértices del grafo, "Aristas" la lista de lados y, "Vertice1" y "Vertice2" los nodos sobre los cuales se busca el camino de longitud más corta.

Ejemplo 7.65

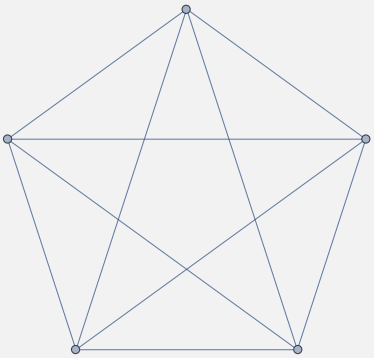
Ejecute el comando **`CDFPropertiesGrafo`** sobre:



con los nodos 1 y 5.

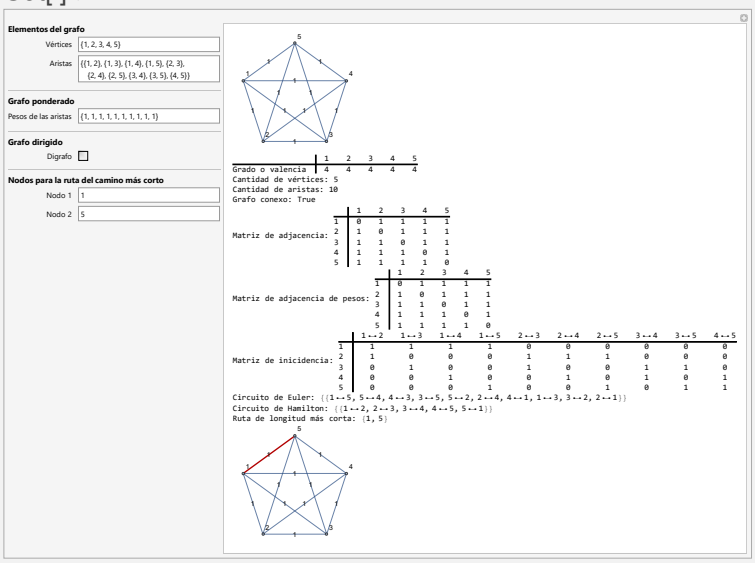
Solución:

In[ ] :=

`grafo =`  **`;`**

```
CDFPropertiesGrafo[VertexList[grafo],
AristasWolframSystemToCombinatorica[EdgeList[grafo]], 1, 5]
```

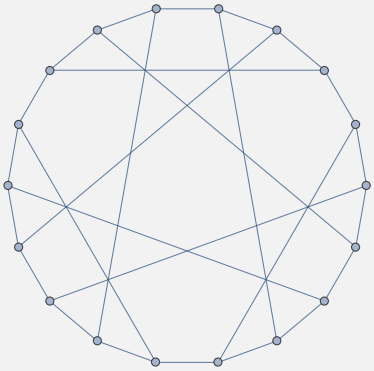
Out[ ] :=



N El grafo de este ejemplo se puede crear con código mediante el comando **GrafoCompleto[5]**.

Ejemplo 7.66

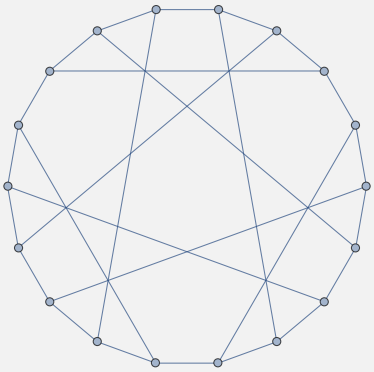
Corra la instrucción **CDFPropertiesGrafo** en el grafo:



con los nodos 1 y 7.

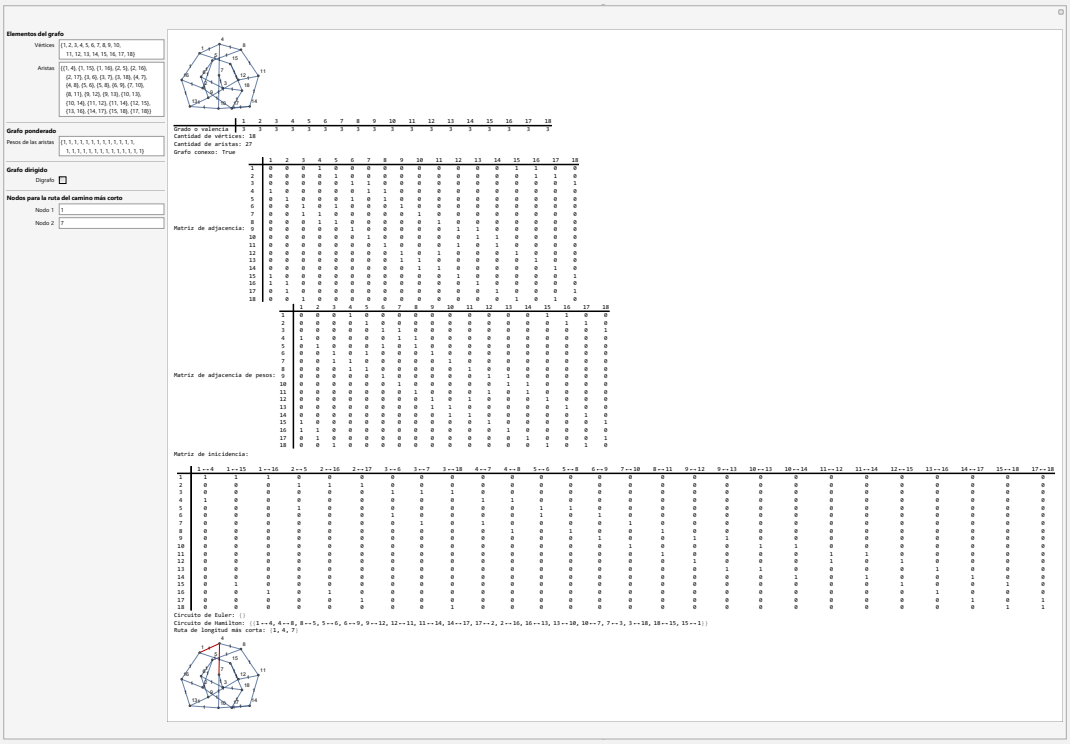
Solución:

In[ ] :=

grafo =  **;**

```
CDFPropertiesGrafo[VertexList[grafo],
AristasWolframSystemToCombinatorica[EdgeList[grafo]], 1, 7]
```

Out[ ] :=



N El grafo del ejemplo se puede construir directamente en *Mathematica* ejecutando **`GraphData["PappusGraph"]`**.

Explicación en video

34) **Ruta**: **construye** una **ruta** sobre un **grafo** "G" dado, de un vértice "Nodo1" a un vértice "Nodo2" definidos por el usuario. El **camino** se **muestra gráficamente** y se calcula la **longitud** de la **trayectoria**.

Sintaxis: **`Ruta[G, Nodo1, Nodo2]`**.

Ejemplo 7.67

Determine una ruta sobre el grafo octaedro (en "Combinatorica": **`OctahedralGraph`**) del nodo 2 al 6, utilizando el comando **`Ruta`**.

Solución:

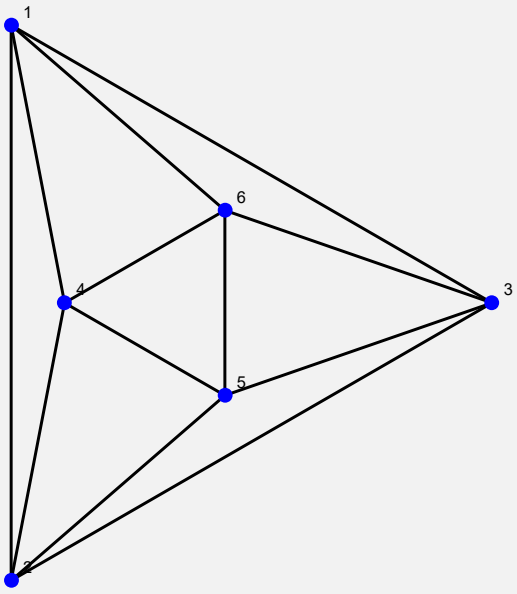
En *Mathematica*:

In[ ] :=

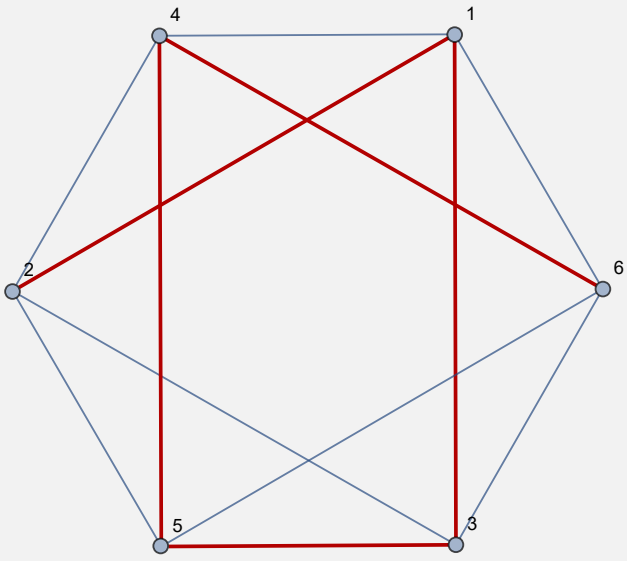
```
Quiet[<<Combinatorica`]
```

```
ShowGraph[grafo = SetGraphOptions[OctahedralGraph,
VertexColor -> Blue, EdgeColor -> Black], VertexLabel -> True,
PlotRange -> 0.1]
Ruta[grafo, 2, 6]
```

Out[ ] :=



Una ruta corresponde a: {{2, 1}, {1, 3}, {3, 5}, {5, 4}, {4, 6}}
Su longitud es: 5



N Es importante mencionar que la ruta resaltada en el último grafo mostrado en el *Out* [ ], se realiza siempre en el "Wolfram System" de *Mathematica*. Por este motivo el aspecto del grafo cambió con relación al original (pues éste se creó mediante el paquete "Combinatorica"), sin embargo, ambos grafos son equivalentes (tienen las mismas aristas y los mismos nodos).

Ejemplo 7.68

Con las aristas de un grafo seudoaleatorio de orden 20×50, construya un grafo ponderado con pesos seudoaleatorios enteros de uno a diez y cuyos vértices se encuentran encerrados por discos. Halle en este grafo una ruta del vértice 2 al 10 empleando la instrucción **`Ruta`**.

Solución:

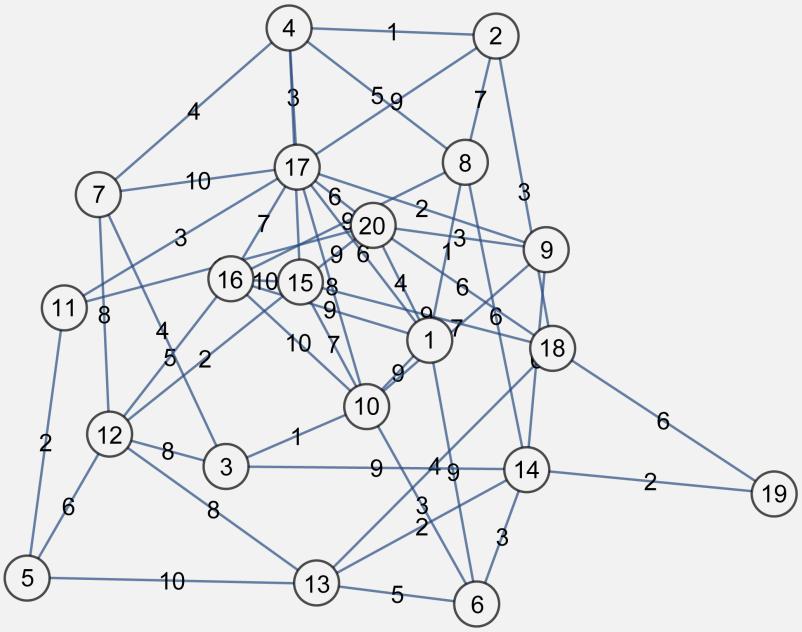
En el software:

In[ ] :=

```
grafo = GrafoRandom[20, 50];
aristas = AristasWolframSystemToCombinatorica[EdgeList[grafo]]
grafo = Grafo[aristas, pesos -> RandomInteger[{1, 10},
Length[aristas]], mostrarpesos -> True, shape -> True]
PesosAristas[grafo]
Ruta[grafo, 2, 10]
```

Out[ ] :=

{{1, 6}, {1, 8}, {1, 10}, {1, 16}, {1, 17}, {1, 20}, {2, 4}, {2, 8}, {2, 17}, {2, 18}, {3, 7}, {3, 10}, {3, 12}, {3, 14}, {4, 7}, {4, 8}, {4, 15}, {4, 17}, {5, 11}, {5, 12}, {5, 13}, {6, 10}, {6, 13}, {6, 14}, {7, 12}, {7, 17}, {8, 14}, {8, 16}, {9, 10}, {9, 14}, {9, 17}, {9, 20}, {10, 15}, {10, 16}, {10, 17}, {11, 17}, {11, 20}, {12, 13}, {12, 15}, {12, 16}, {13, 14}, {13, 18}, {14, 19}, {15, 16}, {15, 18}, {15, 20}, {16, 17}, {17, 20}, {18, 19}, {18, 20}}

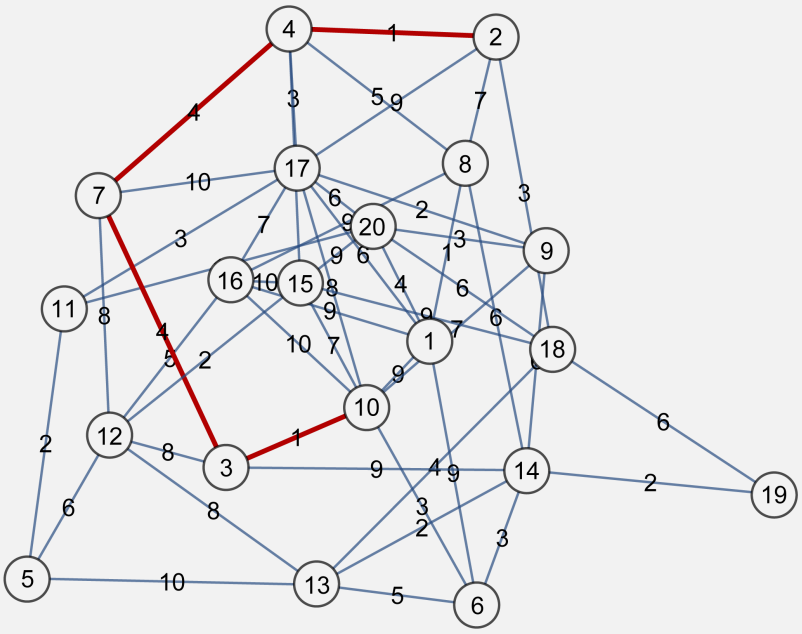


{{1 ●—● 6, 9}, {1 ●—● 8, 1}, {1 ●—● 10, 9}, {1 ●—● 16, 9}, {1 ●—● 17, 6}, {1 ●—● 20, 4}, {2 ●—● 4, 1}, {2

●—● 8, 7}, {2 ●—● 17, 9}, {2 ●—● 18, 3}, {3 ●—● 7, 4}, {3 ●—● 10, 1}, {3 ●—● 12, 8}, {3 ●—● 14, 9}, {4 ●—● 7, 4}, {4 ●—● 8, 5}, {4 ●—● 15, 7}, {4 ●—● 17, 3}, {5 ●—● 11, 2}, {5 ●—● 12, 6}, {5 ●—● 13, 10}, {6 ●—● 10, 3}, {6 ●—● 13, 5}, {6 ●—● 14, 3}, {7 ●—● 12, 8}, {7 ●—● 17, 10}, {8 ●—● 14, 6}, {8 ●—● 16, 9}, {9 ●—● 10, 7}, {9 ●—● 14, 6}, {9 ●—● 17, 2}, {9 ●—● 20, 3}, {10 ●—● 15, 7}, {10 ●—● 16, 10}, {10 ●—● 17, 8}, {11 ●—● 17, 3}, {11 ●—● 20, 1}, {12 ●—● 13, 8}, {12 ●—● 15, 2}, {12 ●—● 16, 5}, {13 ●—● 14, 2}, {13 ●—● 18, 4}, {14 ●—● 19, 2}, {15 ●—● 16, 10}, {15 ●—● 18, 9}, {15 ●—● 20, 9}, {16 ●—● 17, 7}, {17 ●—● 20, 6}, {18 ●—● 19, 6}, {18 ●—● 20, 6}}

Una ruta corresponde a: {{2, 4}, {4, 7}, {7, 3}, {3, 10}}

Su longitud es: 10



N Al ser el grafo ponderado, la longitud de la ruta encontrada corresponde a la suma de los pesos de cada una de las aristas que integra. Lo anterior se puede comprobar mediante el resultado mostrado por **`PesosAristas`**.

Explicación en video

35) **RutaMCorta**: **construye** una **ruta** de **longitud más corta** sobre un **grafo** "G" dado, de un vértice "Nodo1" a un vértice "Nodo2" definidos por el usuario. El **camino** se **muestra gráficamente** y se calcula la **longitud** de la **trayectoria**.

Sintaxis: **`RutaMCorta[G, Nodo1, Nodo2]`**.

Ejemplo 7.69

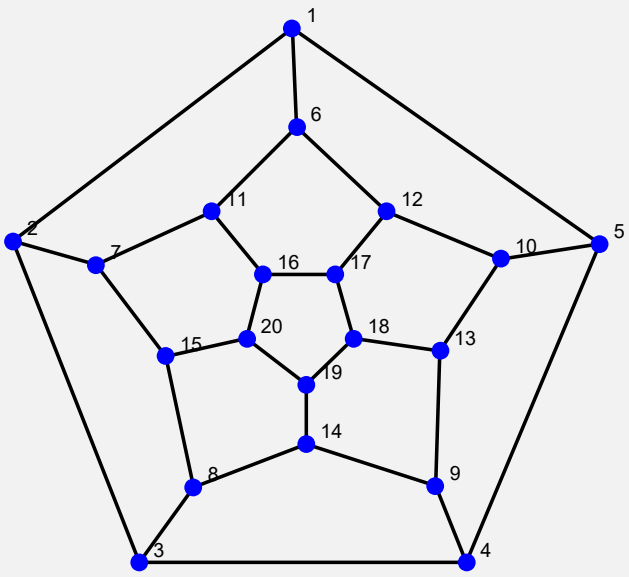
Encuentre una ruta de longitud más corta sobre el grafo dodecaedro del vértice 1 al 16.

Solución:

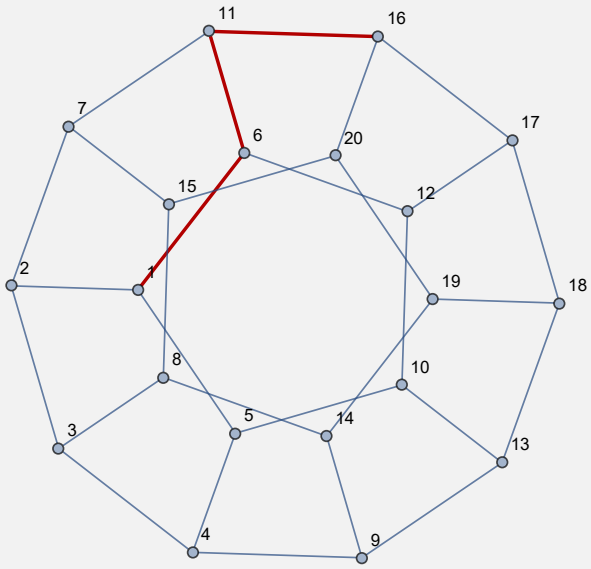
In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
RutaMCorta[grafo, 1, 16]
```

Out[ ] :=



Una ruta de longitud más corta corresponde a: {{1, 6}, {6, 11}, {11, 16}}
Su longitud es: 3



N **RutaMCorta** se asemeja a la instrucción **Ruta**. Si se procesa un grafo creado con el paquete "Combinatorica" (como en este ejemplo), la ruta sobre el grafo la traza en el "Wolfram System", por lo que cambia no su contenido pero sí su aspecto.

Ejemplo 7.70

Genere un grafo ponderado con pesos seudoaleatorios enteros de uno a diez y cuyos vértices se encuentran encerrados por discos, con los lados de un grafo seudoaleatorio de orden 20×50. Determine en este grafo un camino de longitud más corta, del vértice 2 al 10. Utilice el comando **PesosAristas** para verificar la longitud.

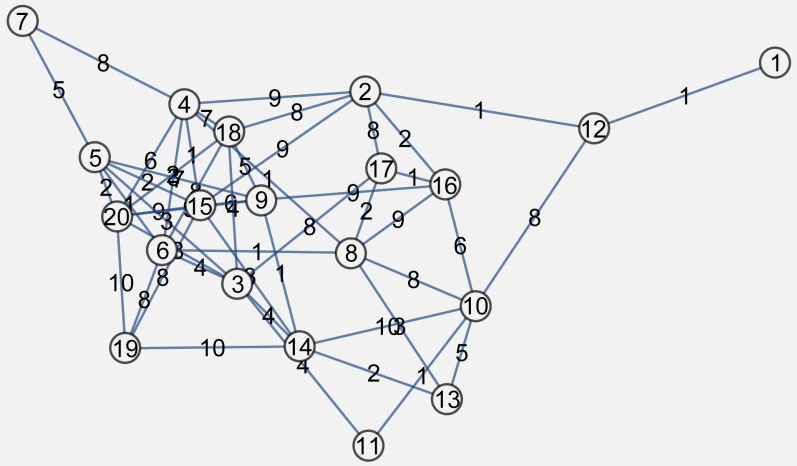
Solución:

In[ ] :=

```
grafo = GrafoRandom[20, 50];
aristas = AristasWolframSystemToCombinatorica[EdgeList[grafo]]
grafo = Grafo[aristas, pesos -> RandomInteger[{1, 10},
Length[aristas]], mostrarpesos -> True, shape -> True]
PesosAristas[grafo]
RutaMCorta[grafo, 2, 10]
```

Out[ ] :=

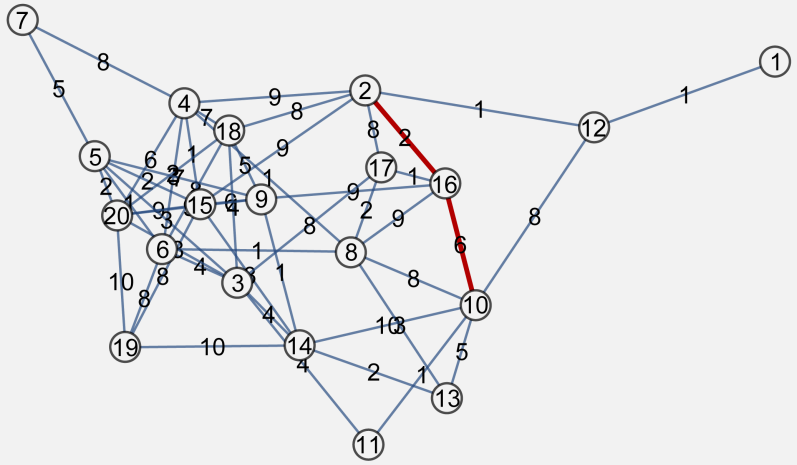
{{1, 12}, {2, 4}, {2, 12}, {2, 15}, {2, 16}, {2, 17}, {2, 18}, {3, 5}, {3, 6}, {3, 11}, {3, 14}, {3, 17}, {3, 18}, {3, 20}, {4, 6}, {4, 7}, {4, 8}, {4, 15}, {4, 18}, {4, 20}, {5, 6}, {5, 7}, {5, 9}, {5, 15}, {5, 20}, {6, 8}, {6, 18}, {6, 19}, {8, 10}, {8, 13}, {8, 16}, {8, 17}, {9, 14}, {9, 15}, {9, 16}, {9, 18}, {9, 20}, {10, 11}, {10, 12}, {10, 13}, {10, 14}, {10, 16}, {13, 14}, {14, 15}, {14, 19}, {15, 19}, {15, 20}, {16, 17}, {18, 20}, {19, 20}}



{{1 ●—● 12, 1}, {2 ●—● 4, 9}, {2 ●—● 12, 1}, {2 ●—● 15, 9}, {2 ●—● 16, 2}, {2 ●—● 17, 8}, {2 ●—● 18, 8}, {3 ●—● 5, 3}, {3 ●—● 6, 4}, {3 ●—● 11, 4}, {3 ●—● 14, 4}, {3 ●—● 17, 8}, {3 ●—● 18, 4}, {3 ●—● 20, 8}, {4 ●—● 6, 3}, {4 ●—● 7, 8}, {4 ●—● 8, 1}, {4 ●—● 15, 1}, {4 ●—● 18, 7}, {4 ●—● 20, 6}, {5 ●—● 6, 1}, {5 ●—● 7, 5}, {5 ●—● 9, 7}, {5 ●—● 15, 2}, {5 ●—● 20, 2}, {6 ●—● 8, 1}, {6 ●—● 18, 8}, {6 ●—● 19, 8}, {8 ●—● 10, 8}, {8 ●—● 13, 3}, {8 ●—● 16, 9}, {8 ●—● 17, 2}, {9 ●—● 14, 1}, {9 ●—● 15, 6}, {9 ●—● 16, 9}, {9 ●—● 18, 5}, {9 ●—● 20, 9}, {10 ●—● 11, 1}, {10 ●—● 12, 8}, {10 ●—● 13, 5}, {10 ●—● 14, 10}, {10 ●—● 16, 6}, {13 ●—● 14, 2}, {14 ●—● 15, 8}, {14 ●—● 19, 10}, {15 ●—● 19, 8}, {15 ●—● 20, 9}, {16 ●—● 17, 1}, {18 ●—● 20, 2}, {19 ●—● 20, 10}}

Una ruta de longitud más corta corresponde a: {{2, 16}, {16, 10}}

Su longitud es: 8.



Se observa en la salida arrojada por **PesosAristas**, el peso de los lados 2 ●—● 16 y 16 ●—● 10, que corresponden a 2 y 6, respectivamente. Por lo tanto, la longitud del camino más corto es 8, al sumar los pesos anteriormente señalados.

Explicación en video

36) **CantRutas**: **cuenta** la **cantidad total** de **rutas** de **cualquier longitud**, de un nodo "a" a un nodo "b", sobre un **grafo** "G" **simple** (sin aristas múltiples, ni lazos). El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, a través del uso del **paquete** "Combinatorica" (**sin aristas mixtas**: dirigidas y no dirigidas). La instrucción proporciona la **opción** "`rutas -> True`" que retorna un **vector** con la **cantidad de caminos** y las **rutas** correspondientes.

Sintaxis: `CantRutas[G, a, b]`, o bien, `CantRutas[G, a, b, rutas -> True]`.

Ejemplo 7.71

Construya un grafo seudoaleatorio de tamaño 20×30 y determine la cantidad de caminos del vértice 2 al 4. Muestre además, las trayectorias.

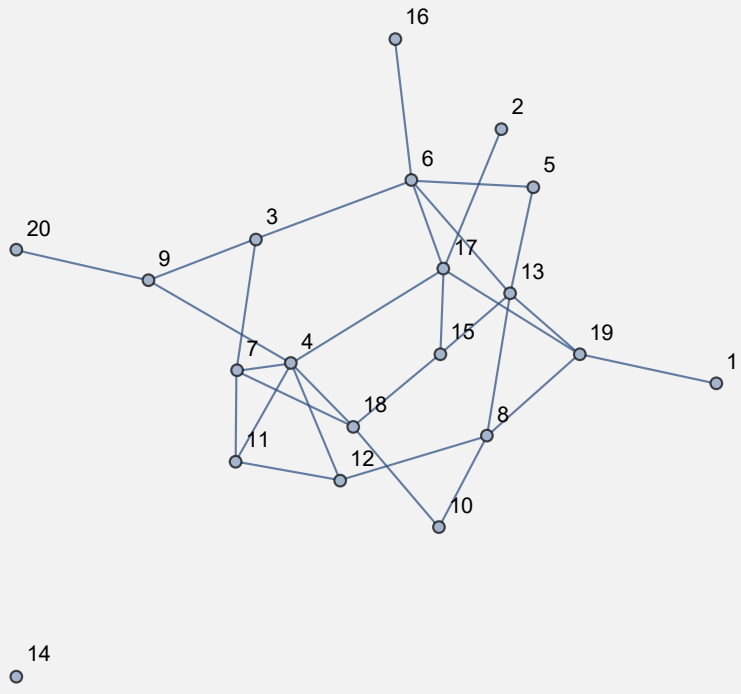
Solución:

En *Mathematica*:

In[ ] :=

```
grafo = GrafoRandom[20, 30]
CantRutas[grafo, 2, 4]
CantRutas[grafo, 2, 4, rutas -> True]
```

Out[ ] :=



218

```
{218, {{{2, 17}, {17, 4}}, {{2, 17}, {17, 15}, {15, 18}, {18, 4}}, ··· 215 ···, {{2, 17}, {17, 6}, {6, 5},
   {5, 13}, {13, 15}, {15, 18}, {18, 10}, {10, 8}, {8, 12}, {12, 11}, {11, 7}, {7, 3}, {3, 9}, {9, 4}}}}
```

salida grande | **Mostrar menos** | **Mostrar más** | **Mostrar salida completa** | **Establecer límite de tamaño...**

Las rutas en detalle no las arroja el software de manera explícita por sus dimensiones. El usuario puede presionar para ello, el botón "Mostrar salida completa".

Ejemplo 7.72

Encuentre la cantidad de rutas y los caminos sobre un grafo completo de orden diez, del nodo 1 al 5. Halle además, mediante el uso de *Mathematica*, una fórmula para calcular la cantidad de rutas entre dos nodos distintos en un grafo completo de orden "n".

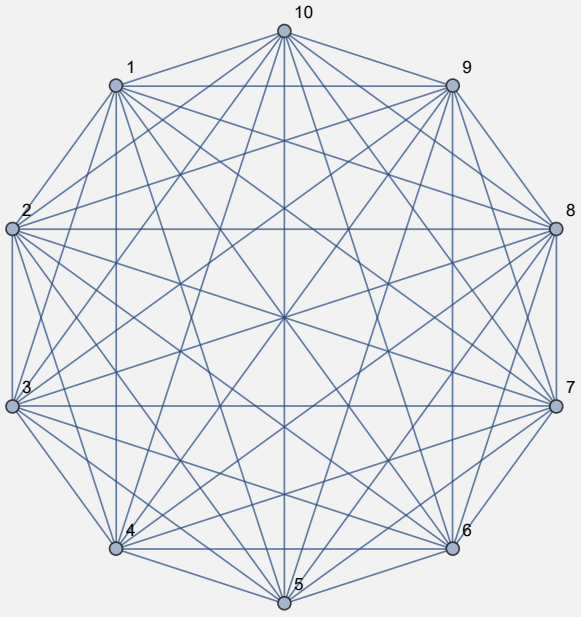
Solución:

En el software:

In[ ] :=

```
grafo = System`CompleteGraph[10, VertexLabels -> "Name",
ImagePadding -> 10]
CantRutas[grafo, 1, 5]
CantRutas[grafo, 1, 5, rutas -> True]
```

Out[ ] :=



109601

```
{109 601, {{{1, 5}}, {{1, 10}, {10, 5}}, {{1, 9}, {9, 5}}, ··· 109 596 ···,
  {{1, 2}, {2, 3}, {3, 4}, {4, 6}, {6, 7}, {7, 8}, {8, 10}, {10, 9}, {9, 5}},
  {{1, 2}, {2, 3}, {3, 4}, {4, 6}, {6, 7}, {7, 8}, {8, 9}, {9, 10}, {10, 5}}}}
```

salida grande | **Mostrar menos** | **Mostrar más** | **Mostrar salida completa** | **Establecer límite de tamaño...**

Para hallar la fórmula solicitada en este ejemplo, se puede proceder como sigue:

In[ ] :=
```
Table[CantRutas[System`CompleteGraph[i], 1, 2], {i, 2, 10}]
FindSequenceFunction[Table[CantRutas[System`CompleteGraph[i], 1, 2],
{i, 2, 10}], n]
```

Out[ ] :=
{1, 2, 5, 16, 65, 326, 1957, 13700, 109601}
e Gamma[n, 1]

N **`FindSequenceFunction`** toma la sucesión de cantidad de caminos de nueve grafos completos del orden dos al diez y mediante estos datos, determina una fórmula para generar la secuencia, es decir: *e* Gamma[n, 1]. **`Gamma[n, 1]`** es un comando de *Mathematica* que representa la función *gamma de Euler:* $\Gamma(n,1)$.

Explicación en video

37) **CantMRutas**: recibe un **grafo simple** "G" devolviendo una **matriz** donde cada una de sus entradas **cuenta** la **cantidad máxima** de **rutas de longitud** "k" (con **repetición de aristas**), entre **cualquier par** de vértices sobre "G". El arreglo bidimensional es el resultado de **elevar** a la "k" una **matriz de adyacencia** del grafo. La instrucción facilita la **propiedad** "**`all -> True`**" que retorna una **animación** con todas las potencias de la matriz de adyacencia **comenzando** con la **potencia uno** y hasta la **potencia** "k".

Sintaxis: **`CantMRutas[G, k]`**, o bien, **`CantMRutas[G, k, all -> True]`**. El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o, a través del **paquete** "Combinatorica".

Ejemplo 7.73

Considere el grafo octaedro, halle una matriz que muestre la cantidad máxima de trayectorias de longitud diez entre dos nodos cualesquiera. Además, genere una animación con todas las matrices desde la potencia uno hasta la diez.

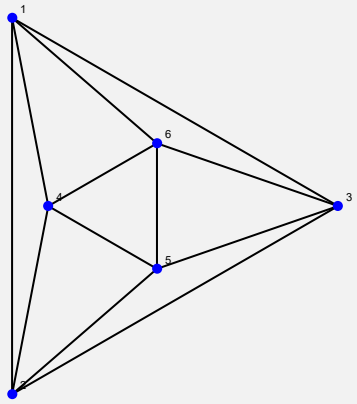
Solución:

Al emplear la instrucción **`CantMRutas`**:

In[ ] :=
```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[OctahedralGraph, VertexColor -> Blue,
EdgeColor -> Black], VertexLabel -> True, PlotRange -> 0.1]
CantMRutas[grafo, 10]
```

```
CantMRutas[grafo, 10, all -> True]
```

Out[ ] :=



	1	2	3	4	5	6
1	175 104	174 592	174 592	174 592	175 104	174 592
2	174 592	175 104	174 592	174 592	174 592	175 104
3	174 592	174 592	175 104	175 104	174 592	174 592
4	174 592	174 592	175 104	175 104	174 592	174 592
5	175 104	174 592	174 592	174 592	175 104	174 592
6	174 592	175 104	174 592	174 592	174 592	175 104

k

7

	1	2	3	4	5	6
1	2688	2752	2752	2752	2688	2752
2	2752	2688	2752	2752	2752	2688
3	2752	2752	2688	2688	2752	2752
4	2752	2752	2688	2688	2752	2752
5	2688	2752	2752	2752	2688	2752
6	2752	2688	2752	2752	2752	2688

Ejemplo 7.74

Sea un grafo con: aristas = {{1, 2}, {1, 98}, {2, 3}, {2, 51}, {3, 4}, {3, 28}, {3, 50}, {4, 5}, {4, 17}, {4, 27}, {5, 6}, {5, 12}, {5, 16}, {6, 11}, {7, 8}, {7, 104}, {8, 9}, {8, 57}, {8, 103}, {9, 10}, {9, 34}, {9, 56}, {10, 11}, {10, 23}, {10, 33}, {11, 12}, {11, 22}, {13, 14}, {13, 110}, {14, 15}, {14, 63}, {14, 109}, {15, 16}, {15, 40}, {15, 62}, {16, 17}, {16, 39}, {17, 18}, {17, 24}, {18, 23}, {19, 20}, {19, 116}, {20, 21}, {20, 69}, {20, 115}, {21, 22}, {21, 46}, {21, 68}, {22, 23}, {22, 45}, {23, 24}, {25, 26}, {26, 27}, {26, 75}, {26, 121}, {27, 28}, {27, 74}, {28, 29}, {28, 41}, {29, 30}, {29, 36}, {29, 40}, {30, 35}, {31, 32}, {31, 128}, {32, 33}, {32, 81}, {32, 127}, {33, 34}, {33, 80}, {34, 35}, {34, 47}, {35, 36}, {35, 46}, {37, 38}, {37, 134}, {38, 39}, {38, 87}, {38, 133}, {39, 40}, {39, 86}, {40, 41}, {41, 42}, {41, 48}, {42, 47}, {43, 44}, {43, 140}, {44, 45}, {44, 93}, {44, 139}, {45, 46}, {45, 92}, {46, 47}, {47, 48}, {49, 50}, {49, 146}, {50, 51}, {50, 145}, {51, 52}, {51, 76}, {52, 53}, {52, 65}, {52, 75}, {53, 54}, {53, 60}, {53, 64}, {54, 59}, {55, 56}, {55, 152}, {56, 57}, {56, 151}, {57, 58}, {57, 82}, {58, 59}, {58, 71}, {58, 81}, {59, 60}}. Construya el grafo a través del uso del paquete “Combinatorica”, encuentre una matriz que muestre la cantidad máxima

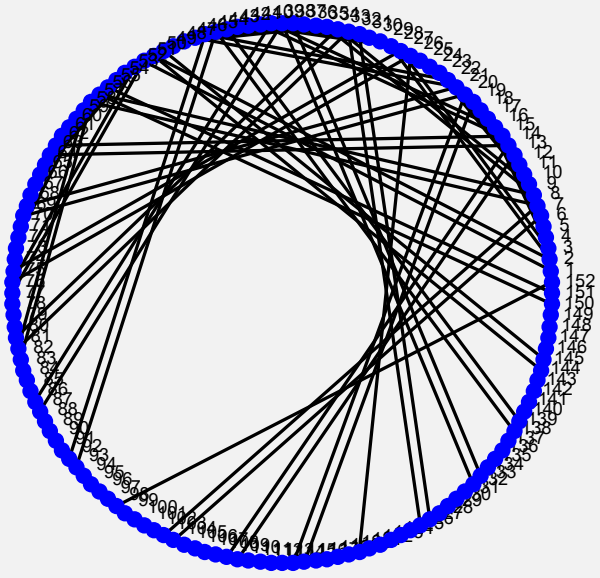
de caminos de longitud diez entre dos vértices cualesquiera. Además, despliegue una animación con todas las matrices desde la potencia uno hasta la diez.

Solución:

In[ ] :=

```
aristas = {{1, 2}, {1, 98}, {2, 3}, {2, 51}, {3, 4}, {3, 28}, {3, 50},
{4, 5}, {4, 17}, {4, 27}, {5, 6}, {5, 12}, {5, 16}, {6, 11}, {7, 8}, {7,
104}, {8, 9}, {8, 57}, {8, 103}, {9, 10}, {9, 34}, {9, 56}, {10, 11},
{10, 23}, {10, 33}, {11, 12}, {11, 22}, {13, 14}, {13, 110}, {14, 15},
{14, 63}, {14, 109}, {15, 16}, {15, 40}, {15, 62}, {16, 17}, {16, 39},
{17, 18}, {17, 24}, {18, 23}, {19, 20}, {19, 116}, {20, 21}, {20, 69},
{20, 115}, {21, 22}, {21, 46}, {21, 68}, {22, 23}, {22, 45}, {23, 24},
{25, 26}, {26, 27}, {26, 75}, {26, 121}, {27, 28}, {27, 74}, {28, 29},
{28, 41}, {29, 30}, {29, 36}, {29, 40}, {30, 35}, {31, 32}, {31, 128},
{32, 33}, {32, 81}, {32, 127}, {33, 34}, {33, 80}, {34, 35}, {34, 47},
{35, 36}, {35, 46}, {37, 38}, {37, 134}, {38, 39}, {38, 87}, {38, 133},
{39, 40}, {39, 86}, {40, 41}, {41, 42}, {41, 48}, {42, 47}, {43, 44},
{43, 140}, {44, 45}, {44, 93}, {44, 139}, {45, 46}, {45, 92}, {46, 47},
{47, 48}, {49, 50}, {49, 146}, {50, 51}, {50, 145}, {51, 52}, {51, 76},
{52, 53}, {52, 65}, {52, 75}, {53, 54}, {53, 60}, {53, 64}, {54, 59},
{55, 56}, {55, 152}, {56, 57}, {56, 151}, {57, 58}, {57, 82}, {58, 59},
{58, 71}, {58, 81}, {59, 60}};
GrafoC[aristas]
CantMRutas[G, 10]
CantMRutas[G, 10, all -> True]
```

Out[ ] :=



··· 1 ···

salida grande | Mostrar menos | Mostrar más | Mostrar salida completa | Establecer límite de tamaño...

k

	1	2	3	4	5	6	7	8	9	10	11
1	0	1	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0	0	0
4	0	0	1	0	1	0	0	0	0	0	0
5	0	0	0	1	0	1	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	1
7	0	0	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	1	0	1	0	0
9	0	0	0	0	0	0	0	1	0	1	0
10	0	0	0	0	0	0	0	0	1	0	1
11	0	0	0	0	0	1	0	0	0	1	0
12	0	0	0	0	1	0	0	0	0	0	1
13	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0

El *Out* [ ] no aparece completo pues cada matriz es de tamaño 152×152.

Explicación en video

38) **GrafoConexoQ**: **retorna** "`True`" si al recibir un **grafo** "G" como parámetro éste es **conexo** o "`False`", en caso contrario. El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, mediante el uso del **paquete** "Combinatorica".

Sintaxis: `GrafoConexoQ[G]`.

Ejemplo 7.75

Establezca con ayuda de software si el grafo `DeBruijnGraph[2, 5]` es o no conexo.

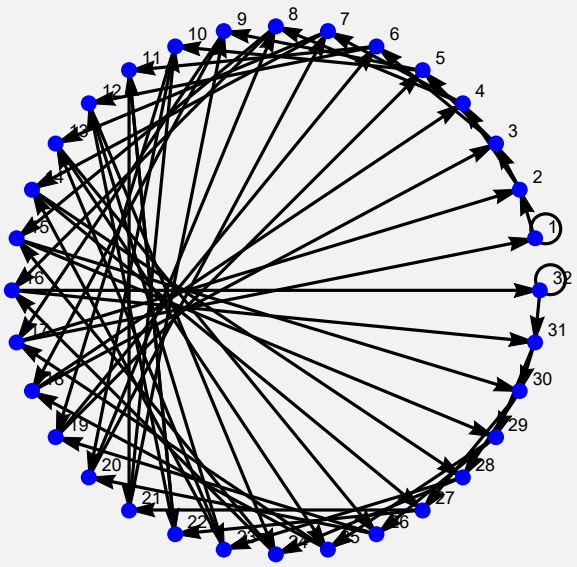
Solución:

Al utilizar el paquete "Combinatorica" y el comando `GrafoConexoQ` se obtiene:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DeBruijnGraph[2, 5],
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
GrafoConexoQ[grafo]
```

Out[ ] :=



True

Ejemplo 7.76

Emplee el comando **GrafoConexoQ** sobre un grafo seudoaleatorio de orden 10×10, creado en el "Wolfram System" y por medio del paquete "Combinatorica".

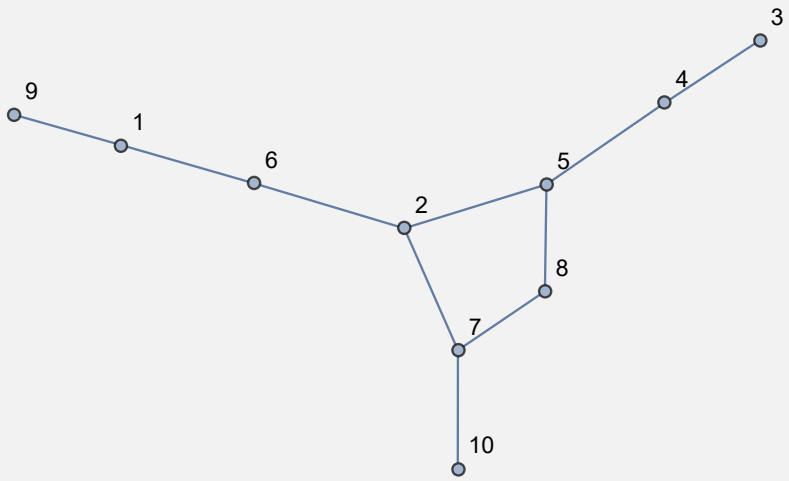
Solución:

En *Mathematica*:

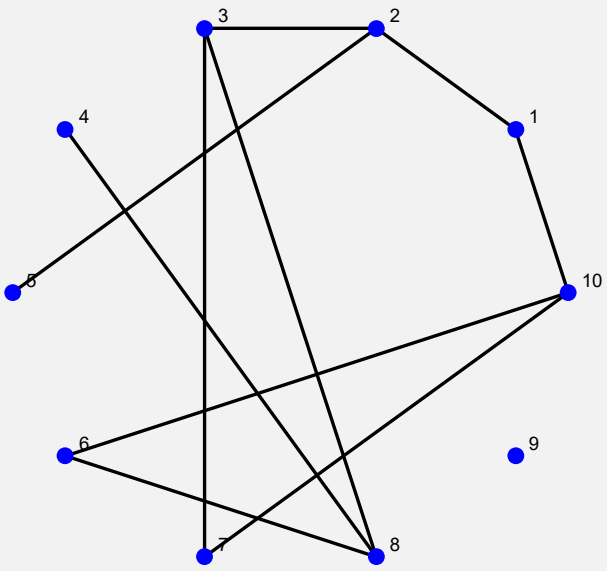
In[ ] :=

```
grafo = GrafoRandom[10, 10]
GrafoConexoQ[grafo]
GrafoRandom[10, 10, combinatorica -> True]
GrafoConexoQ[G]
```

Out[ ] :=



True



False

N Como la salida es seudoaleatoria, el alumno al correr el mismo código, podría conseguir grafos distintos y por consiguiente, otras respuestas lógicas.

Explicación en video

39) **GrafoSimpleQ**: **retorna** "`True`" si al recibir un **grafo** "G" como parámetro éste es **simple** (no tiene lazos ni aristas múltiples) o "`False`", en caso contrario. El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, mediante el uso del **paquete** "Combinatorica".

Sintaxis: `GrafoSimpleQ[G]`.

Ejemplo 7.77

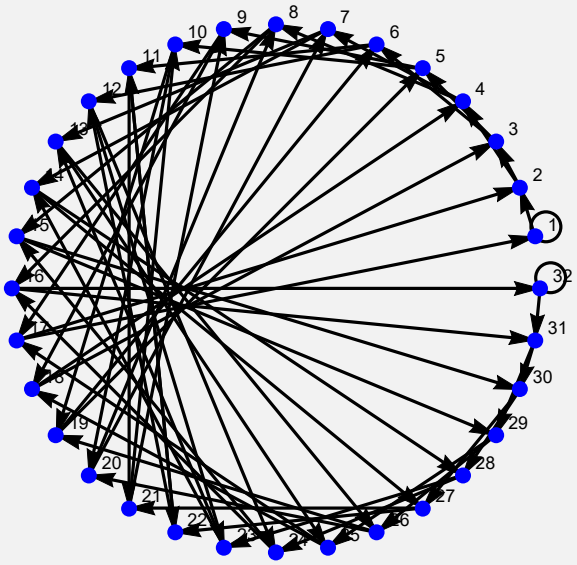
¿El grafo `DeBruijnGraph[2, 5]` es simple?

Solución:

```
In[ ]:=
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DeBruijnGraph[2, 5],
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
```

```
PlotRange -> 0.1]
GrafoSimpleQ[grafo]
```

Out[ ] :=



False

Se concluye que no es simple (al contener lazos).

Ejemplo 7.78

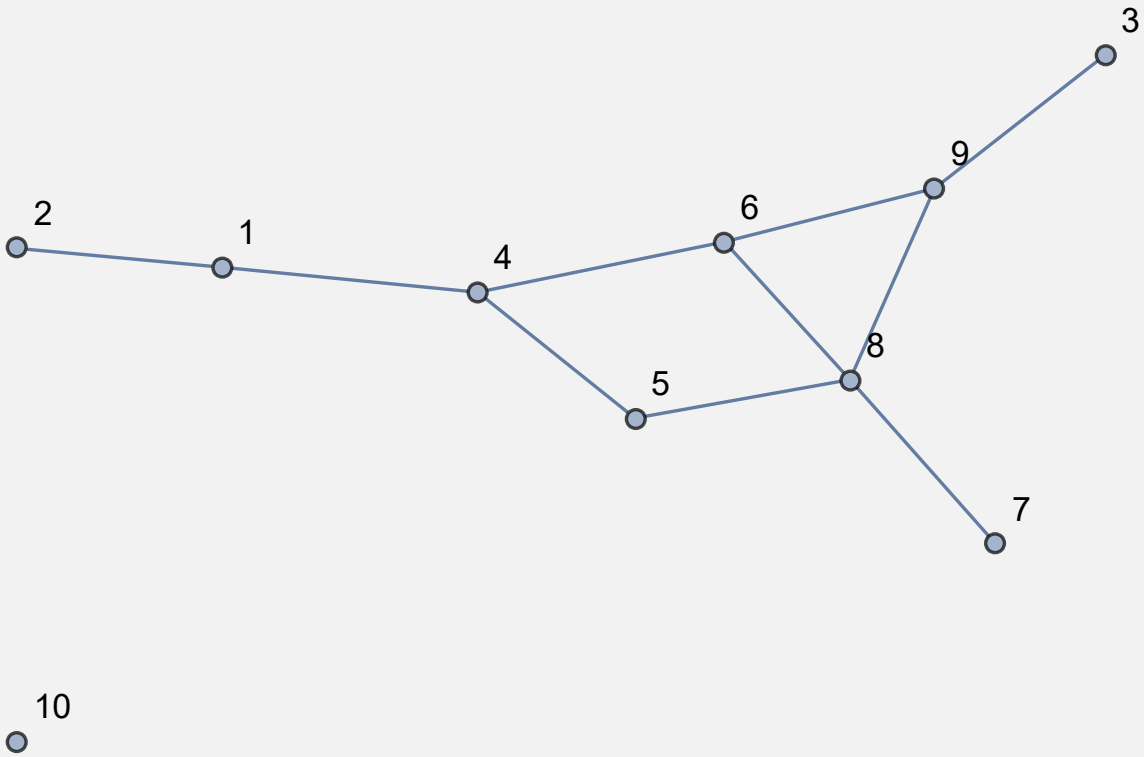
Utilice la instrucción **GrafoSimpleQ** sobre un grafo seudoaleatorio de orden 10×10, creado en el "Wolfram System" y a través del paquete "Combinatorica".

Solución:

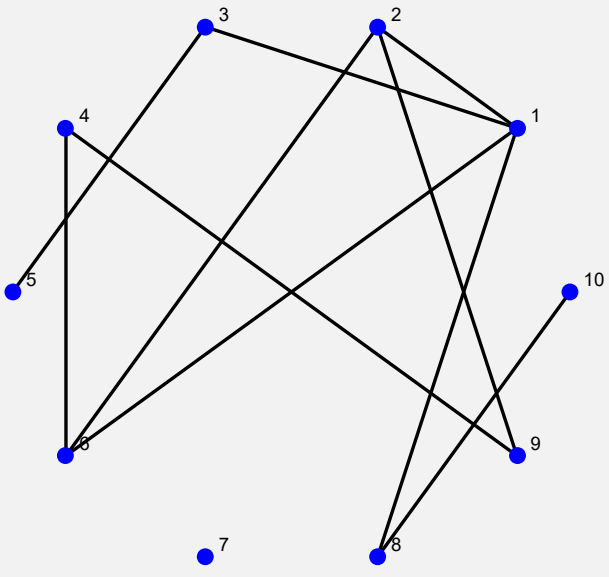
In[ ] :=

```
grafo = GrafoRandom[10, 10]
GrafoSimpleQ[grafo]
GrafoRandom[10, 10, combinatorica -> True]
GrafoSimpleQ[G]
```

Out[ ] :=



True



True

Ⓝ A pesar de la salida seudoaleatoria, el estudiante debe tener claro que **GrafoRandom** siempre genera un grafo simple, bajo la excepción de incluir la opción "**simple->False**".

Explicación en video

40) **GrafoPonderadoQ**: **retorna** "**True**" si al recibir un **grafo** "G" como parámetro éste es **ponderado** (todas las aristas tienen **pesos asignados**) o "**False**", en caso contrario. El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, mediante el uso del **paquete** "Combinatorica".

Sintaxis: `GrafoPonderadoQ[G]`.

Ejemplo 7.79

Verifique mediante **GrafoPonderadoQ**, que un grafo cuyas aristas vienen dadas por la relación binaria: $aRb \Leftrightarrow a-b \geq 2$, definida en el conjunto $A=\{1,2,3,\ldots,30\}$, sin y con pesos seudoaleatorios reales de uno a diez es no ponderado y ponderado, respectivamente.

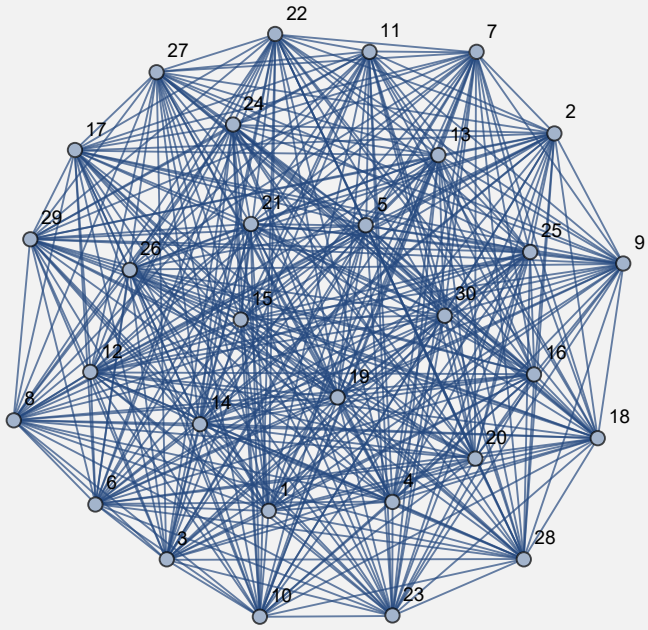
Solución:

En el software:

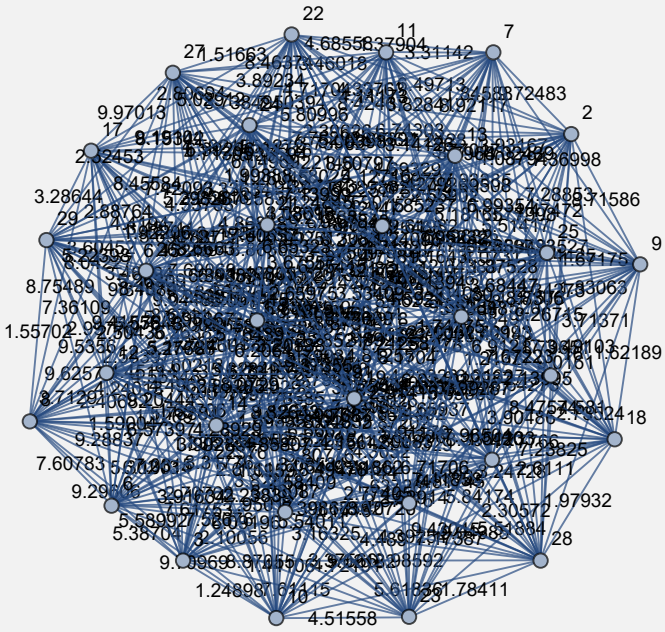
In[ ] :=

```
A = Table[i, {i, 1, 30}];
aristas = RelBin["a-b>=2", A, A];
grafo = Grafo[aristas]
GrafoPonderadoQ[grafo]
grafo = Grafo[aristas, pesos->RandomReal[{1, 10}, Length[aristas]],
mostrarpesos->True]
GrafoPonderadoQ[grafo]
```

Out[ ] :=



False



True

Ejemplo 7.80

Construya en "Combinatorica" un grafo cuyas aristas corresponden a $A \times A$, con $A = \{1,2,3,4,5\}$ ¿Es el grafo ponderado? Asigne pesos seudoaleatorios reales de uno a diez y verifique a través del comando **`GrafoPonderadoQ`**, que el grafo obtenido tiene pesos.

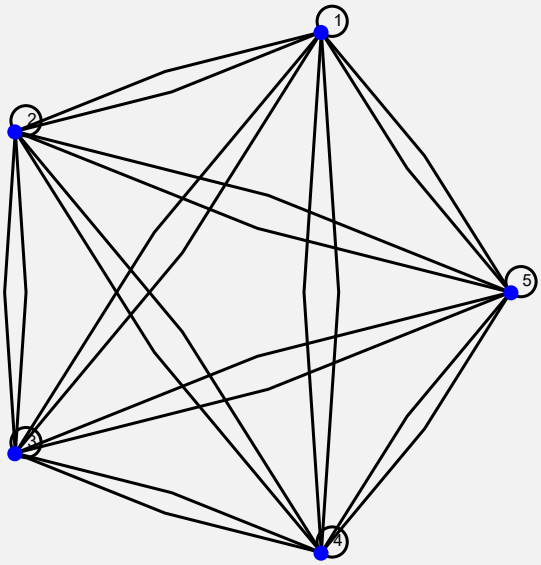
Solución:

En *Mathematica*:

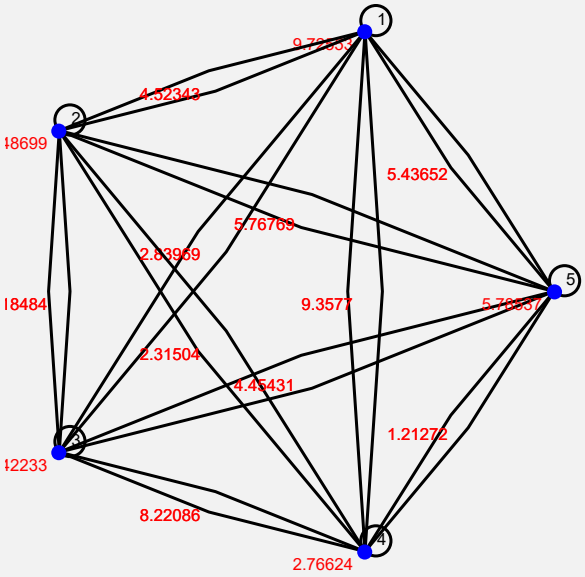
In[ ] :=

```
A = {1, 2, 3, 4, 5};
aristas = PC[A, A];
GrafoC[aristas]
GrafoPonderadoQ[G]
GrafoC[aristas, pesos -> RandomReal[{1, 10}, Length[aristas]],
mostrarpesos -> True]
GrafoPonderadoQ[G]
```

Out[ ] :=



True



True

Ⓝ Se observa que el primer grafo es considerado ponderado pese a no tener asignados pesos en sus lados, esto ocurre pues para el software *Mathematica*, cualquier grafo creado con el paquete "Combinatorica" es ponderado. Lo anterior no aplica en grafos construidos mediante el "Wolfram System".

Explicación en video

41) **GrafoDirigidoQ**: **retorna** "`True`" si al recibir un **grafo** "G" como parámetro éste es **dirigido** o "`False`", en caso contrario. El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, mediante el uso del **paquete** "Combinatorica".

Sintaxis: `GrafoDirigidoQ[G]`.

Ejemplo 7.81

Con ayuda de software responda: ¿es el grafo `DeBruijnGraph[2, 5]` dirigido?

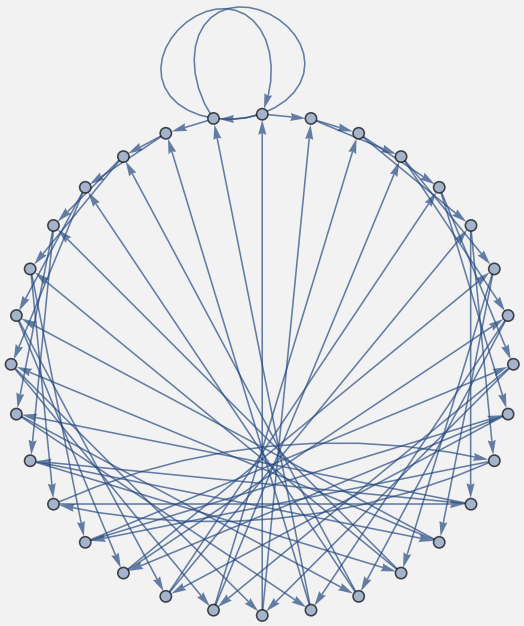
Solución:

En el "Wolfram System" de *Mathematica*, `DeBruijnGraph` corre sin ningún problema, es decir, para crear el grafo `DeBruijnGraph[2, 5]` no es necesario abrir el paquete "Combinatorica". En este ejemplo, se procederá de esta forma:

In[ ] :=

```
grafo = System`DeBruijnGraph[2, 5]
GrafoDirigidoQ[grafo]
```

Out[ ] :=



True

Ejemplo 7.82

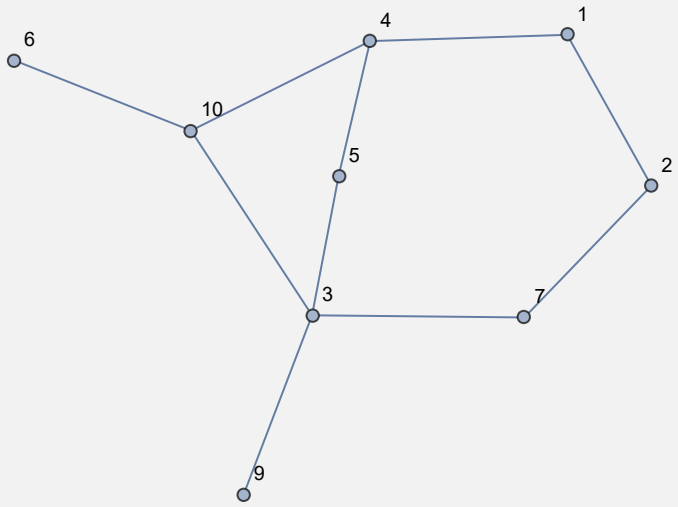
Verifique que un grafo construido al invocar **`GrafoRandom[10, 10]`**, con o sin "Combinatorica" no es dirigido.

Solución:

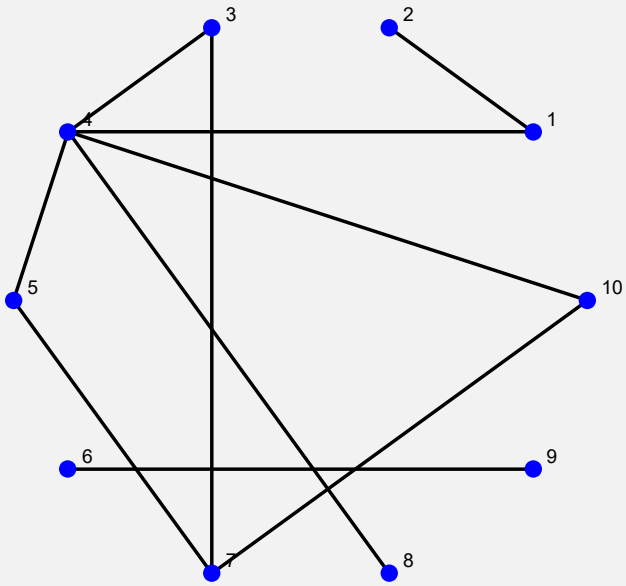
In[ ] :=

```
grafo = GrafoRandom[10, 10]
GrafoDirigidoQ[grafo]
GrafoRandom[10, 10, combinatorica -> True]
GrafoDirigidoQ[G]
```

Out[ ] :=



False



False

N ¡Recuerde!: **`GrafoRandom`** no construye por defecto grafos dirigidos.

Explicación en video

42) **GrafoCentro**: **encuentra** la lista de **nodos** con **excentricidad mínima**, en un **grafo conexo** "G" recibido como parámetro (llamados **puntos centrales** pues su **excentricidad** es **igual** al **radio** del **grafo**). El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o bien, mediante el uso del **paquete** "Combinatorica". El comando proporciona al usuario la **opción** "`mostrar -> True`" que retorna además de la lista de vértices, el **subgrafo** que los **contiene**. En "Combinatorica" este subgrafo queda por defecto almacenado en una **variable denominada** "G".

Sintaxis: `GrafoCentro[G]`, o bien, `GrafoCentro[G, mostrar -> True]`.

Ejemplo 7.83

Determine los puntos centrales de un grafo creado con el paquete "Combinatorica" cuyos lados corresponden al conjunto $A \times A$, siendo $A = \{1,2,3,4,5\}$. Resuelva el mismo problema asignando pesos seudoaleatorios reales de uno a diez y construyendo otro grafo dirigido ponderado. En cada caso muestre el subgrafo que contiene los vértices centrales.

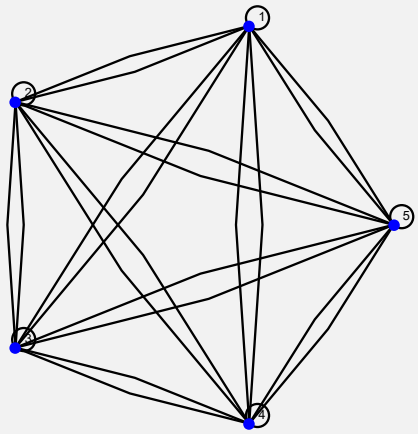
Solución:

En *Mathematica*:

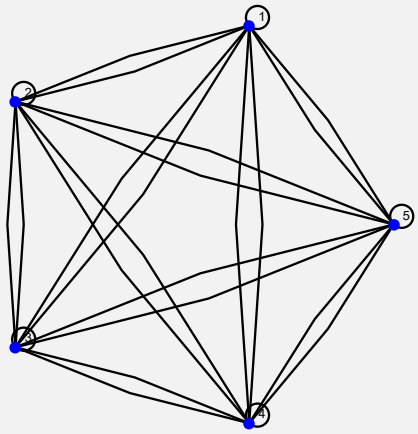
In[ ] :=

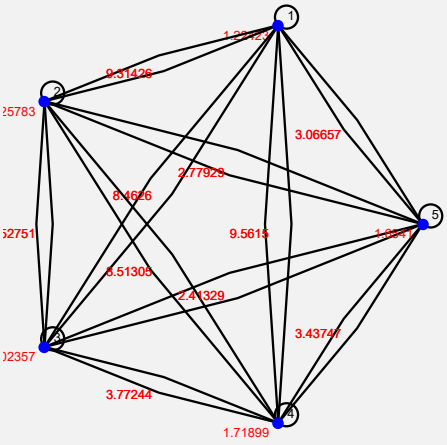
```
A = {1, 2, 3, 4, 5};
aristas = PC[A, A];
GrafoC[aristas]
G1 = G;
GrafoCentro[G1]
GrafoCentro[G1, mostrar -> True]
GrafoC[aristas, pesos -> RandomReal[{1, 10}, Length[aristas]],
mostrarpesos -> True]
G2 = G;
GrafoCentro[G2]
GrafoCentro[G2, mostrar -> True]
GrafoC[aristas, dirigido -> True, pesos -> RandomReal[{1, 10},
Length[aristas]], mostrarpesos -> True]
G3 = G;
GrafoCentro[G3]
GrafoCentro[G3, mostrar -> True]
```

Out[ ] :=

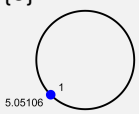


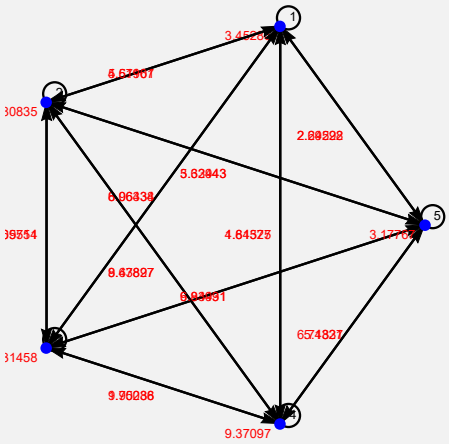
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5}





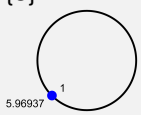
{5}
{5}





{3}

{3}



Ejemplo 7.84

Genere un grafo a través de **GrafoRandomConexo[20, 50]** y determine sus nodos centrales, además, del subgrafo que los contiene. Utilizando las aristas de **GrafoRandomConexo[20, 50]**, construya otro grafo con pesos seudoaleatorios reales de uno a diez en sus lados y para él, encuentre sus puntos centrales.

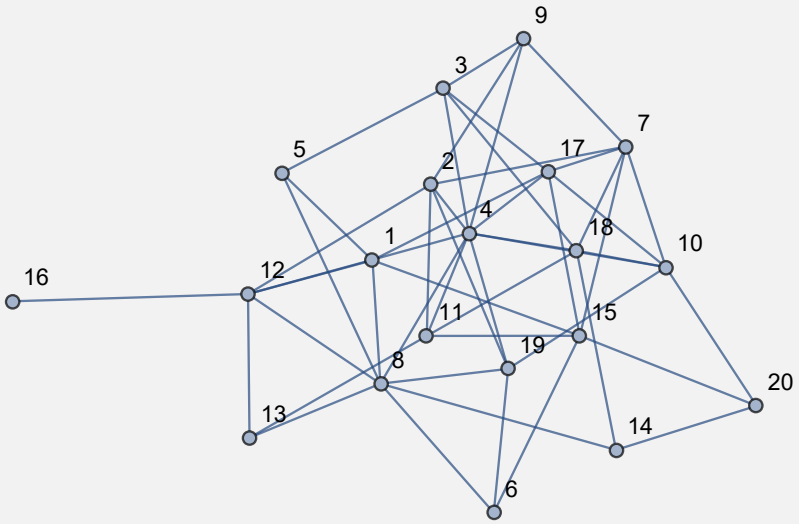
Solución:

En el software:

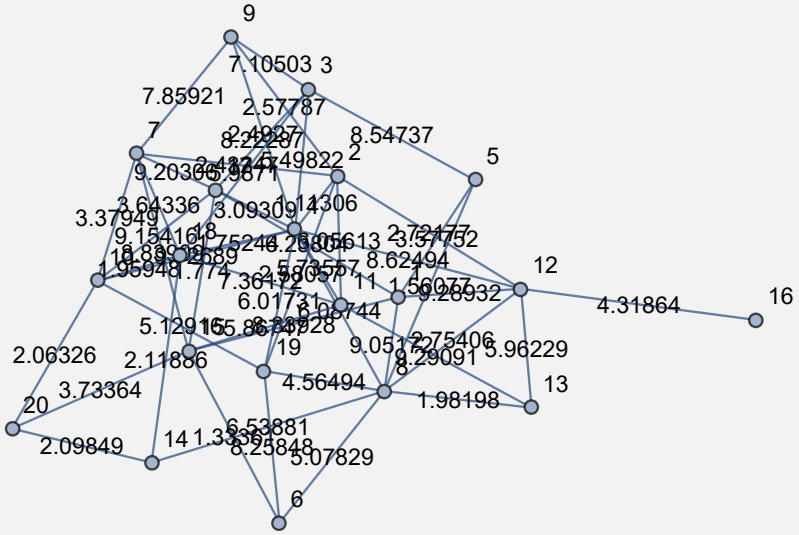
In[ ] :=

```
grafo = GrafoRandomConexo[20, 50]
GrafoCentro[grafo, mostrar -> True]
grafo = Grafo[AristasWolframSystemToCombinatorica[EdgeList[grafo]],
pesos -> RandomReal[{1, 10}, EdgeCount[grafo]], mostrarpesos -> True]
GrafoCentro[grafo]
```

Out[ ] :=



{4}



{4}

Ⓝ Se aclara al estudiante que los puntos centrales en un grafo ponderado y otro no ponderado con las mismas aristas, no necesariamente tienen los mismos vértices centrales como ha ocurrido en este ejemplo.

Explicación en video

43) **GrafoRadio**: **determina** el **valor mínimo** de **todas** las **excentricidades** en un **grafo conexo** "G" recibido como parámetro. El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o bien, mediante el uso del **paquete** "Combinatorica".

Sintaxis: `GrafoRadio[G]`.

Ejemplo 7.85

Halle el radio de un grafo construido con el paquete "Combinatorica" cuyos lados corresponden al conjunto $A \times A$, siendo $A = \{1,2,3,4,5\}$. Resuelva el mismo problema asignando pesos seudoaleatorios reales de uno a diez y generando otro grafo dirigido ponderado.

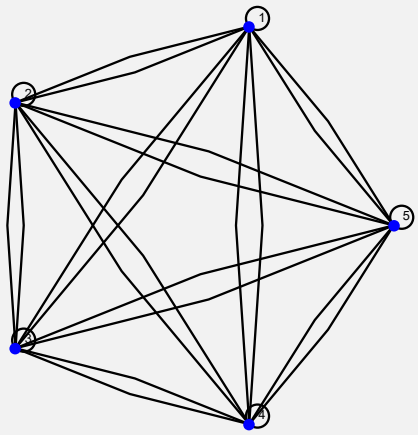
Solución:

Este ejemplo se resuelve de forma similar al tras anterior, pero empleando el comando **GrafoRadio**:

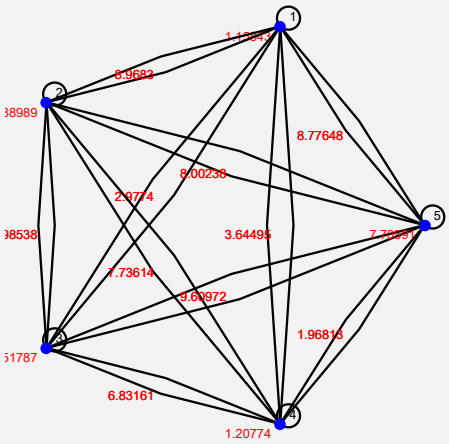
In[ ] :=

```
A = {1, 2, 3, 4, 5};
aristas = PC[A, A];
GrafoC[aristas]
GrafoRadio[G]
GrafoC[aristas, pesos -> RandomReal[{1, 10}, Length[aristas]],
mostrarpesos -> True]
GrafoRadio[G]
GrafoC[aristas, dirigido -> True, pesos -> RandomReal[{1, 10},
Length[aristas]], mostrarpesos -> True]
GrafoRadio[G]
```

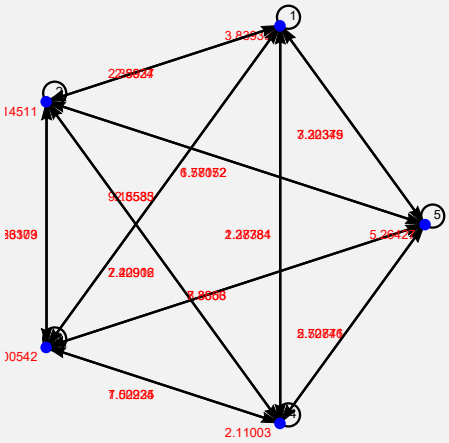
Out[ ] :=



1



7.73614



3.22379

Ejemplo 7.86

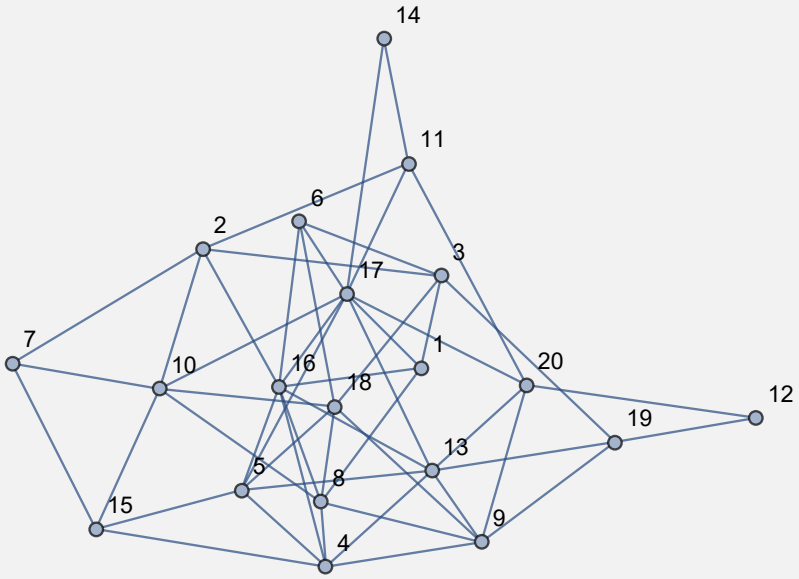
Considerando un grafo obtenido por **GrafoRandomConexo[20, 50]**, encuentre su radio. Empleando sus aristas construya otro grafo con pesos seudoaleatorios reales de uno a diez y determine su radio.

Solución:

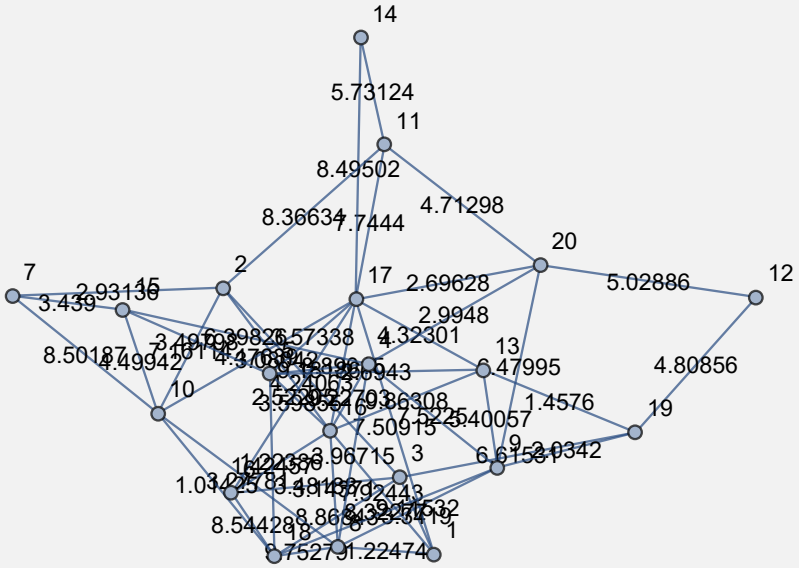
In[ ] :=

```
grafo = GrafoRandomConexo[20, 50]
GrafoRadio[grafo]
grafo = Grafo[AristasWolframSystemToCombinatorica[EdgeList[grafo]],
pesos -> RandomReal[{1, 10}, EdgeCount[grafo]], mostrarpesos -> True]
GrafoRadio[grafo]
```

Out[ ] :=



2



9.75769

> (N) Cuando el grafo es ponderado **`GrafoRadio`** calcula el radio en función de los pesos asignados a cada una de sus aristas.

Explicación en video

44) **GrafoDiametro: determina** el **valor máximo** de **todas** las **excentricidades** en un **grafo conexo** "G" recibido como parámetro, es decir, la **longitud** del **camino más corto** para **unir** los **dos vértices más alejados**. El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o bien, mediante el uso del **paquete** "Combinatorica".

Sintaxis: `GrafoDiametro[G]`.

Ejemplo 7.87

Calcule el diámetro de un grafo con aristas $A \times A$, $A = \{1,2,3,4,5\}$, ponderado y con pesos seudoaleatorios reales de uno a diez. Construya el grafo mediante el uso del paquete "Combinatorica".

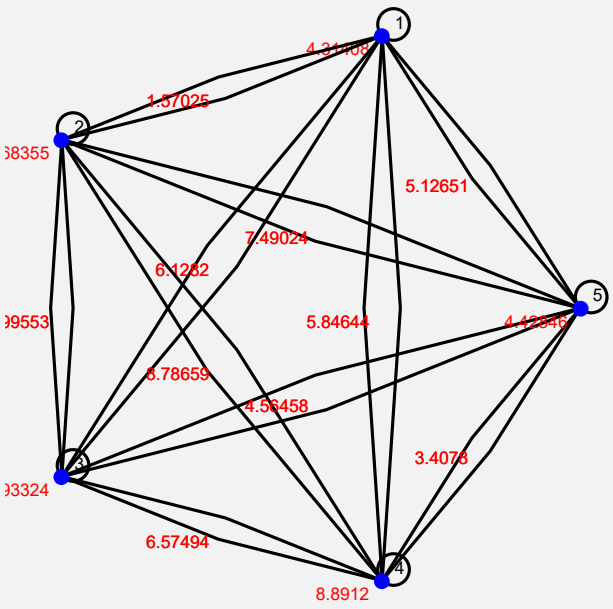
Solución:

En *Mathematica*:

In[ ] :=

```
A = {1, 2, 3, 4, 5};
aristas = PC[A, A];
GrafoC[aristas, pesos -> RandomReal[{1, 10}, Length[aristas]],
mostrarpesos -> True]
GrafoDiametro[G]
```

Out[ ] :=



7.69844

Ejemplo 7.88

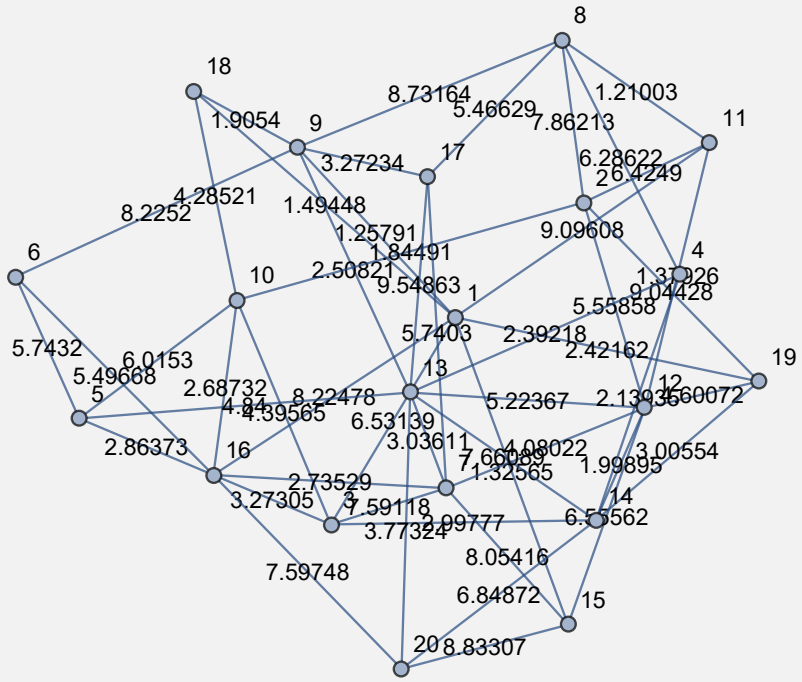
Halle el diámetro de un grafo ponderado con pesos seudoaleatorios reales de uno a diez, cuyos lados corresponden a las aristas de un grafo generado mediante la instrucción `GrafoRandomConexo[20, 50]`.

Solución:

In[ ] :=

```
grafo = GrafoRandomConexo[20, 50]
grafo = Grafo[AristasWolframSystemToCombinatorica[EdgeList[grafo]],
pesos -> RandomReal[{1, 10}, EdgeCount[grafo]], mostrarpesos -> True]
GrafoDiametro[grafo]
```

Out[ ] :=



14.9015

Explicación en video

45) **CRDGrafosSA**: **determina** el **centro**, el **radio** y el **diámetro** de un **máximo** de "n" **grafos conexos seudoaleatorios**. El valor de "n" se recibe como parámetro del comando. La instrucción tiene un objetivo didáctico con la intención de **comprender los conceptos** de los términos anteriormente señalados.

Sintaxis: `CRDGrafosSA[n]`, "n" debe ser un número **entero mayor o igual a cinco**. Dependiendo de este dato se pueden retornar **grafos repetidos**.

Ejemplo 7.89

Invoque la función **CRDGrafosSA** con $n = 5$.

Solución:

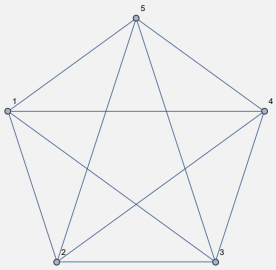
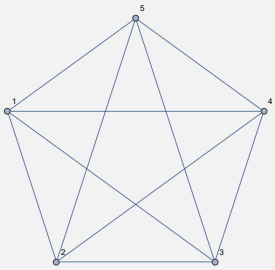
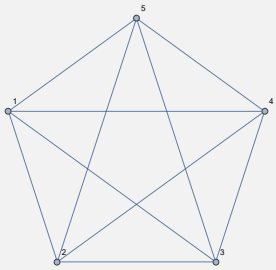
En el software:

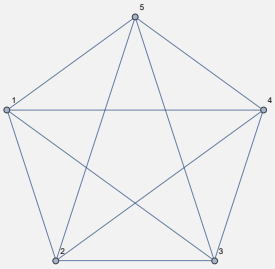
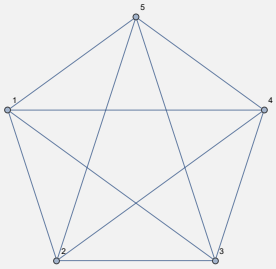
In[ ] :=

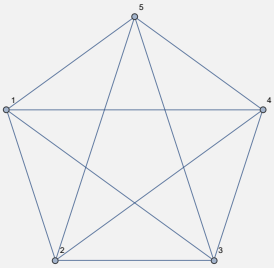
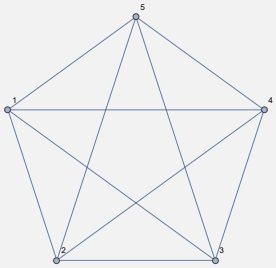
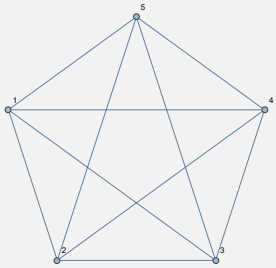
```
CRDGrafosSA[5]
```

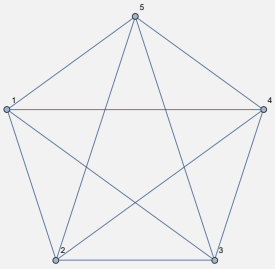
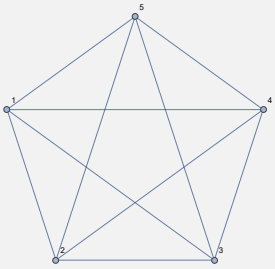
Out[ ] :=

Grafo, centro, subgrafo centro, radio y diámetro, respectivamente. Se obtuvieron 5 casos:

{{  , {1, 2, 3, 4, 5},  , 1, 1}, {  ,

{1, 2, 3, 4, 5},  , 1, 1}, {  , {1, 2, 3, 4, 5},

 , 1, 1}, {  , {1, 2, 3, 4, 5},  , 1,

1}, {  , {1, 2, 3, 4, 5},  , 1, 1}}

N En este ejercicio, **CRDGrafosSA[5]** a retornado grafos conexos seudoaleatorios iguales.

Ejemplo 7.90

Halle diez grafos conexos seudoaleatorios y determine sus puntos centrales, el subgrafo que los contiene, su radio y diámetro, respectivamente.

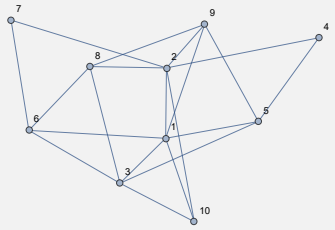
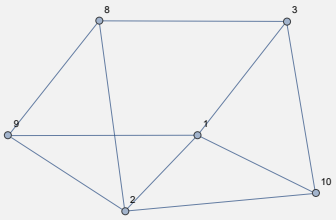
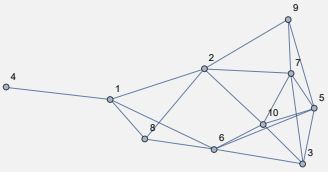
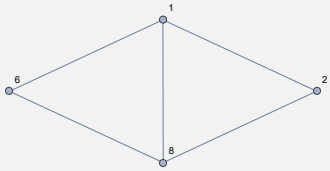
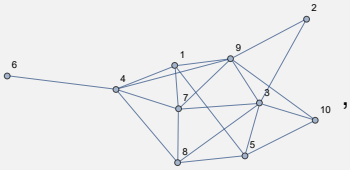
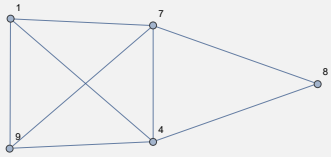
Solución:

Al correr **`CRDGrafosSA[10]`**:

In[ ] :=

```
CRDGrafosSA[10]
```

Out[ ] :=

{{  , {1, 2, 3, 8, 9, 10},  , 2, 3}, {  , {1, 2, 6, 8},  , 2, 3}, ..., {  , {1, 4, 7, 8, 9},  , 2, 3}}

Se omite el *Out* [ ] completo por su tamaño.

Explicación en video

46) **Excentricidad**: **encuentra** la **excentricidad** de **cada uno** de los **vértices** en un **grafo conexo** "G" recibido como parámetro, es decir, para cada nodo "v" de "G", **calcula** el **valor máximo** de las **longitudes mínimas** de los **caminos** desde "v" a los demás vértices de "G". El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o bien, mediante el uso del **paquete** "Combinatorica".

Sintaxis: **`Excentricidad[G]`**.

Ejemplo 7.91

Encuentre la excentricidad sobre cada uno de los nodos de un grafo construido con "Combinatorica", cuyas aristas corresponden al conjunto $A \times A$, con $A = \{1,2,3,4,5\}$, dirigido y con pesos seudoaleatorios reales de uno a diez.

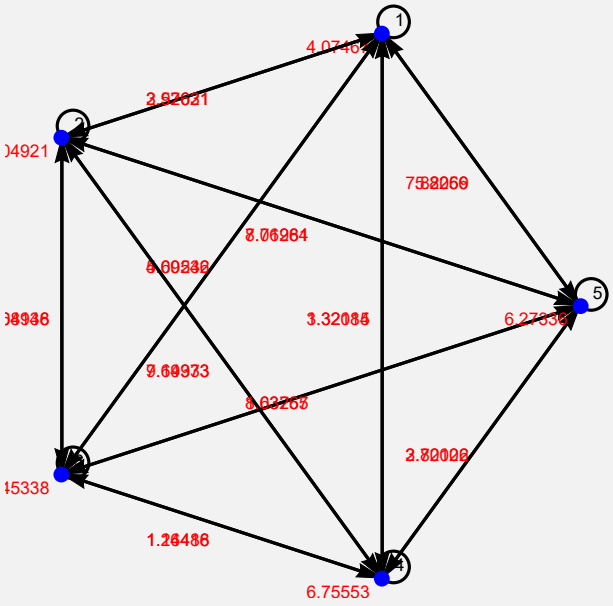
Solución:

En *Mathematica*:

In[ ] :=

```
A = {1, 2, 3, 4, 5};
aristas = PC[A, A];
GrafoC[aristas, dirigido -> True, pesos -> RandomReal[{1, 10},
Length[aristas]], mostrarpesos -> True]
Excentricidad[G]
```

Out[ ] :=



	1	2	3	4	5
Excentricidad	7.12221	6.0473	4.96592	5.29115	6.28213

Ejemplo 7.92

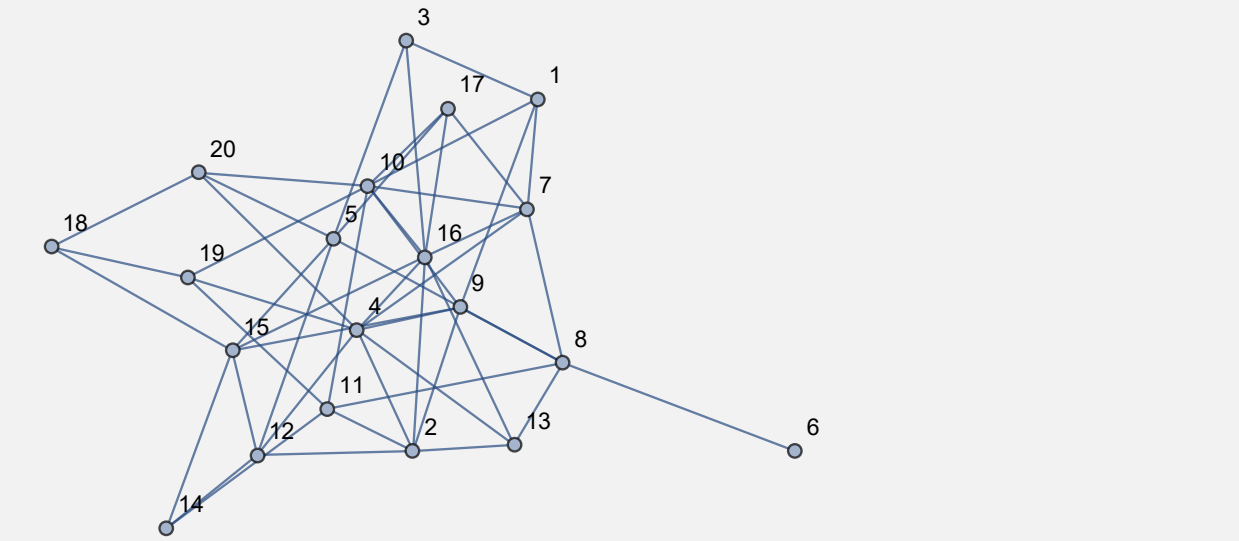
Ejecute el comando **Excentricidad** sobre un grafo devuelto por **GrafoRandomConexo[20, 50]**.

Solución:

In[ ] :=

```
grafo = GrafoRandomConexo[20, 50]
Excentricidad[grafo]
```

Out[ ] :=



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Excentricidad	3	3	3	3	3	4	3	3	2	3	3	3	3	3	3	3	3	4	3	3

Explicación en video

47) **GrafoCompleto**: **traza** un **grafo completo** con "n" nodos. Presenta la **opción** "**`analizar->True`**" que **muestra** una **animación** con **distintos grafos completos** hasta el orden "n", indicando además en cada caso: la **valencia** de los vértices, la **cantidad de aristas** y, si el grafo posee **circuitos de Euler** y de **Hamilton**.

Sintaxis: **`GrafoCompleto[n]`**, o bien, **`GrafoCompleto[n, analizar->True]`**.

Ejemplo 7.93

Construya un grafo completo de orden 20.

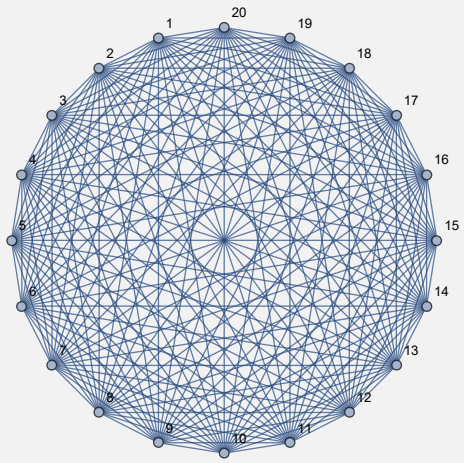
Solución:

En *Mathematica*:

In[ ] :=

```
GrafoCompleto[20]
```

Out[ ] :=



Ejemplo 7.94

Genere y analice los grafos completos desde el orden 1 hasta el 20.

Solución:

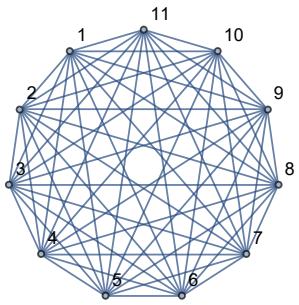
En este ejercicio se debe emplear la opción "**analizar->True**" del comando **GrafoCompleto**:

In[ ] :=

```
GrafoCompleto[20, analizar->True]
```

Out[ ] :=

Orden del grafo



	1	2	3	4	5	6	7	8	9	10	11
Grado o valencia	10	10	10	10	10	10	10	10	10	10	10

Cantidad de aristas: 55 = $5^1 \cdot 11^1$

Tiene circuitos de Euler: True

Tiene circuitos de Hamilton: True

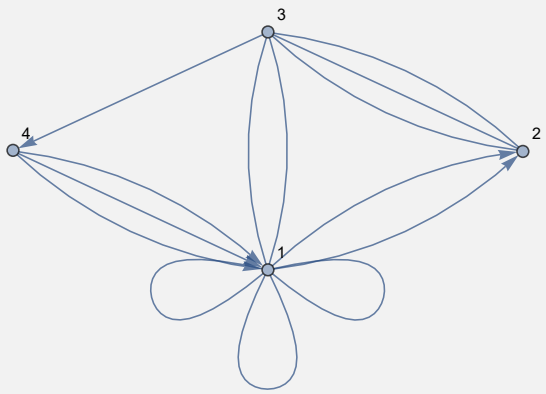
Explicación en video

48) **GrafoCompletoQ**: **determina** si un grafo "G" recibido como parámetro **es o no completo**, retornando "`True`" o "`False`" según corresponda. El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o a través del uso del **paquete** "Combinatorica".

Sintaxis: `GrafoCompletoQ[G]`.

Ejemplo 7.95

Determine mediante el uso de software si el siguiente grafo es completo.



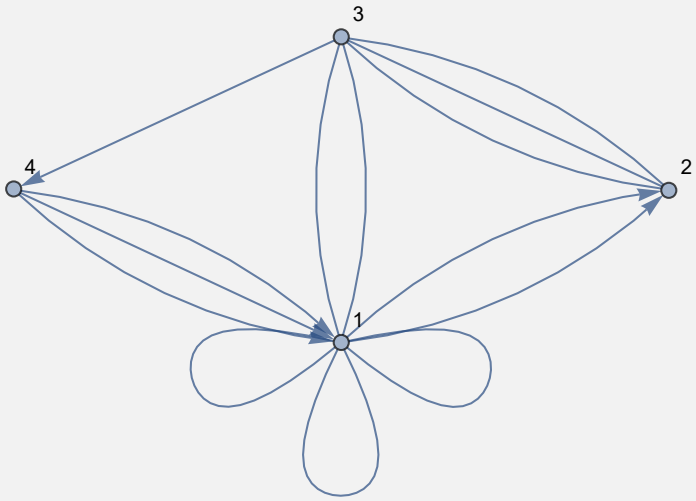
Solución:

Al emplear la instrucción **`GrafoCompletoQ`** se tiene:

In[ ] :=

```
grafo = System`Graph[{1 <-> 1, 1 <-> 1, 1 <-> 1, 1 -> 2, 1 -> 2, 2
<-> 3, 2 <-> 3, 2 <-> 3, 3 <-> 1, 3 <-> 1, 3 -> 4, 4 -> 1, 4 ->
1, 4 -> 1}, VertexLabels -> "Name", ImagePadding -> 10]
GrafoCompletoQ[grafo]
```

Out[ ] :=



False

N El grafo por consiguiente no es completo. En este ejemplo, se procedió a crear el grafo original mediante el comando **Graph** nativo de *Mathematica*, sin embargo, se pudo haber utilizado también, la instrucción **Grafo** de *VilCretas*.

Ejemplo 7.96

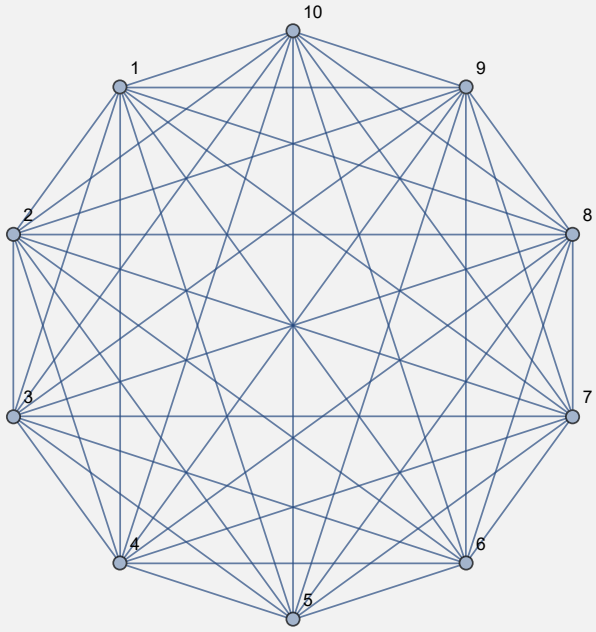
Verifique que lo retornado por **GrafoCompleto[10]** es un grafo completo ¿Qué ocurre con la salida de **GrafoCompleto[10, analizar -> True]** al utilizarla como argumento en **GrafoCompletoQ**?

Solución:

In[ ] :=

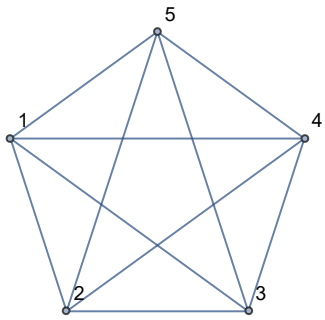
```
grafo = GrafoCompleto[10]
GrafoCompletoQ[grafo]
grafo = GrafoCompleto[10, analizar -> True]
GrafoCompletoQ[grafo]
```

Out[ ] :=



True

Orden del grafo



	1	2	3	4	5
Grado o valencia	4	4	4	4	4

Cantidad de aristas: $10 = 2^1 \cdot 5^1$
Tiene circuitos de Euler: True
Tiene circuitos de Hamilton: True

N Se observa en el ejemplo, cómo **GrafoCompletoQ** no ejecuta (devuelve un valor nulo) al procesar la animación retornada por **GrafoCompleto[10, analizar -> True]**. Esto ocurre pues la animación no es un grafo para el software *Mathematica*.

Explicación en video

49) **GrafoVacio**: **construye** un **grafo vacío** con "n" nodos. Por defecto el grafo queda almacenado en una **variable denominada** "G". La instrucción brinda la **opción** "**`analizar->True`**" que **muestra** una **animación** con **distintos grafos vacíos** hasta el orden "n".

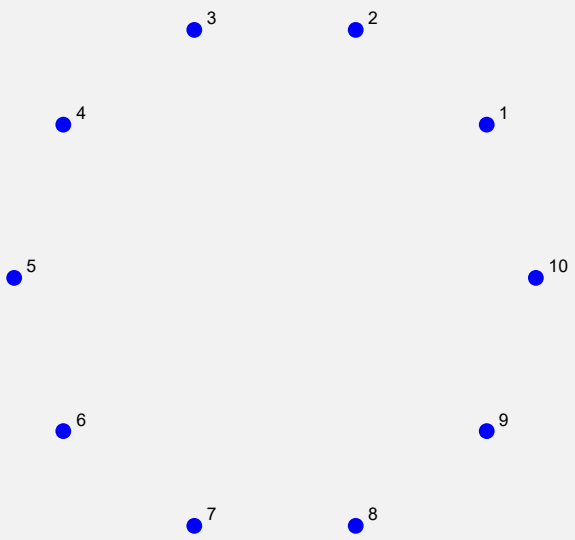
Sintaxis: **`GrafoVacio[n]`**, o bien, **`GrafoVacio[n, analizar->True]`**.

Ejemplo 7.97

Genere un grafo vacío con 10 vértices.

Solución:

In[ ] :=
```
GrafoVacio[10]
```
Out[ ] :=



Ejemplo 7.98

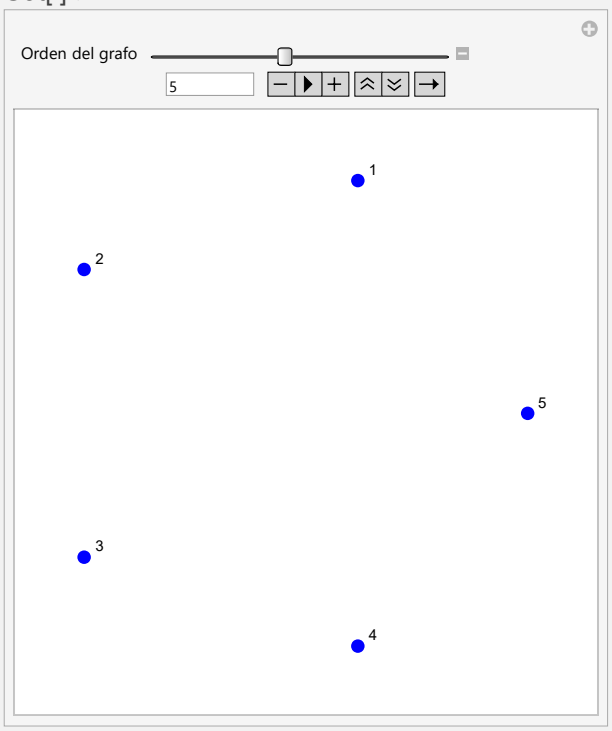
En una animación muestre todos los grafos vacíos desde el orden 1 hasta el 10.

Solución:

Para resolver lo solicitado se usa la opción "**`analizar->True`**":

In[ ] :=
```
GrafoVacio[10, analizar->True]
```

Out[ ] :=



Explicación en video

50) **GrafoBipartitoCompleto**: **traza** un **grafo bipartito completo** con "n" nodos izquierdos y "m" vértices derechos. Integra la **opción** "**`analizar -> True`**" que **muestra** una **animación** con **distintos grafos bipartitos completos** hasta el orden "n" por "m", indicando además en cada caso: la **valencia** de los nodos, la **cantidad de aristas** y, si el grafo posee **circuitos de Euler** y de **Hamilton**.

Sintaxis: **`GrafoBipartitoCompleto[n, m]`**, o bien, **`GrafoBipartitoCompleto[n, m, analizar -> True]`**.

Ejemplo 7.99

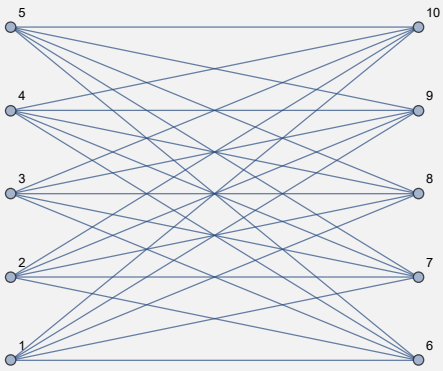
Construya un grafo bipartito completo de orden 5×5.

Solución:

In[ ] :=

```
GrafoBipartitoCompleto[5, 5]
```

Out[ ] :=



N Se dilucida al examinar el *Out* [ ], cómo el grafo bipartito completo retornado se creó en el "Wolfram System" de *Mathematica*. Se recuerda al lector que otra manera de obtener una salida similar empleando el paquete "Combinatorica", la constituye el uso de **`CompleteKPartiteGraph`**.

Ejemplo 7.100

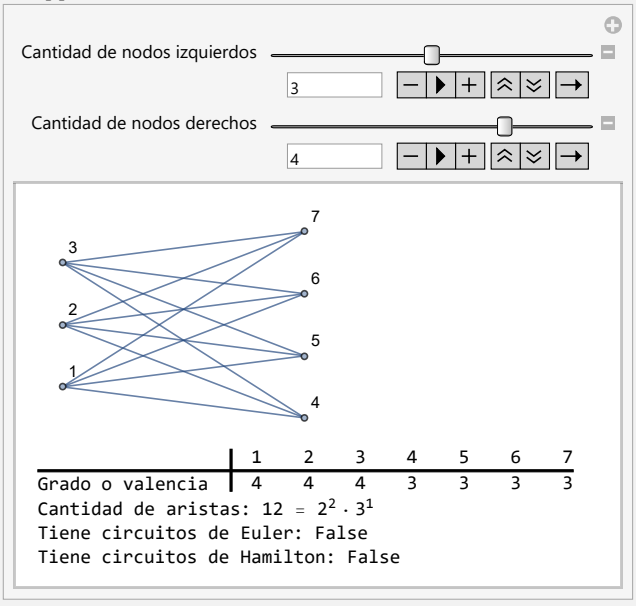
Analice el grafo del ejemplo anterior, a través del empleo de la opción "**`analizar -> True`**".

Solución:

In[ ] :=

```
GrafoBipartitoCompleto[5, 5, analizar -> True]
```

Out[ ] :=



Ⓝ "**analizar->True**" es una herramienta interesante que permite explorar características especiales en cierto tipo de grafos. A este respecto, se podría analizar si existe una fórmula que calcule las valencias sobre cada uno de los nodos, la cantidad de lados, o bien, se puedan garantizar condiciones necesarias y/o suficientes que determinen la existencia de circuitos de *Euler* o de *Hamilton*.

Explicación en video

51) **GrafoBipartitoCompletoQ**: **retorna** "**True**" si un **grafo** "G" recibido como parámetro es **bipartito completo** o "**False**" en caso contrario. El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o a través del uso del **paquete** "Combinatorica".

Sintaxis: **GrafoBipartitoCompletoQ[G]**. Si el comando devuelve "**True**" añade en un **vector**, el **orden** del grafo bipartito completo. **No acepta** grafos **dirigidos**.

Ejemplo 7.101

Determine si el grafo dodecaedro es bipartito completo.

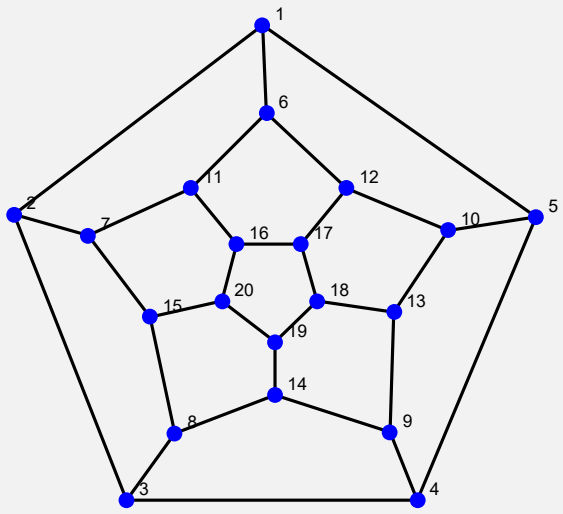
Solución:

En *Mathematica*:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
GrafoBipartitoCompletoQ[grafo]
```

Out[ ] :=



False

Ejemplo 7.102

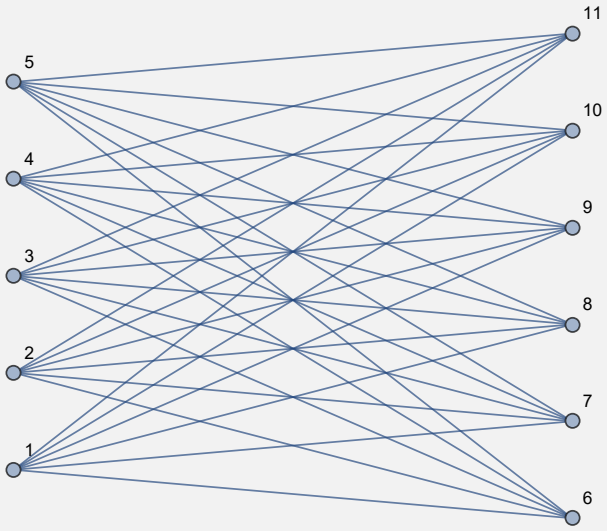
Verifique que **GrafoBipartitoCompletoQ** retorna "**True**" en **GrafoBipartito-Completo[5, 6]**.

Solución:

In[ ] :=

```
grafo = GrafoBipartitoCompleto[5, 6]
GrafoBipartitoCompletoQ[grafo]
```

Out[ ] :=



{True, {5, 6}}

N La salida confirma que el grafo es bipartito completo, pero además, muestra el orden correspondiente del grafo.

Explicación en video

52) **GrafoBipartitoQ**: **retorna** "`True`" si un **grafo** "G" recibido como parámetro es **bipartito, no necesariamente completo**, o "`False`" en caso contrario. El grafo pudo haber sido **creado** con o sin "Combinatorica".

Sintaxis: `GrafoBipartitoQ[G]`. **Acepta** grafos **dirigidos**.

Ejemplo 7.103

¿Es el grafo dodecaedro un grafo bipartito (no necesariamente completo)?

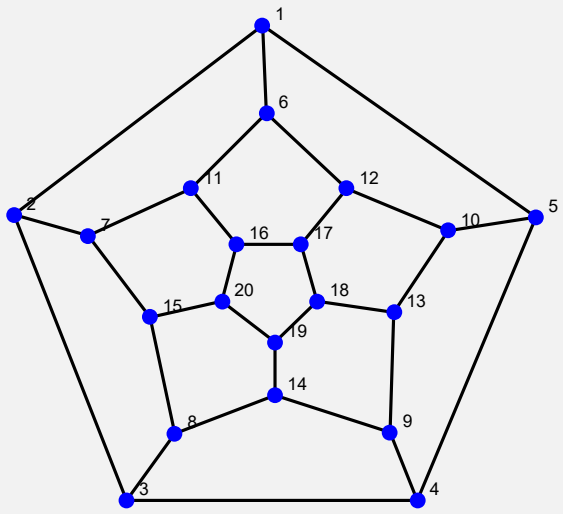
Solución:

Al recurrir al comando **`GrafoBipartitoQ`** se obtiene:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
GrafoBipartitoQ[grafo]
```

Out[ ] :=



False

Se concluye que no es bipartito.

Ejemplo 7.104

Considere un grafo con: aristas = {{1, 7}, {1, 8}, {1, 9}, {1, 10}, {1, 11}, {2, 6}, {2, 7}, {2, 8}, {2, 9}, {2, 10}, {2, 11}, {3, 6}, {3, 7}, {3, 8}, {3, 9}, {3, 10}, {3, 11}, {4, 6}, {4, 7}, {4, 8}, {4, 9}, {4, 10}, {4, 11}, {5, 6}, {5, 7}, {5, 8}, {5, 9}, {5, 10}, {5, 11}}. Determine si es bipartito.

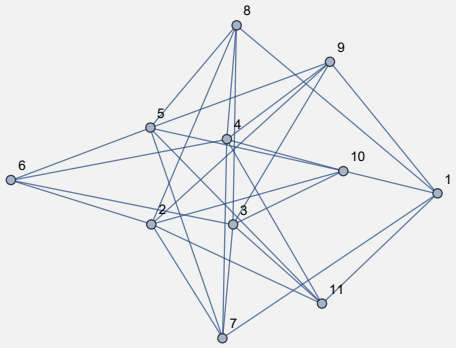
Solución:

En el software:

In[ ] :=

```
grafo = Grafo[{{1, 7}, {1, 8}, {1, 9}, {1, 10}, {1, 11}, {2, 6}, {2,
7}, {2, 8}, {2, 9}, {2, 10}, {2, 11}, {3, 6}, {3, 7}, {3, 8}, {3, 9},
{3, 10}, {3, 11}, {4, 6}, {4, 7}, {4, 8}, {4, 9}, {4, 10}, {4, 11},
{5, 6}, {5, 7}, {5, 8}, {5, 9}, {5, 10}, {5, 11}}]
GrafoBipartitoQ[grafo]
```

Out[ ] :=



True

N En este ejercicio la forma que toma el grafo no lo hace parecer bipartito, sin embargo, sí lo es, corresponde a **`GrafoBipartitoCompleto[5, 6]`** exceptuando la arista {1, 6}.

Explicación en video

53) **GrafoMariposa**: **crea** un **grafo mariposa** de orden "n". Integra la **opción** "**`analizar -> True`**" que **muestra** una **animación** con **distintos grafos mariposa** hasta el orden "n", indicando además en ca-

da caso: la **valencia** de los vértices, la **cantidad de aristas** y, si el grafo posee **circuitos de Euler** y de **Hamilton**.

Sintaxis: `GrafoMariposa[n]`, o bien, `GrafoMariposa[n, analizar -> True]`.

Ejemplo 7.105

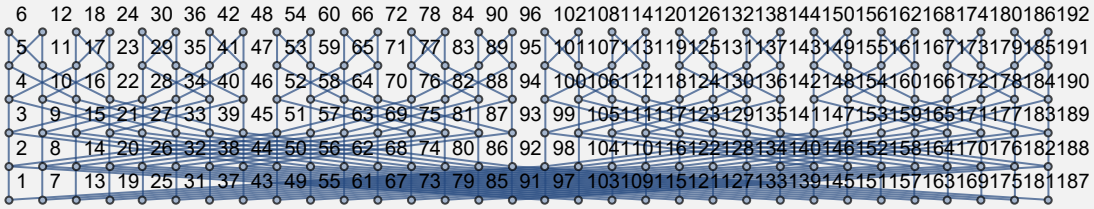
Genere un grafo mariposa de orden 5.

Solución:

In[ ] :=

```
GrafoMariposa[5]
```

Out[ ] :=



Ejemplo 7.106

Analice los grafos mariposa desde el orden 1 hasta el 5.

Solución:

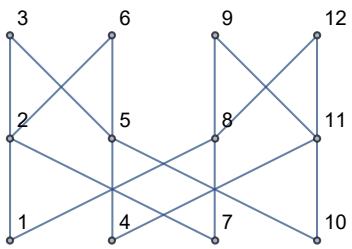
En el software se emplea la opción "`analizar -> True`":

In[ ] :=

```
GrafoMariposa[5, analizar -> True]
```

Out[ ] :=

Orden del grafo



	1	2	3	4	5	6	7	8	9	10	11	12
Grado o valencia	2	4	2	2	4	2	2	4	2	2	4	2

Cantidad de aristas: 16 = 2^4

Tiene circuitos de Euler: True

Tiene circuitos de Hamilton: False

Explicación en video

54) **GrafoRueda**: **genera** un **grafo rueda** con "n" nodos. Brinda la **opción** "**analizar -> True**" que **muestra** una **animación** con **distintos grafos rueda** hasta el orden "n", indicando además en cada caso: la **valencia** de los vértices, la **cantidad de aristas** y, si el grafo posee **circuitos de Euler** y de **Hamilton**.

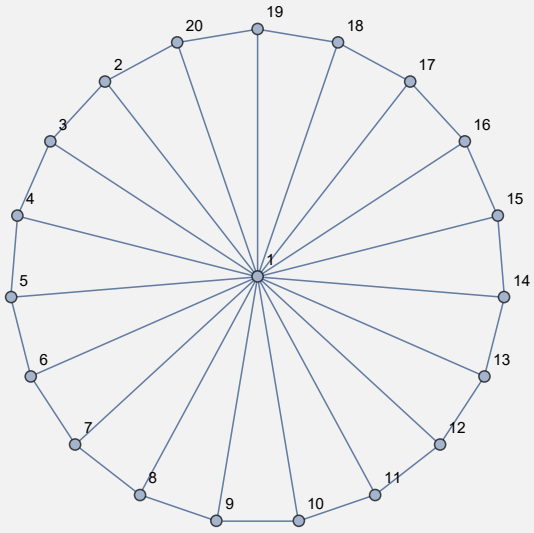
Sintaxis: `GrafoRueda[n]`, o bien, `GrafoRueda[n, analizar -> True]`.

Ejemplo 7.107

Muestre el grafo rueda de orden 20.

Solución:

In[ ] :=
```
GrafoRueda[20]
```
Out[ ] :=



Ejemplo 7.108

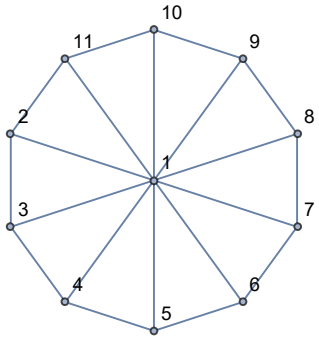
Utilizando la opción "**analizar -> True**" genere todos los grafos rueda desde el orden 1 hasta el 20.

Solución:

In[ ] :=
```
GrafoRueda[20, analizar -> True]
```
Out[ ] :=

Orden del grafo

11



	1	2	3	4	5	6	7	8	9	10	11
Grado o valencia	10	3	3	3	3	3	3	3	3	3	3

Cantidad de aristas: 20 = $2^2 \cdot 5^1$

Tiene circuitos de Euler: False

Tiene circuitos de Hamilton: True

Explicación en video

55) **GrafoEstrella**: **construye** un **grafo estrella** con "n" nodos. Proporciona la **opción** "**`analizar -> True`**" que **muestra** una **animación** con **distintos grafos estrella** hasta el orden "n", indicando además en cada caso: la **valencia** de los vértices, la **cantidad de aristas** y, si el grafo posee **circuitos de Euler** y de **Hamilton**.

Sintaxis: **`GrafoEstrella[n]`**, o bien, **`GrafoEstrella[n, analizar -> True]`**.

Ejemplo 7.109

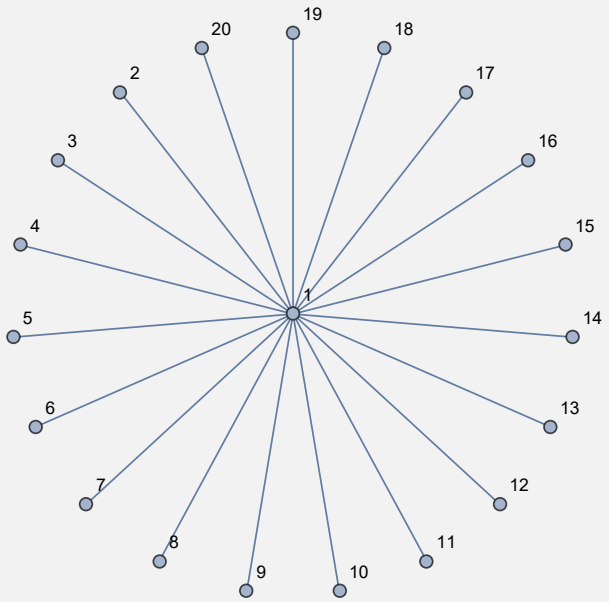
Construya el grafo estrella de orden 20.

Solución:

In[ ] :=

```
GrafoEstrella[20]
```

Out[ ] :=



Ejemplo 7.110

Analice distintos grafos estrella desde el orden 1 hasta el 20.

Solución:

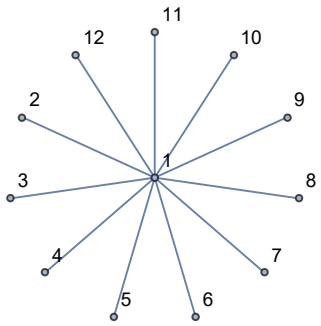
En *Mathematica*:

In[ ] :=

```
GrafoEstrella[20, analizar->True]
```

Out[ ] :=

Orden del grafo



	1	2	3	4	5	6	7	8	9	10	11	12
Grado o valencia	11	1	1	1	1	1	1	1	1	1	1	1

Cantidad de aristas: 11 = 11^1

Tiene circuitos de Euler: False

Tiene circuitos de Hamilton: False

Explicación en video

56) **GrafoCamino**: **crea** un **grafo** "path" o **camino** con "n" nodos. Facilita la **opción** "**`analizar -> True`**" que **muestra** una **animación** con **distintos grafos camino** hasta el orden "n", indicando además en cada caso: la **valencia** de los vértices, la **cantidad de aristas** y, si el grafo posee **circuitos de Euler** y de **Hamilton**.

Sintaxis: **`GrafoCamino[n]`**, o bien, **`GrafoCamino[n, analizar -> True]`**.

Ejemplo 7.111

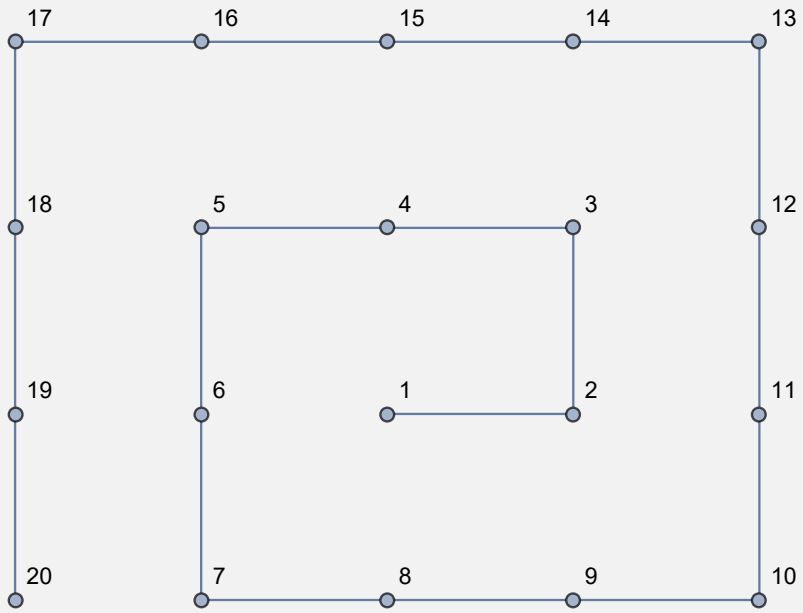
Genere el grafo camino con 20 nodos.

Solución:

In[ ] :=

```
GrafoCamino[20]
```

Out[ ] :=



Ejemplo 7.112

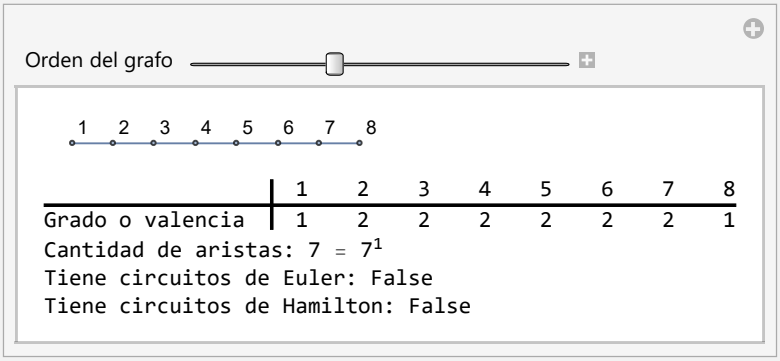
Analice distintos grafos "path" desde el orden 1 hasta el 20.

Solución:

In[ ] :=

```
GrafoCamino[20, analizar -> True]
```

Out[ ] :=



Explicación en video

57) **GrafoCiclo**: **traza** un **grafo ciclo** o **circuito** con "n" nodos. Proporciona la **opción** "**analizar -> True**" que **muestra** una **animación** con **distintos grafos ciclo** hasta el orden "n", indicando además en cada caso: la **valencia** de los vértices, la **cantidad de aristas** y, si el grafo posee **circuitos de Euler** y de **Hamilton**.

Sintaxis: `GrafoCiclo[n]`, o bien, `GrafoCiclo[n, analizar -> True]`.

Ejemplo 7.113

Construya un grafo circuito con 20 vértices.

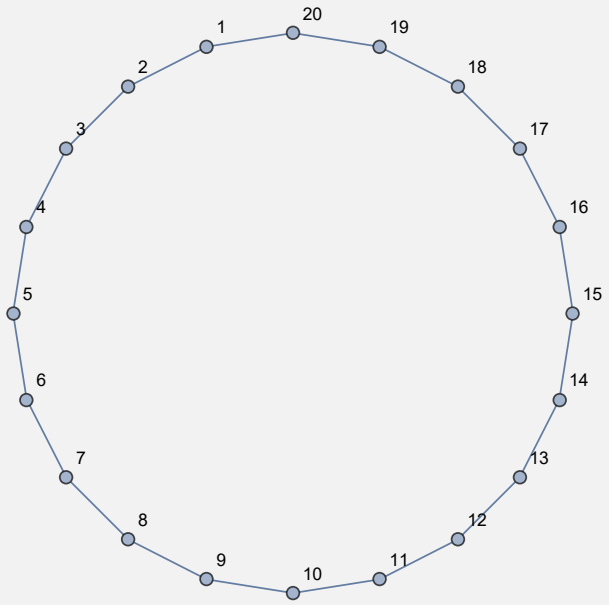
Solución:

En el software:

In[ ] :=

```
GrafoCiclo[20]
```

Out[ ] :=



Ejemplo 7.114

Obtenga distintos grafos ciclo comenzando con el orden 1 y hasta el 20.

Solución:

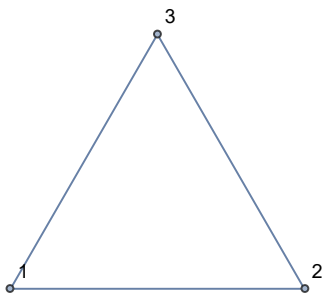
Al utilizar la opción "**analizar -> True**":

In[ ] :=

```
GrafoCiclo[20, analizar -> True]
```

Out[ ] :=

Orden del grafo



	1	2	3
Grado o valencia	2	2	2

Cantidad de aristas: 3 = 3^1

Tiene circuitos de Euler: True

Tiene circuitos de Hamilton: True

Explicación en video

58) **GrafoRed**: **traza** un **grafo red** o **cuadrícula** con "n" filas y "m" columnas ("n*m" nodos). Posee la **opción** "`analizar->True`" que **muestra** una **animación** con **distintos grafos red** hasta el orden "n" por "m", indicando además en cada caso: la **valencia** de los vértices, la **cantidad de aristas** y, si el grafo posee **circuitos de Euler** y de **Hamilton**.

Sintaxis: `GrafoRed[n, m]`, o bien, `GrafoRed[n, m, analizar->True]`.

Ejemplo 7.115

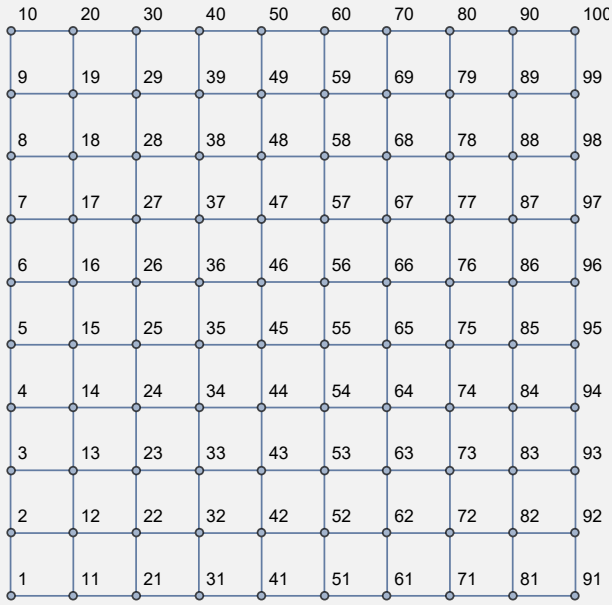
Construya un grafo red de orden 10×10.

Solución:

In[ ] :=

```
GrafoRed[10, 10]
```

Out[ ] :=



Ejemplo 7.116

Analice un grafo red 10×10.

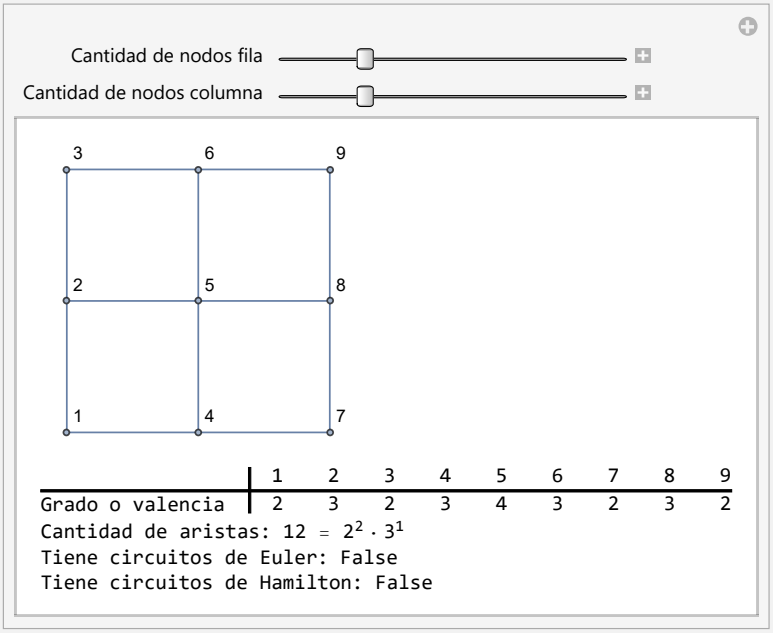
Solución:

En *Mathematica*:

In[ ] :=

```
GrafoRed[10, 10, analizar -> True]
```

Out[ ] :=



Explicación en video

59) **GrafoRegularQ**: **retorna** "`True`" si se recibe como parámetro un **grafo** "G" **regular** y "`False`", en caso contrario. Un grafo es **regular** si es **no dirigido** donde todos los **vértices** tienen la **misma valencia** (valor que corresponde a su grado u orden).

Sintaxis: `GrafoRegularQ[G]`.

Ejemplo 7.117

Genere en el "Wolfram System" de *Mathematica* el grafo **`RandomGraph[DegreeGraphDistribution[Table[6, {6}]]]`**. Determine si es regular. Convierta el grafo al ambiente del paquete "Combinatorica". Verifique la misma salida lógica del comando **`GrafoRegularQ`**.

Solución:

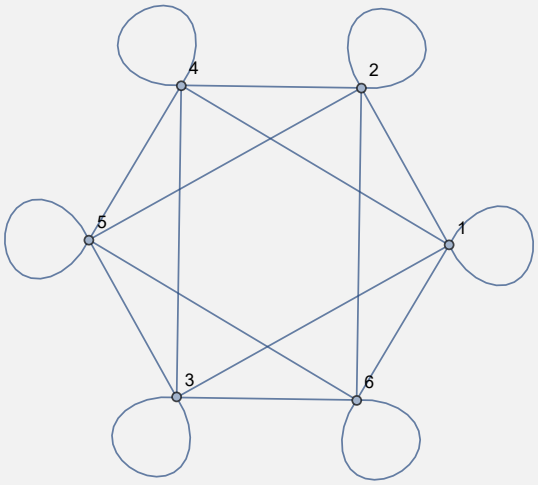
En el software:

In[ ] :=

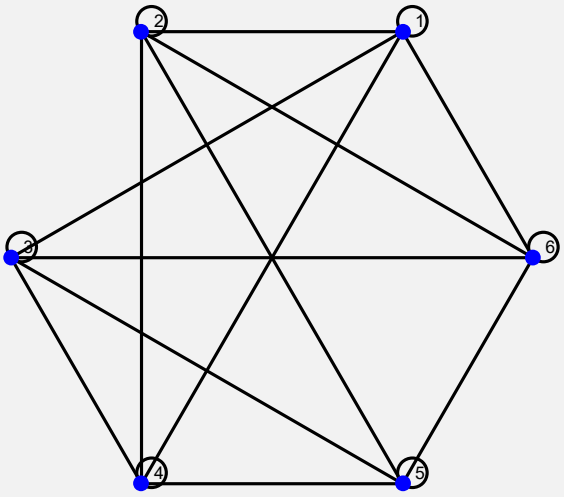
```
grafo = System`RandomGraph[DegreeGraphDistribution[Table[6, {6}]],
VertexLabels -> "Name", ImagePadding -> 10]
GrafoRegularQ[grafo]
GraphToCombinatorica[grafo]
```

GrafoRegularQ[G]

Out[ ] :=



True



True

N Se recuerda al estudiante la funcionalidad de la instrucción **GraphTo-Combinatorica**, convirtiendo en este ejemplo, el grafo **Random-Graph[DegreeGraphDistribution[Table[6, {6}]]]** del "Wolfram System" al paquete "Combinatorica".

Ejemplo 7.118

Compruebe que un grafo red de orden 8×8 no es regular. Resuelva el ejercicio en ambos ambientes: el provisto por "Wolfram System" y el facilitado por el paquete "Combinatorica".

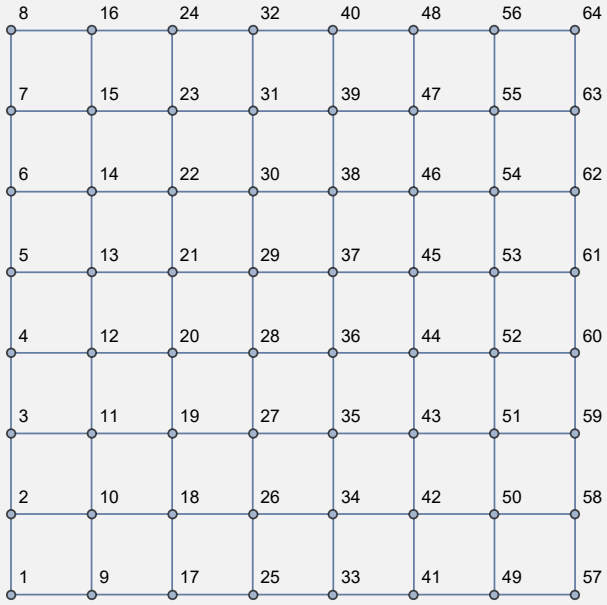
Solución:

Al emplear **`GrafoRegularQ`**:

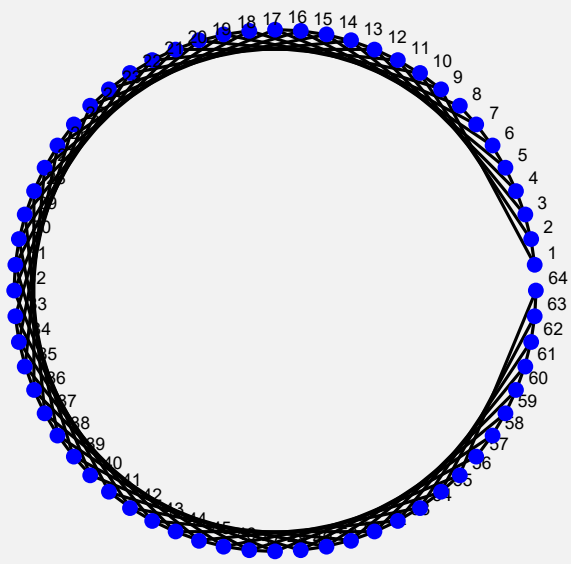
In[ ] :=

```
grafo = GrafoRed[8, 8]
GrafoRegularQ[grafo]
GraphToCombinatorica[grafo]
GrafoRegularQ[G]
```

Out[ ] :=



False



False

Explicación en video

60) **GrafoRegular**: **crea** si existe, un **grafo seudoaleatorio regular** de orden "n" con "m" nodos. Un grafo es **regular** si es **no dirigido** donde todos los **vértices** tienen la **misma valencia** (valor que corresponde a su grado u orden).

Sintaxis: `GrafoRegular[n, m]`.

Ejemplo 7.119

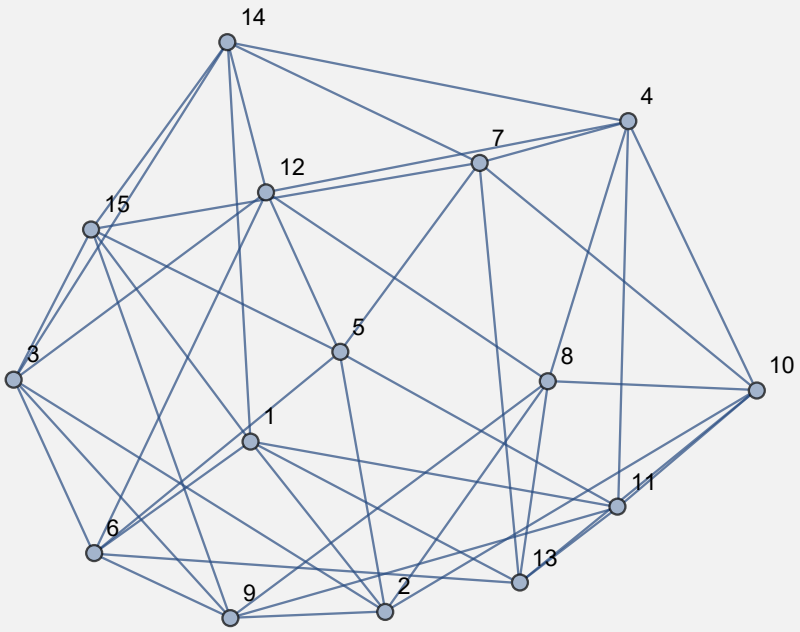
Construya un grafo regular donde todos los vértices tengan valencia 6 y con 15 nodos.

Solución:

In[ ] :=

```
GrafoRegular[6, 15]
```

Out[ ] :=



El *Out* [] es seudoaleatorio por lo que el grafo cambiará cada vez que se ejecuta el mismo *In* [].

Ejemplo 7.120

Muestre un grafo regular de orden 10 con 10 vértices.

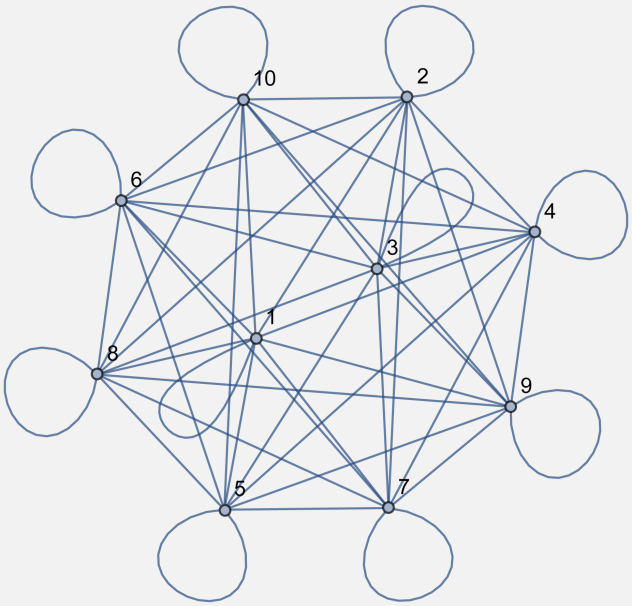
Solución:

En *Mathematica*:

In[ ] :=

```
GrafoRegular[10, 10]
```

Out[ ] :=



Explicación en video

61) **GrafoDato**: **genera** una **animación** para **analizar** un **grafo** de alguno de los **siguientes tipos**: "ChvatalGraph", "CubicalGraph", "DodecahedralGraph", "FranklinGraph", "FruchtGraph", "GroetzschGraph", "HeawoodGraph", "HerschelGraph", "IcosahedralGraph", "LeviGraph", "McGeeGraph", "NoPerfectMatchingGraph", "OctahedralGraph", "PetersenGraph", "RobertsonGraph", "SmallestCyclicGroupGraph", "TetrahedralGraph", "TutteGraph", "UniquelyThreeColorableGraph" y "WaltherGraph". El usuario **selecciona** el tipo de grafo de un "combo" suministrado y se **muestra** en **cada caso**: el **grafo**, la **valencia** de los vértices, la **cantidad de aristas** y, si posee **circuitos de Euler** y de **Hamilton**. También, se brinda la **opción** "`tipo -> G`" la cual permite **construir** el **grafo** especificado en "G", siendo "G" un "string" que contiene el **nombre** correspondiente de alguno de los **citados** anteriormente.

Sintaxis: `GrafoDato[]`, o bien, `GrafoDato[tipo -> G]`.

Ejemplo 7.121

Muestre el *CDF* generado al invocar `GrafoDato[]`. Seleccione del combo distitos grafos y observe la salida.

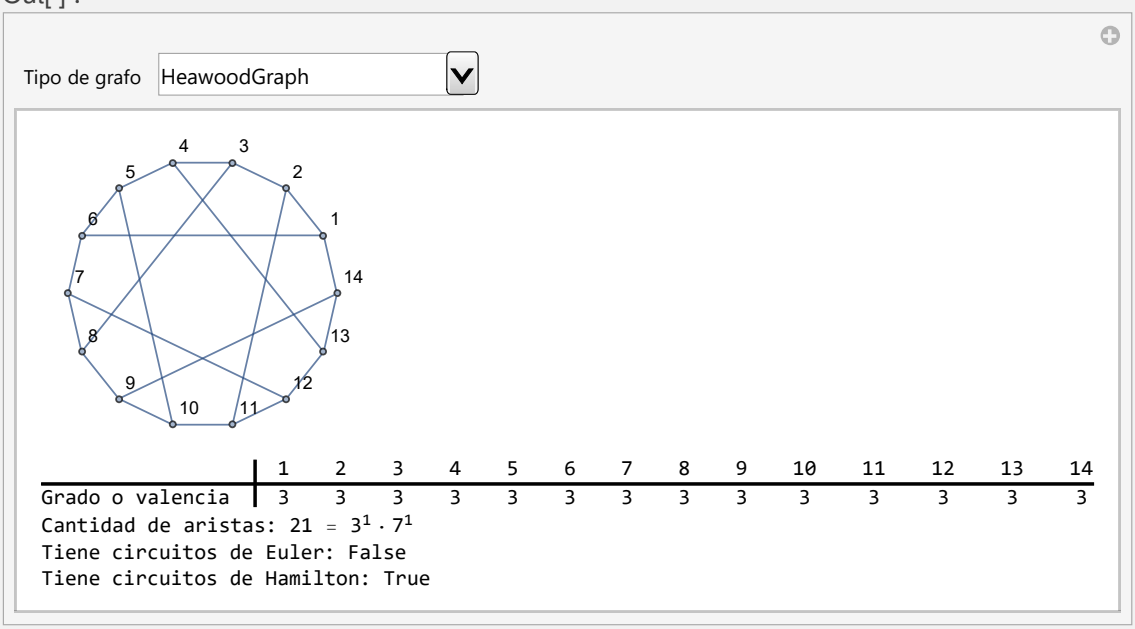
Solución:

En el software:

In[ ] :=

```
GrafoDato[]
```

Out[ ] :=



Ejemplo 7.122

Construya el grafo "TutteGraph" mediante el comando **`GrafoDato`**. Retorne la lista de todos sus lados.

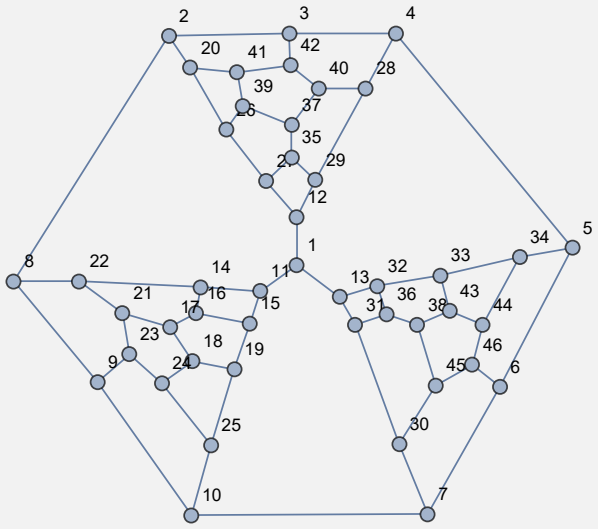
Solución:

Al utilizar la opción "tipo" de **`GrafoDato`**, se obtiene:

In[ ] :=

```
grafo = GrafoDato[tipo -> "TutteGraph"]
EdgeList[grafo]
```

Out[ ] :=



{1 ●—● 11, 1 ●—● 12, 1 ●—● 13, 2 ●—● 3, 2 ●—● 8, 2 ●—● 20, 3 ●—● 4, 3 ●—● 42, 4 ●—● 5, 4 ●—● 28, 5 ●—● 6, 5 ●—● 34, 6 ●—● 7, 6 ●—● 46, 7 ●—● 10, 7 ●—● 30, 8 ●—● 9, 8 ●—● 22, 9 ●—● 10, 9 ●—● 23, 10 ●—● 25, 11 ●—● 14, 11 ●—● 15, 12 ●—● 27, 12 ●—● 29, 13 ●—● 31, 13 ●—● 32, 14 ●—● 16, 14 ●—● 22, 15 ●—● 16, 15 ●—● 19, 16 ●—● 17, 17 ●—● 18, 17 ●—● 21, 18 ●—● 19, 18 ●—● 24, 19 ●—● 25, 20 ●—● 26, 20 ●—● 41, 21 ●—● 22, 21 ●—● 23, 23 ●—● 24, 24 ●—● 25, 26 ●—● 27, 26 ●—● 39, 27 ●—● 35, 28 ●—● 29, 28 ●—● 40, 29 ●—● 35, 30 ●—● 31, 30 ●—● 45, 31 ●—● 36, 32 ●—● 33, 32 ●—● 36, 33 ●—● 34, 33 ●—● 43, 34 ●—● 44, 35 ●—● 37, 36 ●—● 38, 37 ●—● 39, 37 ●—● 40, 38 ●—● 43, 38 ●—● 45, 39 ●—● 41, 40 ●—● 42, 41 ●—● 42, 43 ●—● 44, 44 ●—● 46, 45 ●—● 46}

Explicación en video

62) **GrafoCircuitosQ**: **retorna** "**`True`**" si un **grafo** "G" recibido como parámetro posee **circuitos** y "**`False`**", en caso contrario. El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o a través del uso del **paquete** "Combinatorica".

Sintaxis: `GrafoCircuitosQ[G]`.

Ejemplo 7.123

Determine si $K_{5,6}$ posee circuitos.

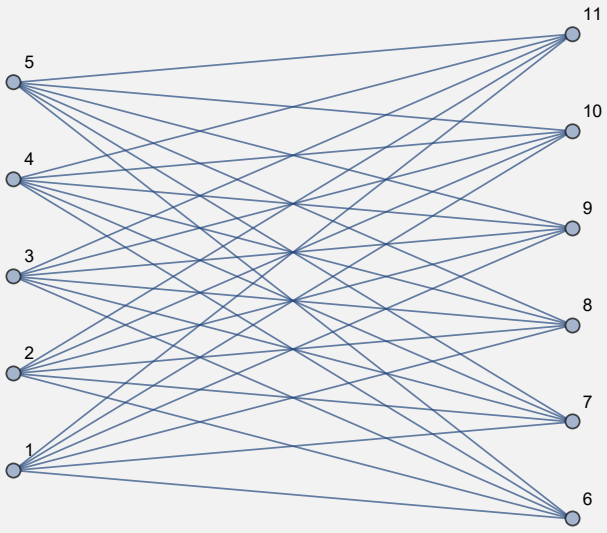
Solución:

En el software se utilizará la instrucción **`GrafoBipartitoCompleto`**, para crear el grafo correspondiente:

In[ ] :=

```
grafo = GrafoBipartitoCompleto[5, 6]
GrafoCircuitosQ[grafo]
```

Out[ ] :=



True

Sí posee circuitos.

Ejemplo 7.124

Construya un grafo en "Combinatorica" mediante el comando **`GrafoRandomConexo`** de orden 10×9 y a través de **`GrafoCircuitosQ`**, establezca si posee o no circuitos.

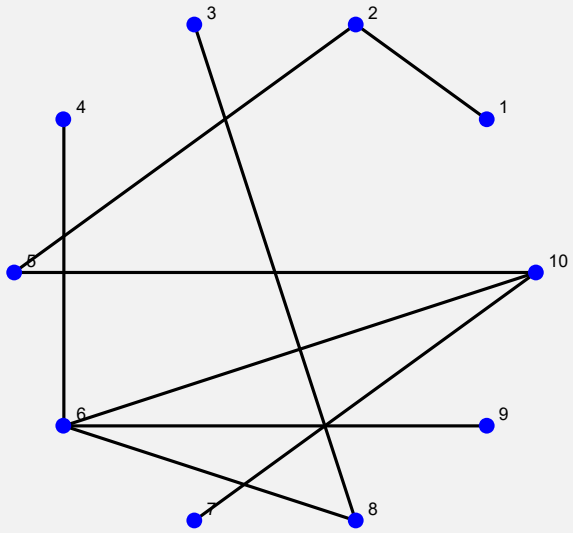
Solución:

En *Mathematica*:

In[ ] :=

```
GrafoRandomConexo[10, 9, combinatorica -> True]
GrafoCircuitosQ[G]
```

Out[ ] :=



False

Se concluye que el grafo mostrado no posee circuitos.

Explicación en video

63) **CircuitoL**: recibe un **grafo** "G" **devolviendo** un **circuito de longitud** "k" (con "k" **mayor o igual a dos** (en caso de **aristas múltiples y sin considerar lazos**)). El comando brinda la **propiedad** "`ruta -> True`" que retorna una **animación** con el **circuito correspondiente** sobre "G".

Sintaxis: `CircuitoL[G, k]`, o bien, `CircuitoL[G, k, ruta -> True]`. El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o, a través del **paquete** "Combinatorica". La **longitud** está en función de la **cantidad** de **aristas** contenidas y **no** de los **pesos** que podría poseer "G".

Ejemplo 7.125

Considere el grafo dodecaedro. Halle sobre él un circuito de longitud diez. Resuelva el problema en el ambiente provisto por el paquete "Combinatorica" y en el "Wolfram System" de *Mathematica*. Muestre en cada caso el circuito mediante una animación.

Solución:

Se empleará el comando de *VilCretas* `CombinatoricaToGraph` para convertir el grafo dodecaedro construido en "Combinatorica" al "Wolfram System". Luego:

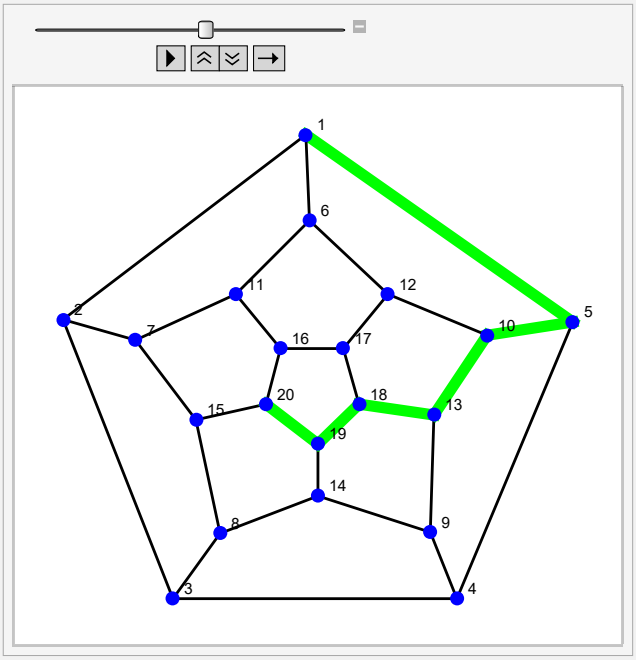
In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
```

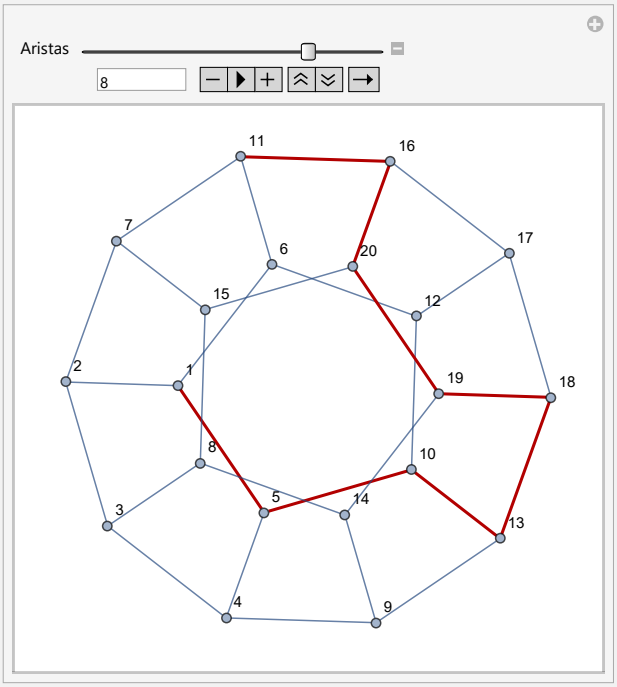
```
VertexColor -> Blue, EdgeColor -> Black], VertexLabel -> True,
PlotRange -> 0.1];
CircuitoL[grafo, 10]
CircuitoL[grafo, 10, ruta -> True]
grafo = CombinatoricaToGraph[grafo];
CircuitoL[grafo, 10]
CircuitoL[grafo, 10, ruta -> True]
```

Out[ ]:=

{1 ●—● 5, 5 ●—● 10, 10 ●—● 13, 13 ●—● 18, 18 ●—● 19, 19 ●—● 20, 20 ●—● 16, 16 ●—● 11, 11 ●—● 6, 6 ●—● 1}



{1 ●—● 5, 5 ●—● 10, 10 ●—● 13, 13 ●—● 18, 18 ●—● 19, 19 ●—● 20, 20 ●—● 16, 16 ●—● 11, 11 ●—● 6, 6 ●—● 1}



N Como es de esperarse, la instrucción **`CircuitoL`** retorna el mismo circuito con "Combinatorica" y en el "Wolfram System". Se pudo haber procedido de forma equivalente en este ejercicio, de manera inversa, es decir, construyendo el grafo en el "Wolfram System" de *Mathematica* y posteriormente, convirtiéndolo al ambiente suministrado por el paquete "Combinatorica" .

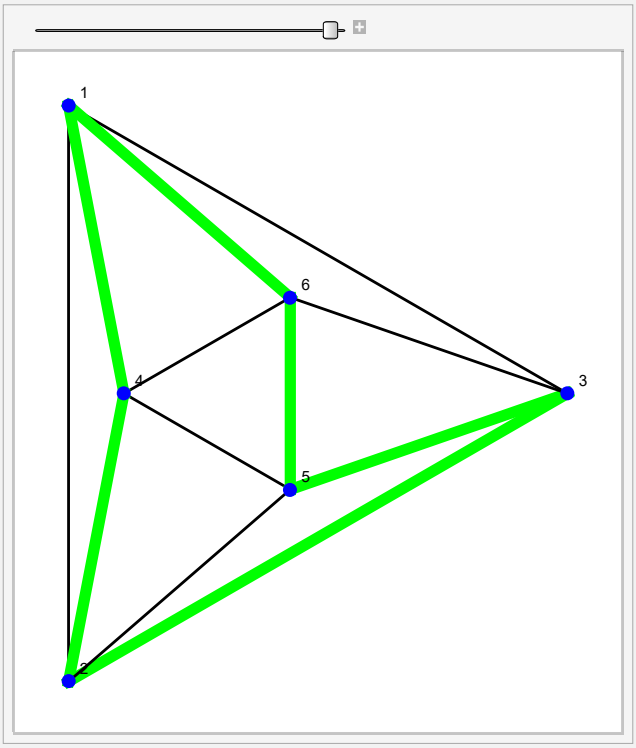
Ejemplo 7.126

Construya el grafo octaedro y determine sobre él un circuito de longitud seis. Adicionalmente, exhiba el circuito a través de una animación.

Solución:

```
In[ ] :=
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[OctahedralGraph,
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1];
CircuitoL[grafo, 6]
CircuitoL[grafo, 6, ruta->True]
```

```
Out[ ] :=
{1 ●—● 6, 6 ●—● 5, 5 ●—● 3, 3 ●—● 2, 2 ●—● 4, 4 ●—● 1}
```



Explicación en video

64) **Circuito**: recibe un **grafo** "G" **devolviendo** si **existe** un **circuito**. La instrucción proporciona la **opción** "**`ruta -> True`**" que retorna una **animación** con el **circuito correspondiente** sobre "G".

Sintaxis: **`Circuito[G]`**, o bien, **`Circuito[G, ruta -> True]`**. El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o, a través del **paquete** "Combinatorica".

Ejemplo 7.127

Halle un circuito y muestre la ruta correspondiente sobre el grafo dodecaedro. Resuelva el problema con o sin el uso del paquete "Combinatorica".

Solución:

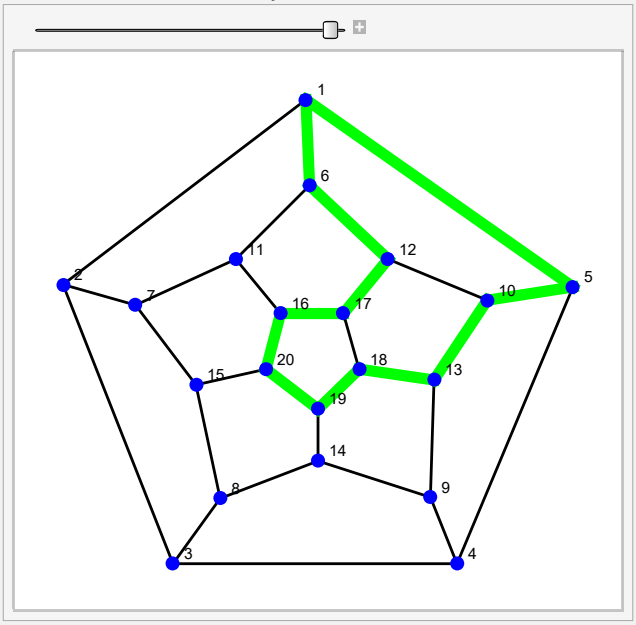
Empleando el comando **`Circuito`**, se tiene:

```
In[ ]:=
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor -> Blue, EdgeColor -> Black], VertexLabel -> True,
PlotRange -> 0.1];
Circuito[grafo]
```

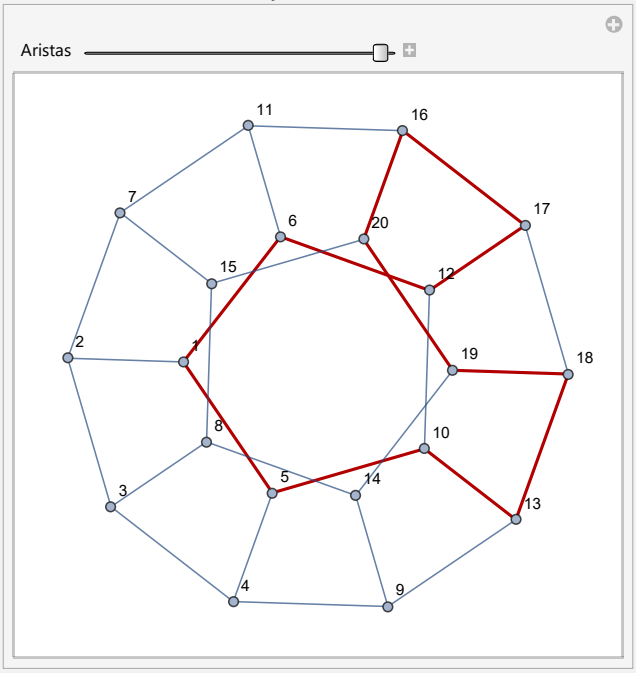
```
Circuito[grafo, ruta -> True]
grafo = CombinatoricaToGraph[grafo];
Circuito[grafo]
Circuito[grafo, ruta -> True]
```

Out[ ] :=

{1 ●—● 5, 5 ●—● 10, 10 ●—● 13, 13 ●—● 18, 18 ●—● 19, 19 ●—● 20, 20 ●—● 16, 16 ●—● 17, 17 ●—● 12, 12 ●—● 6, 6 ●—● 1}



{1 ●—● 5, 5 ●—● 10, 10 ●—● 13, 13 ●—● 18, 18 ●—● 19, 19 ●—● 20, 20 ●—● 16, 16 ●—● 17, 17 ●—● 12, 12 ●—● 6, 6 ●—● 1}



N El lector debe observar que en ambos ambientes, el circuito arrojado mediante la instrucción **`Circuito`** es el mismo.

Ejemplo 7.128

Construya el grafo **`GrafoFronteraCountries["France"]`**. A través del uso del comando **`Circuito`**, halle un circuito y muestre la ruta correspondiente.

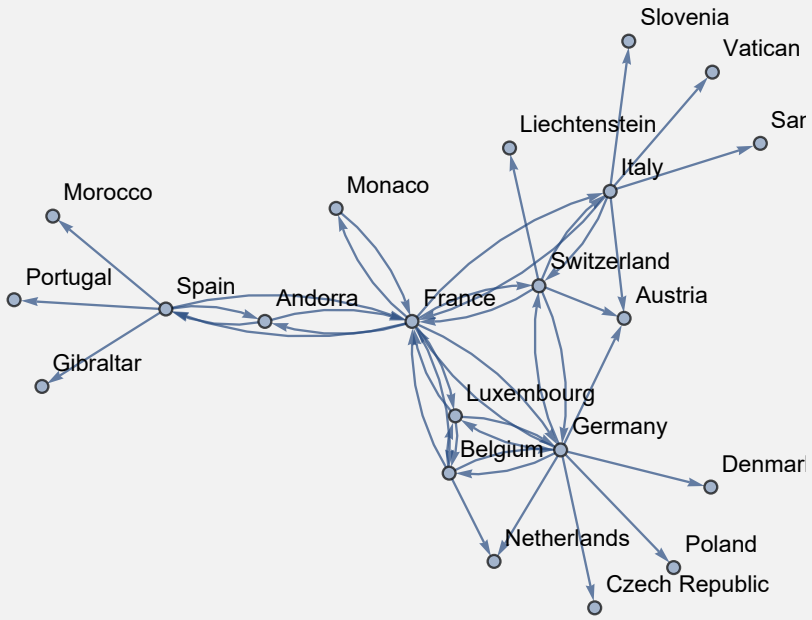
Solución:

En el software:

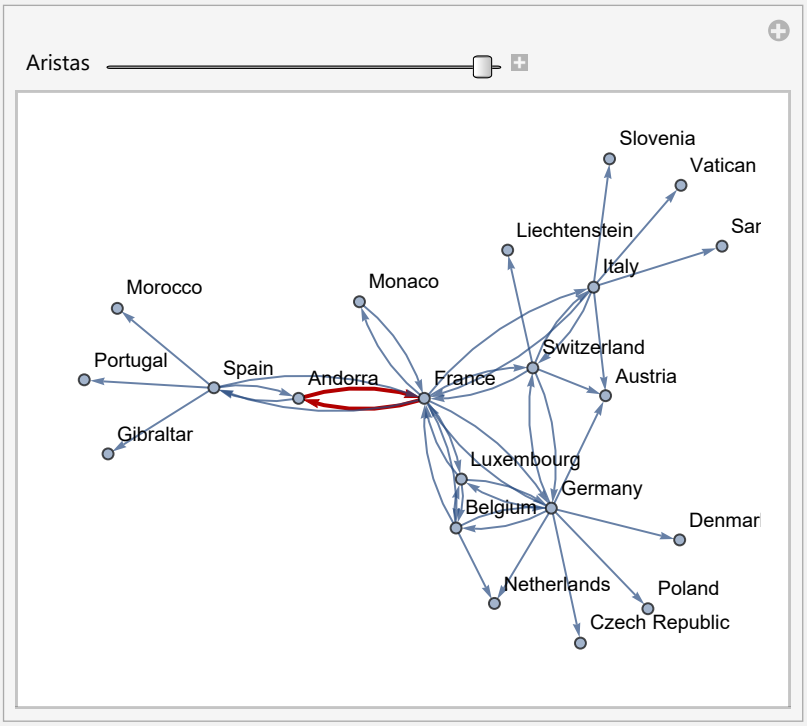
In[ ] :=

```
grafo = GrafoFronteraCountries["France"]
Circuito[grafo]
Circuito[grafo, ruta -> True]
```

Out[ ] :=



{ **Andorra** (country) ●–> **France** (country) , **France** (country) ●–> **Andorra** (country) }



Explicación en video

65) **LongitudCircuitoSmall**: **determina** la **longitud del circuito más pequeño** sobre un **grafo simple** "G" recibido como parámetro. El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, mediante el uso del **paquete** "Combinatorica".

Sintaxis: `LongitudCircuitoSmall[G]`. La **longitud** está en función de la **cantidad** de **aristas** contenidas y **no** de los **pesos** que podría poseer "G".

Ejemplo 7.129

Genere un grafo seudoaleatorio de orden 20×50 con y sin el paquete "Combinatorica". Mediante la instrucción **`LongitudCircuitoSmall`**, halle en cada caso, la longitud del circuito más pequeño.

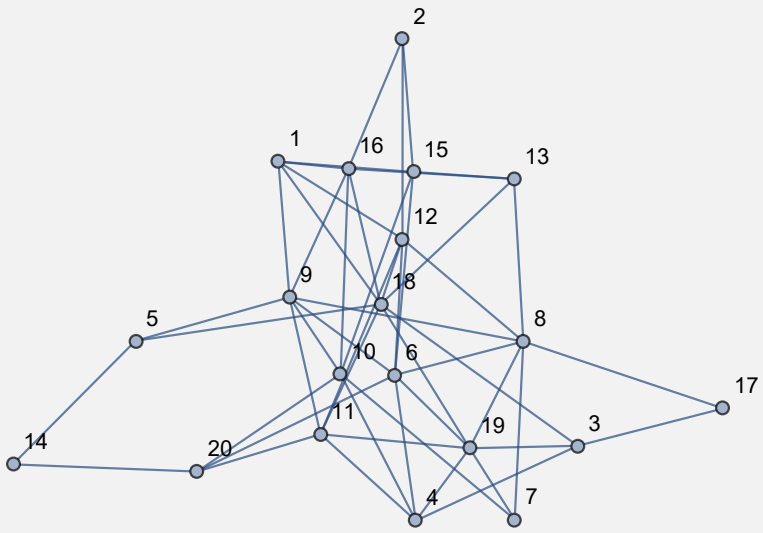
Solución:

En *Mathematica*:

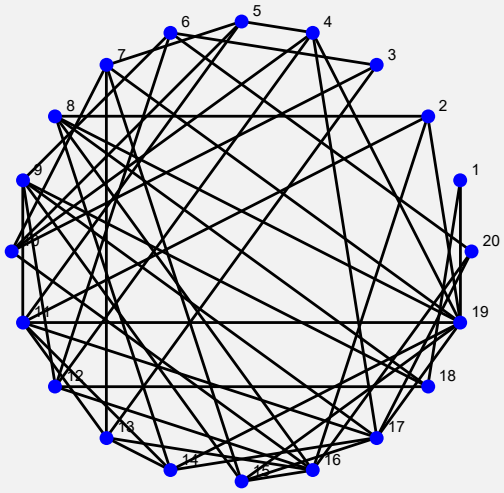
In[ ] :=

```
grafo = GrafoRandom[20, 50]
LongitudCircuitoSmall[grafo]
GrafoRandom[20, 50, combinatorica -> True]
LongitudCircuitoSmall[G]
```

Out[ ] :=



3



3

Ejemplo 7.130

En un grafo completo de orden 15, determine la longitud del circuito más pequeño. Verifique la misma salida utilizando el paquete "Combinatorica".

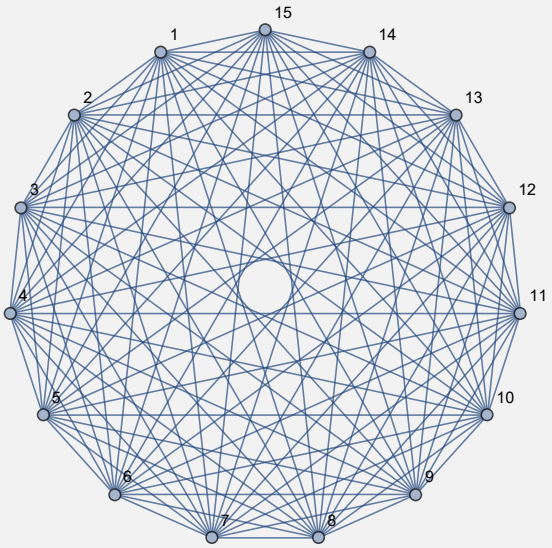
Solución:

En el software se construirá el grafo completo, usando el comando de *VilCretas* **GrafoCompleto**:

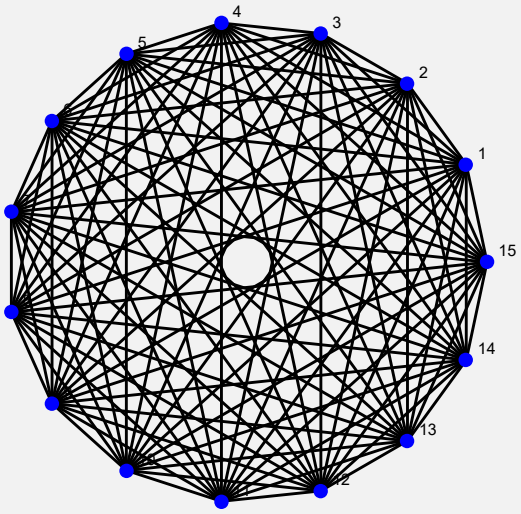
In[ ] :=

```
grafo = GrafoCompleto[15]
LongitudCircuitoSmall[grafo]
GraphToCombinatorica[grafo]
LongitudCircuitoSmall[G]
```

Out[ ] :=



3



3

Explicación en video

66) **GrafoKönigsberg**: **genera** el **grafo** asociado al **problema** de los "siete puentes de Königsberg" (actualmente "Kaliningrado"). Presenta **tres opciones**: "`dimensions3d->True`" construye el grafo en **tercera dimensión**, "`grados->True`" muestra el **grafo** y las **valencias** e, "`image->True`" **crea** el **grafo** con una **imagen** de **fondo** de lo que fue en el pasado, la **región** de los "siete puentes de Königsberg".

Sintaxis: **`GrafoKönigsberg[]`**, **`GrafoKönigsberg[grados->True]`**, **`GrafoKönigsberg[dimensions3d->True]`**, **`GrafoKönigsberg[dimensions3d->True, grados->True]`**, **`GrafoKönigsberg[image->True]`**, o bien,

```
GrafoKönigsberg[image->True, grados->True]
```

El uso de las tres opciones **simultáneamente** no genera **ningún error**, salvo el hecho de que "**`image->True`**" **automáticamente** toma a "**`dimensions3d->False`**" **sobreponiendo** la **asignación** "**`dimensions3d->True`**".

Ejemplo 7.131

Corra en el programa *Mathematica*:

- **`GrafoKönigsberg[dimensions3d->True]`**
- **`GrafoKönigsberg[dimensions3d->True, grados->True]`**

Determine en cada invocación si la salida corresponde a un grafo.

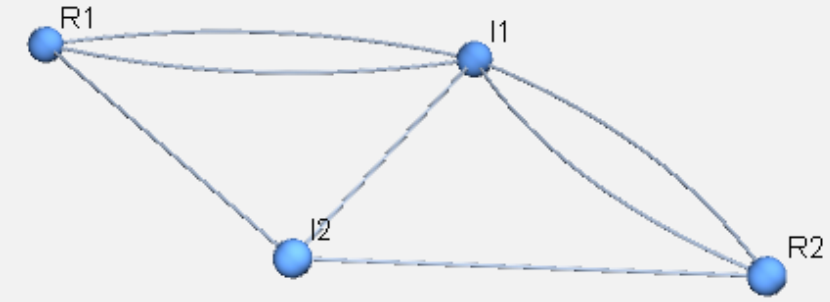
Solución:

Para constatar o no en el *Out*[ ], que el resultado es un grafo, se recurrirá al empleo de **`GrafoQ`**:

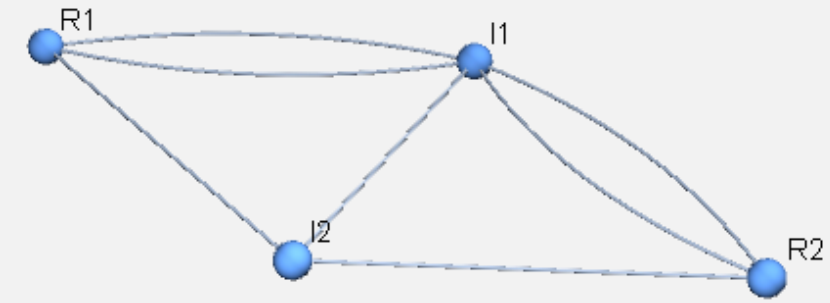
In[ ] :=

```
grafo = GrafoKönigsberg[dimensions3d->True]
GrafoQ[grafo]
grafo = GrafoKönigsberg[dimensions3d->True, grados->True]
GrafoQ[grafo]
```

Out[ ] :=



True



	R2	I1	I2	R1
Grado o valencia	3	5	3	3

False

N La salida de `GrafoKönigsberg[dimensions3d->True, grados->True]` no es interpretada por parte del software *Mathematica*, como un grafo, al incluir una tabla de valencias.

Ejemplo 7.132

Genere el grafo del problema de los siete puentes de *Königsberg*, mediante una imagen. Además, construya el grafo como una imagen y muestre la tabla de grados ¿La imagen es un grafo en el software?

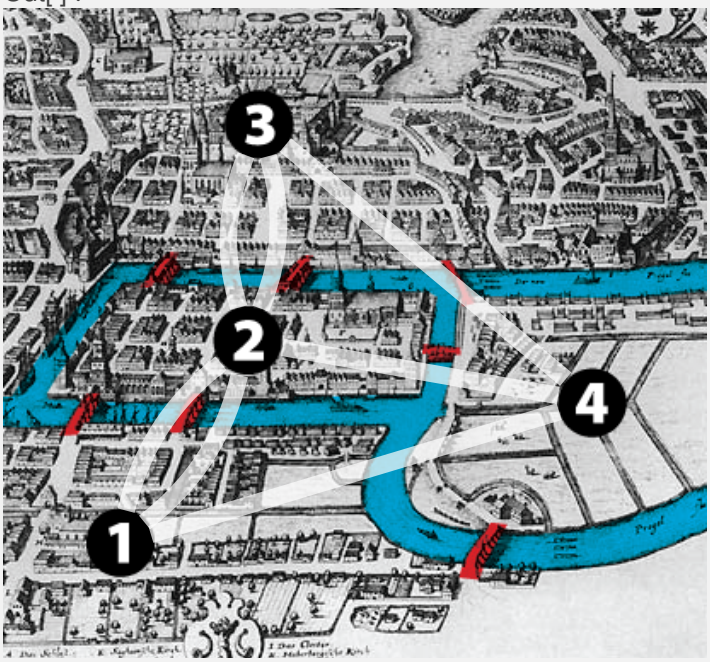
Solución:

En *Mathematica*:

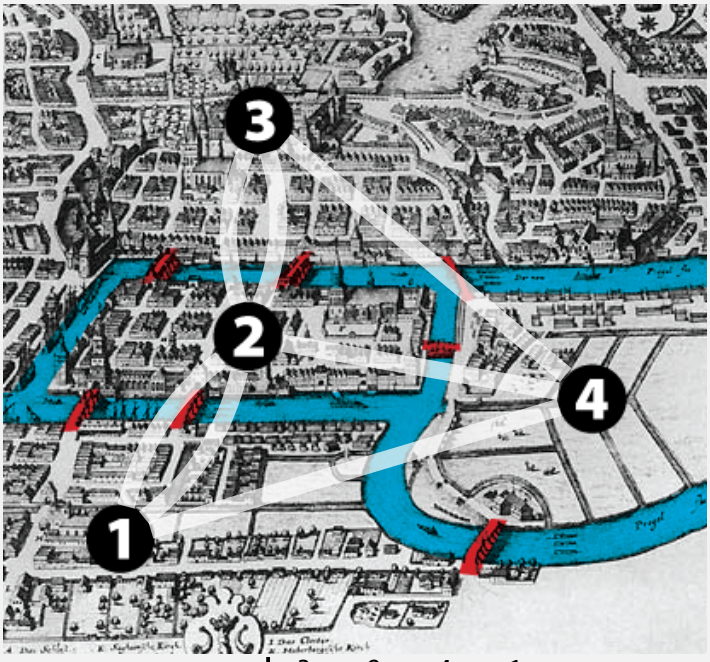
In[ ] :=

```
grafo = GrafoKönigsberg[image->True]
GrafoQ[grafo]
grafo = GrafoKönigsberg[image->True, grados->True]
```

Out[ ] :=



False



	3	2	4	1
Grado o valencia	3	5	3	3

N Se observa que la imagen no es un grafo para el software *Mathematica*. Por otra parte, los nodos en el grafo imagen corresponden al conjunto $\{1,2,3,4\}$, por lo que la tabla de valencias de este ejemplo, no tiene las mismas etiquetas de la obtenida en el ejercicio anterior.

Explicación en video

67) **CircuitoEulerQ**: función booleana que determina si un grafo "G" **posee** un **circuito de Euler**. El grafo pudo haber sido **creado** con o sin el **paquete** "Combinatorica".

Sintaxis: `CircuitoEulerQ[G]`.

Ejemplo 7.133

Determine si el grafo "ChvatalGraph" posee un circuito de *Euler*.

Solución:

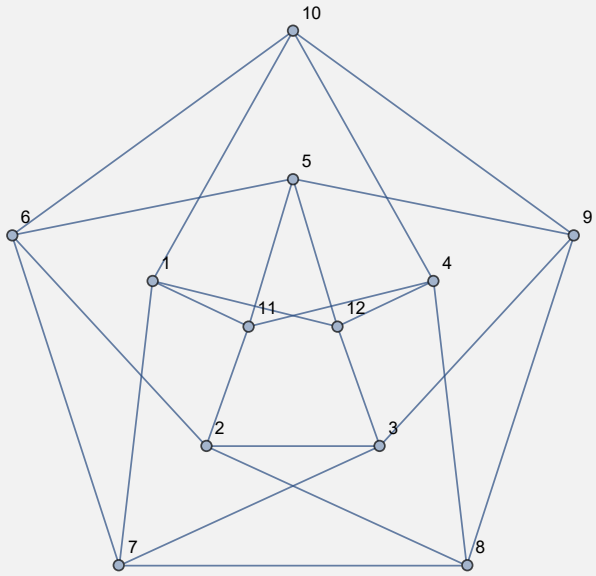
Al utilizar los comandos **`GrafoDato`** y **`CircuitoEulerQ`**:

In[ ] :=

```
grafo = GrafoDato[tipo -> "ChvatalGraph"]
```

```
CircuitoEulerQ[grafo]
```

Out[ ] :=



True

Se concluye que sí posee circuitos de *Euler*.

Ejemplo 7.134

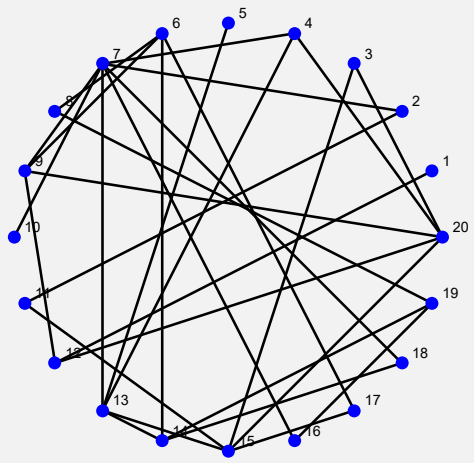
Establezca si un grafo construido al invocar **`GrafoRandomConexo[20, 30, combinatorica -> True]`**, tiene o no un circuito euleriano.

Solución:

In[ ] :=

```
GrafoRandomConexo[20, 30, combinatorica -> True]
CircuitoEulerQ[G]
```

Out[ ] :=



False

En el grafo mostrado no hay circuitos de *Euler*.

Explicación en video

68) **Fleury**: **genera** un **circuito de Euler** sobre un grafo "G" **simple**, **conexo no dirigido**, con todas sus **valencias pares**, utilizando como base el **algoritmo de Fleury**. El comando muestra **iteración por iteración** cómo se va **construyendo** el **circuito**. El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, mediante el uso del **paquete** "Combinatorica".

Sintaxis: `Fleury[G]`.

Ejemplo 7.135

Sea un grafo con aristas: {{a, b}, {a, c}, {c, b}, {b, d}, {c, d}, {b, e}, {c, e}}. Encuentre un circuito de *Euler* sobre este grafo a través de la instrucción **`Fleury`**.

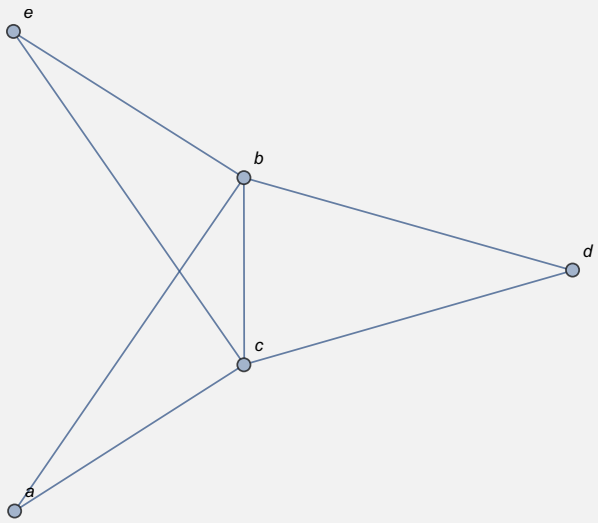
Solución:

En *Mathematica*:

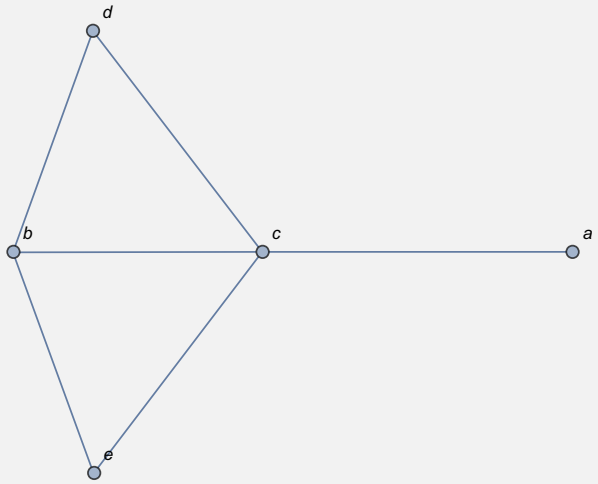
In[ ] :=

```
grafo = Grafo[{{a, b}, {a, c}, {c, b}, {b, d}, {c, d}, {b, e}, {c,
e}}]
Fleury[grafo]
```

Out[ ] :=

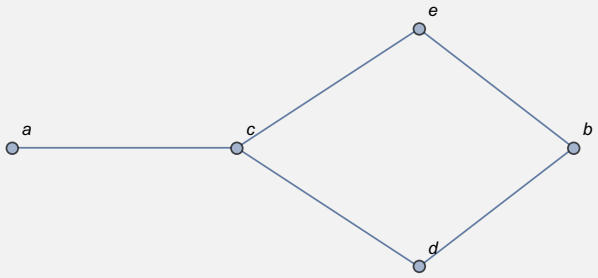


Iteración: 1
Aristas añadidas: {{a, b}}



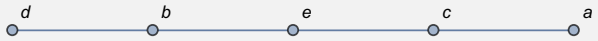
Iteración: 2

Aristas añadidas: {{a, b}, {b, c}}



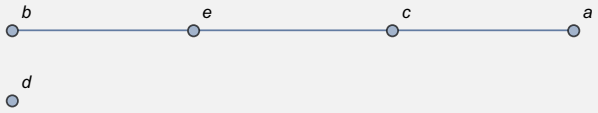
Iteración: 3

Aristas añadidas: {{a, b}, {b, c}, {c, d}}



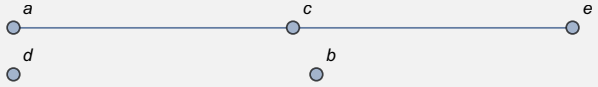
Iteración: 4

Aristas añadidas: {{a, b}, {b, c}, {c, d}, {d, b}}



Iteración: 5

Aristas añadidas: {{a, b}, {b, c}, {c, d}, {d, b}, {b, e}}

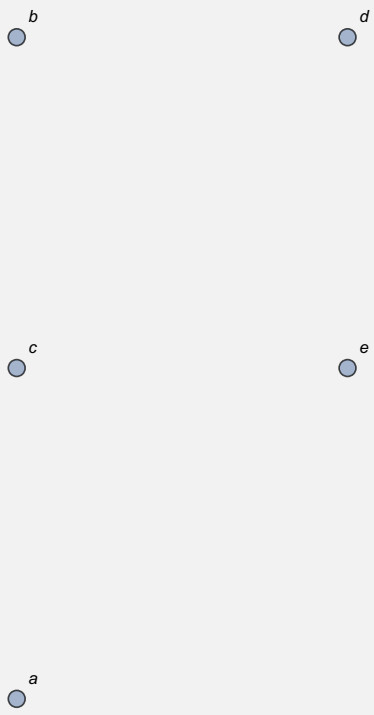


Iteración: 6

Aristas añadidas: {{a, b}, {b, c}, {c, d}, {d, b}, {b, e}, {e, c}}

Iteración: 7

Aristas añadidas: {{a, b}, {b, c}, {c, d}, {d, b}, {b, e}, {e, c}, {c, a}}



Un circuito de Euler mediante una representación de nodos es: {a, b, c, d, b, e, c, a}
El mismo circuito representado a través de aristas es: {{a, b}, {b, c}, {c, d}, {d, b}, {b, e}, {e, c}, {c, a}}

N En cada iteración, el grafo mostrado se obtiene al eliminar del grafo anterior, la arista añadida al circuito.

Ejemplo 7.136

Construya mediante el uso del paquete "Combinatorica" un grafo con aristas: {{1, 2}, {1, 3}, {3, 2}, {3, 4}, {3, 5}, {5, 4}}. Halle por el algoritmo de *Fleury* un circuito euleriano.

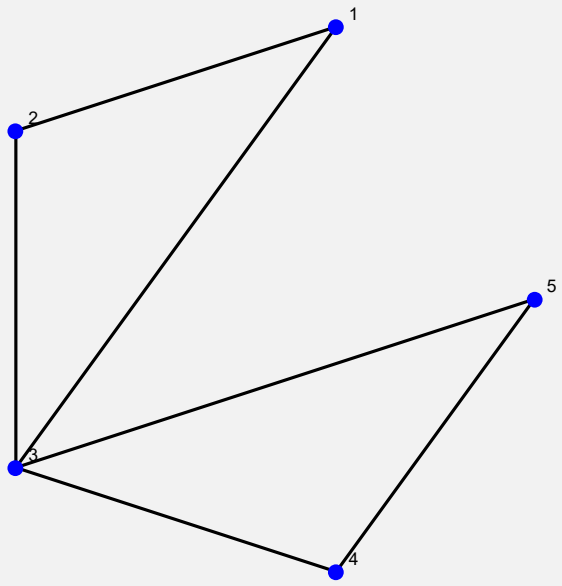
Solución:

Se creará el grafo por medio del comando **GrafoC**, luego:

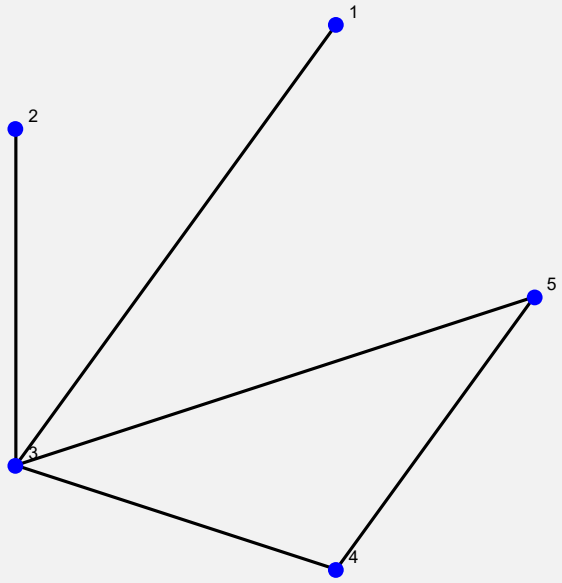
In[ ] :=

```
GrafoC[{{1, 2}, {1, 3}, {3, 2}, {3, 4}, {3, 5}, {5, 4}}]
Fleury[G]
```

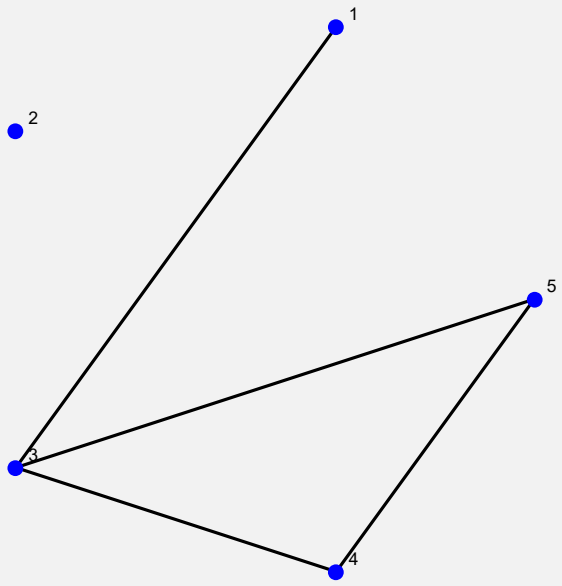
Out[ ] :=



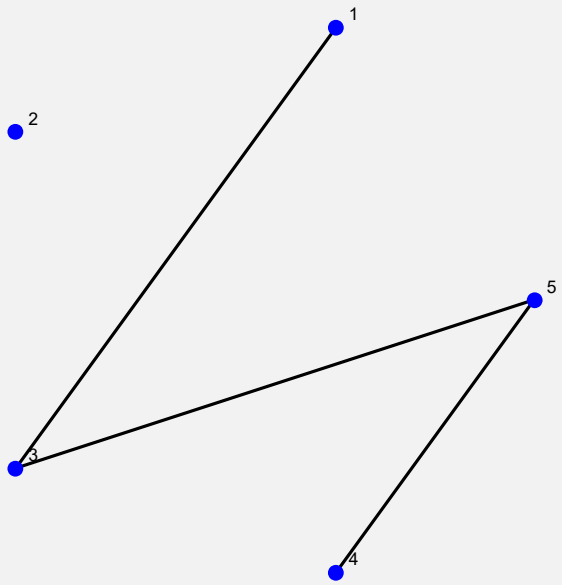
Iteración: 1
Aristas añadidas: {{1, 2}}



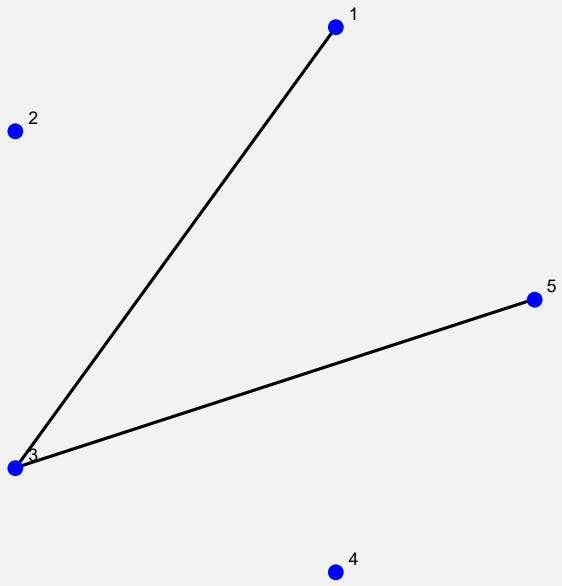
Iteración: 2
Aristas añadidas: {{1, 2}, {2, 3}}



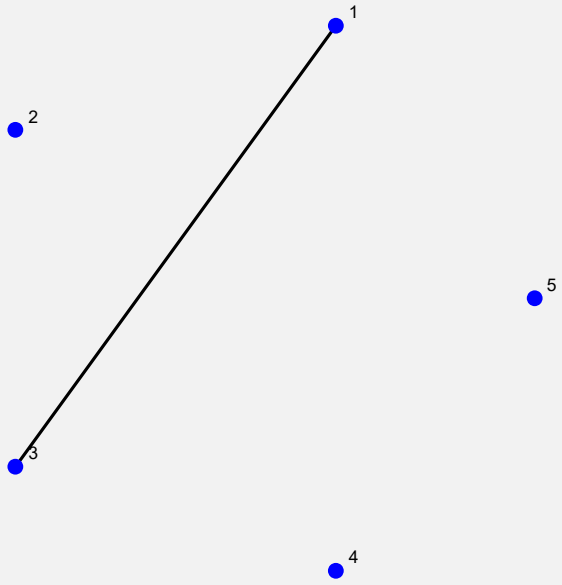
Iteración: 3
Aristas añadidas: {{1, 2}, {2, 3}, {3, 4}}



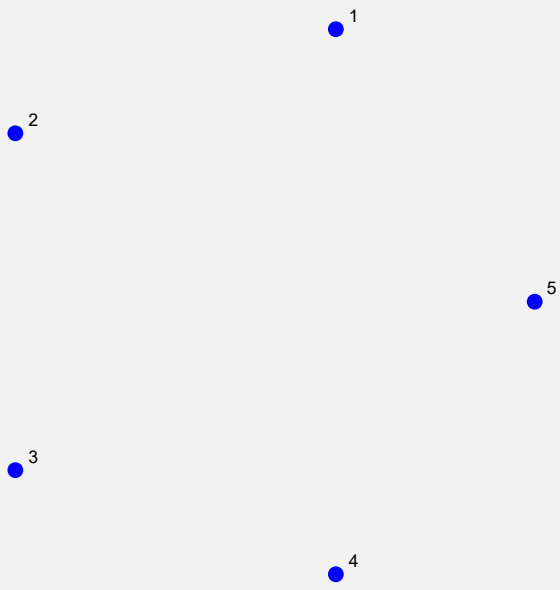
Iteración: 4
Aristas añadidas: {{1, 2}, {2, 3}, {3, 4}, {4, 5}}



Iteración: 5
Aristas añadidas: {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 3}}



Iteración: 6
Aristas añadidas: {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 3}, {3, 1}}



Un circuito de Euler mediante una representación de nodos es: {1, 2, 3, 4, 5, 3, 1}
El mismo circuito representado a través de aristas es: {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 3}, {3, 1}}

Explicación en video

69) **CircuitosEuler**: **encuentra** "n" **circuitos de Euler** distintos sobre un grafo "G", si es que existen. El comando resuelve el problema tanto si el grafo se **creó** con el **paquete** "Combinatorica", así como también, si fue **generado** en el "Wolfram System" de *Mathematica*. Brinda la **opción** "`ruta -> True`" que despliega adicionalmente a través de una **animación**, **uno** de los **circuitos** elegido de manera **seudoaleatoria**.

Sintaxis: `CircuitosEuler[G, n]`, o bien, `CircuitosEuler[G, n, ruta -> True]`.

Ejemplo 7.137

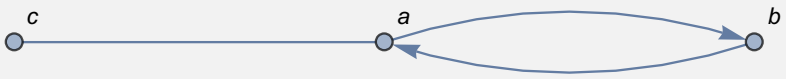
Considere un grafo con aristas: {a ●–> b, b ●–> a, a ●–● c}. Halle un circuito de *Euler* y muestre su recorrido mediante una animación.

Solución:

En el software:

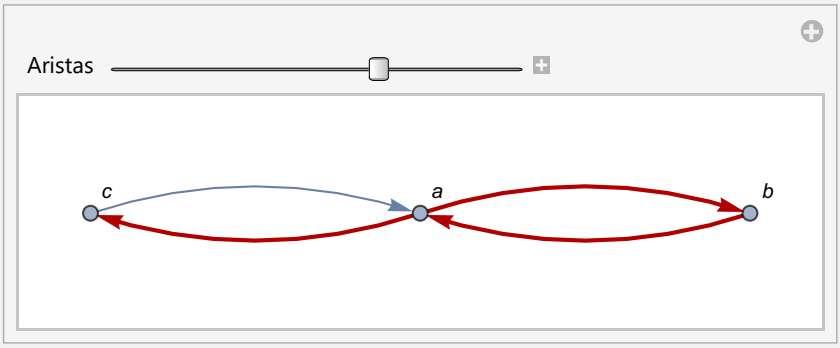
```
In[ ] :=
grafo = System`Graph[{a -> b, b -> a, a <-> c},
VertexLabels -> "Name", ImagePadding -> 10]
CircuitosEuler[grafo, 1, ruta -> True]
```

Out[ ] :=



{{a ●–> b, b ●–> a, a ●–> c, c ●–> a}}

Ruta de la animación: {a ●–> b, b ●–> a, a ●–> c, c ●–> a}



Ejemplo 7.138

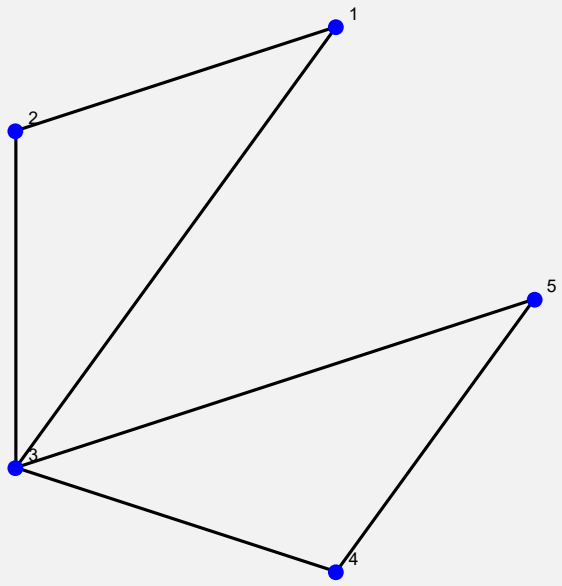
En el ambiente facilitado por el paquete "Combinatorica", crea un grafo con aristas: {{1, 2}, {1, 3}, {3, 2}, {3, 4}, {3, 5}, {5, 4}}. Encuentre tres circuitos de *Euler* distintos y en una animación, muestre uno de ellos.

Solución:

In[ ] :=

```
GrafoC[{{1, 2}, {1, 3}, {3, 2}, {3, 4}, {3, 5}, {5, 4}}]
CircuitosEuler[G, 3]
CircuitosEuler[G, 3, ruta->True]
```

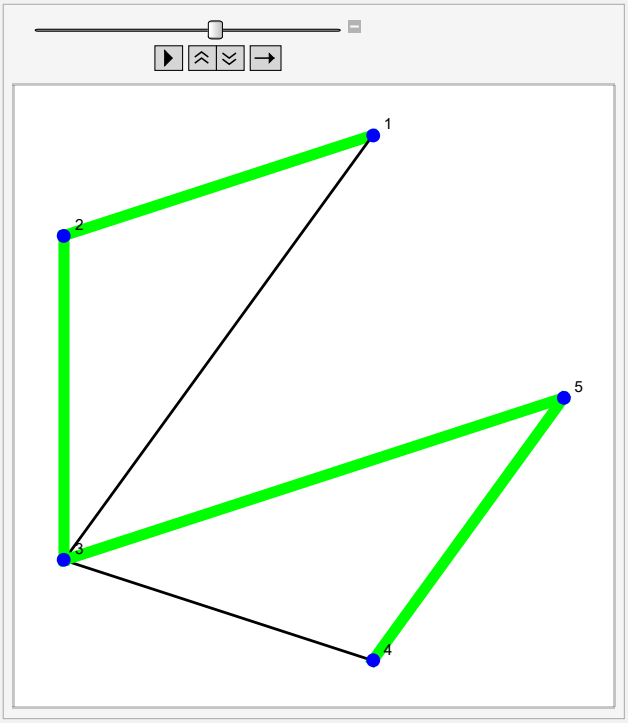
Out[ ] :=



{{1 ●—● 2, 2 ●—● 3, 3 ●—● 5, 5 ●—● 4, 4 ●—● 3, 3 ●—● 1}, {1 ●—● 2, 2 ●—● 3, 3 ●—● 4, 4 ●—● 5, 5 ●—● 3, 3 ●—● 1}}

{{1 ●—● 2, 2 ●—● 3, 3 ●—● 5, 5 ●—● 4, 4 ●—● 3, 3 ●—● 1}, {1 ●—● 2, 2 ●—● 3, 3 ●—● 4, 4 ●—● 5, 5 ●—● 3, 3 ●—● 1}}

Ruta de la animación: {1 ●—● 2, 2 ●—● 3, 3 ●—● 4, 4 ●—● 5, 5 ●—● 3, 3 ●—● 1}



N En este ejemplo, el software no fue capaz de encontrar tres circuitos eulerianos distintos, retornando solamente dos. En ocasiones la instrucción **`CircuitosEuler`** presenta este comportamiento.

Explicación en video

70) **RutaEulerQ**: función booleana que retorna "`True`" si un grafo "G" **no dirigido** recibido como parámetro **posee** una **ruta de Euler** o "`False`", en caso contrario.

Sintaxis: **`RutaEulerQ[G]`**.

Ejemplo 7.139

Utilice la invocación **`GrafoRandomConexo[4, 5]`** para crear un grafo ¿Existe sobre él una ruta de *Euler*?

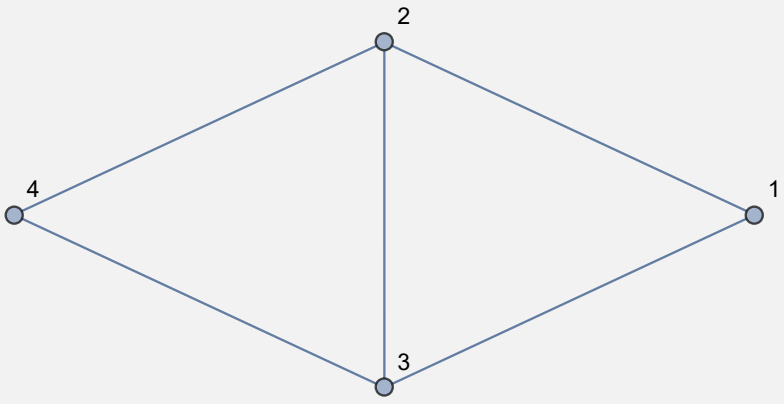
Solución:

Al emplear el comando **`RutaEulerQ`**:

In[ ] :=

```
grafo = GrafoRandomConexo[4, 5]
RutaEulerQ[grafo]
```

Out[ ] :=



True

Sí contiene una ruta euleriana.

Ejemplo 7.140

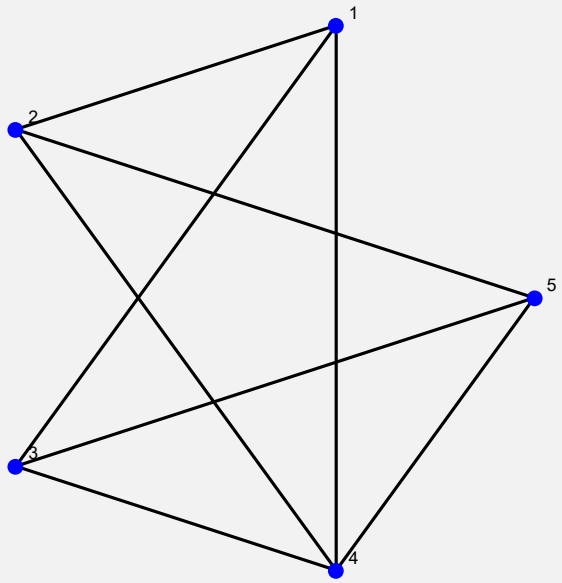
Construya en el ambiente provisto por el paquete "Combinatorica", un grafo seudoaleatorio conexo de orden 5×8. Establezca si sobre él existe o no una ruta de *Euler*.

Solución:

En *Mathematica*, se recurre a la opción "**combinatorica -> True**" de **GrafoRandomConexo**:

```
In[ ] :=
GrafoRandomConexo[5, 8, combinatorica -> True]
RutaEulerQ[G]
```

Out[ ] :=



False

Se concluye que el grafo no posee una ruta de *Euler*. Como la salida es seudoaleatoria podría ocurrir que al volver a correr la misma entrada, se obtenga otro grafo, donde sí se dé la existencia de una ruta euleriana.

Explicación en video

71) **RutaEuler**: **retorna** una **ruta de Euler** sobre un grafo "G" **simple** (**sin aristas múltiples**, **ni lazos**) **no dirigido** que la contenga. El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, a través del uso del **paquete** "Combinatorica". El comando brinda la **propiedad**

"**ruta -> True**" que retorna una **animación** con la **ruta correspondiente** sobre "G".

Sintaxis: **RutaEuler[G]**, o bien, **RutaEuler[G, ruta -> True]**.

Ejemplo 7.141

Halle una ruta de *Euler* sobre un grafo con lados: {{a, c}, {a, d}, {b, c}, {b, d}, {c, e}, {d, e}}. Genere una animación que la describa.

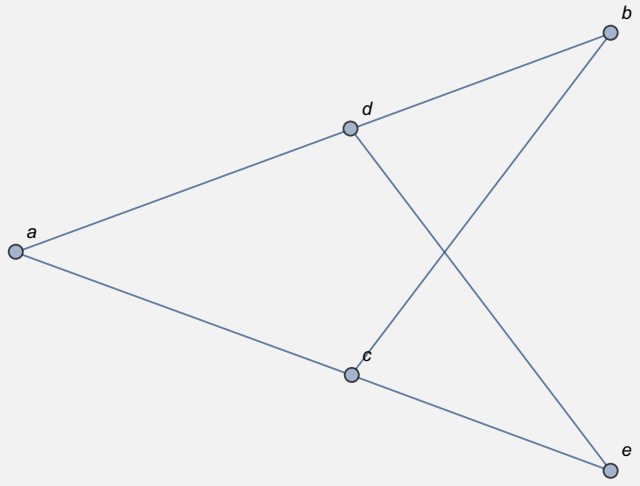
Solución:

En el software:

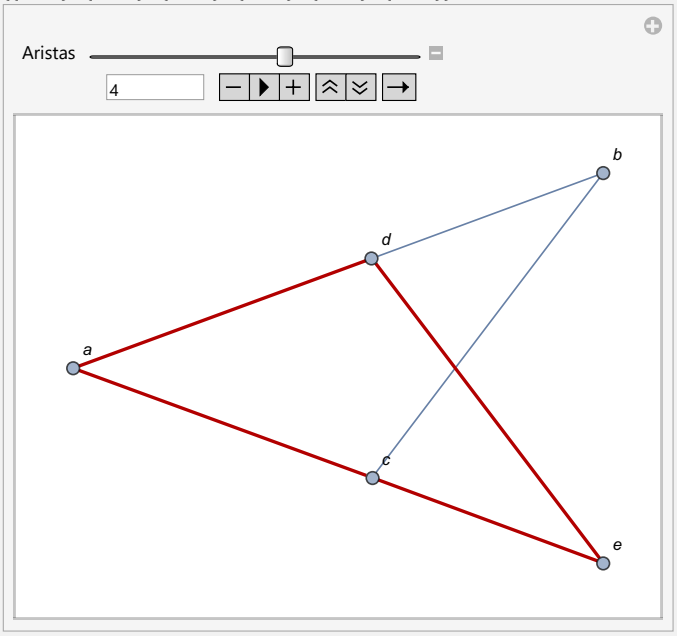
In[ ] :=

```
grafo = Grafo[{{a, c}, {a, d}, {b, c}, {b, d}, {c, e}, {d, e}}]
RutaEuler[grafo]
RutaEuler[grafo, ruta -> True]
```

Out[ ] :=



{{c, a}, {a, d}, {d, e}, {e, c}, {c, b}, {b, d}}



Ejemplo 7.142

Encuentre si es posible una ruta euleriana sobre un grafo seudoaleatorio de orden 4×5, generado con "Combinatorica". Muestre el recorrido a través de una animación.

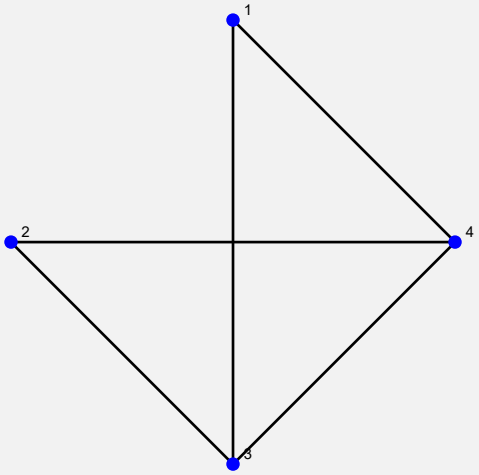
Solución:

Al recurrir al comando **RutaEuler**, se obtiene:

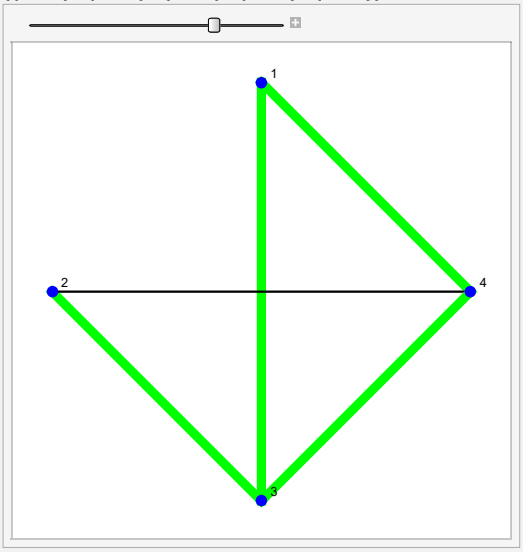
In[ ] :=

```
grafo = GrafoRandomConexo[4, 5, combinatorica -> True]
RutaEuler[grafo]
RutaEuler[grafo, ruta -> True]
```

Out[ ] :=



{{3, 1}, {1, 4}, {4, 3}, {3, 2}, {2, 4}}



Explicación en video

72) **CircuitoHamiltonQ**: función booleana que determina si un grafo "G" **posee** un **circuito de Hamilton**. El grafo pudo haber sido **creado** con o sin el **paquete** "Combinatorica".

Sintaxis: `CircuitoHamiltonQ[G]`.

Ejemplo 7.143

Determine si un grafo dirigido creado con "Combinatorica", cuyas aristas son: {{1, 1}, {1, 2}, {2, 4}, {3, 1}, {4, 5}, {5, 3}}, posee o no circuitos de *Hamilton*.

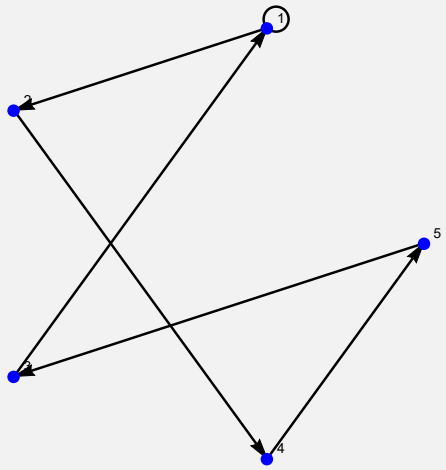
Solución:

En *Mathematica*:

In[ ] :=

```
GrafoC[{{1, 1}, {1, 2}, {2, 4}, {3, 1}, {4, 5}, {5, 3}},
dirigido -> True]
CircuitoHamiltonQ[G]
```

Out[ ] :=



True

N Sí contiene circuitos hamiltonianos. El estudiante es importante que observe, la versatilidad de la instrucción **`CircuitoHamiltonQ`**, al aceptar inclusive grafos dirigidos.

Ejemplo 7.144

Construya el grafo dodecaedro usando el comando **`GrafoDato`** ¿Posee circuitos de *Hamilton*?

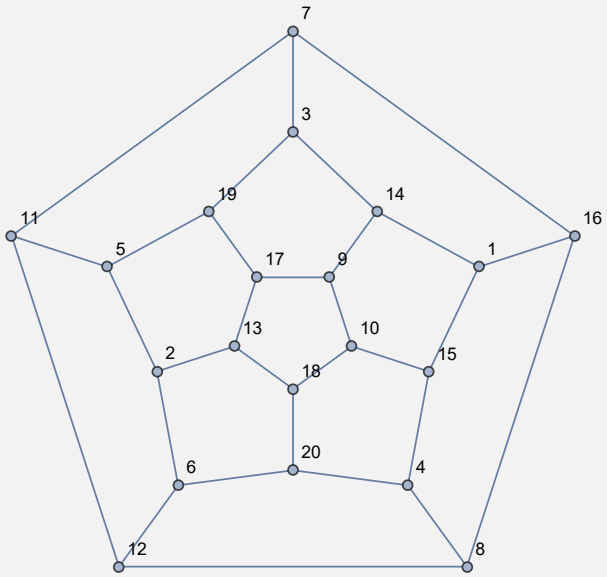
Solución:

En el software:

In[ ] :=

```
grafo = GrafoDato[tipo -> "DodecahedralGraph"]
CircuitoHamiltonQ[grafo]
```

Out[ ] :=



True

Sí es posible encontrar en este grafo, circuitos hamiltonianos.

Explicación en video

73) **CircuitosHamilton**: **encuentra** "n" **circuitos de Hamilton** distintos sobre un grafo "G", si es que existen. El comando resuelve el problema tanto si el grafo se **creó** con el **paquete** "Combinatorica", así como también, si fue **generado** en el "Wolfram System" de *Mathematica*. Brinda la **opción** "**`ruta -> True`**" que despliega adicionalmente a través de una **animación**, **uno** de los **circuitos** elegido de manera **seudoaleatoria**.

Sintaxis: **`CircuitosHamilton[G, n]`**, o bien, **`CircuitosHamilton[G, n, ruta -> True]`**.

Ejemplo 7.145

Halle cinco circuitos de *Hamilton* distintos, sobre un grafo completo de orden 10. Muestre uno de ellos en una animación.

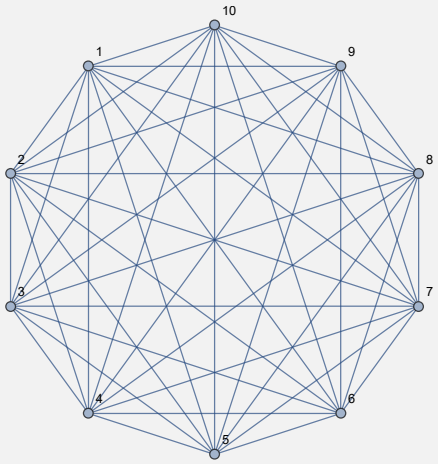
Solución:

`CircuitosHamilton` resuelve lo solicitado:

In[ ] :=

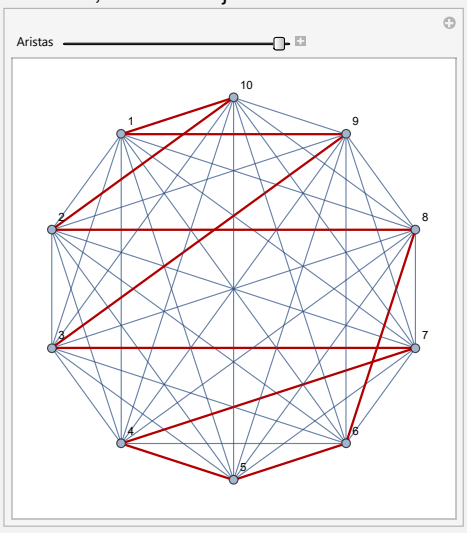
```
grafo = GrafoCompleto[10]
CircuitosHamilton[grafo, 5, ruta -> True]
```

Out[ ] :=



{{1 ●—● 9, 9 ●—● 3, 3 ●—● 7, 7 ●—● 5, 5 ●—● 6, 6 ●—● 4, 4 ●—● 8, 8 ●—● 2, 2 ●—● 10, 10 ●—● 1}, {1 ●—● 9, 9 ●—● 3, 3 ●—● 7, 7 ●—● 6, 6 ●—● 5, 5 ●—● 4, 4 ●—● 8, 8 ●—● 2, 2 ●—● 10, 10 ●—● 1}, {1 ●—● 9, 9 ●—● 3, 3 ●—● 7, 7 ●—● 4, 4 ●—● 6, 6 ●—● 5, 5 ●—● 8, 8 ●—● 2, 2 ●—● 10, 10 ●—● 1}, {1 ●—● 9, 9 ●—● 3, 3 ●—● 7, 7 ●—● 4, 4 ●—● 5, 5 ●—● 6, 6 ●—● 8, 8 ●—● 2, 2 ●—● 10, 10 ●—● 1}, {1 ●—● 9, 9 ●—● 3, 3 ●—● 7, 7 ●—● 6, 6 ●—● 4, 4 ●—● 5, 5 ●—● 8, 8 ●—● 2, 2 ●—● 10, 10 ●—● 1}}

Ruta de la animación: {1 ●—● 9, 9 ●—● 3, 3 ●—● 7, 7 ●—● 4, 4 ●—● 5, 5 ●—● 6, 6 ●—● 8, 8 ●—● 2, 2 ●—● 10, 10 ●—● 1}



Ejemplo 7.146

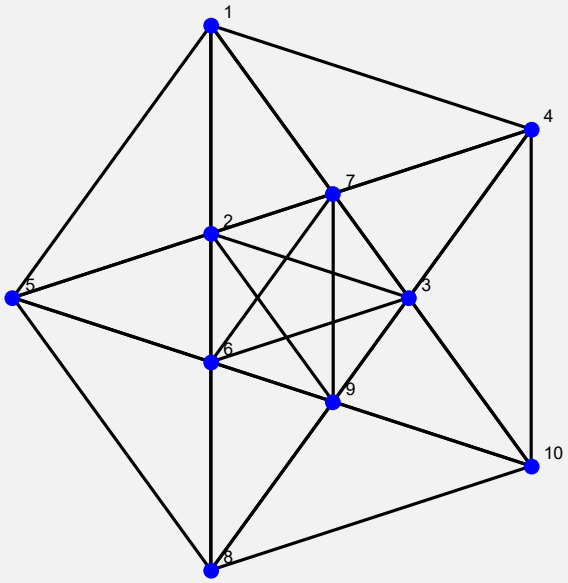
Encuentre si es posible, cinco circuitos de *Hamilton* sobre el grafo de "Combinatorica" **`LineGraph[CompleteGraph[5]]`**. Despliegue uno de ellos a través de una animación.

Solución:

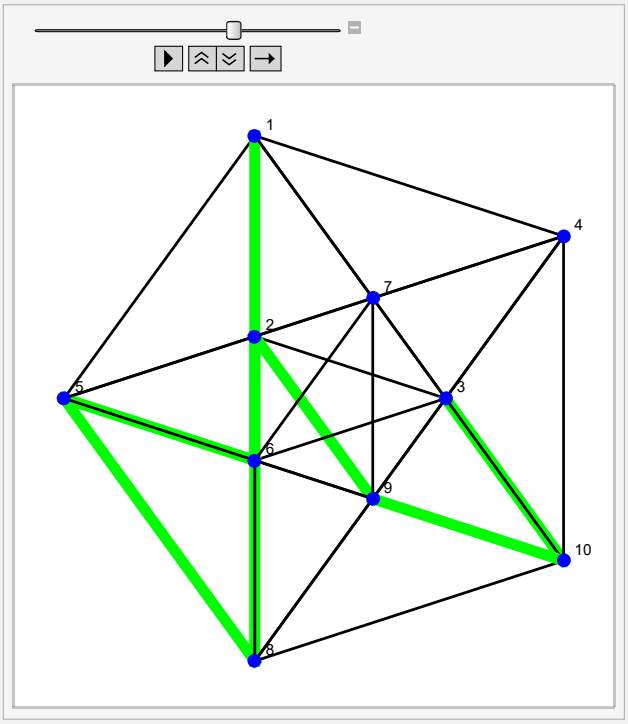
In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[G = SetGraphOptions[LineGraph[CompleteGraph[5]],
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
CircuitosHamilton[G, 5]
CircuitosHamilton[G, 5, ruta->True]
```

Out[ ] :=



{{1 ●—● 6, 6 ●—● 5, 5 ●—● 9, 9 ●—● 2, 2 ●—● 8, 8 ●—● 3, 3 ●—● 10, 10 ●—● 4, 4 ●—● 7, 7 ●—● 1}, {1 ●—● 6, 6 ●—● 3, 3 ●—● 10, 10 ●—● 4, 4 ●—● 9, 9 ●—● 2, 2 ●—● 8, 8 ●—● 5, 5 ●—● 7, 7 ●—● 1}, {1 ●—● 6, 6 ●—● 5, 5 ●—● 9, 9 ●—● 2, 2 ●—● 8, 8 ●—● 10, 10 ●—● 3, 3 ●—● 4, 4 ●—● 7, 7 ●—● 1}, {1 ●—● 6, 6 ●—● 5, 5 ●—● 8, 8 ●—● 2, 2 ●—● 9, 9 ●—● 10, 10 ●—● 3, 3 ●—● 4, 4 ●—● 7, 7 ●—● 1}, {1 ●—● 6, 6 ●—● 5, 5 ●—● 8, 8 ●—● 2, 2 ●—● 9, 9 ●—● 4, 4 ●—● 3, 3 ●—● 10, 10 ●—● 7, 7 ●—● 1}}
{{1 ●—● 6, 6 ●—● 5, 5 ●—● 9, 9 ●—● 2, 2 ●—● 8, 8 ●—● 3, 3 ●—● 10, 10 ●—● 4, 4 ●—● 7, 7 ●—● 1}, {1 ●—● 6, 6 ●—● 3, 3 ●—● 10, 10 ●—● 4, 4 ●—● 9, 9 ●—● 2, 2 ●—● 8, 8 ●—● 5, 5 ●—● 7, 7 ●—● 1}, {1 ●—● 6, 6 ●—● 5, 5 ●—● 9, 9 ●—● 2, 2 ●—● 8, 8 ●—● 10, 10 ●—● 3, 3 ●—● 4, 4 ●—● 7, 7 ●—● 1}, {1 ●—● 6, 6 ●—● 5, 5 ●—● 8, 8 ●—● 2, 2 ●—● 9, 9 ●—● 10, 10 ●—● 3, 3 ●—● 4, 4 ●—● 7, 7 ●—● 1}, {1 ●—● 6, 6 ●—● 5, 5 ●—● 8, 8 ●—● 2, 2 ●—● 9, 9 ●—● 4, 4 ●—● 3, 3 ●—● 10, 10 ●—● 7, 7 ●—● 1}}
Ruta de la animación: {1 ●—● 6, 6 ●—● 5, 5 ●—● 8, 8 ●—● 2, 2 ●—● 9, 9 ●—● 10, 10 ●—● 3, 3 ●—● 4, 4 ●—● 7, 7 ●—● 1}



Explicación en video

74) **RutaHamiltonQ**: función booleana que retorna "`True`" si un **grafo simple no dirigido** "G" recibido como parámetro **posee** una **ruta de Hamilton** o "`False`", en caso contrario.

Sintaxis: `RutaHamiltonQ[G]`.

Ejemplo 7.147

Un grafo con aristas: {{a, b}, {a, c}, {c, b}, {b, d}, {c, d}}, ¿tiene una ruta de *Hamilton*?

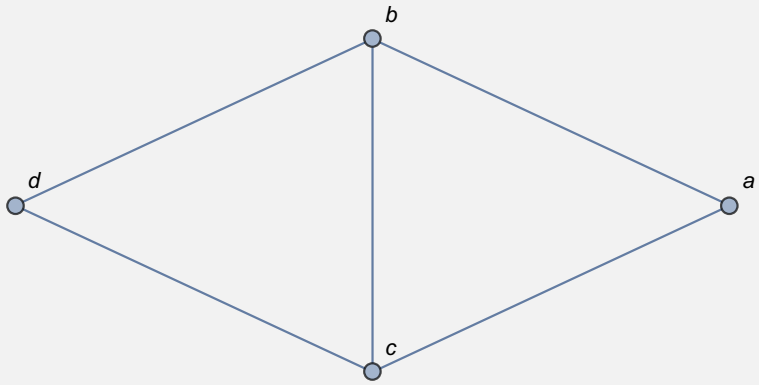
Solución:

Al recurrir al comando **`RutaHamiltonQ`**, se tiene:

In[ ] :=

```
grafo = Grafo[{{a, b}, {a, c}, {c, b}, {b, d}, {c, d}}]
RutaHamiltonQ[grafo]
```

Out[ ] :=



True

Sí posee rutas hamiltonianas.

Ejemplo 7.148

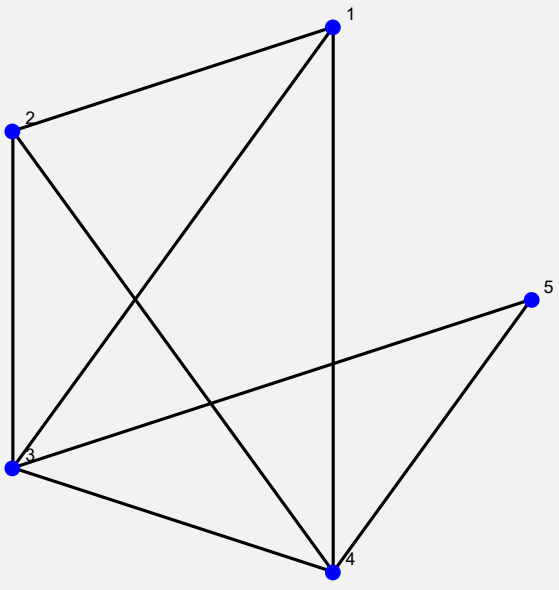
Contiene rutas de *Hamilton* un grafo cuyos lados corresponden a: {{1, 2}, {1, 3}, {3, 2}, {2, 4}, {3, 4}, {3, 5}, {5, 4}, {4, 1}}. Construya el grafo en el ambiente facilitado por el paquete "Combinatorica".

Solución:

In[ ] :=

```
GrafoC[{{1, 2}, {1, 3}, {3, 2}, {2, 4}, {3, 4}, {3, 5}, {5, 4}, {4, 1}}]
RutaHamiltonQ[G]
```

Out[ ] :=



True

Explicación en video

75) **RutaHamilton**: **retorna** una **ruta de Hamilton** sobre un grafo "G" **simple no dirigido** que la contenga. El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, a través del uso del **paquete** "Combinatorica". El comando brinda la **propiedad** "`ruta -> True`" que retorna una **animación** con la **ruta correspondiente** sobre "G".

Sintaxis: `RutaHamilton[G]`, o bien, `RutaHamilton[G, ruta -> True]`.

Ejemplo 7.149

Encuentre una ruta de *Hamilton* en el grafo del ejercicio anterior. Muestre el camino mediante una animación.

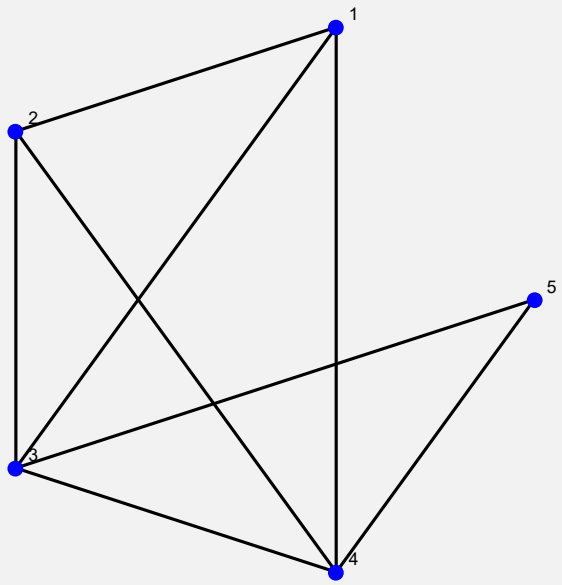
Solución:

Al invocar **`RutaHamilton`**:

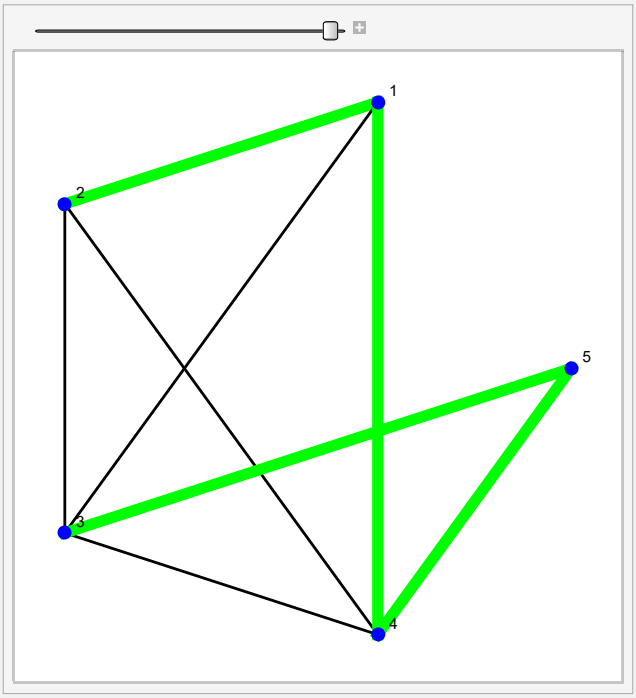
In[ ] :=

```
GrafoC[{{1, 2}, {1, 3}, {3, 2}, {2, 4}, {3, 4}, {3, 5}, {5, 4}, {4, 1}}]
RutaHamilton[G]
RutaHamilton[G, ruta -> True]
```

Out[ ] :=



{{3, 5}, {5, 4}, {4, 1}, {1, 2}}



Ejemplo 7.150

Construya el grafo dodecaedro a través del "Wolfram System" de *Mathematica*. Halle sobre este grafo, una ruta de *Hamilton* y muéstrela por medio de una animación.

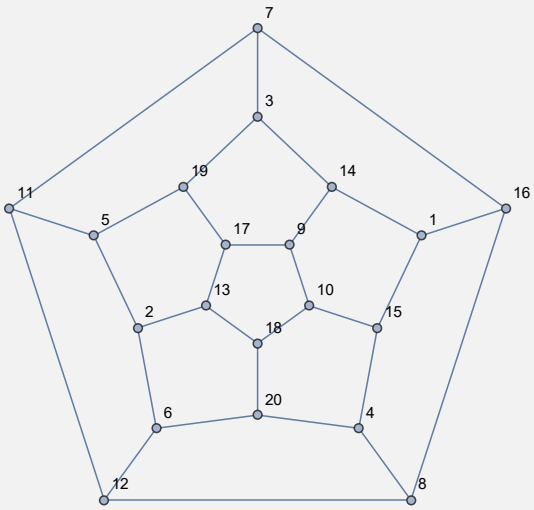
Solución:

Para crear el grafo se utilizará la instrucción **`GrafoDato`**:

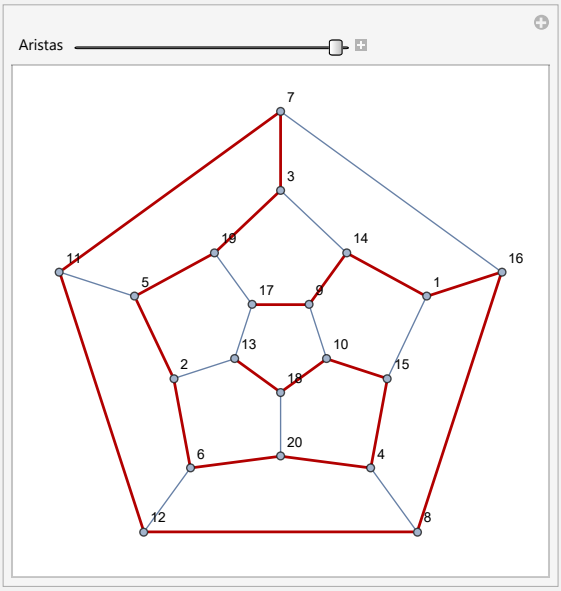
In[ ] :=

```
grafo = GrafoDato[tipo -> "DodecahedralGraph"]
RutaHamilton[grafo]
RutaHamilton[grafo, ruta -> True]
```

Out[ ] :=



{{13, 18}, {18, 10}, {10, 15}, {15, 4}, {4, 20}, {20, 6}, {6, 2}, {2, 5}, {5, 19}, {19, 3}, {3, 7}, {7, 11}, {11, 12}, {12, 8}, {8, 16}, {16, 1}, {1, 14}, {14, 9}, {9, 17}}



Explicación en video

76) **ExisteCircuitoHamilton**: **determina** la **existencia** de un **circuito de Hamilton** en un grafo "G" **no dirigido**, **simple**, **conexo** con una cantidad de vértices **mayor o igual a tres**, por medio de la **propiedad de Ore**, la **propiedad de Dirac** y la **cantidad de aristas** del grafo. Si alguna de las condiciones suficientes **no se satisface**, la instrucción encuentra un **contraejemplo** y si ninguna de ellas es válida, el comando **retorna**: "No hay criterio". Acepta grafos **creados** con o sin "Combinatorica".

Sintaxis: `ExisteCircuitoHamilton[G]`.

Ejemplo 7.151

Sobre un grafo con aristas: {{a, b}, {a, c}, {c, b}, {b, d}, {c, d}, {d, f}, {f, e}, {e, b}, {e, d}, {e, h}, {h, j}, {j, e}, {a, k}}, emplee el comando **ExisteCircuitoHamilton** para garantizar o no, la existencia de un circuito hamiltoniano.

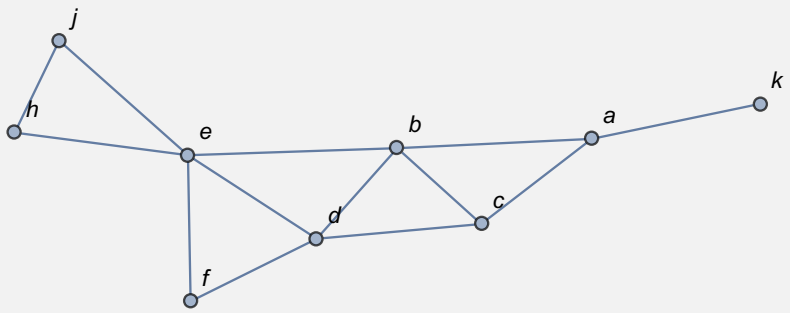
Solución:

Al correr **ExisteCircuitoHamilton**, se obtiene:

In[ ] :=

```
grafo = Grafo[{{a, b}, {a, c}, {c, b}, {b, d}, {c, d}, {d, f}, {f,
e}, {e, b}, {e, d}, {e, h}, {h, j}, {j, e}, {a, k}}]
ExisteCircuitoHamilton[grafo]
```

Out[ ] :=



No se satisface la propiedad de Ore, un contraejemplo ocurre en: Valencia[a] + Valencia[d] = 7 $<$ n = 9

No se cumple la propiedad de Dirac, un contrajemplo ocurre en: Valencia[a] = 3 $<$ $\frac{n}{2} = \frac{9}{2}$

No se satisface la propiedad de la cantidad de aristas, m = 13 $<$ $\frac{n^2-3n+6}{2}$ = 30

No hay criterio, sin embargo, el valor lógico sobre la existencia de un circuito de Hamilton en el grafo es: False

Ejemplo 7.152

En "Combinatorica" construya un grafo cuyos lados son: {{1, 2}, {1, 3}, {3, 2}, {2, 4}, {3, 4}, {3, 5}, {5, 4}, {4, 1}}. Justifique la existencia de circuitos de *Hamilton* sobre este grafo.

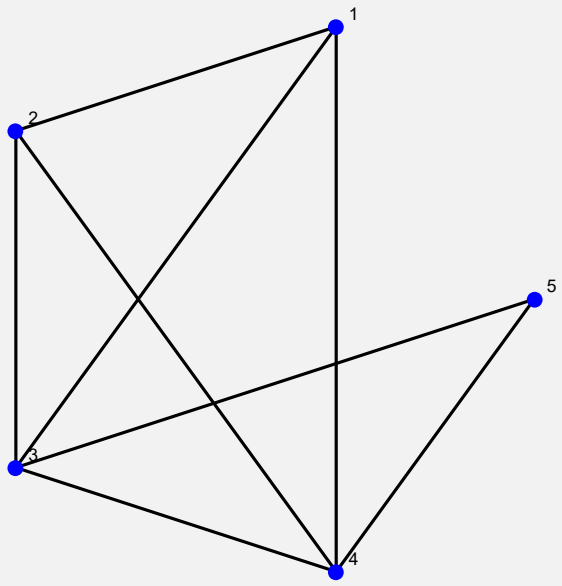
Solución:

En el software:

In[ ] :=

```
GrafoC[{{1, 2}, {1, 3}, {3, 2}, {2, 4}, {3, 4}, {3, 5}, {5, 4}, {4, 1}}]
ExisteCircuitoHamilton[G]
```

Out[ ] :=



No se cumple la propiedad de Dirac, un contrajemplo ocurre en: Valencia[5] = $2 < \frac{n}{2} = \frac{5}{2}$

Se satisface la propiedad de Ore

Se satisface la propiedad de la cantidad de aristas, m = $8 \geq \frac{n^2-3n+6}{2} = 8$

Existe un circuito de Hamilton

Explicación en video

77) **AgenteViajero**: **resuelve** el **problema del agente viajero** sobre un **grafo** "G" **simple**, **no dirigido creado** en el "Wolfram System", o bien, con el **paquete** "Combinatorica". Es decir, la instrucción **encuentra** en caso de existir, un **circuito de Hamilton** de **longitud más corta**. Si el grafo **no** es **ponderado** la **longitud** se asume en **función** de la **cantidad** de **aristas** involucradas. Presenta la **opción** "`ruta -> True`" que muestra adicionalmente una **animación** del **circuito encontrado**.

Sintaxis: `AgenteViajero[G]`, o bien, `AgenteViajero[G, ruta -> True]`. El comando **retorna** la **longitud del circuito** y su **recorrido** representado mediante una secuencia de aristas.

Ejemplo 7.153

Resuelva el problema del agente viajero sobre un grafo de "Combinatorica", cuyas aristas vienen dadas por: {{1, 2}, {1, 3}, {3, 2}, {2, 4}, {3, 4}, {3, 5}, {5, 4}, {4, 1}}, con pesos seudoaleatorios reales de uno a diez. Muestre un circuito hamiltoniano de longitud mínima.

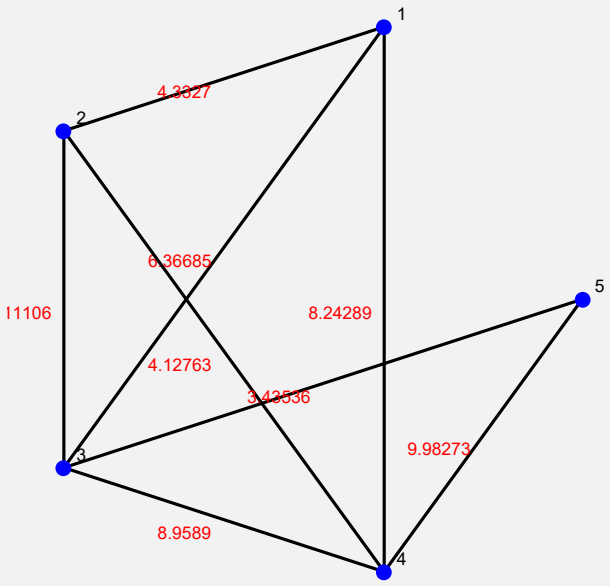
Solución:

En *Mathematica*:

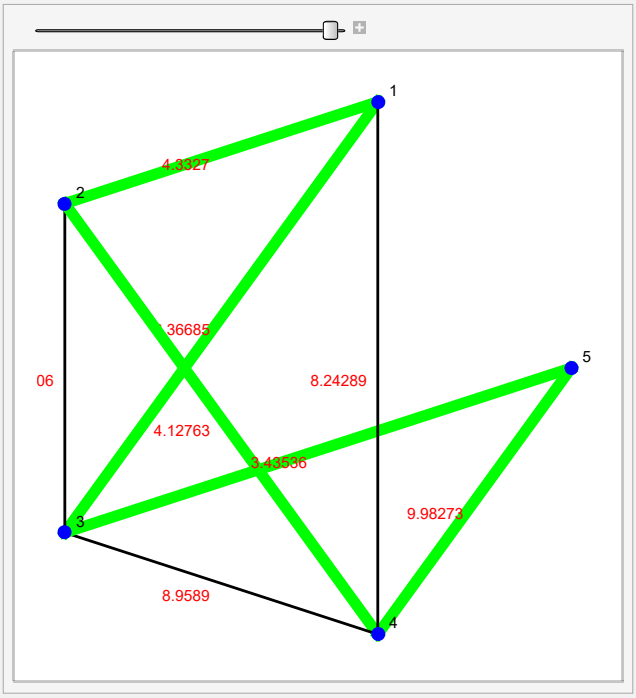
In[ ] :=

```
GrafoC[{{1, 2}, {1, 3}, {3, 2}, {2, 4}, {3, 4}, {3, 5}, {5, 4}, {4,
1}}, pesos -> RandomReal[{1, 10}, 8], mostrarpesos -> True]
AgenteViajero[G]
AgenteViajero[G, ruta -> True]
```

Out[ ] :=



{28.2453, {{1, 2}, {2, 4}, {4, 5}, {5, 3}, {3, 1}}}
{28.2453, {{1, 2}, {2, 4}, {4, 5}, {5, 3}, {3, 1}}}



Ejemplo 7.154

Resuelva el problema del agente viajero sobre un grafo completo de orden 10. Muestre un circuito de *Hamilton* de longitud más corta, mediante una animación.

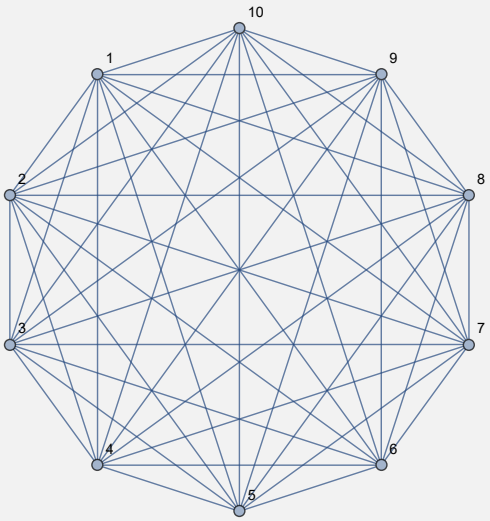
Solución:

En el software:

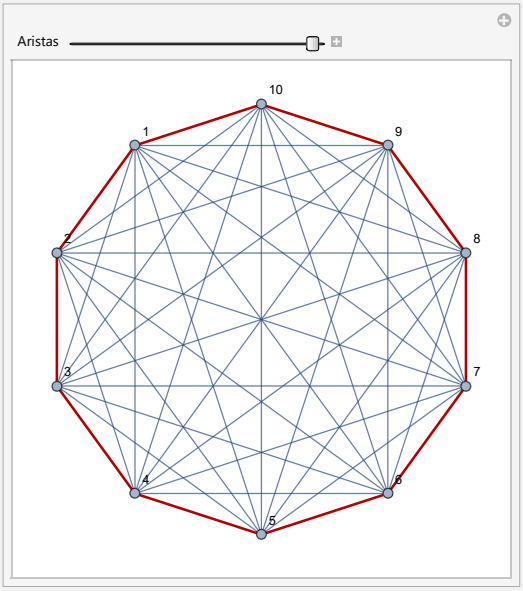
In[ ] :=

```
grafo = GrafoCompleto[10]
AgenteViajero[grafo]
AgenteViajero[grafo, ruta -> True]
```

Out[ ] :=



{10, {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6}, {6, 7}, {7, 8}, {8, 9}, {9, 10}, {10, 1}}}
{10, {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6}, {6, 7}, {7, 8}, {8, 9}, {9, 10}, {10, 1}}}



N El peso 10 arrojado como salida en este ejemplo, corresponde a la cantidad de aristas empleadas en el circuito de *Hamilton* hallado, dado que el grafo no es ponderado.

Explicación en video

78) **CDFAgenteViajeroGrupos**: **genera** una **animación** donde al **seleccionar** un **grupo de países**, se **construye automáticamente** un **circuito de Hamilton** para recorrer cada país y se muestra sobre un **mapa geográfico** con la **distancia total** transitada. En caso de **no existir**, el comando determina un **camino hamiltoniano**.

Sintaxis: `CDFAgenteViajeroGrupos[]`.

Ejemplo 7.155

Seleccione del combo mostrado por **`CDFAgenteViajeroGrupos[]`**, el grupo de países de América Central. Observe el circuito de *Hamilton* obtenido en la salida y la distancia total.

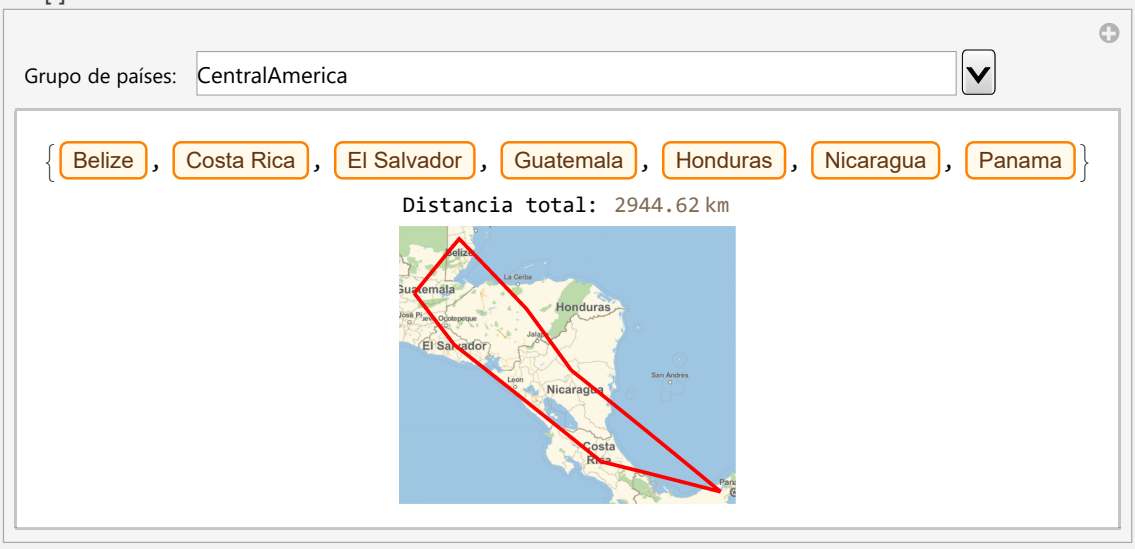
Solución:

En *Mathematica*:

In[ ] :=

```
CDFAgenteViajeroGrupos[]
```

Out[ ] :=



Ejemplo 7.156

Resuelva el problema del agente viajero para los países de Europa Central.

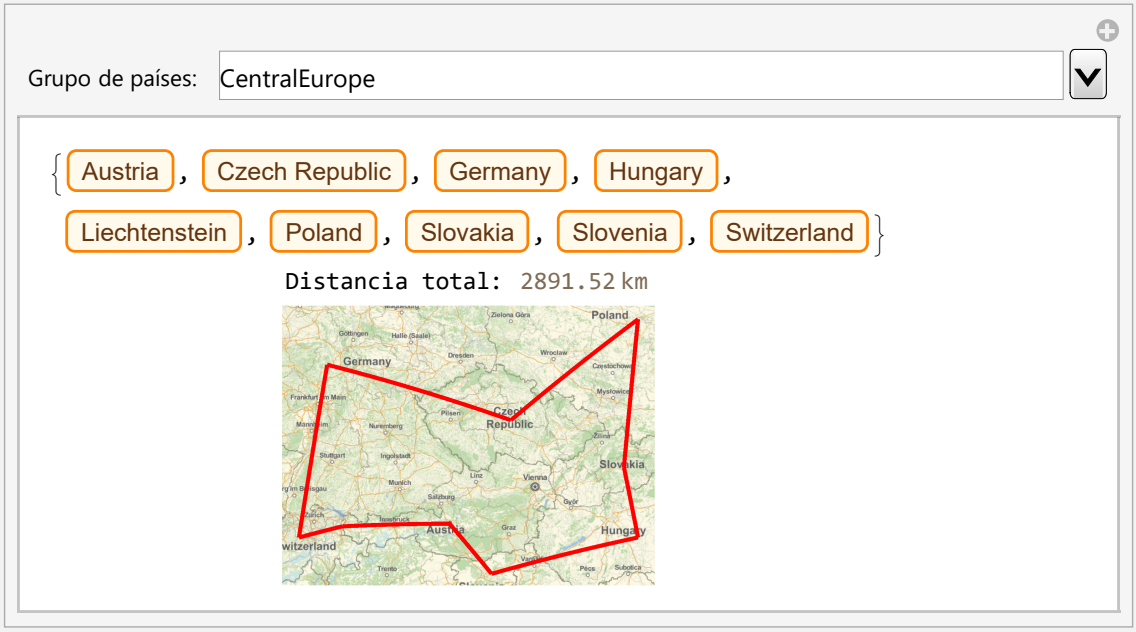
Solución:

En el combo de la animación se escoge "CentralEurope", luego:

In[ ] :=

```
CDFAgenteViajeroGrupos[]
```

Out[ ] :=



Explicación en video

79) **CDFAgenteViajeroRegiones**: **construye** una **animación** donde al **seleccionar** un **país**, se **muestra automáticamente** sobre un **mapa geográfico**, un **circuito de Hamilton** para recorrer todas sus regiones, **añadiendo información** con respecto a la **distancia acumulada**. En caso de **no existir**, la instrucción determina un **camino hamiltoniano**.

Sintaxis: `CDFAgenteViajeroRegiones[]`.

Ejemplo 7.157

Considere el problema del agente viajero sobre las regiones principales de Costa Rica, utilice el comando **`CDFAgenteViajeroRegiones`** para resolverlo.

Solución:

En el software:

In[ ] :=

```
CDFAgenteViajeroRegiones[]
```

Out[ ] :=

Ejemplo 7.158

Solucione el problema del agente viajero en las regiones de República Dominicana.

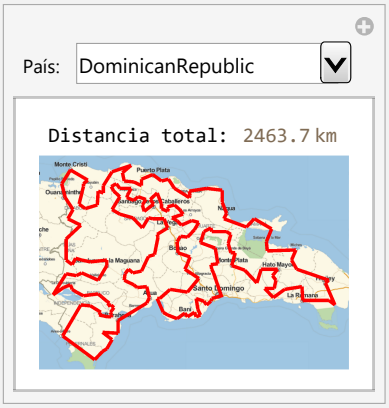
Solución:

Al emplear **`CDFAgenteViajeroRegiones`**, se selecciona del combo "DominicanRepublic", entonces:

In[ ] :=

```
CDFAgenteViajeroRegiones[]
```

Out[ ] :=



Explicación en video

80) **ListaVertices**: **retorna** la **lista** de **todos** los **vértices** de un **grafo** "G" **creado** en el "Wolfram System" de *Mathematica*, o bien, mediante el uso del **paquete** "Combinatorica". Brinda la **opción** "**`cantidad->True`**" que **devuelve** un **vector** con el **número de nodos** del grafo y la **lista de vértices**.

Sintaxis: **`ListaVertices[G]`**, o bien, **`ListaVertices[G, cantidad->True]`**.

Ejemplo 7.159

Mediante el uso de **`ListaVertices`** en un grafo creado al invocar **`GrafoRandomConexo[20, 30, simple->False]`**, halle la lista y la cantidad de nodos.

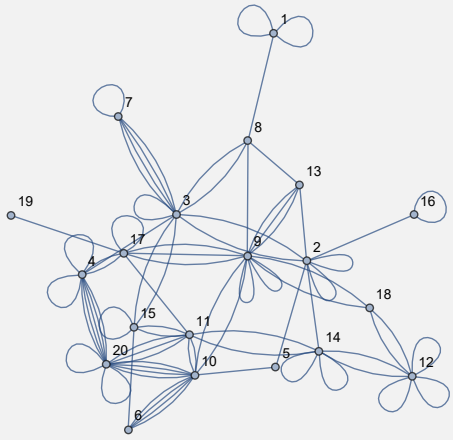
Solución:

En el software se recurrirá a la opción "**`cantidad->True`**" de **`ListaVertices`**:

In[ ] :=

```
grafo = GrafoRandomConexo[20, 30, simple->False]
ListaVertices[grafo, cantidad->True]
```

Out[ ] :=



{20, {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}}

N El primer dato (20) de la lista, corresponde a la cantidad de vértices del grafo.

Ejemplo 7.160

Halle la lista y la cantidad de nodos en el grafo dodecaedro, creado a través del paquete "Combinatorica".

Solución:

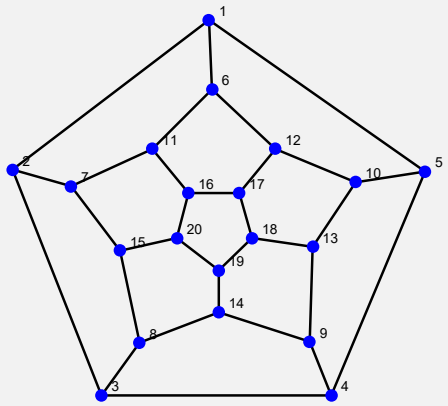
En *Mathematica*:

In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
```

```
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
ListaVertices[grafo]
ListaVertices[grafo, cantidad->True]
```

Out[ ] :=



{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
{20, {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}}

Explicación en video

81) **ListaAristas**: **retorna** la **lista** de **todos** los **lados** de un **grafo** "G" **creado** en el "Wolfram System" de *Mathematica*, o bien, mediante el uso del **paquete** "Combinatorica". Proporciona la **opción** "**`cantidad->True`**" que **muestra** un **vector** con el **número de aristas** del grafo y la **lista de lados**.

Sintaxis: **`ListaAristas[G]`**, o bien, **`ListaAristas[G, cantidad->True]`**.

Ejemplo 7.161

Encuentre la lista y la cantidad de aristas de un grafo creado al invocar **`GrafoRandomConexo[20, 30, simple->False, combinatorica->True]`**.

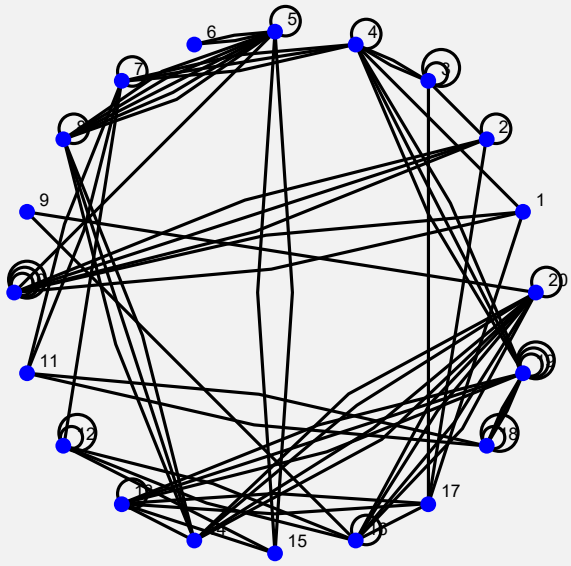
Solución:

Al usar la instrucción **`ListaAristas`**, se tiene:

In[ ] :=

```
GrafoRandomConexo[20, 30, simple->False, combinatorica->True]
ListaAristas[G, cantidad->True]
```

Out[ ] :=



{80, {{1, 4}, {1, 10}, {1, 17}, {2, 3}, {2, 10}, {2, 17}, {3, 4}, {3, 17}, {4, 7}, {4, 19}, {5, 6}, {5, 8}, {5, 10}, {5, 15}, {7, 11}, {7, 12}, {8, 14}, {9, 16}, {9, 20}, {11, 18}, {12, 15}, {12, 16}, {13, 14}, {13, 15}, {13, 17}, {13, 19}, {14, 20}, {16, 17}, {16, 20}, {18, 19}, {4, 19}, {7, 11}, {13, 19}, {5, 15}, {18, 19}, {1, 10}, {5, 8}, {5, 8}, {12, 16}, {16, 20}, {13, 19}, {2, 10}, {8, 14}, {3, 4}, {4, 7}, {4, 19}, {5, 8}, {16, 20}, {4, 7}, {5, 6}, {2, 10}, {13, 17}, {11, 18}, {14, 20}, {8, 14}, {16, 20}, {18, 19}, {5, 8}, {14, 20}, {14, 20}, {19, 19}, {8, 8}, {18, 18}, {19, 19}, {3, 3}, {10, 10}, {19, 19}, {18, 18}, {16, 16}, {2, 2}, {5, 5}, {12, 12}, {10, 10}, {3, 3}, {20, 20}, {13, 13}, {10, 10}, {12, 12}, {7, 7}, {4, 4}}}

Se concluye que el grafo seudoaleatorio posee ochenta lados.

Ejemplo 7.162

¿Cuántas aristas contiene el grafo dodecaedro? Constrúyalo en el "Wolfram System" de *Mathematica*.

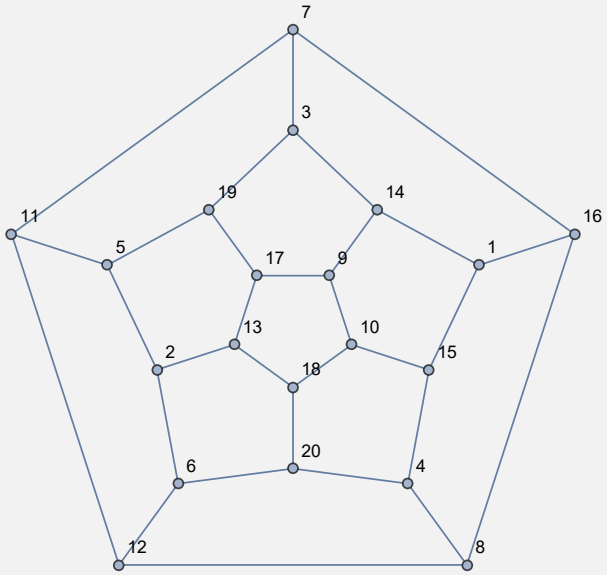
Solución:

Al emplear el comando **`GrafoDato`**:

In[ ] :=

```
grafo = GrafoDato[tipo -> "DodecahedralGraph"]
ListaAristas[grafo, cantidad -> True]
```

Out[ ] :=



{30, {{1, 14}, {1, 15}, {1, 16}, {2, 5}, {2, 6}, {2, 13}, {3, 7}, {3, 14}, {3, 19}, {4, 8}, {4, 15}, {4, 20}, {5, 11}, {5, 19}, {6, 12}, {6, 20}, {7, 11}, {7, 16}, {8, 12}, {8, 16}, {9, 10}, {9, 14}, {9, 17}, {10, 15}, {10, 18}, {11, 12}, {13, 17}, {13, 18}, {17, 19}, {18, 20}}}

El grafo tiene treinta lados.

Explicación en video

82) **LongitudCaminoMC**: **retorna** la **longitud** y **una ruta** de un **camino más corto** entre **dos vértices** "a" y "b" **distintos** de un **grafo** "G", **creado** con o sin "Combinatorica". Integra la **opción** "`ruta->True`" que adicionalmente, muestra una **animación** de un **camino** de **longitud más corta**. Si el grafo **no tiene pesos**, determina la ruta en **función** de la **cantidad** de **aristas** involucradas.

Sintaxis: `LongitudCaminoMC[G, a, b]`, o bien, `LongitudCaminoMC[G, a, b, ruta->True]`.

Ejemplo 7.163

Considerando el nodo 1 y 10, determine una ruta más corta y su longitud, sobre un grafo generado al invocar `GrafoRandomConexo[10, 10, simple->False]`.

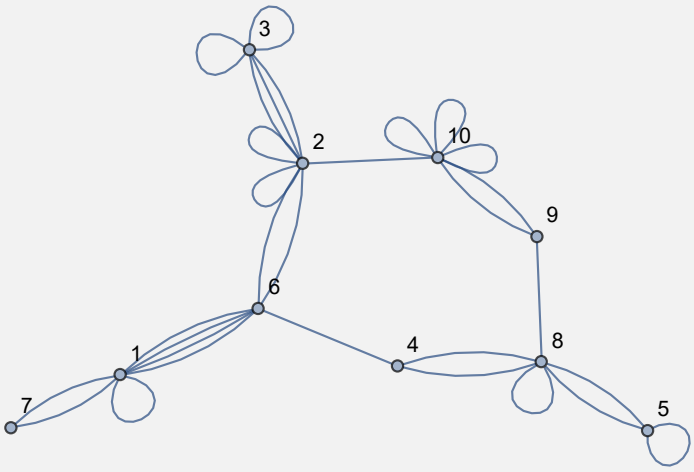
Solución:

La instrucción **LongitudCaminoMC** resuelve lo solicitado en el ejemplo:

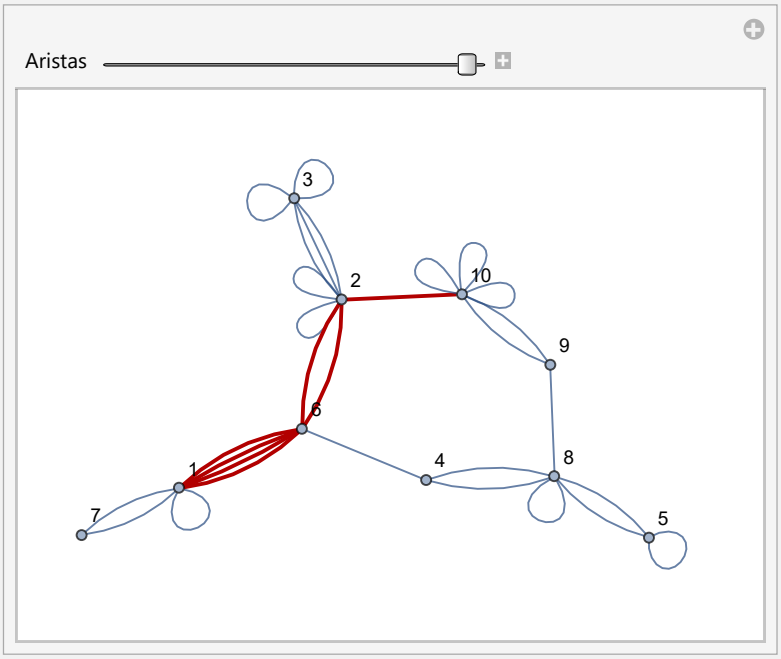
In[ ] :=

```
grafo = GrafoRandomConexo[10, 10, simple->False]
LongitudCaminoMC[grafo, 1, 10, ruta->True]
```

Out[ ] :=



{3, {{1, 6}, {6, 2}, {2, 10}}}



N Como el grafo no es ponderado, la longitud más corta 3, se determinó de acuerdo con la cantidad de lados involucrados en la ruta. Por otra parte, el grafo presenta aristas múltiples, de donde al recorrer en la animación una de ellas, se manifiestan todas de color rojo, dando a entender al usuario que cualquiera es elegible durante la trayectoria.

Ejemplo 7.164

Construya un grafo en "Combinatorica" con los lados del octaedro, asignando pesos seudo-aleatorios reales de uno a diez. Con respecto a los vértices 2 y 6, halle un camino más corto y muéstrelo a través de una animación.

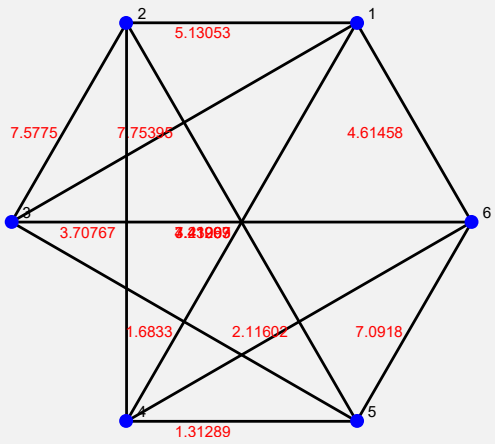
Solución:

En el software:

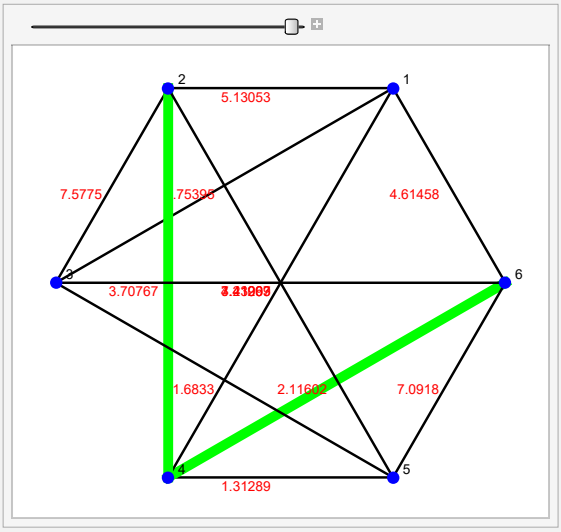
In[ ] :=

```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[OctahedralGraph,
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1];
GrafoC[ListaAristas[grafo], pesos->RandomReal[{1, 10},
ListaAristas[grafo, cantidad->True][[1]]], mostrarpesos->True]
LongitudCaminoMC[G, 2, 6, ruta->True]
```

Out[ ] :=



{5.82368, {{2, 4}, {4, 6}}}



Se aclara al lector que la instrucción **`ListaAristas[grafo, cantidad->True][[1]]`** retorna del vector **`ListaAristas[grafo, cantidad->True]`** su primera componente, es decir, devuelve la cantidad de lados del grafo.

Explicación en video

83) **GrafoSocial**: **crea** un **grafo social** de **amistades** con relación a **dos networking** elegidas por el usuario: "Facebook" o "Twitter".

Sintaxis: **`GrafoSocial["Facebook"]`**, o bien, **`GrafoSocial["Twitter"]`**.

Ejemplo 7.165

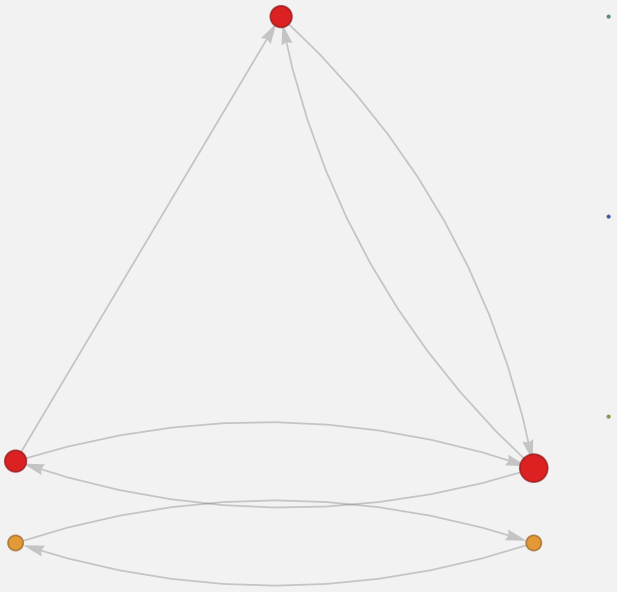
Genere el grafo social vinculado con "Twitter".

Solución:

In[ ] :=

```
GrafoSocial["Twitter"]
```

Out[ ] :=



N El grafo social anterior, se construyó autorizando al software *Mathematica* su acceso a la cuenta de interés, en este caso, los datos de "Twitter" del autor del presente trabajo. Al correr **`GrafoSocial["Twitter"]`** automáticamente se solicita el permiso correspondiente al usuario. Naturalmente, el estudiante obtendrá otro grafo distinto, de acuerdo con sus gustos y preferencias asociados a su cuenta personal de "Twitter".

Ejemplo 7.166

Construya el grafo social de amistades de "Facebook".

Solución:

En *Mathematica*, se procede así:

In[ ] :=

```
GrafoSocial["Facebook"]
```

N No se comparte el *Out*[ ] exhibido, dado que arroja una imagen en blanco. *Facebook* ha cambiado sus políticas de *API* (*Application Programming Interface*) por lo que actualmente devuelve un conjunto de datos mucho más restringido. Momentáneamente esta opción de **`GrafoSocial`** ha quedado supeditada a la normalización de estas políticas.

Explicación en video

84) **AlDijkstra**: ejecuta **paso a paso** el **algoritmo de Dijkstra** o de la **longitud del camino más corto** con respecto a **dos vértices distintos** "a" y "b", sobre un **grafo** "G" **conexo** y **simple**, creado o no, a través del **paquete** "Combinatorica". Retorna además, una ruta de **recorrido más corto**.

Sintaxis: **`AlDijkstra[G, a, b]`**.

Ejemplo 7.167

Considere un grafo con aristas: {{a, b}, {a, c}, {c, b}, {b, d}, {c, d}, {d, f}, {b, g}} y pesos seudoaleatorios reales de uno a diez. Muestre iteración por iteración el uso del algoritmo de *Dijkstra* para encontrar la longitud del camino más corto entre los nodos "a" y "f".

Solución:

Al utilizar la instrucción **`AlDijkstra`**, se obtiene:

In[ ] :=

```
grafo = Grafo[{{a, b}, {a, c}, {c, b}, {b, d}, {c, d}, {d, f}, {b, g}}, pesos -> RandomReal[{1, 10}, 7], mostrarpesos -> True]
AlDijkstra[grafo, a, f]
```

Out[ ] :=

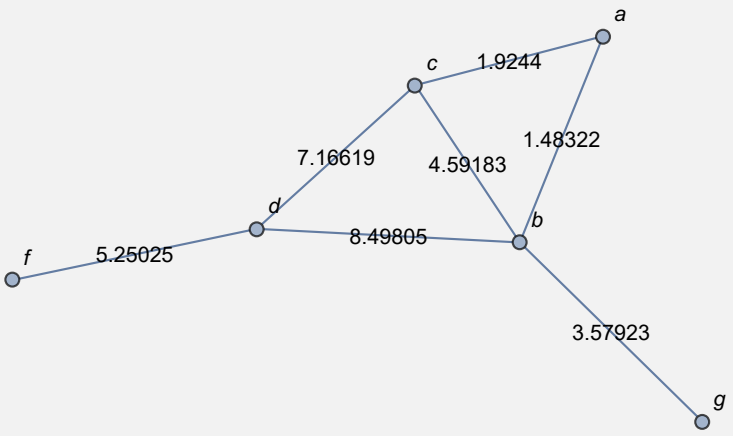


Tabla de pesos del grafo

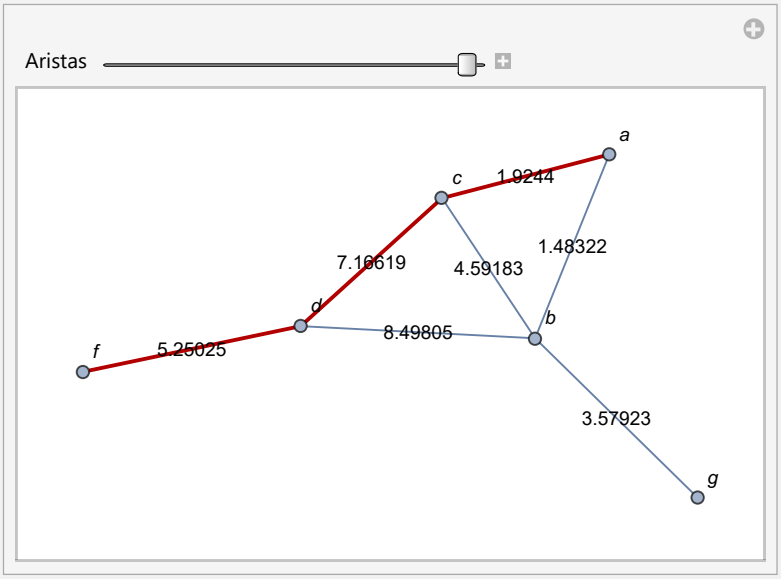
	a	b	c	d	f	g
a	0	1.48322	1.9244	0	0	0
b	1.48322	0	4.59183	8.49805	0	3.57923
c	1.9244	4.59183	0	7.16619	0	0
d	0	8.49805	7.16619	0	5.25025	0
f	0	0	0	5.25025	0	0
g	0	3.57923	0	0	0	0

Inicialización: {{a, 0}, {b, ∞}, {c, ∞}, {d, ∞}, {f, ∞}, {g, ∞}}
Se seleccionó el nodo: a
Lista actual de vértices: {b, c, d, f, g}
Lista actual de marcas: {{b, 1.48322}, {c, 1.9244}, {d, ∞}, {f, ∞}, {g, ∞}}
Se seleccionó el nodo: b
Lista actual de vértices: {c, d, f, g}
Lista actual de marcas: {{c, 1.9244}, {d, 9.98127}, {f, ∞}, {g, 5.06246}}
Se seleccionó el nodo: c
Lista actual de vértices: {d, f, g}
Lista actual de marcas: {{d, 9.09059}, {f, ∞}, {g, 5.06246}}
Se seleccionó el nodo: g
Lista actual de vértices: {d, f}
Lista actual de marcas: {{d, 9.09059}, {f, ∞}}
Se seleccionó el nodo: d
Lista actual de vértices: {f}
Lista actual de marcas: {{f, 14.3408}}
Se seleccionó el nodo: f
La longitud del camino más corto es: 14.3408
Lista actual de vértices: {}
Lista actual de marcas: {}

En resumen:

a	b	c	d	f	g
0	∞	∞	∞	∞	∞
0	1.48322	1.9244	∞	∞	∞
0	1.48322	1.9244	9.98127	∞	5.06246
0	1.48322	1.9244	9.09059	∞	5.06246
0	1.48322	1.9244	9.09059	∞	5.06246
0	1.48322	1.9244	9.09059	14.3408	5.06246
0	1.48322	1.9244	9.09059	14.3408	5.06246

{14.3408, {{a, c}, {c, d}, {d, f}}}



Ejemplo 7.168

Aplique paso a paso el algoritmo de *Dijkstra* sobre un grafo con aristas: {{a, b}, {a, c}, {a, f}, {a, g}, {b, c}, {b, d}, {b, e}, {c, d}, {c, e}, {c, f}, {c, g}, {d, e}, {d, h}, {e, f}, {e, g}, {e, h}, {f, g}, {f, h}, {g, h}} y pesos seudoaleatorios reales de uno a diez, tomando para ello los vértices "a" y "h".

Solución:

En *Mathematica*:

In[ ] :=

```
grafo = Grafo[{{a, b}, {a, c}, {a, f}, {a, g}, {b, c}, {b, d}, {b,
e}, {c, d}, {c, e}, {c, f}, {c, g}, {d, e}, {d, h}, {e, f}, {e, g},
{e, h}, {f, g}, {f, h}, {g, h}}, pesos -> RandomReal[{1, 10}, 19],
mostrarpesos -> True]
AlDijkstra[grafo, a, h]
```

Out[ ] :=

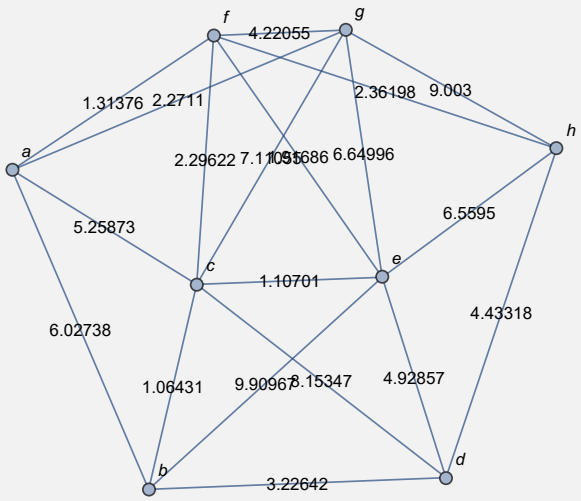


Tabla de pesos del grafo

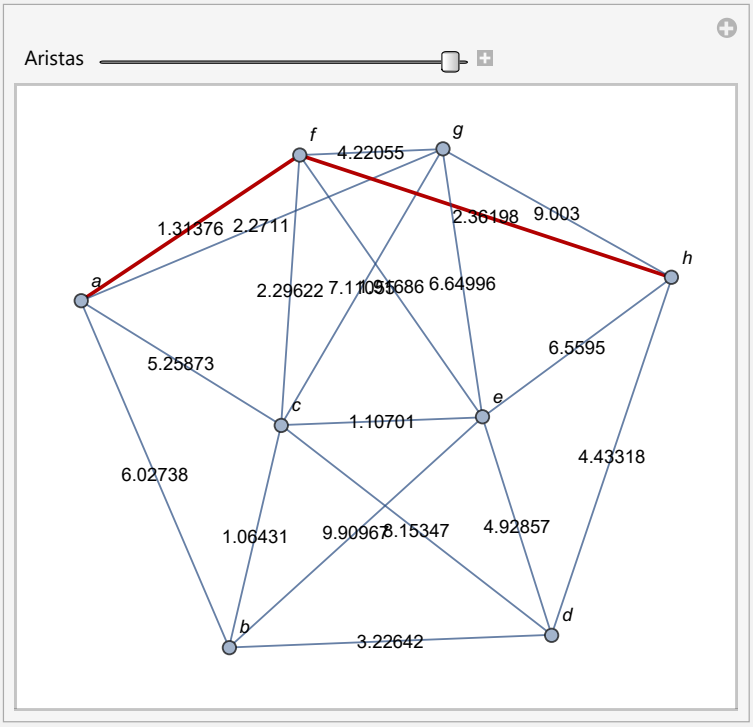
	a	b	c	f	g	d	e	h
a	0	6.02738	5.25873	1.31376	2.2711	0	0	0
b	6.02738	0	1.06431	0	0	3.22642	9.90967	0
c	5.25873	1.06431	0	2.29622	7.11055	8.15347	1.10701	0
f	1.31376	0	2.29622	0	4.22055	0	1.91686	2.36198
g	2.2711	0	7.11055	4.22055	0	0	6.64996	9.003
d	0	3.22642	8.15347	0	0	0	4.92857	4.43318
e	0	9.90967	1.10701	1.91686	6.64996	4.92857	0	6.5595
h	0	0	0	2.36198	9.003	4.43318	6.5595	0

Inicialización: {{a, 0}, {b, ∞}, {c, ∞}, {f, ∞}, {g, ∞}, {d, ∞}, {e, ∞}, {h, ∞}}
Se seleccionó el nodo: a
Lista actual de vértices: {b, c, f, g, d, e, h}
Lista actual de marcas: {{b, 6.02738}, {c, 5.25873}, {f, 1.31376}, {g, 2.2711}, {d, ∞}, {e, ∞}, {h, ∞}}
Se seleccionó el nodo: f
Lista actual de vértices: {b, c, g, d, e, h}
Lista actual de marcas: {{b, 6.02738}, {c, 3.60998}, {g, 2.2711}, {d, ∞}, {e, 3.23062}, {h, 3.67574}}
Se seleccionó el nodo: g
Lista actual de vértices: {b, c, d, e, h}
Lista actual de marcas: {{b, 6.02738}, {c, 3.60998}, {d, ∞}, {e, 3.23062}, {h, 3.67574}}
Se seleccionó el nodo: e
Lista actual de vértices: {b, c, d, h}
Lista actual de marcas: {{b, 6.02738}, {c, 3.60998}, {d, 8.15919}, {h, 3.67574}}
Se seleccionó el nodo: c
Lista actual de vértices: {b, d, h}
Lista actual de marcas: {{b, 4.67429}, {d, 8.15919}, {h, 3.67574}}
Se seleccionó el nodo: h
La longitud del camino más corto es: 3.67574
Lista actual de vértices: {b, d}
Lista actual de marcas: {{b, 4.67429}, {d, 8.10892}}

En resumen:

a	b	c	f	g	d	e	h
0	∞	∞	∞	∞	∞	∞	∞
0	6.02738	5.25873	1.31376	2.2711	∞	∞	∞
0	6.02738	3.60998	1.31376	2.2711	∞	3.23062	3.67574
0	6.02738	3.60998	1.31376	2.2711	∞	3.23062	3.67574
0	6.02738	3.60998	1.31376	2.2711	8.15919	3.23062	3.67574
0	4.67429	3.60998	1.31376	2.2711	8.15919	3.23062	3.67574
0	4.67429	3.60998	1.31376	2.2711	8.10892	3.23062	3.67574

{3.67574, {{a, f}, {f, h}}}



Explicación en video

85) **GrafoPonderadoNQ**: **retorna** "`True`" si al recibir un **grafo** "G" como parámetro todas las aristas tienen **pesos numéricos** asignados o "`False`", en caso contrario. El grafo pudo haber sido **creado** tanto en el "Wolfram System" de *Mathematica*, como también, mediante el uso del **paquete** "Combinatorica".

Sintaxis: `GrafoPonderadoNQ[G]`.

Ejemplo 7.169

Determine el valor lógico del comando **GrafoPonderadoNQ** sobre un grafo cuyas aristas vienen dadas por la relación binaria: $aRb \Leftrightarrow a - b \geq 2$, con $A = \{1,2,3,\ldots,30\}$ y asignando pesos seudoaleatorios reales de uno a diez.

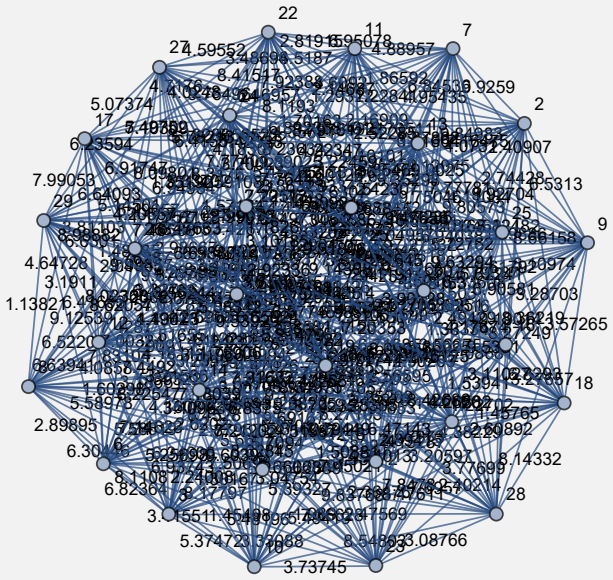
Solución:

En *Mathematica*:

In[ ] :=

```
A = Table[i, {i, 1, 30}];
aristas = RelBin["a-b>=2", A, A];
grafo = Grafo[aristas, pesos -> RandomReal[{1, 10}, Length[aristas]],
mostrarpesos -> True]
GrafoPonderadoNQ[grafo]
```

Out[ ] :=



True

Ejemplo 7.170

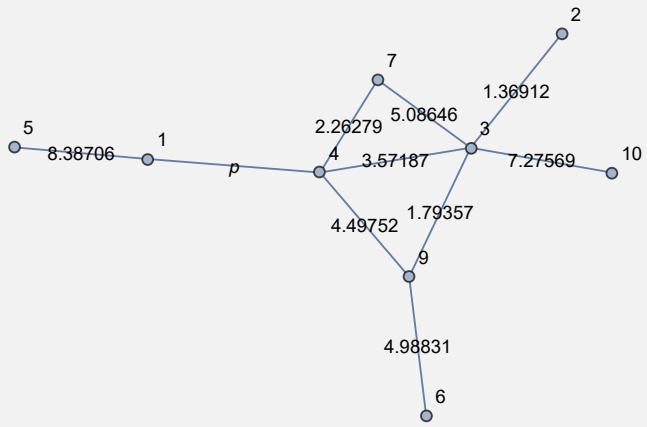
Considere un grafo seudoaleatorio de orden 10×10, con pesos seudoaleatorios reales de uno a diez. Reemplace la primera ponderación por la letra "p", responda a través del uso de software: ¿es ponderado numérico el grafo obtenido?

Solución:

In[ ] :=

```
grafo = GrafoRandom[10, 10];
grafo = Grafo[AristasWolframSystemToCombinatorica[EdgeList[grafo]],
pesos -> ReplacePart[RandomReal[{1, 10}, EdgeCount[grafo]], p, 1],
mostrarpesos -> True]
GrafoPonderadoNQ[grafo]
```

Out[ ] :=



False

N En el código anterior, **`ReplacePart[RandomReal[{1, 10}, EdgeCount[grafo]], p, 1]`** reemplaza en la lista de pesos **`RandomReal[{1, 10}, EdgeCount[grafo]]`** el dato ubicado en la posición 1 por el carácter "p".

Explicación en video

86) **GrafoVilCretas**: **construye** un **grafo** que muestra la **palabra** "VilCretas". Representa un **ejemplo particular** de un grafo con múltiples nodos y aristas.

Sintaxis: **`GrafoVilCretas[]`**.

Ejemplo 7.171

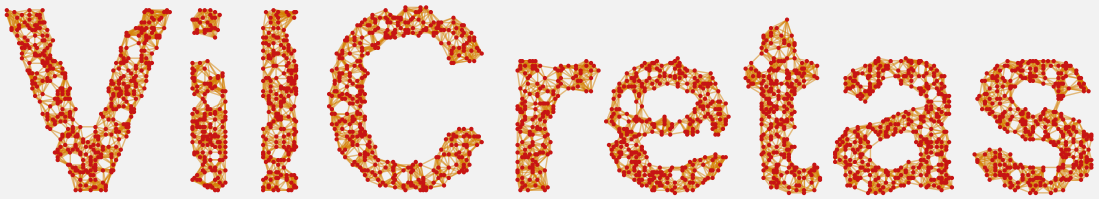
Genere el grafo **`GrafoVilCretas[]`** y halle la cantidad de vértices que contiene.

Solución:

In[ ] :=

```
grafo = GrafoVilCretas[]
Length[ListaVertices[grafo]]
```

Out[ ] :=



2500

N Se utilizó el comando **Length** para determinar la cantidad de nodos, en lugar de la opción "**cantidad->True**" de **ListaVertices**, con la finalidad de evitar mostrar la lista completa de vértices, dadas sus dimensiones.

Ejemplo 7.172

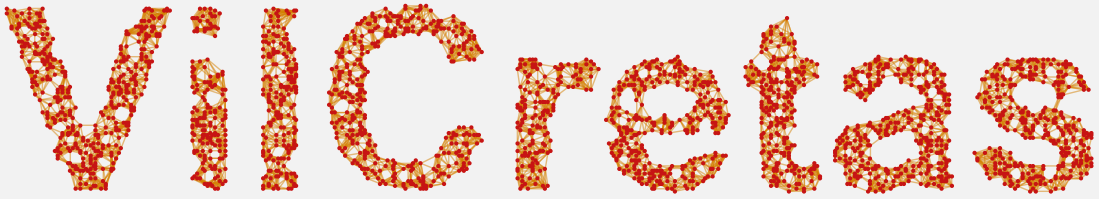
Construya el grafo **GrafoVilCretas[]** y encuentre la cantidad de aristas que posee.

Solución:

In[ ] :=

```
grafo = GrafoVilCretas[]
Length[ListaAristas[grafo]]
```

Out[ ] :=



14628

N En este ejercicio, se empleó la instrucción **Length** para retornar el número de lados del grafo, en sustitución de la opción "**cantidad->True**" del comando **ListaAristas**, con el objetivo de evitar la lista completa que contiene 14628 aristas. También es importante señalar, que el grafo **GrafoVilCretas[]** cambia en cada ejecución la cantidad de aristas y su posición, por lo que si se vuelve a invocar el mismo *In*[], muy probablemente el número de lados será distinto.

Explicación en video

87) **RutaQ**: función **booleana** que retorna "`True`" si una lista "L" de **aristas** representa un **camino o trayectoria** sobre un grafo "G" y "`False`", en caso contrario.

Sintaxis: `RutaQ[G, L]`.

Ejemplo 7.173

Determine si el conjunto {{a, a}, {a, b}, {b, c}, {c, d}} representa una trayectoria sobre un grafo con aristas: {{a, a}, {a, b}, {b, c}, {c, d}, {b, a}} y pesos: {1, 2, 3, 4, 5}, respectivamente.

Solución:

In[ ] :=
```
grafo = Grafo[{{a, a}, {a, b}, {b, c}, {c, d}, {b, a}}, pesos -> {1,
2, 3, 4, 5}, mostrarpesos -> True]
RutaQ[grafo, {{a, a}, {a, b}, {b, c}, {c, d}}]
```

Out[ ] :=

True

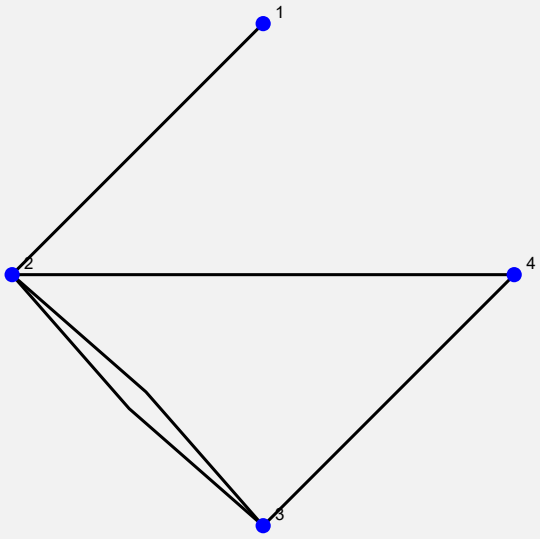
Ejemplo 7.174

El conjunto {{1, 2}, {2, 3}, {3, 2}, {2, 5}}, ¿representa una trayectoria sobre el grafo de "Combinatorica", con aristas: {{1, 2}, {2, 3}, {2, 3}, {3, 4}, {4, 2}}?

Solución:

In[ ] :=
```
GrafoC[{{1, 2}, {2, 3}, {2, 3}, {3, 4}, {4, 2}}]
RutaQ[G, {{1, 2}, {2, 3}, {3, 2}, {2, 5}}]
```

Out[ ] :=



False

Explicación en video

88) PesoRuta: calcula el **peso** asociado a una **ruta o camino** "L" sobre un grafo "G" (con o sin "Combinatorica"). Presenta la opción "**ruta -> True**" que **resalta la trayectoria** sobre el grafo.

Sintaxis: **PesoRuta[G, L]**, o bien, **PesoRuta[G, L, ruta -> True]**.

Ejemplo 7.175

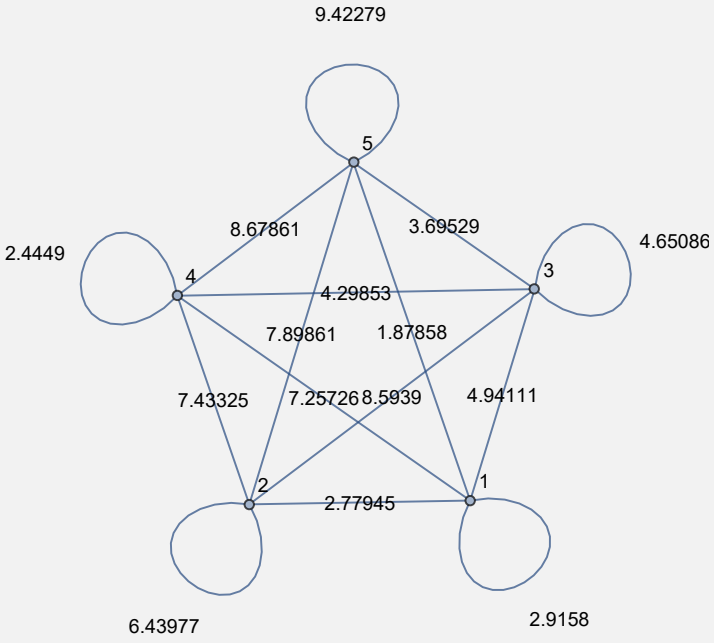
Considere un grafo cuyas aristas vienen dadas por: {{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {2, 2}, {2, 3}, {2, 4}, {2, 5}, {3, 3}, {3, 4}, {3, 5}, {4, 4}, {4, 5}, {5, 5}} con pesos seudoaleatorios reales de uno a diez. Determine el peso de la trayectoria {{1, 3}, {3, 4}, {4, 5}, {5, 1}}, mostrando la ruta.

Solución:

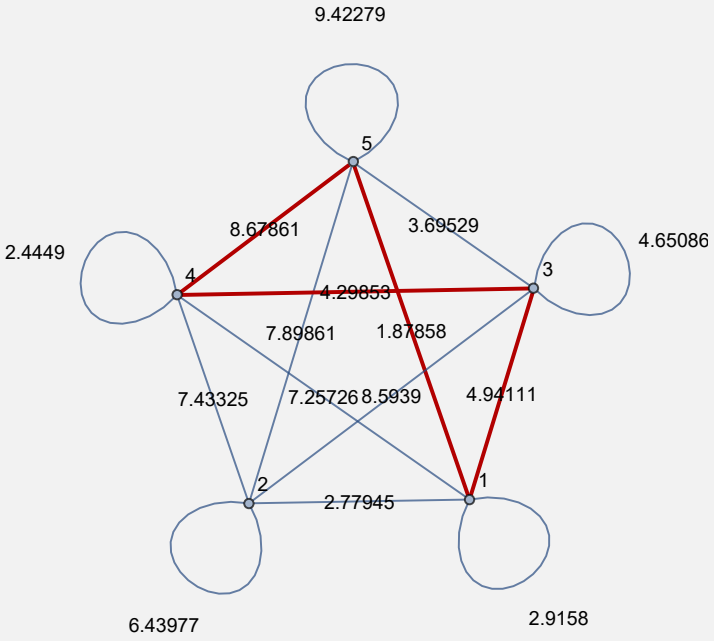
In[ ] :=

```
aristas = {{1, 1}, {1, 2}, {1, 3}, {1, 4}, {1, 5}, {2, 2}, {2, 3},
{2, 4}, {2, 5}, {3, 3}, {3, 4}, {3, 5}, {4, 4}, {4, 5}, {5, 5}};
grafo = Grafo[aristas, pesos -> RandomReal[{1, 10}, Length[aristas]],
mostrarpesos -> True]
PesoRuta[grafo, {{1, 3}, {3, 4}, {4, 5}, {5, 1}}, ruta -> True]
```

Out[ ] :=



19.7968



Ejemplo 7.176

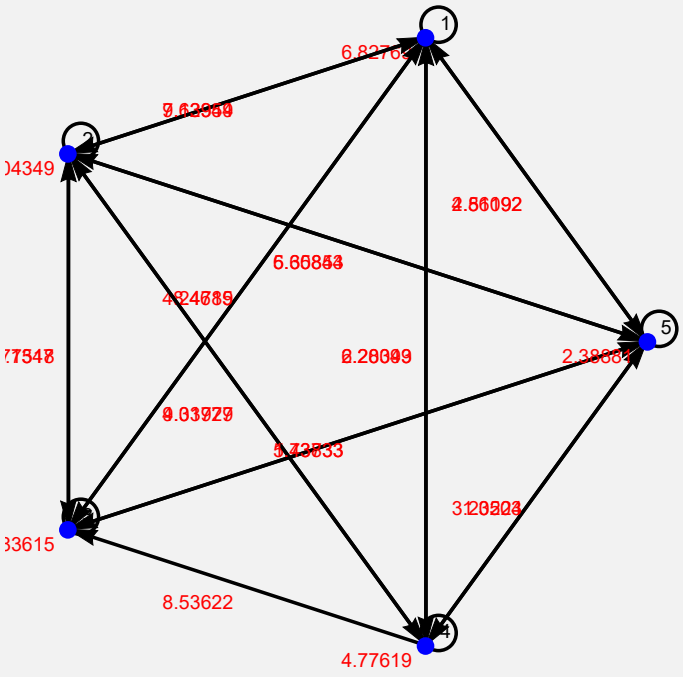
Si $A = \{1,2,3,4,5\}$, sea un grafo dirigido en "Combinatorica" donde sus lados corresponden a: $A \times A - \{(3,4)\}$ con pesos seudoaleatorios reales de uno a diez. Halle el peso de la trayectoria $\{\{3,3\},\{3,2\},\{2,5\},\{5,1\}\}$, mostrando la ruta.

Solución:

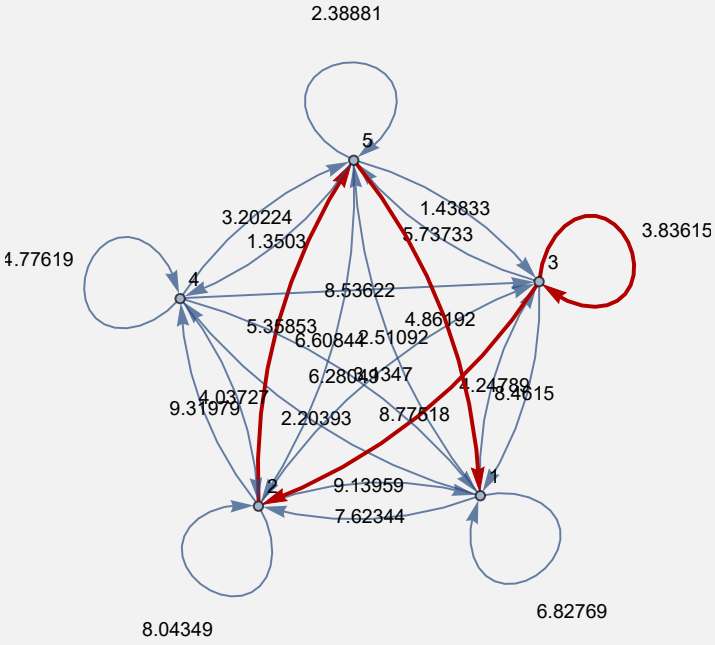
In[ ] :=

```
A = {1, 2, 3, 4, 5};
aristas = Complement[PC[A, A], {{3, 4}}];
GrafoC[aristas, dirigido -> True, pesos -> RandomReal[{1, 10},
Length[aristas]], mostrarpesos -> True]
PesoRuta[G, {{3, 3}, {3, 2}, {2, 5}, {5, 1}}, ruta -> True]
```

Out[ ] :=



22.8318



Explicación en video

Aporte pedagógico

En esta sección se ha **mostrado** el funcionamiento de **diversos comandos** que **permiten** al estudiante **profundizar** la aplicación de **importantes algoritmos** vinculados con el tema de grafos. Por ejemplo, **`Fleury`** constituye una instrucción que facilita la exploración del algoritmo de *Fleury* para **construir circuitos eulerianos**, **`ExisteCircuitoHamilton`** aplica **propiedades suficientes** que garantizan la existencia de un circuito de *Hamilton* y **`AlDijkstra`** expone **iteración por iteración** el uso del algoritmo de la longitud del camino más corto. Comandos como éstos, integrados en lecciones de laboratorio adecuadamente planificadas, pueden brindar contribuciones teóricas muy significativas.

Aporte de investigación

En este aspecto se destacan las funciones **implementadas** en *VilCretas* que hacen uso de los **datos geográficos** proporcionados por la empresa *Wolfram Research*. Algunas de ellas son: **`GrafoCountryRegions`**, **`GrafoFronteraCountries`**, **`CDFAgenteViajeroGrupos`** y **`CDFAgenteViajeroRegiones`**. Todas estas instrucciones toman **datos reales** para **mostrar grafos** y **resolver problemas** que brindan aproximaciones concordantes con las aplicaciones de la teoría de grafos, en **contextos auténticos**.

Ejercicios

Resuelva los siguientes ejercicios utilizando como apoyo el paquete *VilCretas*.

1) Genere en el "Wolfram System" de *Mathematica* los grafos no dirigidos con el conjunto de aristas dado a continuación:

   a) G_1 : Aristas={{1, 4}, {2, 3}, {2, 4}, {2, 5}, {2, 6}, {3, 5}, {3, 7}, {3, 8}, {4, 5}, {4, 10}, {5, 8}, {5, 10}, {6, 9}, {7, 9}, {8, 10}}, en tercera dimensión.

   b) G_2 : Aristas={{1, 3}, {1, 4}, {1, 5}, {1, 10}, {2, 3}, {2, 5}, {2, 8}, {2, 9}, {2, 10}, {3, 6}, {3, 10}, {4, 5}, {4, 9}, {5, 7}, {5, 9}, {6, 8}, {6, 9}, {7, 8}, {7, 10}, {8, 9}}, con pesos seudoaleatorios enteros en el intervalo $[1,10]$.

   c) G_3 : Aristas={{1, 5}, {2, 7}, {2, 11}, {2, 18}, {3, 19}, {4, 8}, {4, 13}, {4, 16}, {5, 18}, {6, 9}, {7, 13}, {9, 10}, {9, 15}, {10, 17}, {10, 19}, {11, 12}, {11, 15}, {11, 17}, {12, 14}, {14, 16}}, en tercera dimensión y con "**`shape -> True`**".

   d) G_4 : Aristas={{1, 2}, {1, 22}, {1, 25}, {1, 28}, {3, 7}, {3, 8}, {3, 16}, {3, 18}, {4, 12}, {4, 14}, {4, 15}, {4, 25}, {5, 25}, {5, 30}, {6, 7}, {6, 9}, {6, 23}, {6, 26}, {7, 23}, {8, 16}, {9, 10}, {9, 13}, {9, 14}, {9, 18}, {11, 18}, {11, 23}, {12, 27}, {13, 18}, {14, 17}, {14, 24}, {15, 19}, {15, 20}, {15, 24}, {16, 26}, {19, 21}, {19, 27}, {20, 27}, {21, 24}, {25, 30}, {28, 29}}, con pesos seudoaleatorios reales en el intervalo $[1,10]$ y "**`shape -> True`**".

e) G_5 : Aristas={{1, 17}, {1, 24}, {1, 28}, {2, 5}, {2, 10}, {2, 16}, {2, 29}, {3, 17}, {3, 29}, {4, 9}, {5, 10}, {5, 11}, {5, 13}, {5, 15}, {5, 17}, {5, 28}, {6, 14}, {6, 21}, {6, 30}, {7, 17}, {8, 9}, {8, 22}, {9, 14}, {9, 15}, {9, 18}, {9, 27}, {9, 28}, {10, 15}, {11, 26}, {12, 14}, {12, 18}, {12, 19}, {12, 26}, {13, 15}, {13, 29}, {14, 19}, {14, 27}, {15, 21}, {16, 26}, {17, 20}, {18, 29}, {19, 29}, {20, 21}, {21, 27}, {22, 25}, {22, 27}, {22, 28}, {23, 24}, {25, 28}, {28, 29}}, en tercera dimensión, con pesos seudoaleatorios reales de $[1,10]$ y "**shape -> True**".

f) G_6 : Aristas={{1, 25}, {2, 14}, {2, 27}, {2, 33}, {2, 46}, {3, 31}, {3, 47}, {3, 49}, {4, 10}, {5, 11}, {5, 12}, {5, 37}, {5, 40}, {5, 45}, {6, 25}, {6, 36}, {7, 15}, {7, 24}, {7, 26}, {7, 45}, {8, 41}, {8, 42}, {9, 15}, {9, 47}, {10, 11}, {11, 34}, {11, 41}, {12, 13}, {12, 36}, {12, 49}, {12, 50}, {13, 23}, {13, 29}, {13, 48}, {14, 28}, {15, 37}, {15, 43}, {16, 18}, {16, 19}, {16, 26}, {16, 36}, {16, 39}, {17, 28}, {19, 29}, {19, 35}, {20, 49}, {21, 24}, {21, 37}, {21, 40}, {22, 30}, {22, 35}, {25, 28}, {26, 28}, {26, 50}, {27, 33}, {29, 39}, {30, 41}, {31, 43}, {32, 36}, {32, 39}, {32, 42}, {33, 34}, {33, 43}, {33, 45}, {34, 38}, {34, 43}, {39, 50}, {42, 46}, {44, 50}, {45, 49}}, en tercera dimensión y con pesos seudoaleatorios reales de $[1,10]$.

2) Represente haciendo uso del comando **GrafoC** los grafos expuestos en el ejercicio 1.

3) Represente como grafos dirigidos en el "Wolfram System" y recurriendo al paquete "Combinatorica", los grafos del ejercicio 1.

4) Obtenga los pesos asociados a los lados de los grafos G_2, G_4, G_5 y G_6 empleando la instrucción **PesosAristas**.

5) Mediante **GraphToCombinatorica** convierta los grafos del ejercicio 1 al ambiente provisto por el paquete "Combinatorica".

6) Usando **CombinatoricaToGraph** retorne los grafos obtenidos en el ejercicio 2 a través del ambiente suministrado por el "Wolfram System" de *Mathematica*.

7) Determine una trayectoria del nodo 1 al vértice 10 en los grafos del ejercicio 1, usando los comandos **Ruta** y **RutaMCorta** de *VilCretas*.

8) Halle a través de **CantRutas** el número de rutas del nodo 1 al vértice 10 en los grafos del ejercicio 1.

9) Determine mediante **CantMRutas** el número de trayectorias de longitud seis del nodo 1 al vértice 10 en los grafos del ejercicio 1, suponiendo válida la repetición de aristas.

10) Encuentre si es posible un máximo de tres caminos del nodo 1 al vértice 10 en los grafos del ejercicio 1. Sugerencia: emplee las instrucciones **GeneraRutas** y **GeneraRutasGraphSimple** de *VilCretas*.

11) Anime utilizando los comandos **AnimarGrafo** y **AnimarGrafoWithCombinatorica**, una de las trayectorias obtenidas en el ejercicio anterior. La ruta debe seleccionarse de forma seudoaleatoria empleando la sentencia **RandomChoice**.

12) Muestre la ruta del ejercicio anterior a través de la sentencia **ResaltarRuta**.

13) Construya un grafo representado por la matriz de adyacencia **RandomInteger[{0, 1}, {10, 10}]**, cuyos vértices son números enteros consecutivos de 1 a 10.

14) Determine una matriz de adyacencia y de incidencia que represente los grafos expuestos en el ejercicio 1. Sugerencia: utilice **`MGrafo`** y **`MIGrafo`**, emplee la opción "**`table ->`** **`True`**".

15) Encuentre un grafo representado por las siguientes matrices de adyacencia de pesos, cuyos vértices son números enteros consecutivos de 1 a 10:

a)
$$\begin{pmatrix} 0 & 16 & 16 & 0 & 13 & 3 & 0 & 9 & 6 & 0 \\ 16 & 0 & 0 & 10 & 5 & 0 & 0 & 13 & 0 & 3 \\ 16 & 0 & 0 & 0 & 0 & 11 & 11 & 0 & 0 & 17 \\ 0 & 10 & 0 & 0 & 0 & 0 & 6 & 12 & 0 & 0 \\ 13 & 5 & 0 & 0 & 0 & 0 & 8 & 0 & 0 & 10 \\ 3 & 0 & 11 & 0 & 0 & 0 & 14 & 0 & 0 & 18 \\ 0 & 0 & 11 & 6 & 8 & 14 & 0 & 14 & 0 & 0 \\ 9 & 13 & 0 & 12 & 0 & 0 & 14 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 17 & 0 & 10 & 18 & 0 & 0 & 0 & 0 \end{pmatrix}$$

b)
$$\begin{pmatrix} 0 & 0 & 0 & 17 & 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 11 & 0 & 16 & 16 & 0 & 0 \\ 0 & 0 & 0 & 18 & 12 & 0 & 0 & 16 & 0 & 2 \\ 17 & 0 & 18 & 0 & 7 & 0 & 0 & 0 & 6 & 0 \\ 0 & 11 & 12 & 7 & 0 & 14 & 4 & 0 & 19 & 0 \\ 0 & 0 & 0 & 0 & 14 & 0 & 0 & 8 & 0 & 1 \\ 0 & 16 & 0 & 0 & 4 & 0 & 0 & 0 & 12 & 20 \\ 6 & 16 & 16 & 0 & 0 & 8 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 6 & 19 & 0 & 12 & 5 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 & 1 & 20 & 0 & 1 & 0 \end{pmatrix}$$

16) Halle una matriz de adyacencia de pesos que represente los grafos G_2, G_4, G_5 y G_6.

17) Construya un grafo de regiones usando **`GrafoCountryRegions`** para los países: México y USA.

18) Genere un grafo dirigido con los países fronterizos a: México y USA. Sugerencia: use la instrucción **`GrafoFronteraCountries`**.

19) Determine el conjunto de aristas, vértices y su cantidad, sobre los grafos retornados en el ejercicio 1.

20) Halle los grados de cada uno de los nodos, en los grafos expuestos en el ejercicio 1.

21) Encuentre el centro, radio, diámetro y las excentricidades de los vértices, sobre los grafos compartidos en el ejercicio 1.

22) Analice las condiciones que se deben satisfacer en caso de ser posible, para garantizar la existencia de un circuito de *Euler*, una ruta de *Euler*, un circuito de *Hamilton* y una ruta de *Hamilton*, en los grafos: completo, bipartito completo, mariposa, rueda, estrella, camino, ciclo, red y regular. Sugerencia: emplee la opción "**`analizar ->`** **`True`**" de los comandos que construyen los tipos indicados, bajo la excepción del grafo regular.

23) Halle un circuito de longitud cinco si existe, sobre los grafos expuestos en el ejercicio 1. Sugerencia: recurra a **`CircuitoL`**, emplee su opción "**`ruta ->`** **`True`**".

24) Encuentre la longitud del circuito más pequeño sobre los grafos del ejercicio 1, a través de la sentencia **`LongitudCircuitoSmall`**.

25) ¿Tienen circuitos de *Euler*, rutas de *Euler*, circuitos de *Hamilton* y rutas de *Hamilton* los grafos del ejercicio 1?, justifique. Sugerencia: puede usar las instrucciones **`CircuitoEulerQ`**, **`CircuitosEuler`**, **`RutaEulerQ`**, **`RutaEuler`**, **`CircuitoHamiltonQ`**, **`CircuitosHamilton`**, **`ExisteCircuitoHamilton`**, **`RutaHamiltonQ`** y **`RutaHamilton`** de *VilCretas*.

26) Resuelva de ser posible, el problema del agente viajero sobre los grafos compartidos en el ejercicio 1.

27) Utilizando la instrucción **`LongitudCaminoMC`** muestre en una animación, un camino más corto entre el vértice 1 y el nodo 10, sobre los grafos del ejercicio 1.

28) Aplique el algoritmo de *Dijkstra* considerando los nodos 1 y 10, en los grafos G_2, G_4, G_5 y G_6.

Capítulo 8

Teoría de árboles con *VilCretas*

Este capítulo introduce **treinta y nueve** comandos de *VilCretas* que facilitan al usuario a través del empleo de software, la construcción de árboles de diferentes tipos, el uso de árboles binarios con algunas de sus aplicaciones y desde luego, la implementación de distintos algoritmos relacionados con el tema, tales como: inserción, buscar primero a lo ancho y a lo largo, *Prim*, *Kruskal* y recorridos en orden prefijo y postfijo. Las herramientas brindadas por *VilCretas* ofrecen alternativas viables para desarrollar una mayor comprensión conceptual y procedimental, abocándose hacia la visualización y manipulación de conceptos y propiedades relacionados con la teoría de árboles.

1) **Arbol**: **construye** un **árbol** a través del "Wolfram System" dadas sus **aristas** como una **matriz** de **pares ordenados**. El comando presenta **cuatro opciones**: "**dirigido->True**", "**pesos->Lista**", "**mostrarpesos->True**" y "**shape->forma**". "dirigido" crea un árbol con **aristas dirigidas**, "pesos" genera un **árbol ponderado**, "mostrarpesos" **muestra** los **pesos** sobre cada uno de sus lados y "shape" brinda una **forma** a los vértices del grafo en cualquiera de las **siguientes alternativas**: "circulo", "triangulo", "cuadrado", "rectangulo", "pentagono", "hexagono" y "octagono".

Sintaxis: **Arbol[Aristas]**, o bien,

```
Arbol[Aristas, dirigido->True, pesos->Lista, mostrarpesos->True,
shape->forma]
```

En esta última invocación, es posible **prescindir** de cualquiera de las opciones.

Ejemplo 8.1

Construya un árbol dirigido con pesos enteros seudoaleatorios de cero a diez, cuyas aristas vienen dadas por: {{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}.

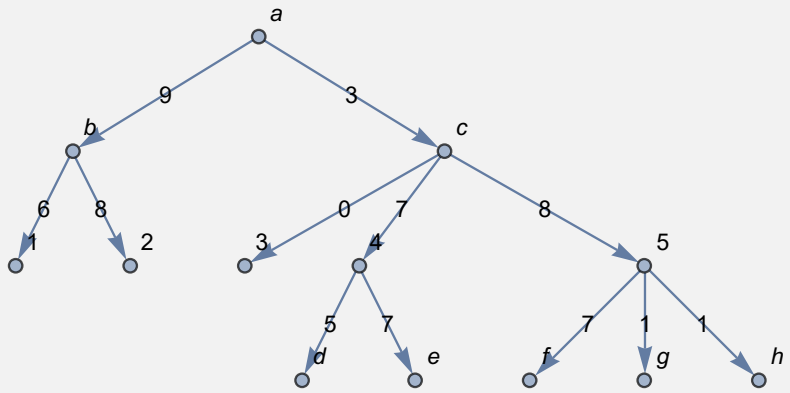
Solución:

Al recurrir a la instrucción **Arbol**:

In[ ] :=

```
Arbol[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c,
5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}, dirigido->True,
pesos->RandomInteger[10, 12], mostrarpesos->True]
```

Out[ ] :=



N El software *Mathematica* selecciona por defecto la raíz.

Ejemplo 8.2

Genere el árbol del ejemplo anterior, añadiendo una forma rectangular a cada uno de los vértices.

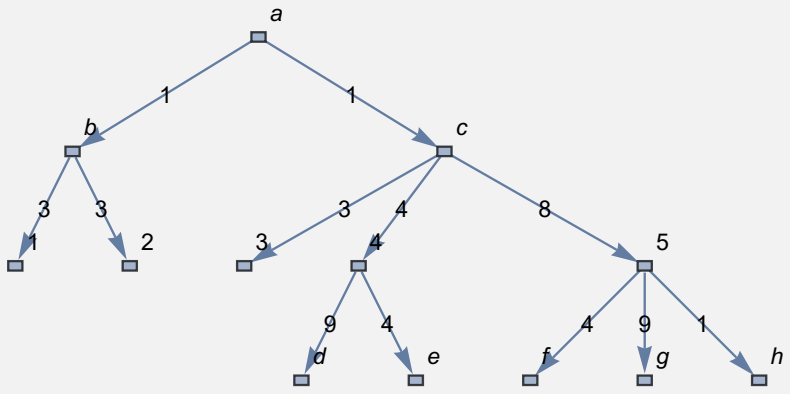
Solución:

En el software:

In[ ] :=

```
Arbol[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c,
5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}, dirigido -> True,
pesos -> RandomInteger[10, 12], mostrarpesos -> True,
shape -> "rectangulo"]
```

Out[ ] :=



N El alumno puede observar la diferencia entre los pesos del árbol mostrado en el *Out*[ ], en comparación con el grafo del ejercicio anterior. Los pesos ¡no coinciden! al asignarse de manera seudoaleatoria.

Explicación en video

2) **ArbolR**: **construye** un **árbol** a través del "Wolfram System" dadas sus **aristas** como una **matriz** de **pares ordenados** y un **vértice** "Raiz", correspondiente al **nodo raíz** del árbol. El comando presenta **cuatro opciones**: "`dirigido -> True`", "`pesos -> Lista`", "`mostrarpesos -> True`" y "`shape -> forma`". "dirigido" crea un árbol con **aristas dirigidas**, "pesos" genera un **árbol ponderado**, "mostrarpesos" **muestra** los **pesos** sobre cada uno de sus lados y "shape" brinda una **forma** a los vértices del grafo en cualquiera de las **siguientes alternativas**: "circulo", "triangulo", "cuadrado", "rectangulo", "pentagono", "hexagono" y "octagono".

Sintaxis: `ArbolR[Aristas, Raiz]`, o bien,

```
ArbolR[Aristas, Raiz, dirigido->True, pesos->Lista, mostrarpesos->True,
shape->forma]
```

En esta última invocación, es posible **prescindir** de cualquiera de las opciones.

Ejemplo 8.3

Construya un árbol con lados: {{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}, dirigido, ponderado con pesos enteros seudoaleatorios de cero a diez y raíz en el nodo 5.

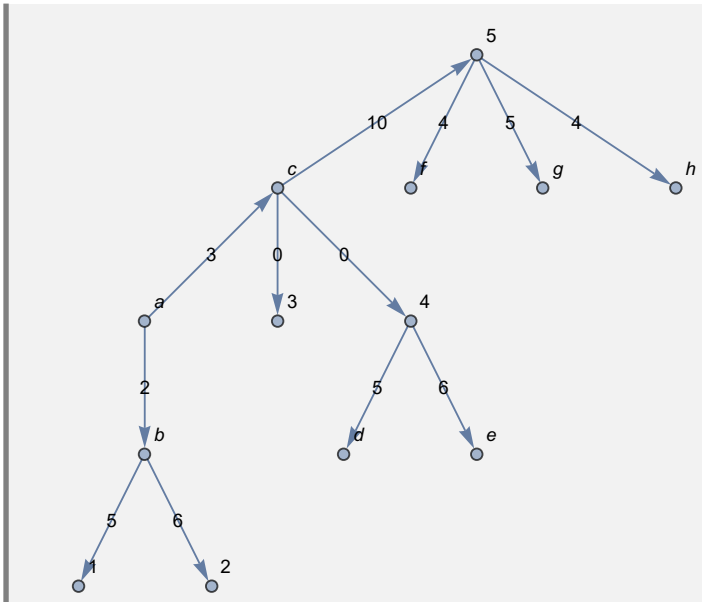
Solución:

Al emplear el comando **ArbolR**, se tiene:

In[ ] :=

```
ArbolR[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5},
{4, d}, {4, e}, {5, f}, {5, g}, {5, h}}, 5, dirigido->True,
pesos->RandomInteger[10, 12], mostrarpesos->True]
```

Out[ ] :=



Ejemplo 8.4

Retorne el árbol del ejercicio anterior, dando forma rectangular a sus nodos.

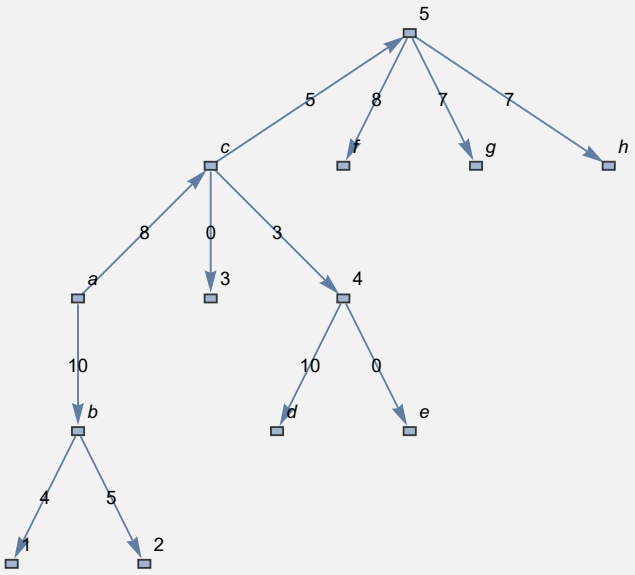
Solución:

En *Mathematica*, se usa la opción "shape":

In[ ] :=

```
ArbolR[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4},
{c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}, 5,
dirigido -> True, pesos -> RandomInteger[10, 12], mostrarpesos -> True,
shape -> "rectangulo"]
```

Out[ ] :=



Explicación en video

3) **ArbolGrafico**: **construye** un **árbol** como una **gráfica** dadas sus **aristas** mediante una **matriz** de **pares ordenados** (los comandos de grafos **no corren** sobre él). La instrucción presenta la **opción** "**`dirigido -> True`**" encargada de crear un árbol con **aristas dirigidas**.

Sintaxis: **`ArbolGrafico[Aristas]`**, o bien, **`ArbolGrafico[Aristas, dirigido -> True]`**.

Ejemplo 8.5

Muestre un árbol gráfico con aristas: {{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9, 8}, {c, p}, {p, j}}.

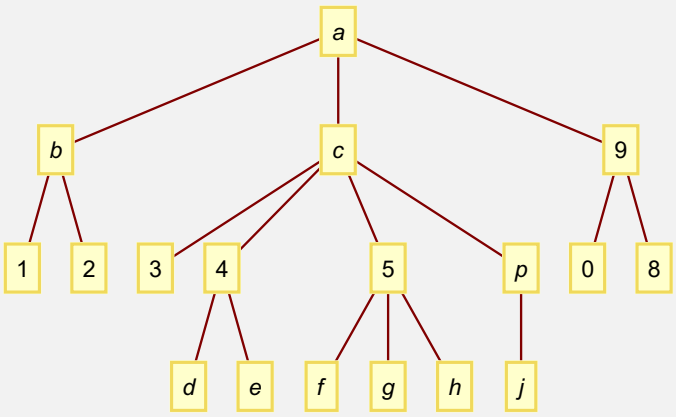
Solución:

Al utilizar la instrucción **`ArbolGrafico`**:

In[ ] :=

```
ArbolGrafico[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5},
{4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9, 8}, {c,
p}, {p, j}}]
```

Out[ ] :=



N Este tipo de árboles en el software *Mathematica* corresponden a un objeto **`Graphics`**. **`Graphics`** es un ambiente que provee el programa para crear toda clase de dibujos. El estudiante por lo tanto, debe tener claro, que estos árboles no son interpretados como un grafo en *Mathematica*.

Ejemplo 8.6

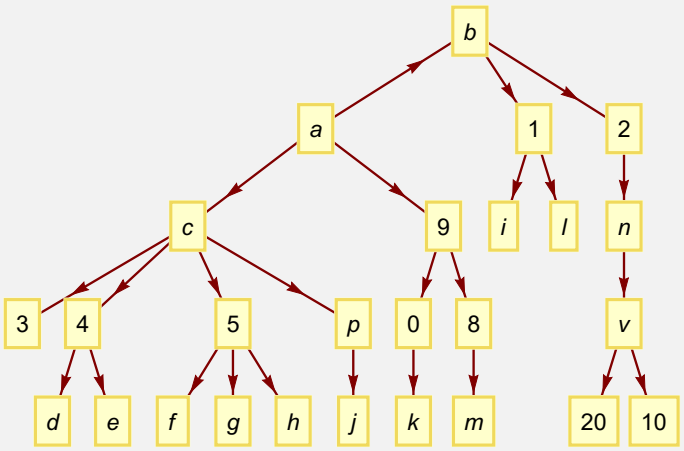
Genere un árbol gráfico dirigido con lados: {{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9, 8}, {c, p}, {p, j}, {1, i}, {1, l}, {8, m}, {0, k}, {2, n}, {n, v}, {v, 20}, {v, 10}}.

Solución:

In[ ] :=

```
ArbolGrafico[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5},
{4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9, 8}, {c,
p}, {p, j}, {1, i}, {1, l}, {8, m}, {0, k}, {2, n}, {n, v}, {v, 20},
{v, 10}}, dirigido -> True]
```

Out[ ] :=



N Al igual que el comando **Arbol**, **ArbolGrafico** elige por defecto la raíz del árbol.

Explicación en video

4) **ArbolGraficoR**: **construye** un **árbol** como una **gráfica** dadas sus **aristas** mediante una **matriz** de **pares ordenados** y un **nodo raíz** (los comandos de grafos **no corren** sobre él). La instrucción presenta la **opción** "**dirigido -> True**" encargada de crear un árbol con **aristas dirigidas**.

Sintaxis: **ArbolGraficoR[Aristas, Raiz]**, o bien, **ArbolGraficoR[Aristas, Raiz, dirigido -> True]**.

Ejemplo 8.7

Construya un árbol gráfico con raíz en el nodo "p", cuyas aristas vienen dadas por: {{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9, 8}, {c, p}, {p, j}}.

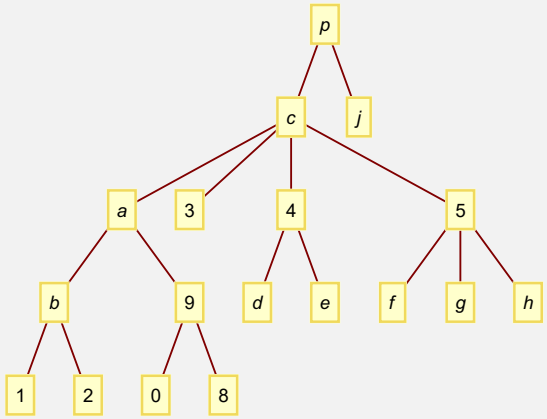
Solución:

En *Mathematica*:

In[ ] :=

```
ArbolGraficoR[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c,
5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9, 8},
{c, p}, {p, j}}, p]
```

Out[ ] :=



Ejemplo 8.8

Muestre un árbol gráfico dirigido con raíz en el vértice "k" y aristas: {{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9, 8}, {c, p}, {p, j}, {1, i}, {1, l}, {8, m}, {0, k}, {2, n}, {n, v}, {v, 20}, {v, 10}}.

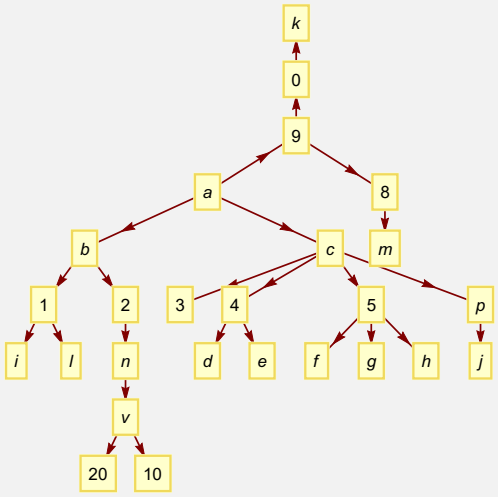
Solución:

Al emplear la opción "dirigido" de **ArbolGraficoR**:

In[ ] :=

```
ArbolGraficoR[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c,
5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9, 8},
{c, p}, {p, j}, {1, i}, {1, l}, {8, m}, {0, k}, {2, n}, {n, v}, {v,
20}, {v, 10}}, k, dirigido -> True]
```

Out[ ] :=



Explicación en video

5) **ArbolQ**: retorna "`True`" si el argumento "T" recibido como parámetro es un **grafo** en *Mathematica* que corresponde a un **árbol** o "`False`", en caso contrario. Acepta grafos creados con el **paquete** "Combinatorica".

Sintaxis: `ArbolQ[T]`.

Ejemplo 8.9

Verifique el valor lógico "`True`", sobre un árbol generado con el comando **`Arbol`**, dirigido, con pesos seudoaleatorios enteros de cero a diez, nodos en forma rectangular y lados: {{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}.

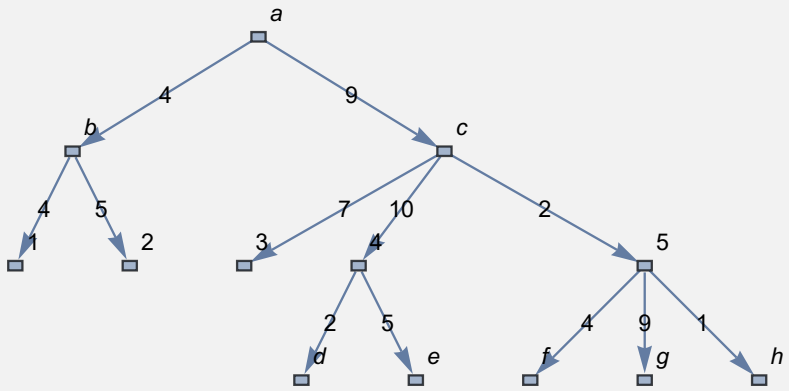
Solución:

En el software:

In[ ] :=

```
arbol = Arbol[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3},
{c, 4}, {c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}},
dirigido -> True, pesos -> RandomInteger[10, 12], mostrarpesos -> True,
shape -> "rectangulo"]
ArbolQ[arbol]
```

Out[ ] :=



True

Ejemplo 8.10

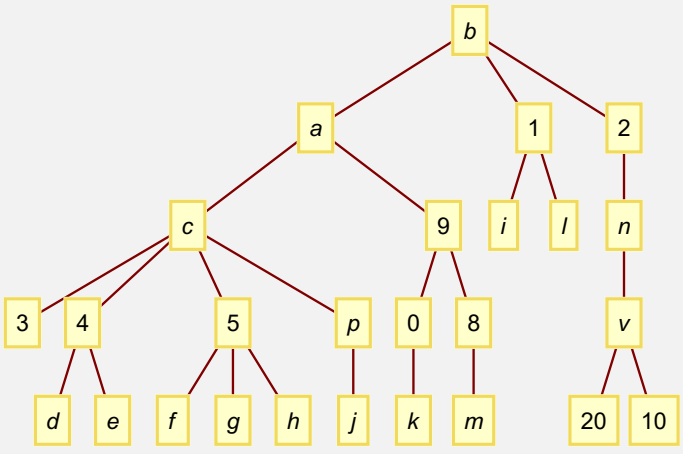
Sea un árbol gráfico con aristas: {{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9, 8}, {c, p}, {p, j}, {1, i}, {1, l}, {8, m}, {0, k}, {2, n}, {n, v}, {v, 20}, {v, 10}}. Determine el valor lógico retornado por la instrucción **`ArbolQ`**.

Solución:

In[ ] :=

```
arbol = ArbolGrafico[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4},
{c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9,
8}, {c, p}, {p, j}, {1, i}, {1, l}, {8, m}, {0, k}, {2, n}, {n, v},
{v, 20}, {v, 10}}]
ArbolQ[arbol]
```

Out[ ] :=



False

N El resultado es False pues el árbol construido, no es un árbol del "Wolfram System" o del paquete "Combinatorica", razón por la cuál no es considerado un grafo en el software *Mathematica*.

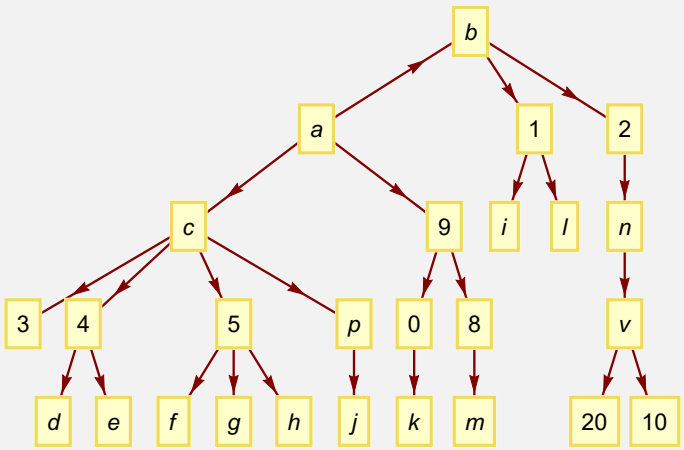
Explicación en video

6) **ArbolGraficoQ**: retorna "**`True`**" si el argumento "T" recibido como parámetro es un **gráfico** que corresponde a un **árbol** o "**`False`**", en caso contrario.

Sintaxis: **`ArbolGraficoQ[T]`**.

Ejemplo 8.11

Con ayuda de software responda: ¿el árbol dado a continuación es gráfico?



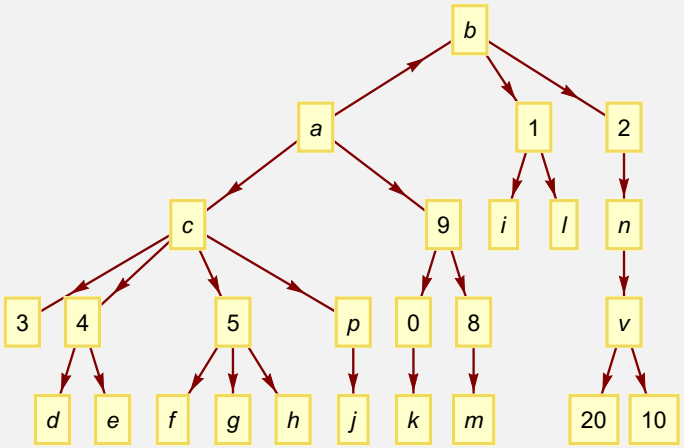
Solución:

Al recurrir a las instrucciones **`ArbolGrafico`** y **`ArbolGraficoQ`**, se obtiene:

In[ ] :=

```
arbol = ArbolGrafico[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4},
{c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9,
8}, {c, p}, {p, j}, {1, i}, {1, l}, {8, m}, {0, k}, {2, n}, {n, v},
{v, 20}, {v, 10}}, dirigido -> True]
ArbolGraficoQ[arbol]
```

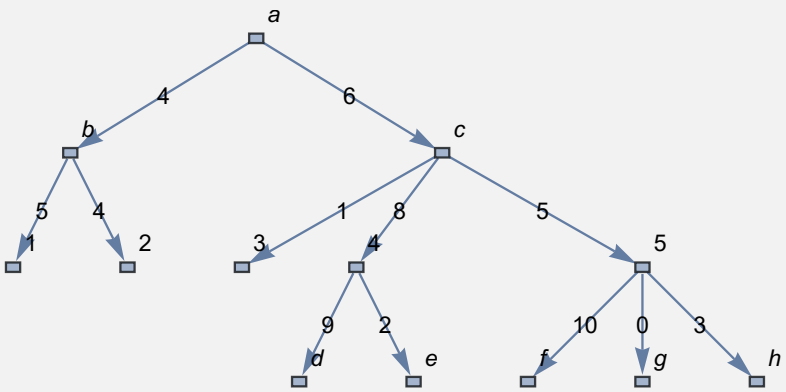
Out[ ] :=



True

Ejemplo 8.12

¿Es el siguiente árbol un árbol gráfico?



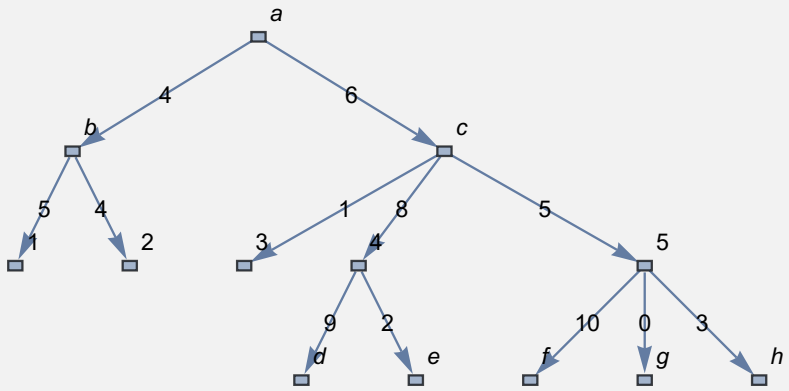
Solución:

En primera instancia se crea el árbol con el comando **Arbol** y posteriormente, se utiliza **ArbolGraficoQ**:

In[ ] :=

```
arbol = Arbol[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4},
{c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}, dirigido -> True,
pesos -> {4, 6, 5, 4, 1, 8, 5, 9, 2, 10, 0, 3}, mostrarpesos -> True,
shape -> "rectangulo"]
ArbolGraficoQ[arbol]
```

Out[ ] :=



False

Devuelve False pues el árbol se creó en el "Wolfram System" de *Mathematica* y por consiguiente, no es un objeto **`Graphics`**.

Explicación en video

7) **ArbolGraficoToArbol**: función que **convierte** un **árbol** "T" **creado** como un **gráfico**, a un **árbol** que **corresponde** a un **grafo** en *Mathematica*. Los nodos deben ser **todos numéricos**, o bien, **todos caracteres**, no aplica una **mezcla** de ambos y **deben estar etiquetados** en el árbol.

Sintaxis: **`ArbolGraficoToArbol[T]`**.

Ejemplo 8.13

Convierta al "Wolfram System", un árbol gráfico con raíz 1 cuyos lados son: {{1, 2}, {1, 3}, {3, 4}, {3, 5}, {4, 6}, {4, 7}, {5, 8}, {5, 9}, {5, 10}}.

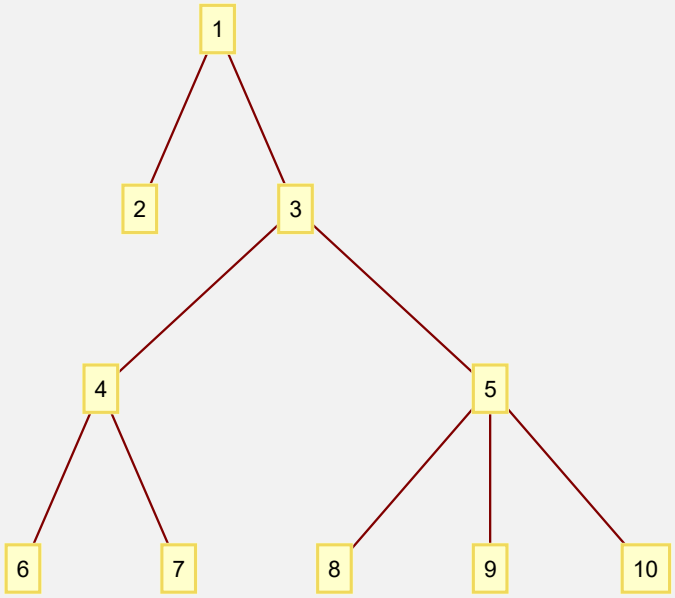
Solución:

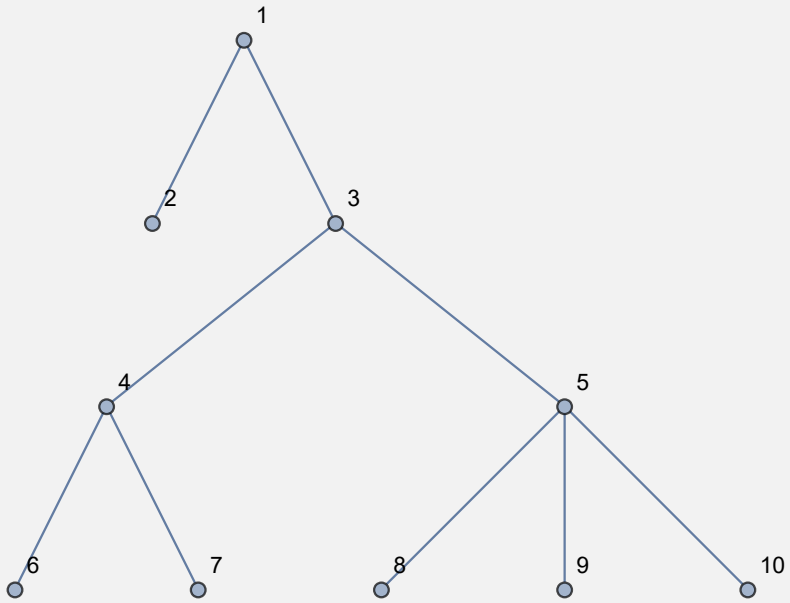
En el software:

In[ ] :=

```
arbol = ArbolGraficoR[{{1, 2}, {1, 3}, {3, 4}, {3, 5}, {4, 6}, {4,
7}, {5, 8}, {5, 9}, {5, 10}}, 1]
ArbolGraficoToArbol[arbol]
```

Out[ ] :=





Ejemplo 8.14

Considere un árbol gráfico dirigido, con raíz 7 y aristas: {{1, 2}, {1, 3}, {1, 11}, {3, 4}, {3, 5}, {3, 15}, {4, 6}, {4, 7}, {5, 8}, {5, 9}, {5, 10}, {10, 12}, {12, 13}, {13, 14}}. Genere otro equivalente en el "Wolfram System".

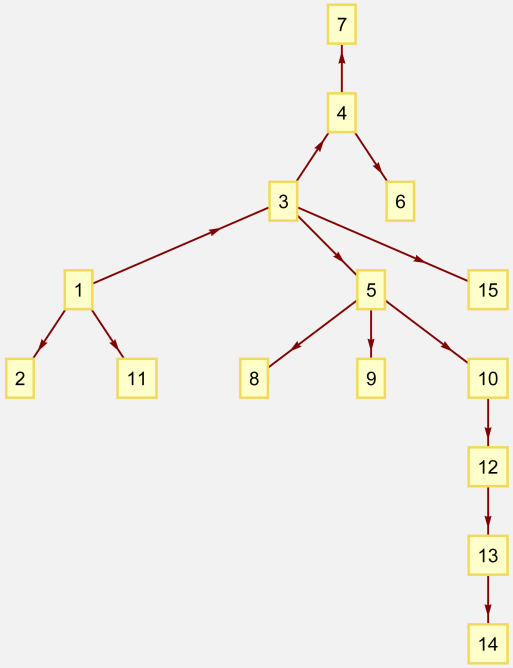
Solución:

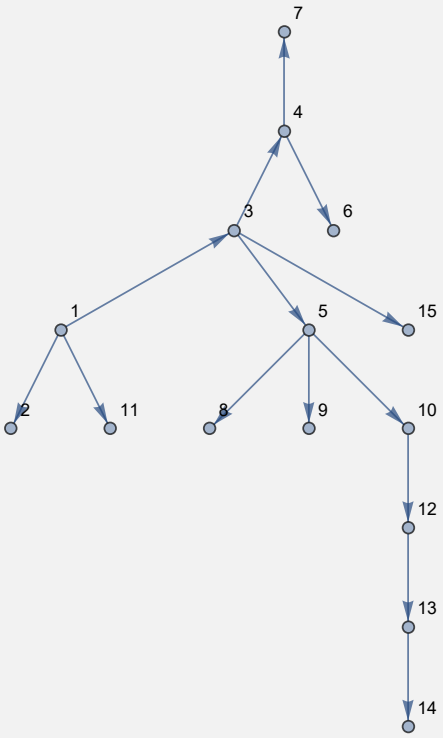
In[ ] :=

```
arbol = ArbolGraficoR[{{1, 2}, {1, 3}, {1, 11}, {3, 4}, {3, 5}, {3,
```

```
15}, {4, 6}, {4, 7}, {5, 8}, {5, 9}, {5, 10}, {10, 12}, {12, 13},
{13, 14}}, 7, dirigido -> True]
ArbolGraficoToArbol[arbol]
```

Out[ ] :=





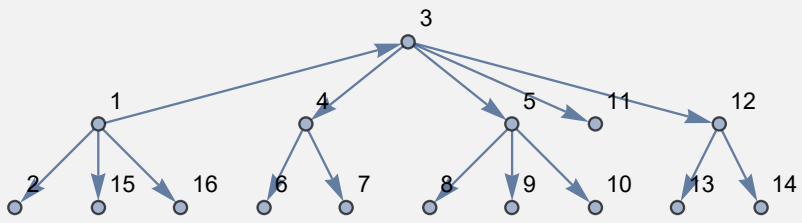
Explicación en video

8) **ArbolToArbolGrafico**: función que **convierte** un árbol "T" **creado** como un **grafo** (**con o sin** "Combinatorica"), a un **árbol** que **corresponde** a un **gráfico**. Brinda la **opción** "**`raiz -> nodo`**" que permite **seleccionar** el **nodo antecesor** a **todos los demás**, en caso de que la instrucción **no sea capaz** de elegir **correctamente** la **raíz**.

Sintaxis: **`ArbolToArbolGrafico[T]`**, o bien, **`ArbolToArbolGrafico[T, raiz -> nodo]`**.

Ejemplo 8.15

Genere el árbol dado a continuación como un árbol gráfico.



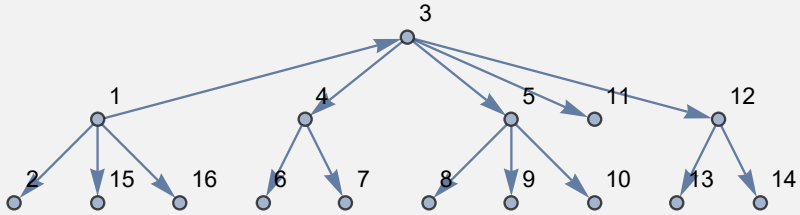
Solución:

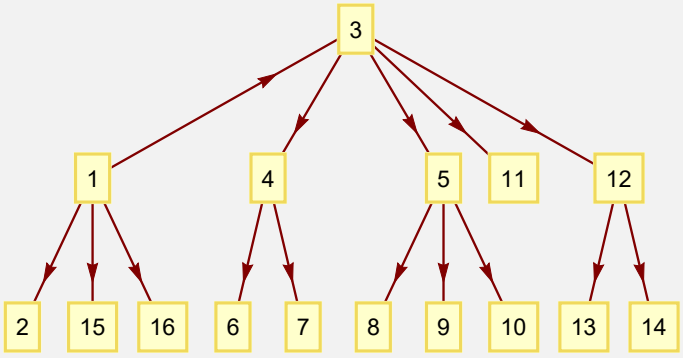
En *Mathematica* el árbol mostrado se creará mediante el uso de la instrucción **`Arbol`**, luego:

In[ ] :=

```
arbol = Arbol[{{1, 2}, {1, 3}, {1, 15}, {1, 16}, {3, 4}, {3, 5}, {3,
11}, {3, 12}, {4, 6}, {4, 7}, {5, 8}, {5, 9}, {5, 10}, {12, 13}, {12,
14}}, dirigido -> True]
ArbolToArbolGrafico[arbol]
```

Out[ ] :=





Ejemplo 8.16

Sea un árbol del "Wolfram System" con raíz 1 y lados: {{1, 2}, {1, 3}, {3, 4}, {3, 5}, {4, 6}, {4, 7}, {5, 8}, {5, 9}, {5, 10}}. Transforme el grafo a un árbol gráfico.

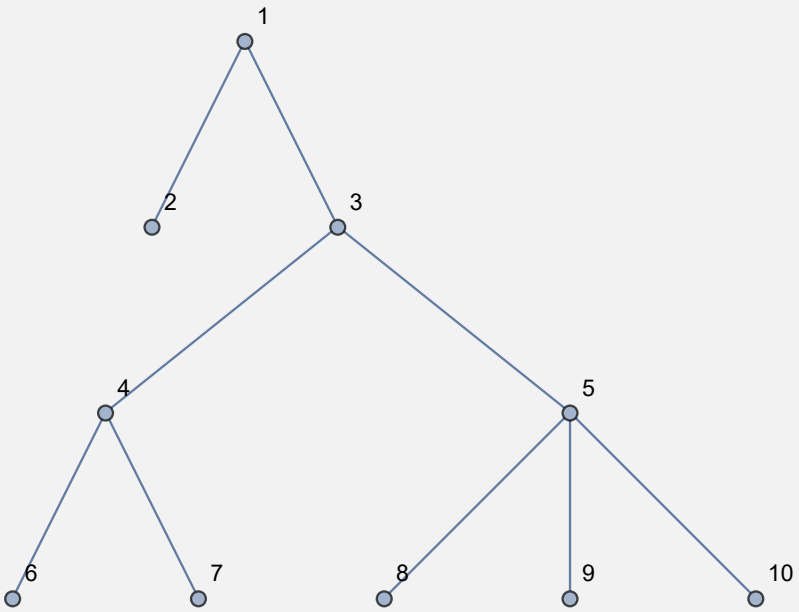
Solución:

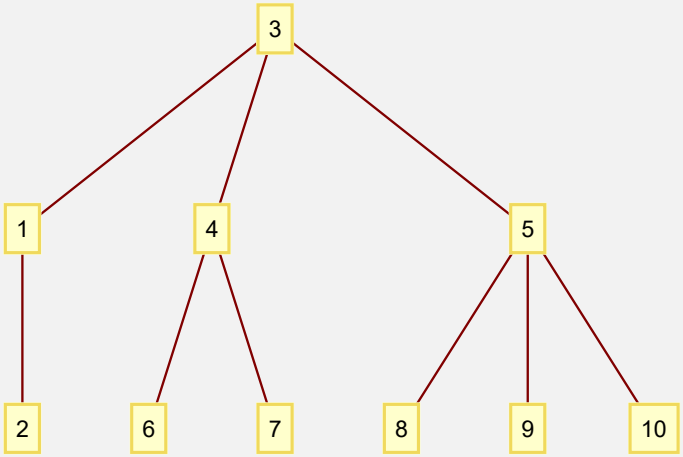
En el software se procede así:

In[ ] :=

```
arbol = ArbolR[{{1, 2}, {1, 3}, {3, 4}, {3, 5}, {4, 6}, {4, 7}, {5,
8}, {5, 9}, {5, 10}}, 1]
ArbolToArbolGrafico[arbol]
```

Out[ ] :=



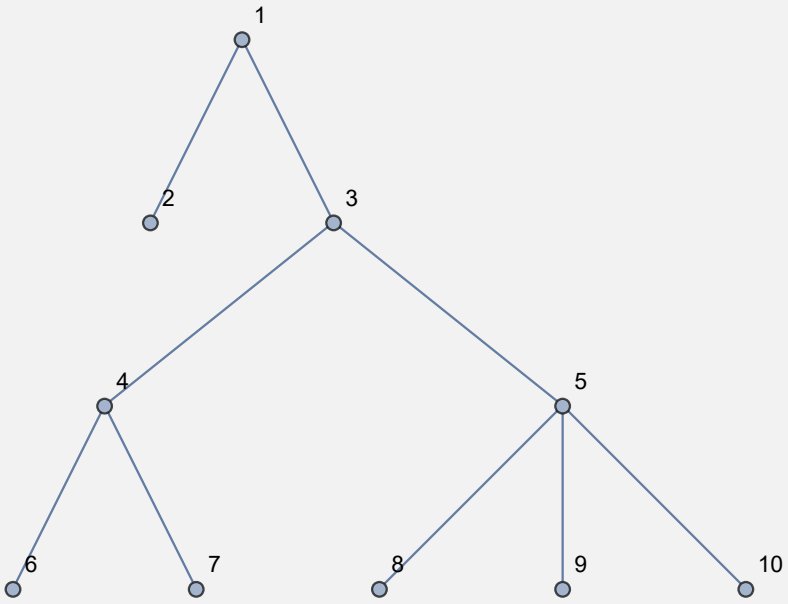


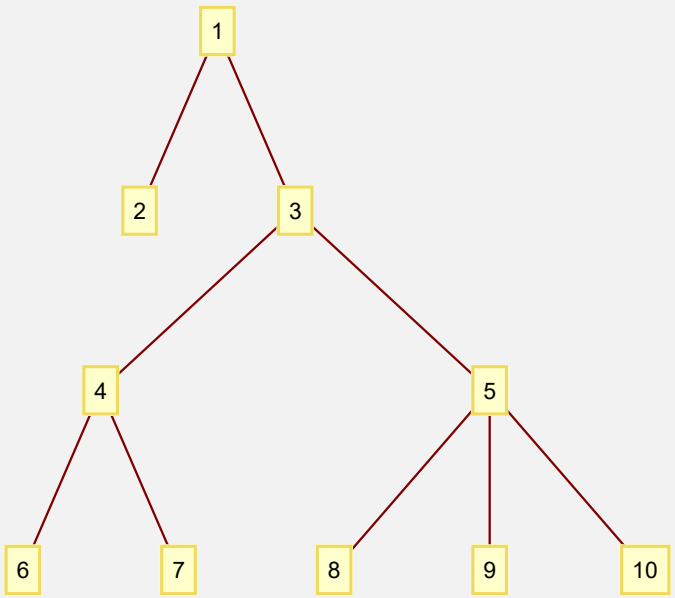
Como la raíz escogida por el comando **`ArbolToArbolGrafico`**, no es 1, se emplea entonces su opción "raiz":

In[ ] :=

```
arbol = ArbolR[{{1, 2}, {1, 3}, {3, 4}, {3, 5}, {4, 6}, {4, 7}, {5,
8}, {5, 9}, {5, 10}}, 1]
ArbolToArbolGrafico[arbol, raiz -> 1]
```

Out[ ] :=





Explicación en video

9) **ArbolRandom**: **construye** un **árbol seudoaleatorio** con "n" **vértices**, por defecto lo genera usando el "Wolfram System" de *Mathematica*. Proporciona al usuario la **opción** "**`combinatorica -> True`**" que muestra el árbol generado en el ambiente provisto por el **paquete** "Combinatorica", quedando almacenado en una **variable denominada** "G". Además, presenta la **alternativa** "**`grafico -> True`**" creando el **árbol seudoaleatorio** como un **gráfico** y no como un grafo.

Sintaxis: **`ArbolRandom[n]`**, o bien, **`ArbolRandom[n, combinatorica -> True]`**, o bien,

```
ArbolRandom[n, grafico->True]
```

Ejemplo 8.17

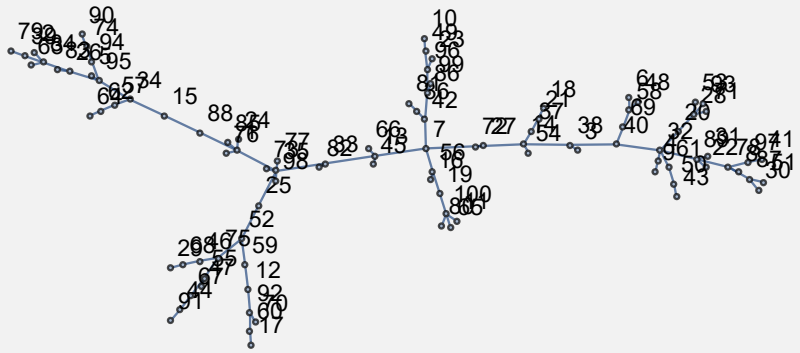
Construya un árbol seudoaleatorio con cien vértices.

Solución:

En el software:

```
In[ ] :=
ArbolRandom[100]
```

Out[ ] :=



Ejemplo 8.18

Haciendo uso del paquete "Combinatorica", muestre un árbol seudoaleatorio con cincuenta nodos.

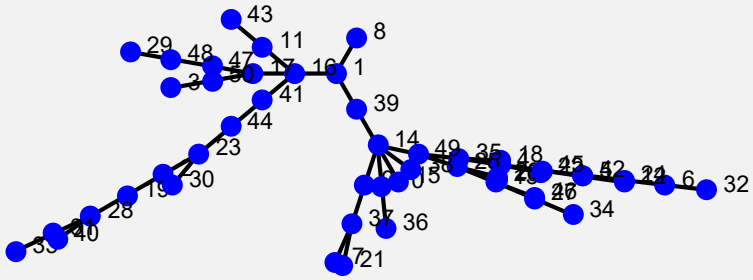
Solución:

La opción "combinatorica" de **`ArbolRandom`**, resuelve lo solicitado:

In[ ] :=

```
ArbolRandom[50, combinatorica -> True]
```

Out[ ] :=



Explicación en video

10) **KArbol**: **genera** un **árbol de orden** "k" con "n" **vértices** usando el "Wolfram System" de *Mathematica*. Proporciona la **opción** "`combinatorica -> True`" que muestra el árbol construido mediante el uso del **paquete** "Combinatorica", quedando almacenado por defecto en una **variable denominada** "G".

Sintaxis: `KArbol[k, n]`, o bien, `KArbol[k, n, combinatorica -> True]`. Si **no** es posible **obtener** el grafo, **retorna** "NaD".

Ejemplo 8.19

Construya mediante **KArbol** un árbol de orden diez con cincuenta vértices.

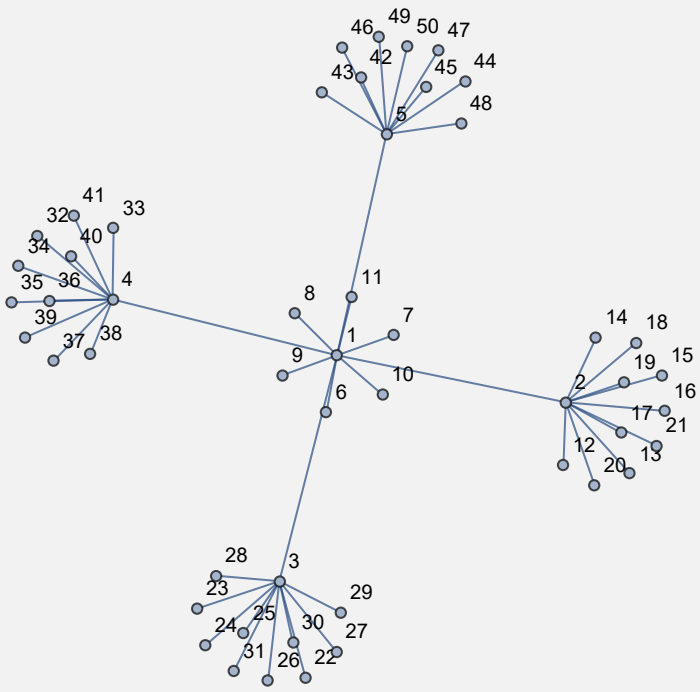
Solución:

En *Mathematica*:

In[ ] :=

```
KArbol[10, 50]
```

Out[ ] :=



Ejemplo 8.20

En el ambiente provisto por el paquete "Combinatorica", retorne un *k*-árbol de orden tres con siete nodos.

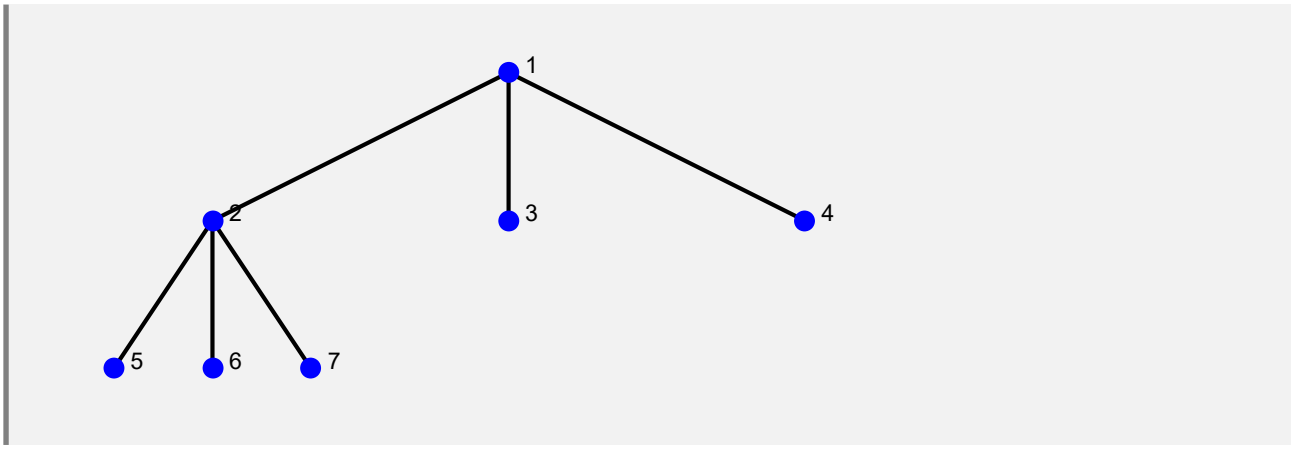
Solución:

Para ello, se recurrirá a la opción "combinatorica" del comando **KArbol**:

In[ ] :=

```
KArbol[3, 7, combinatorica -> True]
```

Out[ ] :=



Explicación en video

11) **KArbolCompletoNivel**: **construye** un "k" **árbol completo** con "n" **niveles** en el "Wolfram System" de *Mathematica*, donde **todos los nodos** por **nivel**, exceptuando las hojas, **tienen** "k" **hijos**.

Sintaxis: `KArbolCompletoNivel[k, n]`. Si **no** es posible **generar** el grafo, **retorna** "NaD".

Ejemplo 8.21

Construya un árbol completo de orden cinco con cuatro niveles.

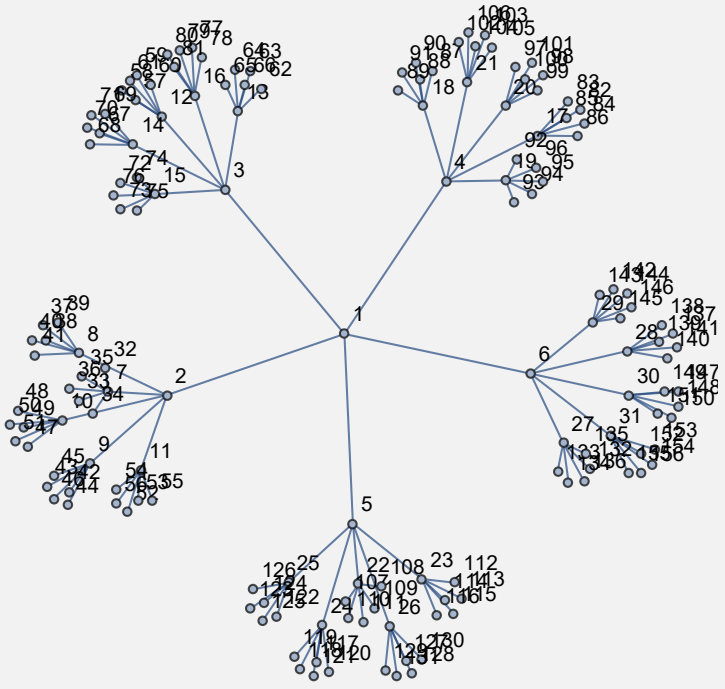
Solución:

En el software:

In[ ] :=

```
KArbolCompletoNivel[5, 4]
```

Out[ ] :=



Ejemplo 8.22

Mediante el uso de la instrucción **`KArbolCompletoNivel`**, genere un 5-árbol con cinco niveles.

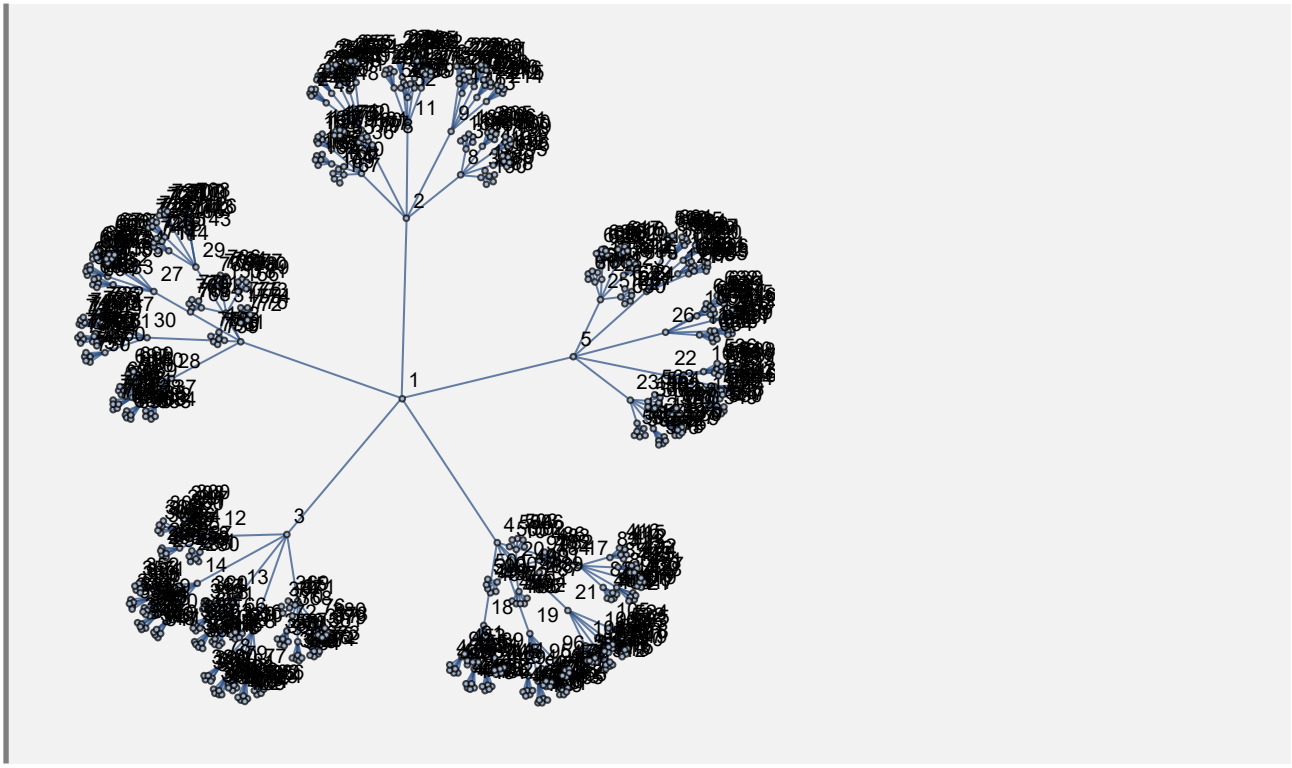
Solución:

En *Mathematica*:

In[ ] :=

```
KArbolCompletoNivel[5, 5]
```

Out[ ] :=



Explicación en video

12) **ArbolBinario**: **genera** un **árbol binario** con "n" **vértices** usando el "Wolfram System" de *Mathematica*. Brinda la **opción** "**`combinatorica -> True`**" que muestra el árbol construido mediante el uso del **paquete** "Combinatorica", quedando almacenado por defecto en una **variable denominada** "G".

Sintaxis: **`ArbolBinario[n]`**, o bien, **`ArbolBinario[n, combinatorica -> True]`**.

Ejemplo 8.23

Obtenga un árbol binario con cincuenta nodos a través del comando **`ArbolBinario`**. Convierta el grafo a un árbol gráfico.

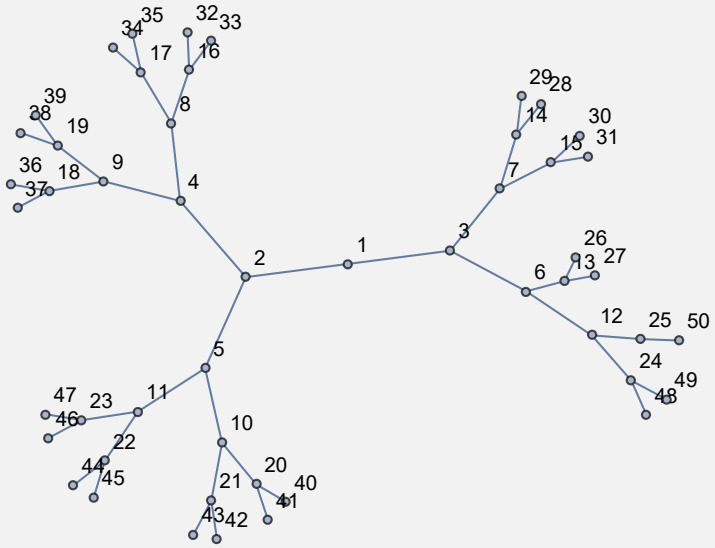
Solución:

El árbol se produce así:

In[ ] :=

```
arbol = ArbolBinario[50]
```

Out[ ] :=

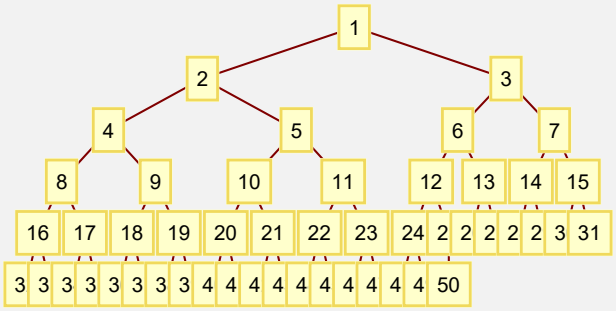


Luego, al estar almacenado el grafo en la variable **arbol**, la conversión solicitada, se logra pasando el contenido de **arbol** a la instrucción **ArbolToArbolGrafico**:

In[ ] :=

```
ArbolToArbolGrafico[arbol]
```

Out[ ] :=



Ejemplo 8.24

Muestre un árbol binario con cincuenta nodos en el ambiente provisto por el paquete "Combinatorica".

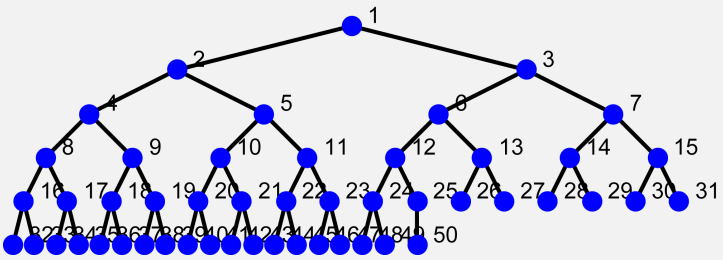
Solución:

Para ello se recurrirá a la opción "**combinatorica->True**":

In[ ] :=

```
ArbolBinario[50, combinatorica->True]
```

Out[ ] :=



Explicación en video

13) **ArbolBinarioCompletoNivel**: **construye** un **árbol binario completo** con "n" **niveles** en el "Wolfram System" de *Mathematica*, donde **todos los nodos** por **nivel**, exceptuando las hojas, **tienen dos hijos**.

Sintaxis: `ArbolBinarioCompletoNivel[n]`. Si **no** logra **generar** el grafo, **retorna** "NaD".

Ejemplo 8.25

Genere un árbol binario completo con seis niveles.

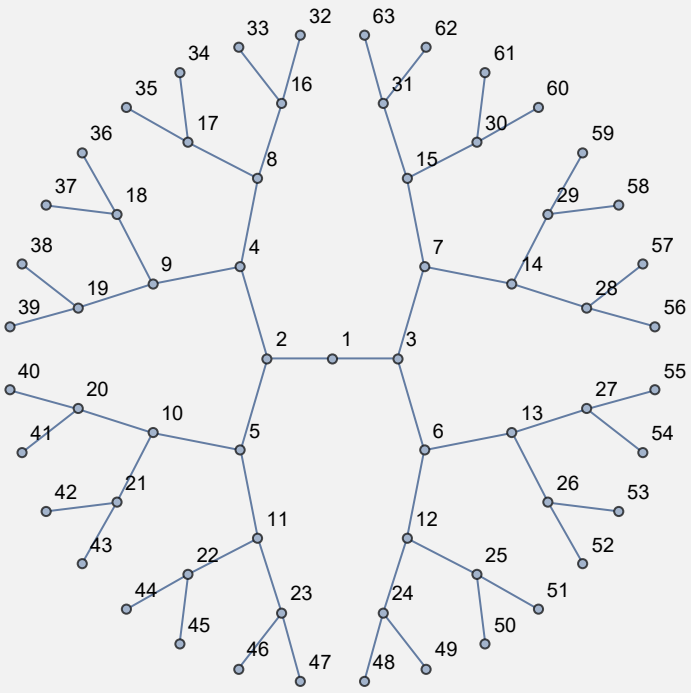
Solución:

En el software:

In[ ] :=

```
ArbolBinarioCompletoNivel[6]
```

Out[ ] :=



Ejemplo 8.26

Construya un árbol binario completo con ocho niveles.

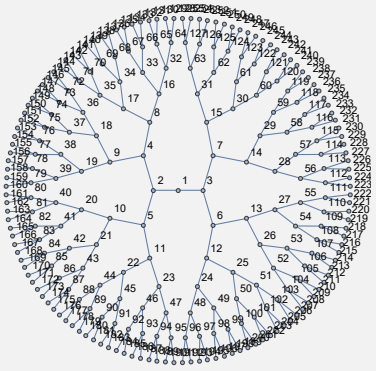
Solución:

En *Mathematica*:

In[ ] :=

```
ArbolBinarioCompletoNivel[8]
```

Out[ ] :=



Explicación en video

14) **ArbolBinarioRandom**: **construye** un **árbol binario seudoaleatorio** con "n" **vértices** en el "Wolfram System" de *Mathematica*. Brinda al usuario la **opción** "**`combinatorica -> True`**" que muestra el árbol generado en el ambiente provisto por el **paquete** "Combinatorica", quedando almacenado por defecto en una **variable denominada** "G". Además, presenta la **alternativa** "**`grafico -> True`**" la cual instaura el **árbol binario seudoaleatorio** como un **gráfico** y no como un grafo.

Sintaxis: **`ArbolBinarioRandom[n]`**, o bien, **`ArbolBinarioRandom[n, combinatorica -> True]`**, o bien,

```
ArbolBinarioRandom[n, grafico->True]
```

Ejemplo 8.27

Forme un árbol binario seudoaleatorio con cien nodos.

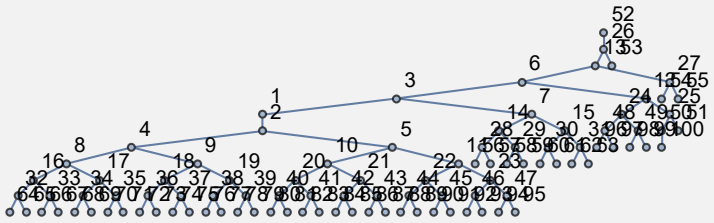
Solución:

Al utilizar el comando **`ArbolBinarioRandom`**:

In[ ] :=

```
ArbolBinarioRandom[100]
```

Out[ ] :=



Ejemplo 8.28

Muestre un árbol binario seudoaleatorio con cincuenta vértices haciendo uso del paquete "Combinatorica".

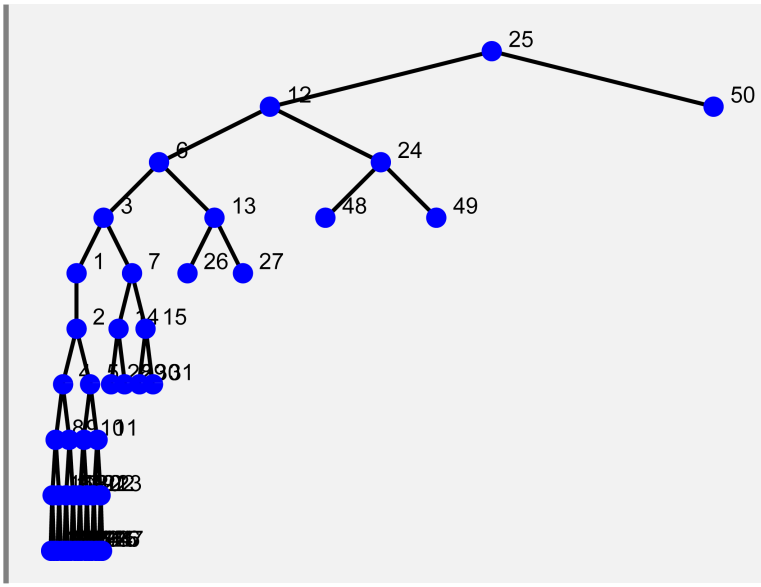
Solución:

Al emplear la opción "**`combinatorica -> True`**", se tiene:

In[ ] :=

```
ArbolBinarioRandom[50, combinatorica -> True]
```

Out[ ] :=



Explicación en video

15) **ArbolBinarioQ**: **función booleana** utilizada para indicar si un **árbol** "T" recibido como parámetro es o no **binario**. El árbol pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, con el **paquete** "Combinatorica", o bien, como un **árbol gráfico** (en este caso, no corre si tiene una **mezcla de números y letras**).

Sintaxis: `ArbolBinarioQ[T]`.

Ejemplo 8.29

Invoque **`ArbolBinario[50]`**. Verifique que su salida es un árbol binario a través de la instrucción **`ArbolBinarioQ`**.

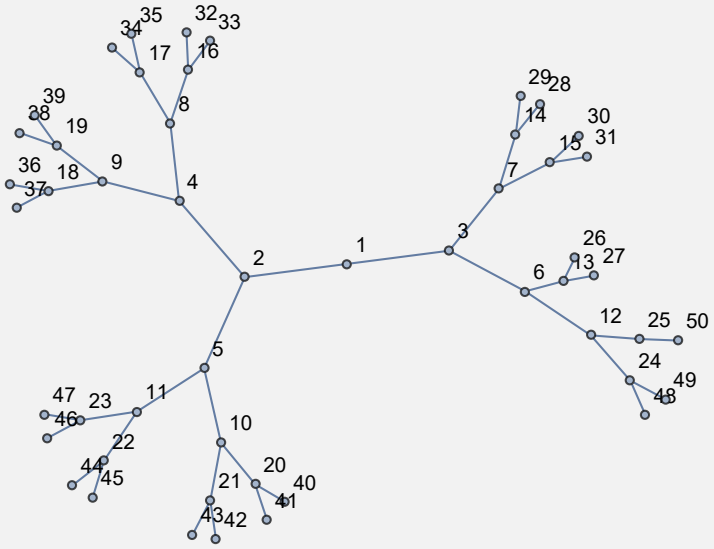
Solución:

En *Mathematica*:

In[ ] :=

```
arbol = ArbolBinario[50]
ArbolBinarioQ[arbol]
```

Out[ ] :=



True

Asumiendo la raíz en el conjunto de nodos: {1, 25}

N Es interesante observar cómo el comando retorna los vértices que pueden corresponder a la raíz del árbol, con la intención de ser binario.

Ejemplo 8.30

Verifique que el grafo **`ArbolBinarioCompletoNivel[6]`** convertido a un árbol gráfico es binario.

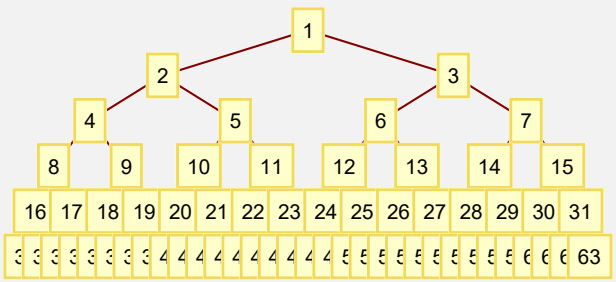
Solución:

Se resolverá el ejemplo, utilizando las instrucciones **`ArbolToArbolGrafico`** y **`ArbolBinarioQ`**:

In[ ] :=

```
arbol = ArbolToArbolGrafico[ArbolBinarioCompletoNivel[6]]
ArbolBinarioQ[arbol]
```

Out[ ] :=



True

Asumiendo la raíz en el conjunto de nodos: {1}

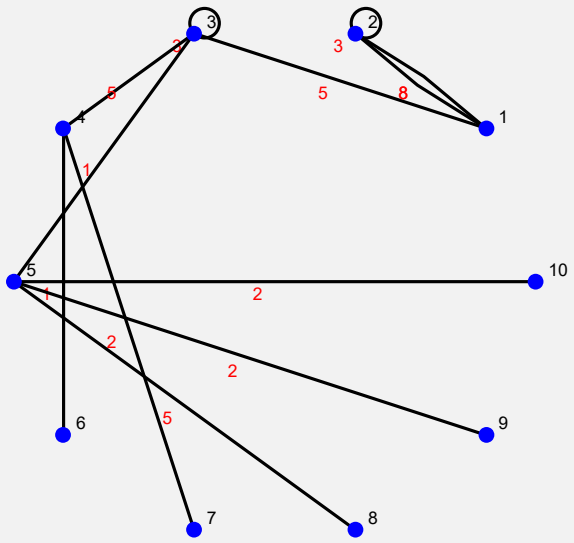
Explicación en video

16) **ArbolJQ**: retorna "`True`" si el argumento "T" recibido como parámetro es un **grafo** en *Mathematica* que corresponde a un **árbol** o "`False`", en caso contrario. Acepta grafos creados con el **paquete** "Combinatorica". La instrucción brinda además, una **justificación** en caso de recibir un grafo que **no corresponde** a un **árbol**.

Sintaxis: `ArbolJQ[T]`.

Ejemplo 8.31

El siguiente grafo, ¿es un árbol?



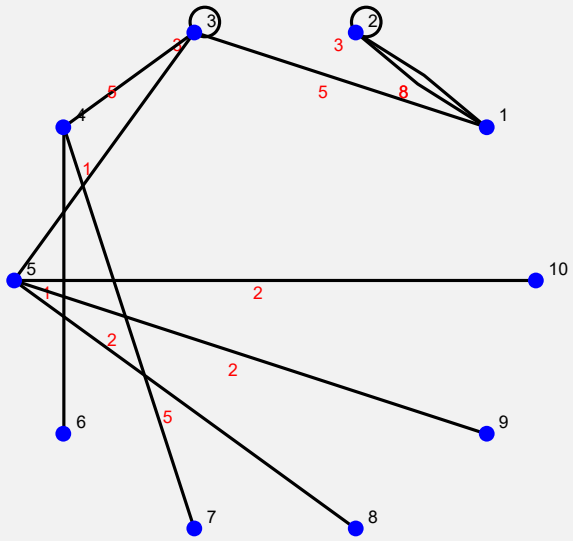
Solución:

Se creará el grafo utilizando **`GrafoC`**:

In[ ] :=

```
GrafoC[{{1, 2}, {1, 3}, {2, 1}, {2, 2}, {3, 3}, {3, 4}, {3, 5}, {4,
6}, {4, 7}, {5, 8}, {5, 9}, {5, 10}}, pesos->{8, 5, 8, 3, 3, 5, 1,
1, 2, 5, 2, 2}, mostrarpesos->True]
ArbolJQ[G]
```

Out[ ] :=

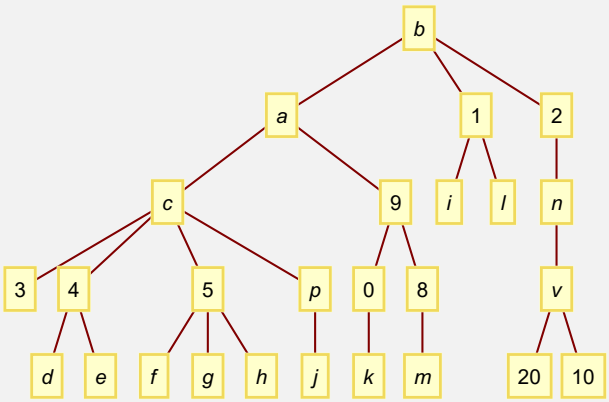


False
Tiene circuitos

N Como el valor lógico es "**`False`**", el comando **`ArbolJQ`** justifica mediante la condición Tiene circuitos, el por qué el grafo no satisface la definición de árbol.

Ejemplo 8.32

Sea el siguiente árbol gráfico:



Verique al usar la instrucción **`ArbolJQ`**, el retorno del valor lógico "**`False`**" ¿Por qué se obtiene este resultado si la gráfica anterior es un árbol?

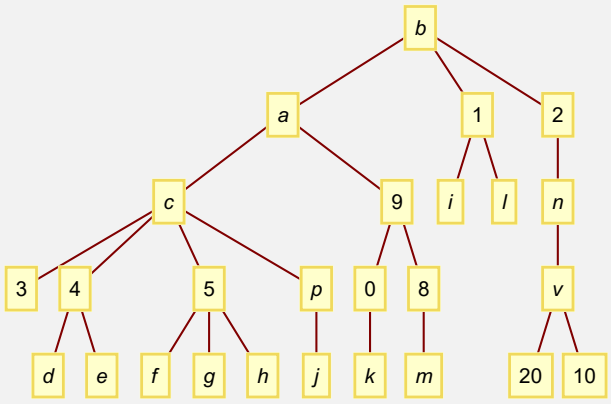
Solución:

En el software:

In[ ] :=

```
arbol = ArbolGrafico[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4},
{c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}, {a, 9}, {9, 0}, {9,
8}, {c, p}, {p, j}, {1, i}, {1, l}, {8, m}, {0, k}, {2, n}, {n, v},
{v, 20}, {v, 10}}]
ArbolJQ[arbol]
```

Out[ ] :=



False

El argumento no es un grafo

N El estudiante ¡debe recordar! que un árbol gráfico es un objeto **`Graphics`** y por consiguiente, como bien lo indica la salida, dicha estructura no es un grafo en el software *Mathematica*. Por esta razón, se regresa un False.

Explicación en video

17) **Hojas**: **retorna** los **vértices finales** u **hojas** de un **árbol** "T" recibido como parámetro (**creado** en el "Wolfram System" o con el **paquete** "Combinatorica"). Se advierte al usuario que la función presupone que la raíz del árbol **no** tiene **valencia igual a uno**, en caso contrario, brinda la **opción** "**`raiz -> nodo`**", donde se **especifica** un vértice como **raíz**. Si este vértice posee valencia uno, ya **no se muestra** como un nodo terminal.

Sintaxis: **`Hojas[T]`**, o bien, **`Hojas[T, raiz -> nodo]`**. **No acepta** árboles **gráficos**.

Ejemplo 8.33

Devuelva los vértices terminales de un árbol con raíz "g", cuyas aristas son: {{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}.

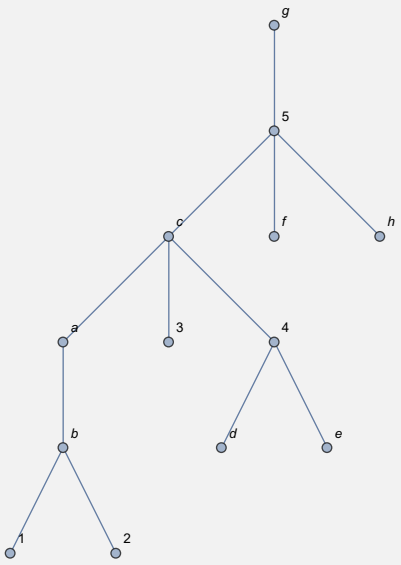
Solución:

En *Mathematica*:

In[ ] :=

```
arbol = ArbolR[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c,
5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}, g]
Hojas[arbol]
```

Out[ ] :=



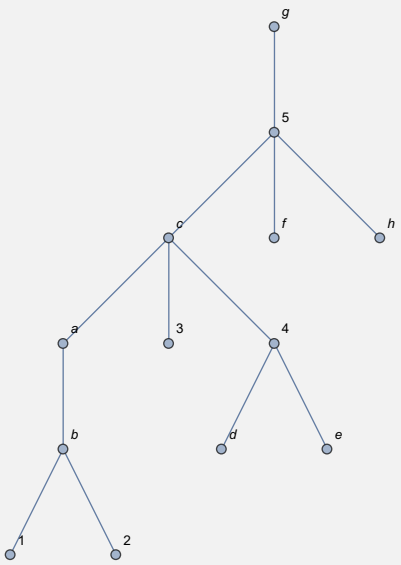
{1, 2, 3, d, e, f, g, h}

El comando **Hojas** incluye en este ejercicio la raíz "g" en el conjunto de nodos finales. Por este motivo, se debe utilizar su opción "raiz" como se detalla a continuación:

In[ ] :=

```
arbol = ArbolR[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c,
5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}, g]
Hojas[arbol, raiz -> g]
```

Out[ ] :=



{1, 2, 3, d, e, f, h}

Ejemplo 8.34

Determine las hojas de un arbol binario con veinte vértices creado en "Combinatorica".

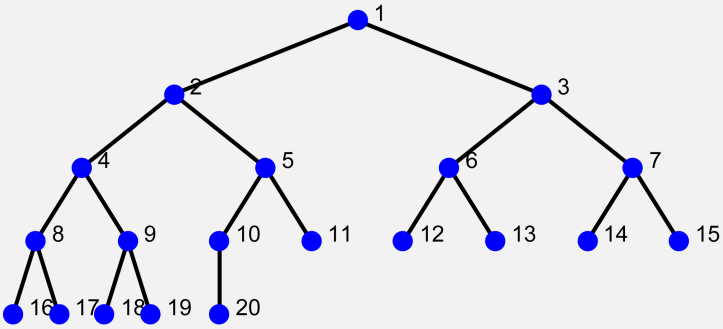
Solución:

En el software:

In[ ] :=

```
ArbolBinario[20, combinatorica -> True]
Hojas[G]
```

Out[ ] :=



{11, 12, 13, 14, 15, 16, 17, 18, 19, 20}

Explicación en video

18) **Altura**: **devuelve** la **altura** de un **árbol** "T" recibido como parámetro (**creado** en el "Wolfram System" o con el **paquete** "Combinatorica").

Sintaxis: `Altura[T, raiz]`, siendo "raiz" la **raíz** del árbol correspondiente. El cálculo se realiza **asumiendo** un **nivel cero** para la **raíz**. **No acepta** árboles **gráficos**.

Ejemplo 8.35

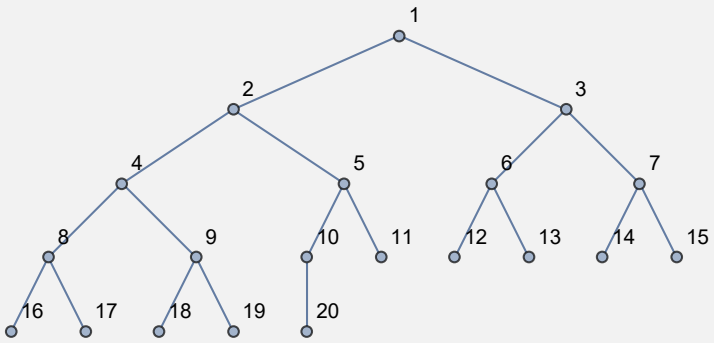
Halle la altura del árbol obtenido al ejecutar **`ArbolBinario[20]`**.

Solución:

In[ ] :=

```
arbol = ArbolBinario[20]
Altura[arbol, 1]
```

Out[ ] :=

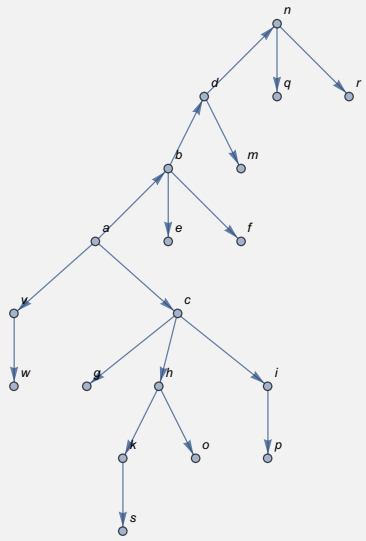


4

La altura del árbol es cuatro.

Ejemplo 8.36

Encuentre la altura de:



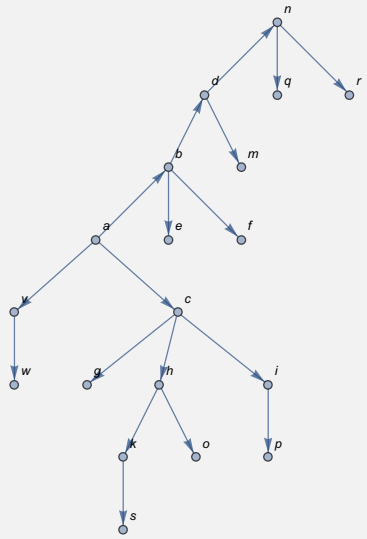
Solución:

El árbol se construirá mediante el uso de la instrucción **ArbolR**, luego:

In[ ] :=

```
arbol = ArbolR[{{a, b}, {a, v}, {v, w}, {a, c}, {b, d}, {b, e}, {b,
f}, {c, g}, {c, h}, {c, i}, {d, m}, {d, n}, {h, k}, {h, o}, {i, p},
{k, s}, {n, q}, {n, r}}, n, dirigido -> True]
Altura[arbol, n]
```

Out[ ] :=



Explicación en video

19) **CantAristasKArbolCompletoNivel**: **determina** el **número de aristas** de un "k" **árbol completo** con "n" **niveles**, donde **todos los nodos** por **nivel**, exceptuando las hojas, **tienen** "k" **hijos**. Proporciona al usuario la **opción** "**mostrar->True**" que **genera** adicionalmente el **árbol correspondiente** si es posible, en caso contrario, **retorna** "NaD".

Sintaxis: `CantAristasKArbolCompletoNivel[k, n]`, o bien,

```
CantAristasKArbolCompletoNivel[k, n, mostrar->True]
```

Ejemplo 8.37

Halle el número de lados de un árbol completo de orden cien con cien niveles. Muestre el grafo si es posible.

Solución:

En *Mathematica*:

```
In[ ] :=
CantAristasKArbolCompletoNivel[100, 100, mostrar->True]

Out[ ] :=
NaD
10101010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101010101100
```

N El NaD indica el impedimento de mostrar el grafo, lo cuál tiene sentido al observar la elevada cantidad de aristas que contiene.

Ejemplo 8.38

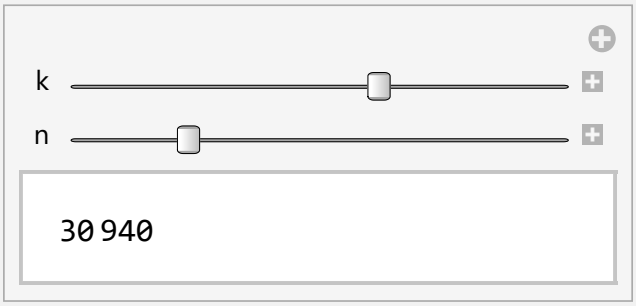
Conjeture a través del uso del software *Mathematica*, una fórmula que calcule el número de aristas de un *k*-árbol completo, en sus "n" niveles.

Solución:

En este ejemplo, se empleará una animación que evalúa en distintos árboles completos, la cantidad de lados que poseen, con el objetivo de generalizar la fórmula solicitada observando los casos particulares:

In[ ] :=
```
Manipulate[CantAristasKArbolCompletoNivel[k, n], {k, 1, 20, 1}, {n,
1, 20, 1}]
```

Out[ ] :=



N Al jugar con los manipuladores del *Out* [], se intuye que la cantidad de aristas viene dada por: $\frac{k^n - k}{k - 1}$ si k es distinto de uno y si es igual a uno, corresponde a: $n - 1$.

Explicación en video

20) **CantNodosKArbolCompletoNivel**: **determina** el **número de vértices** de un "k" **árbol completo** con "n" **niveles**, donde **todos los nodos** por **nivel**, exceptuando las hojas, **tienen** "k" **hijos**. Proporciona al usuario la **opción** "`mostrar -> True`" que **genera** adicionalmente el **árbol correspondiente** si es posible, en caso contrario, **retorna** "NaD".

Sintaxis: `CantNodosKArbolCompletoNivel[k, n]`, o bien,

```
CantNodosKArbolCompletoNivel[k, n, mostrar->True]
```

Ejemplo 8.39

Encuentre la cantidad de nodos de un árbol completo de orden cien con cien niveles. Muestre el grafo si es posible.

Solución:

In[ ] :=
```
CantNodosKArbolCompletoNivel[100, 100, mostrar->True]
```

Out[ ] :=
NaD

```
10101010101010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101010101010101010101010101010101010101010101010101010
```

N En correspondencia con otro ejercicio anterior, el árbol no se puede mostrar y cabe destacar también, el número de vértices tan alto que contiene.

Ejemplo 8.40

Conjeture utilizando el software *Mathematica*, una fórmula que determine el número de nodos de un k-árbol completo, en sus "n" niveles.

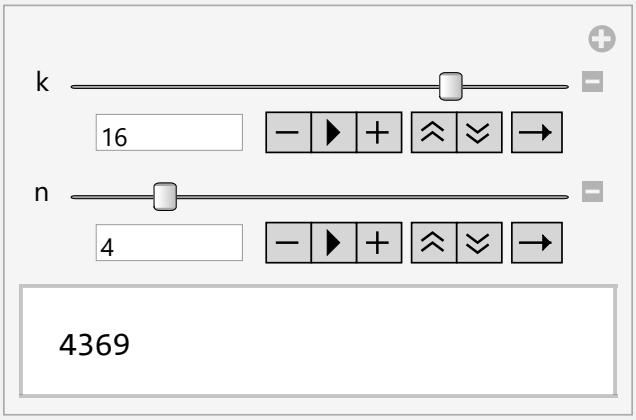
Solución:

Se prodecerá de foma similar al ejercicio tras anterior:

In[ ] :=

```
Manipulate[CantNodosKArbolCompletoNivel[k, n], {k, 1, 20, 1}, {n, 1, 20, 1}]
```

Out[ ] :=



N Se puede inferir a través de un proceso exploratorio en la salida mostrada, que la cantidad de vértices viene dada por: $\frac{k^n - 1}{k - 1}$ si k es distinto de uno y si es igual a uno, corresponde a: n (uno más en comparación con la cantidad de aristas).

Explicación en video

21) **NHojasTotalBinario**: **encuentra** el **número** de **vértices internos**, **hojas** y **nodos** en **total** de un **árbol binario completo** con "n" **niveles**, donde **todos los vértices internos** por **nivel** tienen **dos hijos**. El resultado se **almacena** en un **vector** "{vértices internos, hojas, total}". Presenta la **opción** "**mostrar -> True**" que muestra adicionalmente el **árbol correspondiente** si es posible, en caso contrario, **retorna** "NaD".

Sintaxis: `NHojasTotalBinario[n]`, o bien, `NHojasTotalBinario[n, mostrar -> True]`.

Ejemplo 8.41

Encuentre la cantidad de nodos internos, terminales y vértices en total, en un arbol binario completo con 1000 niveles. Si es posible muestre el árbol.

Solución:

En el software:

In[ ] :=
```
NHojasTotalBinario[1000, mostrar -> True]
```

Out[ ] :=
NaD
{535754303593133660474212524530000905280702405852766803721875194185175
525562468061246599189407847929063797336458776573412593572642846157021
992288787349287401967283887412115492710537302531185570938977091076523
374917909706336993837795827719730385314572855982388432710838302149158
2631219341860283403468 7, 535754303593133660474212524530000905280702405852
766803721875194185175525562468061246599189407847929063797336458776573
412593572642846157021799228878734928740196728388741211549271053730253
118557093897709107652323749179097063369938377958277197303853145728559
823884327108383021491582631219341860283403468 8, 107150860718626732094842
504906000181056140481170553360744375038837035105112493612249319837881
569585812759467291755314682518714528569231404359845775746985748039345
677482423098542107460506237114187795418215304647498358194126739876755
916554394607706291457119647768654216766042983165262438683720566806937
5}

El grafo no es retornado por el comando, dada la cantidad de vértices que posee.

Ejemplo 8.42

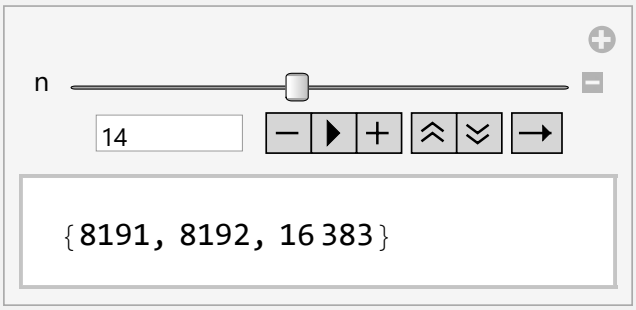
Conjeture a través del uso de software, cuántos vértices internos, hojas y nodos en total, contiene un árbol binario completo, en sus "n" niveles.

Solución:

Al recurrir a una animación, se tiene:

In[ ] :=
```
Manipulate[NHojasTotalBinario[n], {n, 1, 30, 1}]
```

Out[ ] :=



N El manipulador retorna distintos vectores de conteo en el orden de la salida arrojada por la instrucción **`NHojasTotalBinario`**, considerando árboles desde el nivel 1 hasta el 30. De los resultados mostrados, se puede conjeturar que: {vértices internos, hojas, total}=$\{2^{n-1}-1, 2^{n-1}, 2^n-1\}$ si "n" es distinto de uno y si "n" es igual a uno, corresponde a: {1, 0, 1}.

Explicación en video

22) **CDFKArbol**: **genera** una **animación** que muestra un **árbol** de **orden** "k" con "n" **vértices**, **determinando** sus **hojas** y su **altura**. Además, el usuario puede **seleccionar** de un **combo** cuál es la **raíz** del árbol. Adicionalmente, aparecen dos campos de texto donde es posible **cambiar** el **orden** y la **cantidad de nodos**.

Sintaxis: **`CDFKArbol[k, n]`**.

Ejemplo 8.43

Ejecute **`CDFKArbol[8, 12]`** y cambie la raíz al nodo 12.

Solución:

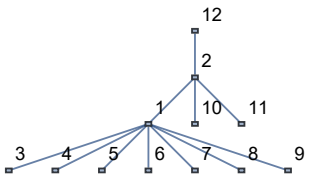
In[ ] :=
```
CDFKArbol[8, 12]
```
Out[ ] :=

Orden y cantidad de vértices del árbol

Orden del árbol 8

Cantidad de nodos 12

Raíz 12




```
Las hojas del árbol son: {3, 4, 5, 6, 7, 8, 9, 10, 11}
La altura del árbol es: 3
```

Ejemplo 8.44

Construya con el comando **`CDFKArbol`** un árbol de orden diez con quince vértices.

Solución:

En *Mathematica*:

In[ ] :=

```
CDFKArbol[10, 15]
```

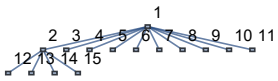
Out[ ] :=

Orden y cantidad de vértices del árbol

Orden del árbol 10

Cantidad de nodos 15

Raíz 1




```
Las hojas del árbol son: {3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}
La altura del árbol es: 2
```

N **`CDFKArbol`** puede proporcionar una ayuda importante al alumno, para interiorizar los conceptos de árbol, raíz, hojas y altura.

Explicación en video

23) **ArbolHuffmanN**: **construye** un **árbol** de **códigos de Huffman** no optimizado, **dado** el "string" que se **desea codificar**. Brinda las **opciones** "**grafico->True**" que muestra el árbol como un **gráfico** y no como un grafo y, "**frecuencias->True**" que **exhibe** un **vector** con las **frecuencias** de **aparición** de cada uno de los caracteres del "string".

Sintaxis: **ArbolHuffmanN[string]**, o bien,

```
ArbolHuffmanN[string, grafico->True, frecuencias->True]
```

pudiendo **prescindir** de cualquiera de las opciones.

Ejemplo 8.45

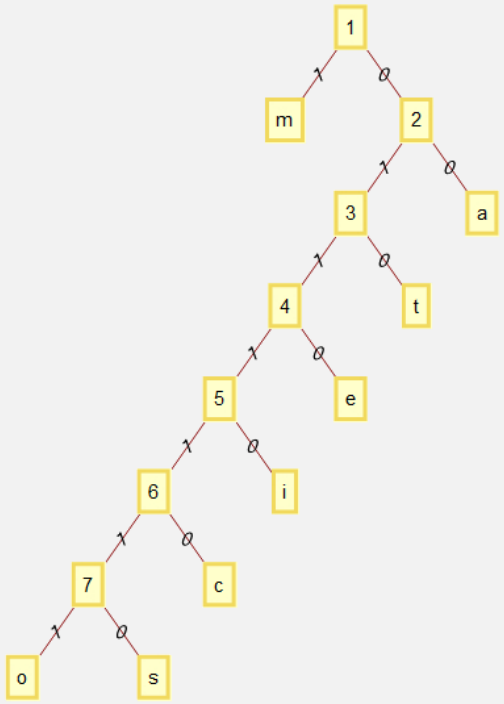
Muestre un árbol de códigos de *Huffman* no optimizado para la palabra "matematicos", mediante un árbol gráfico. Presente las frecuencias de cada uno de sus caracteres.

Solución:

Al recurrir a **ArbolHuffmanN**:

In[ ] :=
```
ArbolHuffmanN["matematicos", grafico->True, frecuencias->True]
```

Out[ ] :=
{{m, 2}, {a, 2}, {t, 2}, {e, 1}, {i, 1}, {c, 1}, {o, 1}, {s, 1}}



N En teoría, el primer carácter del árbol de códigos de *Huffman* puede ser "m", "a" o "t". Se aclara al lector que el comando **`ArbolHuffmanN`** elige el primero en aparición dentro del "string", por lo que en este ejemplo, se manifiesta la "m" al inicio.

Ejemplo 8.46

Considere la hilera de caracteres "massachhusssettsss". Usando **`ArbolHuffmanN`**, devuelva un árbol de códigos de *Huffman* y muestre las frecuencias.

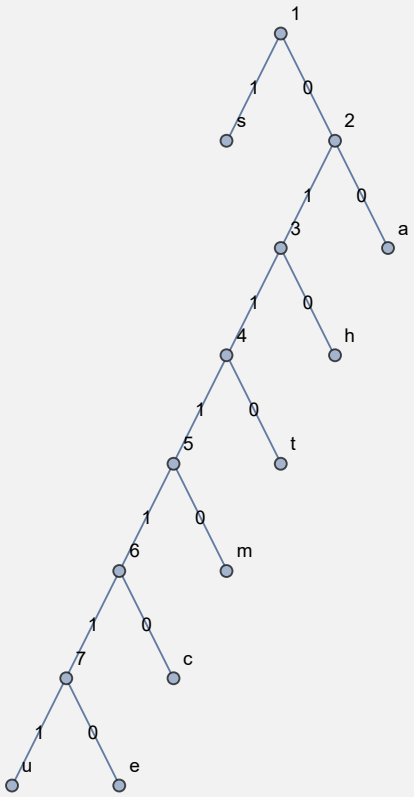
Solución:

In[ ] :=

```
ArbolHuffmanN["massachhusssettsss", frecuencias -> True]
```

Out[ ] :=

{{s, 8}, {a, 2}, {h, 2}, {t, 2}, {m, 1}, {c, 1}, {u, 1}, {e, 1}}



Explicación en video

24) **ArbolHuffman**: **construye** un **árbol** de **códigos de Huffman** optimizado, **dado** el "string" que se **desea codificar**. Proporciona las **opciones** "**`grafico -> True`**" que muestra el árbol como un **gráfico** y no como un grafo y, "**`frecuencias -> True`**" que **exhibe** un **vector** con las **frecuencias utilizadas** por el comando.

Sintaxis: **`ArbolHuffman[string]`**, o bien, **`ArbolHuffman[string, grafico -> True, frecuencias -> True]`**, pudiendo **prescindir** de cualquiera de las opciones.

Ejemplo 8.47

Construya un árbol de códigos de *Huffman* optimizado para la hilera "matematicos" a través de un gráfico. Exhiba las frecuencias de cada uno de sus caracteres.

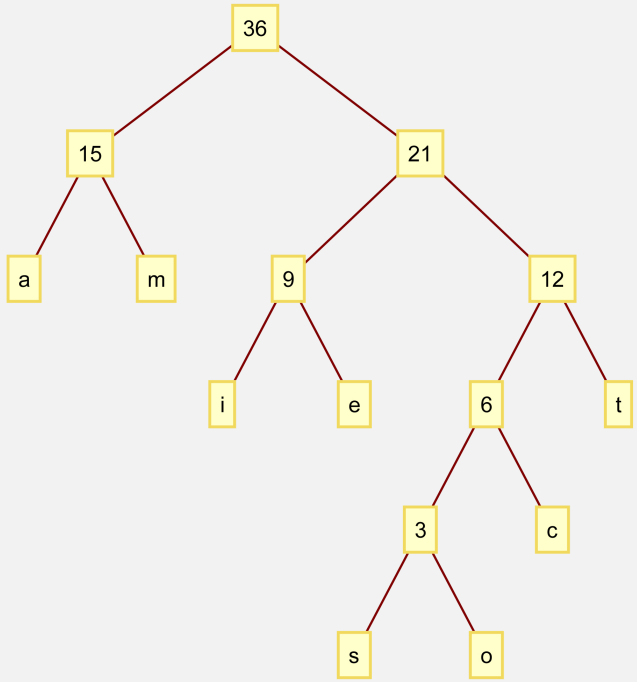
Solución:

En *Mathematica*:

```
In[ ] :=
ArbolHuffman["matematicos", grafico -> True, frecuencias -> True]
```

Out[ ] :=

{{s, 1}, {o, 2}, {c, 3}, {i, 4}, {e, 5}, {t, 6}, {a, 7}, {m, 8}}



N El estudiante observará, cómo las frecuencias son asignadas de manera automática, mediante números naturales consecutivos comenzando en uno.

Ejemplo 8.48

Sea el "string": "massachhusssettsss". Por medio de la instrucción **`ArbolHuffman`**, retorne un árbol de códigos de *Huffman* optimizado y las frecuencias asignadas a cada carácter.

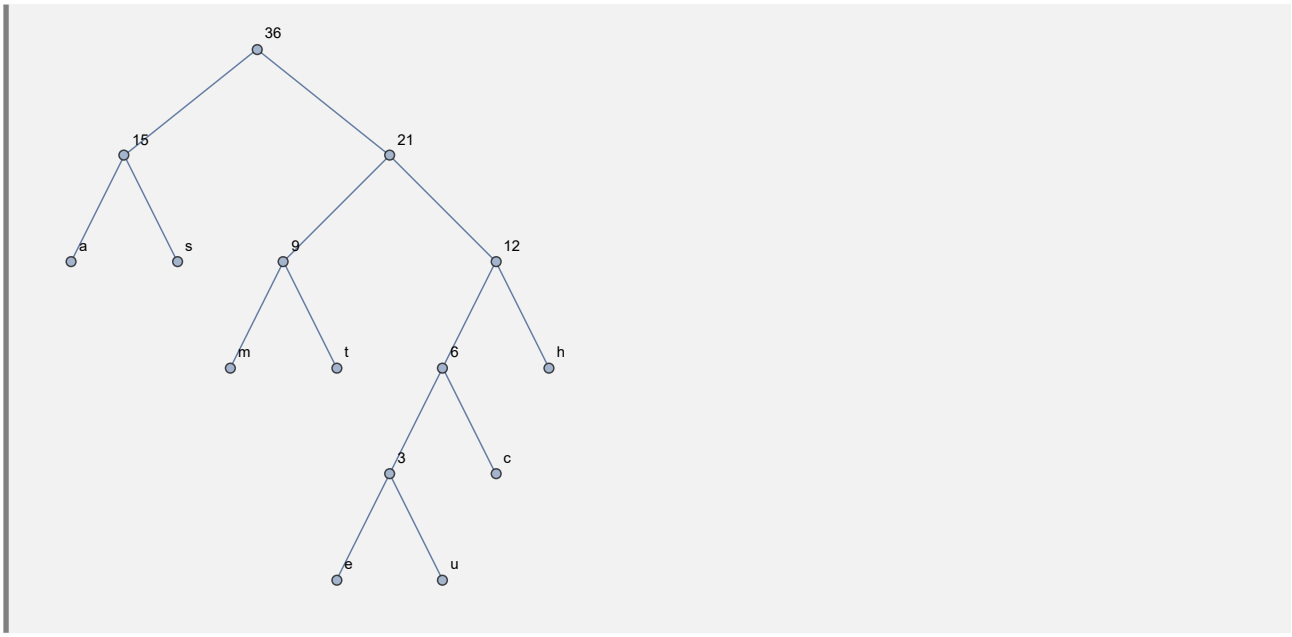
Solución:

In[ ] :=

```
ArbolHuffman["massachhusssettsss", frecuencias -> True]
```

Out[ ] :=

{{e, 1}, {u, 2}, {c, 3}, {m, 4}, {t, 5}, {h, 6}, {a, 7}, {s, 8}}



Explicación en video

25) **ArbolExpAlgebraica**: **construye** un **árbol** para una **expresión algebraica** "ExpAlg" recibida como parámetro. Proporciona al usuario la **opción** "**`grafico -> True`**" que muestra el árbol como un **gráfico** y no como un grafo.

Sintaxis: **`ArbolExpAlgebraica[ExpAlg]`**, o bien, **`ArbolExpAlgebraica[ExpAlg, grafico -> True]`**.

Ejemplo 8.49

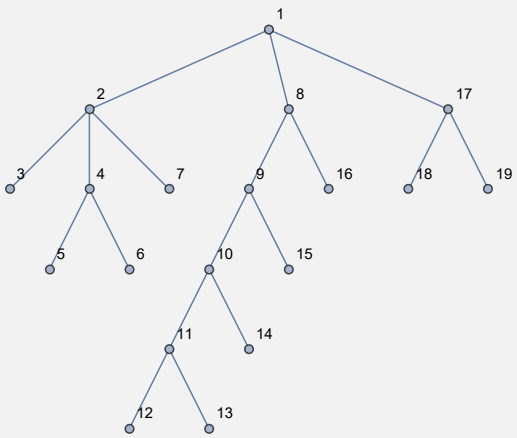
Dada la expresión algebraica: $(((a+b)c+d)e)-((a+b)c+f)$, devuelva un árbol que la represente.

Solución:

Empleando el comando **`ArbolExpAlgebraica`**, se obtiene:

```
In[ ] :=
ArbolExpAlgebraica[(((a + b)*c + d)*e) - ((a + b)*c + f)]

Out[ ] :=
Plus[Times[-1, Plus[a, b], c], Times[Plus[Times[Plus[a, b], c], d], e], Times[-1, f]]
{1 -> Plus, 2 -> Times, 3 -> -1, 4 -> Plus, 5 -> a, 6 -> b, 7 -> c, 8 -> Times, 9 -> Plus, 10 -> Times, 11 -> Plus,
12 -> a, 13 -> b, 14 -> c, 15 -> d, 16 -> e, 17 -> Times, 18 -> -1, 19 -> f}
```



N En el software Plus y Times representan los operadores $+$ y $\cdot$, respectivamente.

Ejemplo 8.50

Construya un árbol gráfico que contenga la expresión algebraica: $(((a+b)c+f)e)-d((a+b)-f+\frac{d}{e})$.

Solución:

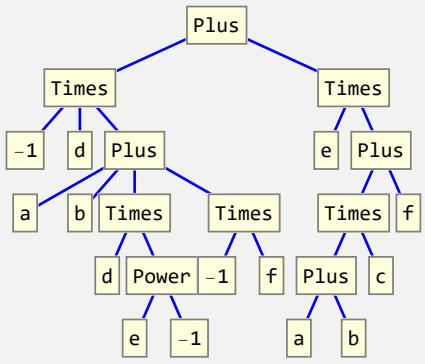
Al usar la opción "**grafico -> True**":

In[ ] :=

```
ArbolExpAlgebraica[(((a + b)*c + f)*e) - d*((a + b) - f + d/e),
grafico -> True]
```

Out[ ] :=

```
Plus[Times[-1, d, Plus[a, b, Times[d, Power[e, -1]], Times[-1, f]]], Times[e, Plus[Times[Plus[a, b], c], f]]]
```



Explicación en video

26) **ArbolBinarioBusqueda**: **construye** un **árbol binario de búsqueda** para almacenar un conjunto de datos **numéricos** o **caracteres** recibidos como parámetro, o bien, una **frase** contenida en un "string". Presenta la **opción** "**`grafico -> True`**" en caso de querer generar el árbol como un **gráfico**.

Sintaxis: **`ArbolBinarioBusqueda[Datos]`**, o bien, **`ArbolBinarioBusqueda[Datos, grafico -> True]`**.

Ejemplo 8.51

Considere la lista de datos: {20, 5, 3, 21, 25, 10, 9, 8, 4, 27, 19, 30}. Genere en *Mathematica* un árbol binario de búsqueda que la almacene.

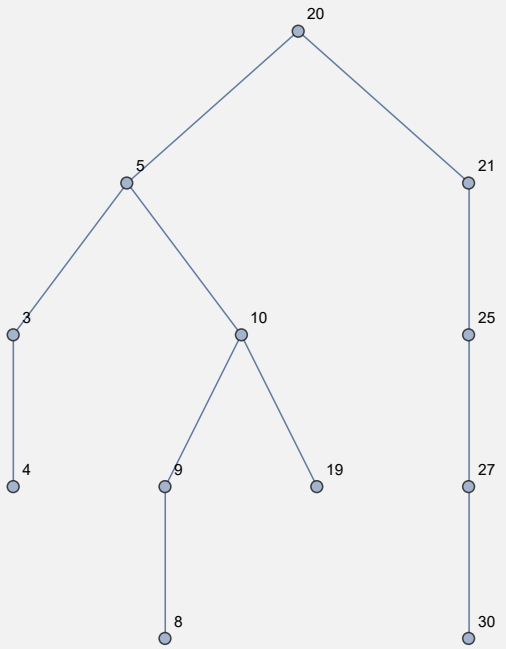
Solución:

En el software:

In[ ] :=

```
ArbolBinarioBusqueda[{20, 5, 3, 21, 25, 10, 9, 8, 4, 27, 19, 30}]
```

Out[ ] :=



Ejemplo 8.52

Utilizando un árbol gráfico binario de búsqueda, guarde los datos: {f, a, d, e, h, k, g, b, j, u}.

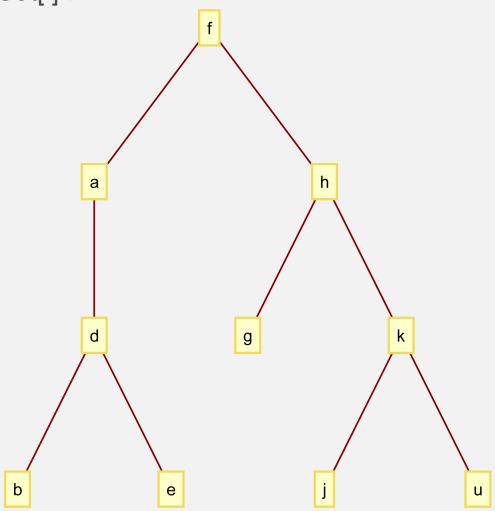
Solución:

El ejercicio se resuelve empleando la opción "**grafico->True**":

In[ ] :=

```
ArbolBinarioBusqueda[{f, a, d, e, h, k, g, b, j, u}, grafico->True]
```

Out[ ] :=



N Es importante recalcar, que la instrucción **ArbolBinarioBusqueda** es capaz de detectar si su argumento es una lista de números, letras o un "string". De acuerdo con ello, realiza posteriormente, la construcción del árbol binario.

Explicación en video

27) **Prefijo**: **recorre** un k-árbol "T" en **orden inicial** (**preorden/prefijo**) mostrando como salida una **lista ordenada** de los **nodos transitados** durante el proceso. "T" pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o bien, con el **paquete** "Combinatorica". Brinda la **opción** "**animacion->True**" que exhibe **paso a paso** el **recorrido prefijo**.

Sintaxis: **Prefijo[T, nodo]**, o, **Prefijo[T, nodo, animacion->True]**, con "nodo" el **vértice raíz** del árbol "T". **No acepta** árboles **gráficos ni dirigidos**.

Ejemplo 8.53

Desarrolle un recorrido preorden sobre un árbol con raíz 5 y aristas: {{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c, 5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}.

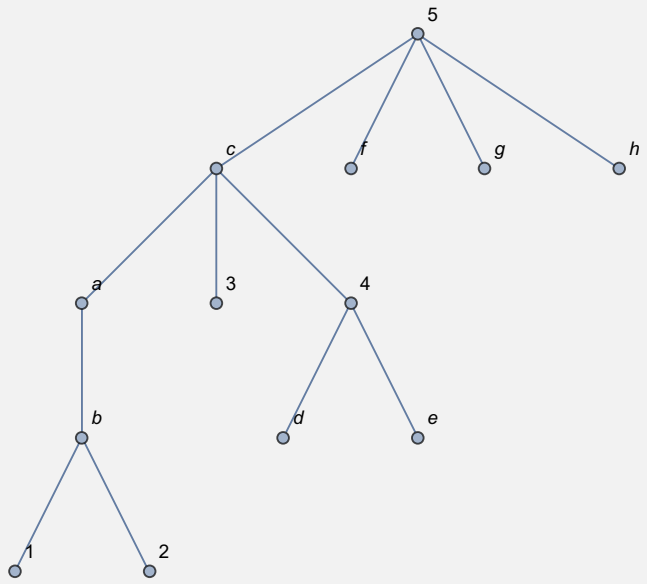
Solución:

En *Mathematica*:

In[ ] :=

```
arbol = ArbolR[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c,
5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}, 5]
Prefijo[arbol, 5]
```

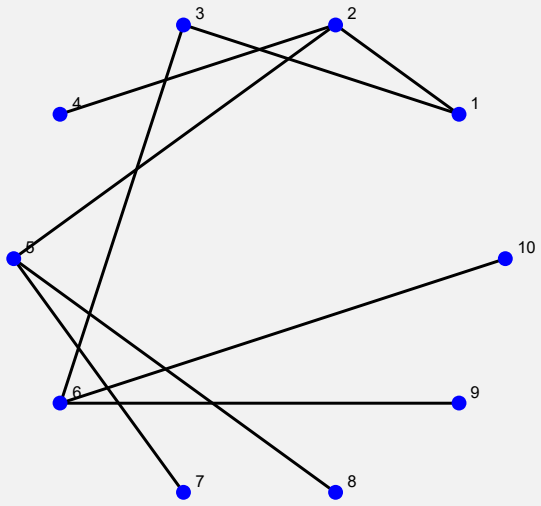
Out[ ] :=



{5, c, a, b, 1, 2, 3, 4, d, e, f, g, h}

Ejemplo 8.54

Mediante una animación, genere el recorrido de orden inicial en el árbol:



Supónga el vértice 6 como la raíz.

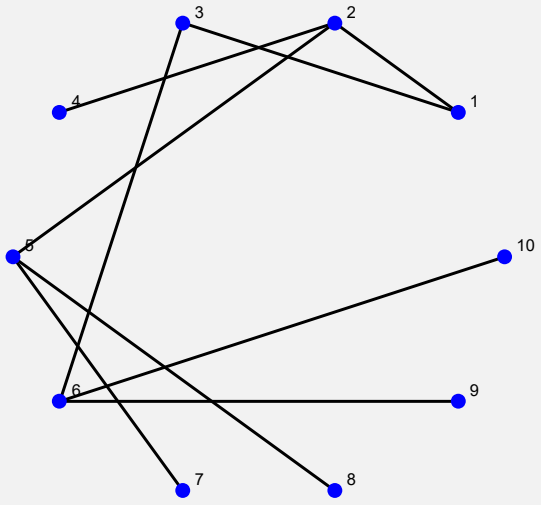
Solución:

Al usar el comando **Prefijo** y su opción "**animacion -> True**":

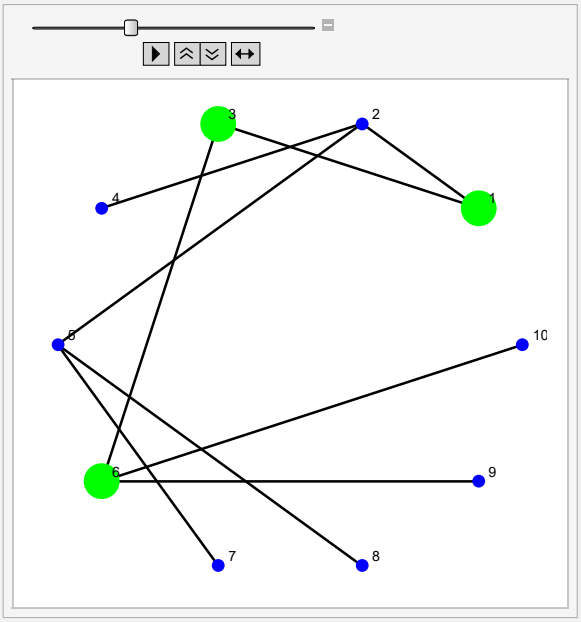
In[ ] :=

```
GrafoC[{{1, 2}, {1, 3}, {2, 4}, {2, 5}, {3, 6}, {5, 7}, {5, 8}, {6,
9}, {6, 10}}]
Prefijo[G, 6, animacion -> True]
```

Out[ ] :=



{6, 3, 1, 2, 4, 5, 7, 8, 9, 10}



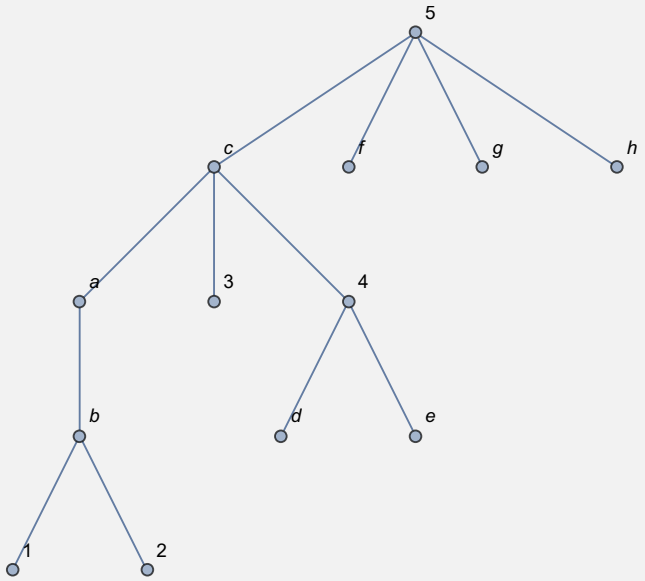
Explicación en video

28) **Postfijo**: **recorre** un k-árbol "T" en **orden final** (**postorden**/**postfijo**) mostrando como salida una **lista ordenada** de los **nodos transitados** durante el proceso. "T" pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o bien, con el **paquete** "Combinatorica". Brinda la **opción** "`animacion->True`" que exhibe **paso a paso** el **recorrido postfijo**.

Sintaxis: `Postfijo[T, nodo]`, o, `Postfijo[T, nodo, animacion->True]`, con "nodo" el **vértice raíz** del árbol "T". **No acepta** árboles **gráficos ni dirigidos**.

Ejemplo 8.55

Muestre una animación con el recorrido de orden final en:



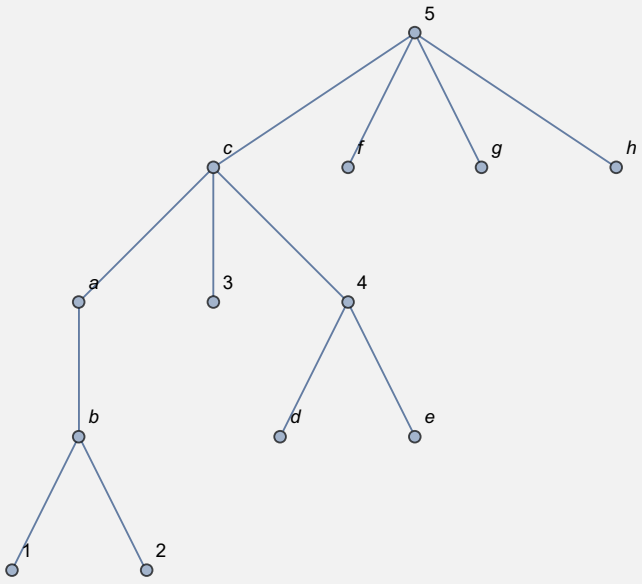
Solución:

Al invocar la instrucción **`Postfijo`**:

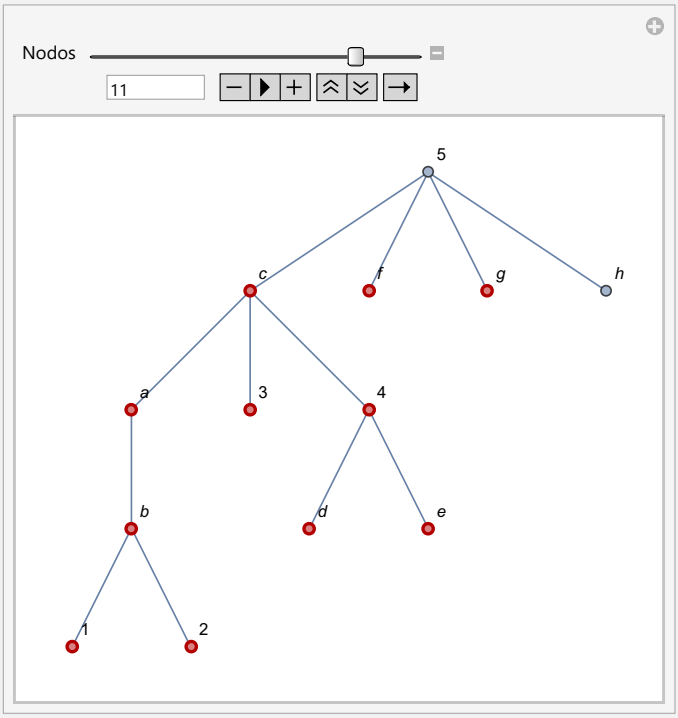
In[ ] :=

```
arbol = ArbolR[{{a, b}, {a, c}, {b, 1}, {b, 2}, {c, 3}, {c, 4}, {c,
5}, {4, d}, {4, e}, {5, f}, {5, g}, {5, h}}, 5]
Postfijo[arbol, 5, animacion -> True]
```

Out[ ] :=



{1, 2, b, a, 3, d, e, 4, c, f, g, h, 5}



Ejemplo 8.56

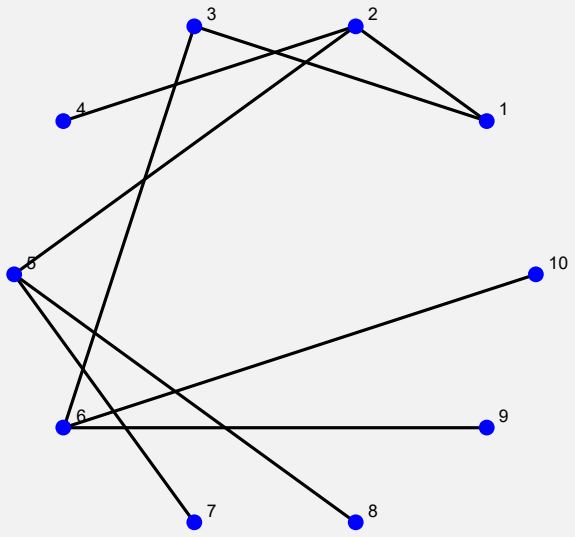
En el árbol con raíz 6 del ejemplo tras anterior, realice un recorrido postorden.

Solución:

In[ ] :=

```
GrafoC[{{1, 2}, {1, 3}, {2, 4}, {2, 5}, {3, 6}, {5, 7}, {5, 8}, {6,
9}, {6, 10}}]
Postfijo[G, 6]
```

Out[ ] :=



{4, 7, 8, 5, 2, 1, 3, 9, 10, 6}

Explicación en video

29) **Polaca**: **genera** la **notación polaca** de una **expresión algebraica** "ExpAlg" recibida como parámetro. El comando **muestra** el **árbol** que **representa** a "ExpAlg" y su **notación polaca**.

Sintaxis: `Polaca[ExpAlg]`.

Ejemplo 8.57

Devuelva la notación polaca de: $(((a+b)c+f)e)-d((a+b)-f+\frac{d}{e})$.

Solución:

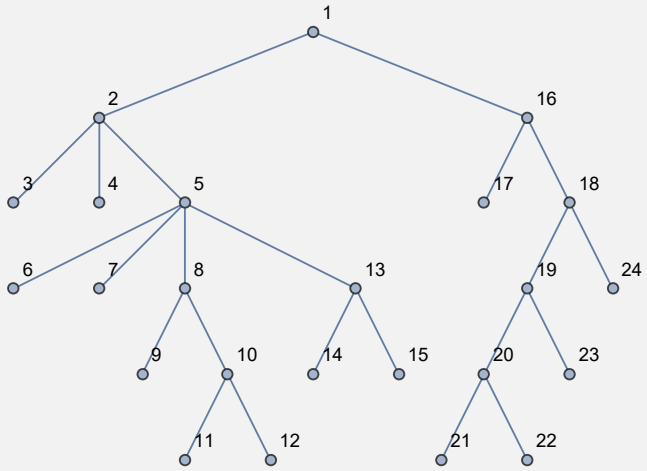
Al utilizar el comando **`Polaca`**, se tiene:

In[ ] :=

```
Polaca[(((a + b) c + f) e) - d*((a + b) - f + d/e)]
```

Out[ ] :=

```
Plus[Times[-1, d, Plus[a, b, Times[d, Power[e, -1]], Times[-1, f]]], Times[e, Plus[Times[Plus[a, b], c], f]]]
{1 -> Plus, 2 -> Times, 3 -> -1, 4 -> d, 5 -> Plus, 6 -> a, 7 -> b, 8 -> Times, 9 -> d, 10 -> Power, 11 -> e,
12 -> -1, 13 -> Times, 14 -> -1, 15 -> f, 16 -> Times, 17 -> e, 18 -> Plus, 19 -> Times, 20 -> Plus, 21 -> a,
22 -> b, 23 -> c, 24 -> f}
```




```
+*-1d+ab*d^e-1*-1f*e+*+abcf
```

N El operador * dentro de la instrucción **`Polaca`** se puede omitir y *Mathematica* de igual forma interpreta correctamente la expresión. En el *Out* [], Power simboliza una potencia.

Ejemplo 8.58

Construya a través del uso de software la notación polaca de: $a\left(b-\frac{c}{d}+e\right)-g\left(f\cdot\frac{a+b}{c+de}\right)+ab$.

Solución:

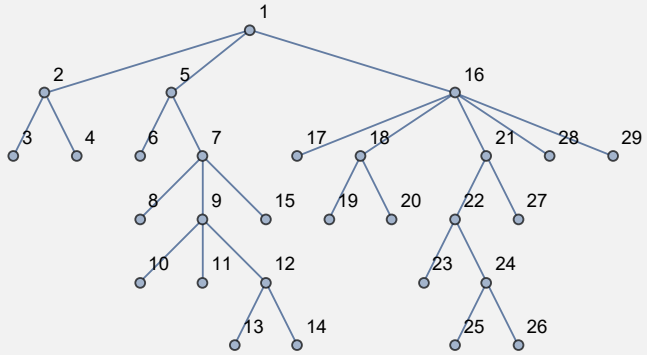
In[ ] :=

```
Polaca[a*(b - c/d + e) - g*(f*(a + b)/(c + d*e)) + a*b]
```

Out[ ] :=

Plus[Times[a, b], Times[a, Plus[b, Times[-1, c, Power[d, -1]], e]], Times[-1, Plus[a, b], Power[Plus[c, Times[d, e]], -1], f, g]]

{1 -> Plus, 2 -> Times, 3 -> a, 4 -> b, 5 -> Times, 6 -> a, 7 -> Plus, 8 -> b, 9 -> Times, 10 -> -1, 11 -> c, 12 -> Power, 13 -> d, 14 -> -1, 15 -> e, 16 -> Times, 17 -> -1, 18 -> Plus, 19 -> a, 20 -> b, 21 -> Power, 22 -> Plus, 23 -> c, 24 -> Times, 25 -> d, 26 -> e, 27 -> -1, 28 -> f, 29 -> g}



+*ab*a+b*-1c^d-1e*-1+ab^+c*de-1fg

Explicación en video

30) **PolacaInversa**: **genera** la **notación polaca inversa** de una **expresión algebraica** "ExpAlg" recibida como parámetro. La instrucción **muestra** el **árbol** que **representa** a "ExpAlg" y su **notación polaca inversa**.

Sintaxis: **`PolacaInversa[ExpAlg]`**.

Ejemplo 8.59

Retorne la notación polaca inversa de la expresión algebraica: $(((a+b)c+f)e)-d((a+b)-f+\frac{d}{e})$.

Solución:

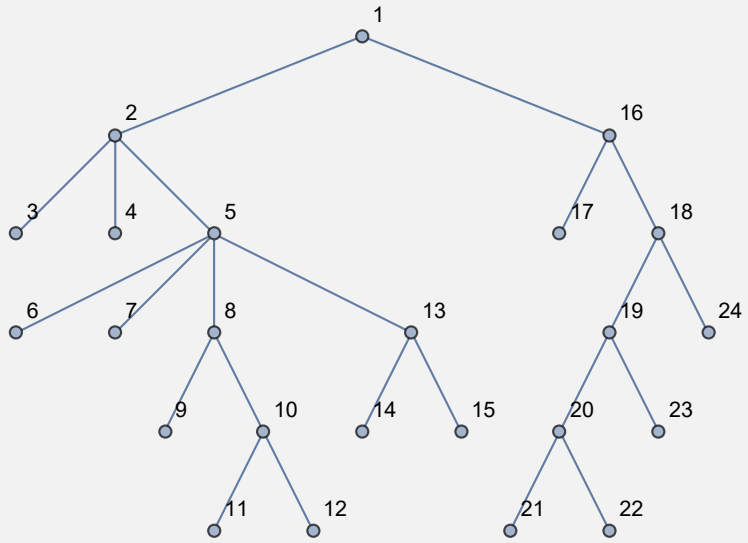
In[ ] :=

```
PolacaInversa[(((a + b)*c + f)*e) - d*((a + b) - f + d/e)]
```

Out[ ] :=

Plus[Times[-1, d, Plus[a, b, Times[d, Power[e, -1]], Times[-1, f]]], Times[e, Plus[Times[Plus[a, b], c], f]]]

{1 -> Plus, 2 -> Times, 3 -> -1, 4 -> d, 5 -> Plus, 6 -> a, 7 -> b, 8 -> Times, 9 -> d, 10 -> Power, 11 -> e, 12 -> -1, 13 -> Times, 14 -> -1, 15 -> f, 16 -> Times, 17 -> e, 18 -> Plus, 19 -> Times, 20 -> Plus, 21 -> a, 22 -> b, 23 -> c, 24 -> f}



-1dabde-1^*-1f*+*eab+c*f+*+

Ejemplo 8.60

Genere en *Mathematica* la notación polaca inversa de: $a\left(b-\frac{c}{d}+e\right)-g\left(f\cdot\frac{a+b}{c+de}\right)+ab$.

Solución:

En el software:

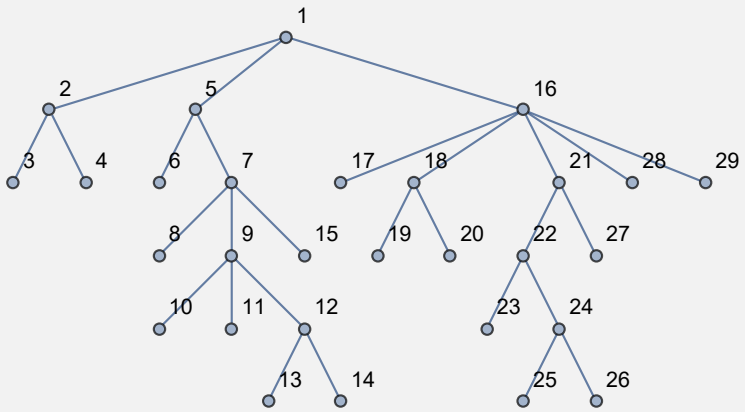
In[ ] :=

```
PolacaInversa[a*(b - c/d + e) - g*(f*(a + b)/(c + d*e)) + a*b]
```

Out[ ] :=

Plus[Times[a, b], Times[a, Plus[b, Times[-1, c, Power[d, -1]], e]], Times[-1, Plus[a, b], Power[Plus[c, Times[d, e]], -1], f, g]]

{1 -> Plus, 2 -> Times, 3 -> a, 4 -> b, 5 -> Times, 6 -> a, 7 -> Plus, 8 -> b, 9 -> Times, 10 -> -1, 11 -> c, 12 -> Power, 13 -> d, 14 -> -1, 15 -> e, 16 -> Times, 17 -> -1, 18 -> Plus, 19 -> a, 20 -> b, 21 -> Power, 22 -> Plus, 23 -> c, 24 -> Times, 25 -> d, 26 -> e, 27 -> -1, 28 -> f, 29 -> g}



ab*ab-1cd-1^*e+*-1ab+cde*+-1^fg*+

Explicación en video

31) **ArbolesGeneradores**: **función** que recibe como parámetro un **grafo conexo no dirigido** "G" y **retorna** un **árbol generador** (o de **expansión**) por **cada uno** de sus **nodos** (cada vértice se **toma** como **raíz**). El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o bien, mediante el uso del **paquete** "Combinatorica".

Sintaxis: `ArbolesGeneradores[G]`.

Ejemplo 8.61

Halle veinte árboles generadores, con raíz en cada uno de los nodos del grafo dodecaedro.

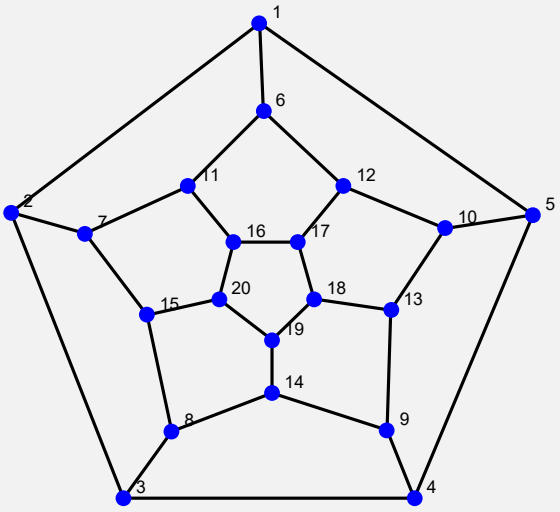
Solución:

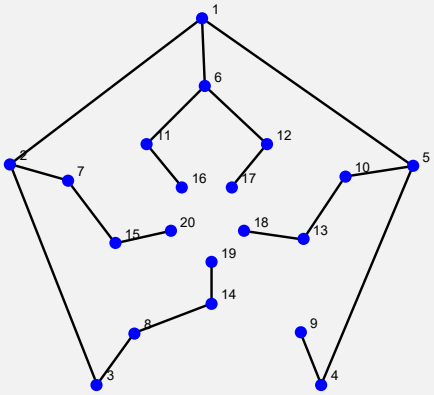
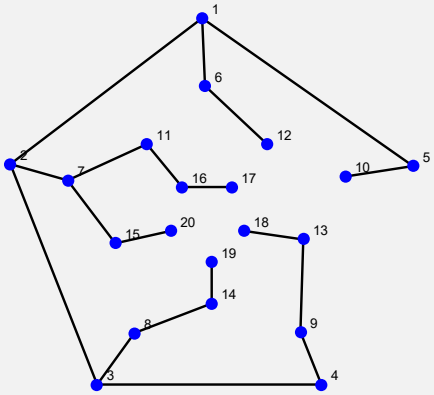
En *Mathematica* el grafo dodecaedro se puede obtener en el "Wolfram System", o bien, utilizando el paquete "Combinatorica". Ambas alternativas son factibles en la resolución del ejemplo. En este caso, se creará el grafo en "Combinatorica":

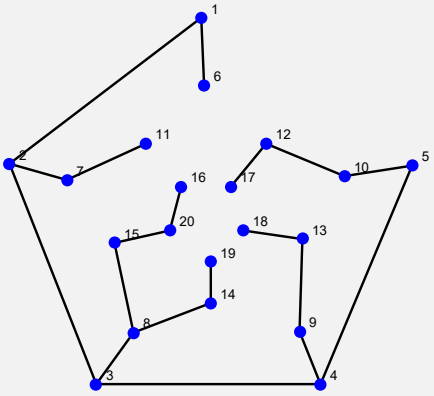
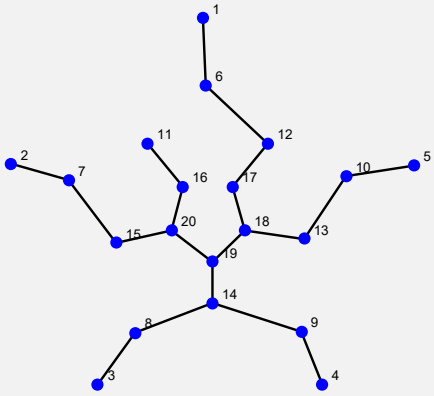
In[ ] :=

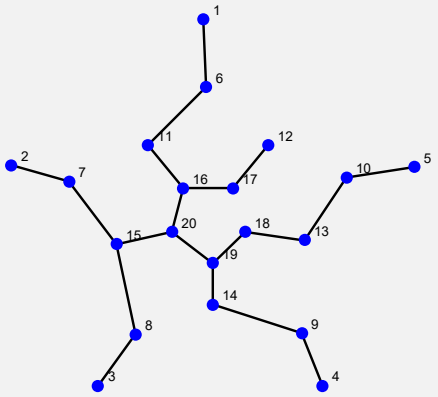
```
Quiet[<<Combinatorica`]
ShowGraph[grafo = SetGraphOptions[DodecahedralGraph,
VertexColor->Blue, EdgeColor->Black], VertexLabel->True,
PlotRange->0.1]
ArbolesGeneradores[grafo]
```

Out[ ] :=


1
6
11
12
2
7
16
17
10
5
15
20
18
13
19
14
8
9
3
4

{ 
1
6
11
12
2
7
16
17
10
5
15
20
18
13
19
14
8
9
3
4
, 
1
6
11
12
2
7
16
17
10
5
15
20
18
13
19
14
8
9
3
4
,


1
6
11
12
2
7
16
17
10
5
15
20
18
13
19
14
8
9
3
4
, ..., 
1
6
11
12
2
7
16
17
10
5
15
20
18
13
19
14
8
9
3
4
,



}

No se muestra el *Out* [ ] completo por su tamaño.

Ejemplo 8.62

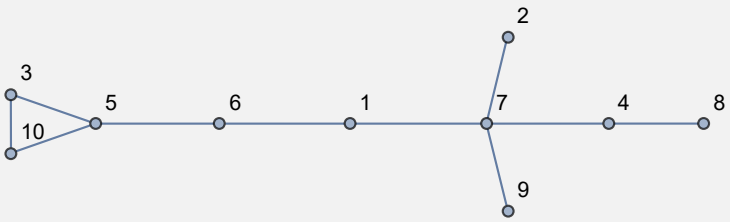
Sobre un grafo seudoaleatorio conexo de orden 10×10, corra la instrucción **`ArbolesGeneradores`**.

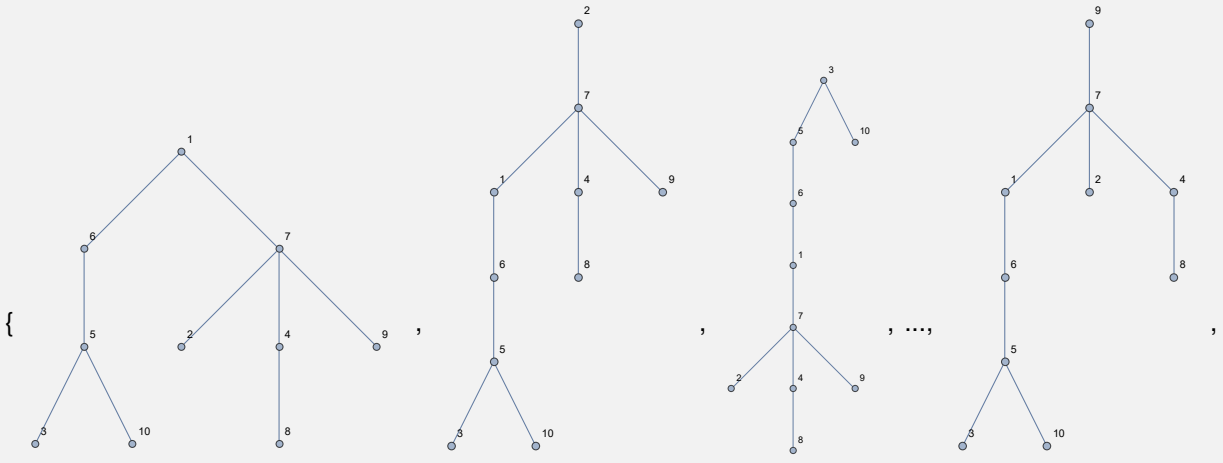
Solución:

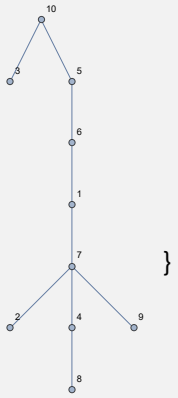
In[ ] :=

```
grafo = GrafoRandomConexo[10, 10]
ArbolesGeneradores[grafo]
```

Out[ ] :=







}

En este ejercicio, en total se construyen de manera automática diez árboles de expansión (uno por cada vértice).

Explicación en video

32) **BuscarPrimeroAncho**: aplica el **algoritmo buscar primero a lo ancho** sobre un **grafo** "G", mostrando como salida una **lista ordenada** de **aristas transitadas** durante el proceso. "G" pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o bien, con el **paquete** "Combinatorica". Brinda las opciones "**`animacion->True`**" que exhibe **paso a paso** el recorrido a lo **ancho** y "**`orden->Lista`**" que **define** un **orden** para los **vértices** en la **aplicación** de la **búsqueda**. Por defecto el orden tomado por el software va **de acuerdo** con el **orden de las aristas** al crear el grafo y puede ser devuelto mediante el uso de la instrucción "**`VertexList[G]`**".

Sintaxis: **`BuscarPrimeroAncho[G]`**, o, **`BuscarPrimeroAncho[G, animacion->True]`**, o,

```
BuscarPrimeroAncho[G, animacion->True, orden->Lista]
```

No acepta grafos **dirigidos**.

Ejemplo 8.63

En un grafo seudoaleatorio conexo no simple de orden 20×50, aplique el algoritmo *buscar primero a lo ancho*.

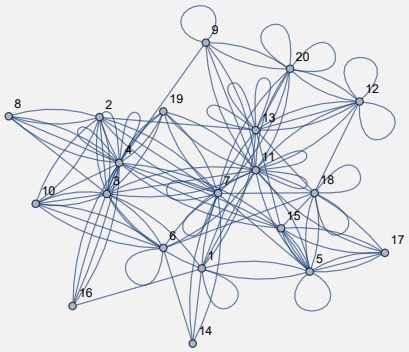
Solución:

En el software:

In[ ] :=

```
grafo = GrafoRandomConexo[20, 50, simple->False]
BuscarPrimeroAncho[grafo]
```

Out[ ] :=



{{1, 3}, {1, 5}, {1, 11}, {1, 13}, {1, 14}, {1, 16}, {2, 3}, {3, 4}, {3, 6}, {3, 8}, {3, 10}, {3, 19}, {5, 7}, {5, 15}, {5, 17}, {5, 18}, {9, 11}, {11, 12}, {11, 20}}

N El recorrido a lo ancho se muestra mediante un conjunto ordenado de aristas. Cabe resaltar que el comando **BuscarPrimeroAncho** por defecto, ha escogido un orden ascendente para los vértices del grafo.

Ejemplo 8.64

Construya en el ambiente facilitado por el paquete "Combinatorica", un grafo con lados: {{1, 2}, {1, 5}, {1, 6}, {2, 3}, {2, 7}, {3, 4}, {3, 8}, {4, 5}, {4, 9}, {5, 10}, {6, 11}, {6, 12}, {7, 11}, {7, 15}, {8, 14}, {8, 15}, {9, 13}, {9, 14}, {10, 12}, {10, 13}, {11, 16}, {12, 17}, {13, 18}, {14, 19}, {15, 20}, {16, 17}, {16, 20}, {17, 18}, {18, 19}, {19, 20}}. Realice una *búsqueda a lo ancho* considerando este grafo y despliegue el resultado con una animación.

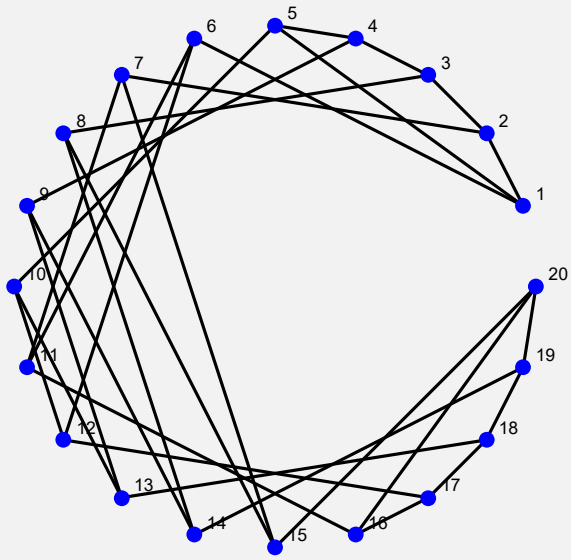
Solución:

El grafo se creará mediante la instrucción **GrafoC**:

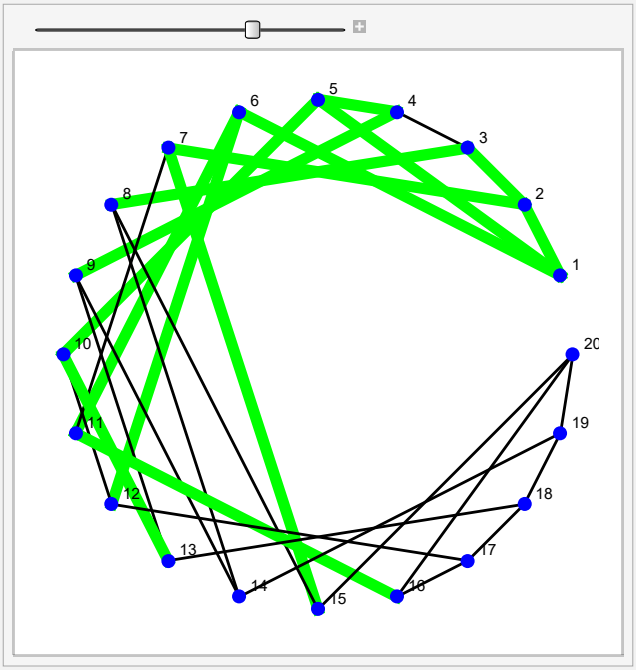
In[ ] :=

```
aristas = {{1, 2}, {1, 5}, {1, 6}, {2, 3}, {2, 7}, {3, 4}, {3, 8},
{4, 5}, {4, 9}, {5, 10}, {6, 11}, {6, 12}, {7, 11}, {7, 15}, {8, 14},
{8, 15}, {9, 13}, {9, 14}, {10, 12}, {10, 13}, {11, 16}, {12, 17},
{13, 18}, {14, 19}, {15, 20}, {16, 17}, {16, 20}, {17, 18}, {18, 19},
{19, 20}};
GrafoC[aristas]
BuscarPrimeroAncho[G, animacion -> True]
```

Out[ ] :=



{{1, 2}, {1, 5}, {1, 6}, {2, 3}, {2, 7}, {5, 4}, {5, 10}, {6, 11}, {6, 12}, {3, 8}, {7, 15}, {4, 9}, {10, 13}, {11, 16}, {12, 17}, {8, 14}, {15, 20}, {13, 18}, {14, 19}}



N El manipulador permite visualizar el recorrido a lo ancho arista por arista.

Explicación en video

33) **BuscarPrimeroLargo**: aplica el **algoritmo buscar primero a lo largo** sobre un **grafo** "G", mostrando como salida una **lista ordenada** de **aristas transitadas** durante el proceso. "G" pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o bien, con el **paquete** "Combinatorica". Brinda las opciones "**`animacion->True`**" que exhibe **paso a paso** el recorrido a lo **largo** y "**`orden->Lista`**" que **define** un **orden** para los **vértices** en la **aplicación** de la **búsqueda**. Por defecto el orden tomado por el software va **de acuerdo** con el **orden de las aristas** al crear el grafo y puede ser devuelto mediante el uso de la instrucción "**`VertexList[G]`**".

Sintaxis: **`BuscarPrimeroLargo[G]`**, o, **`BuscarPrimeroLargo[G, animacion->True]`**, o,

```
BuscarPrimeroLargo[G, animacion->True, orden->Lista]
```

No acepta grafos **dirigidos**.

Ejemplo 8.65

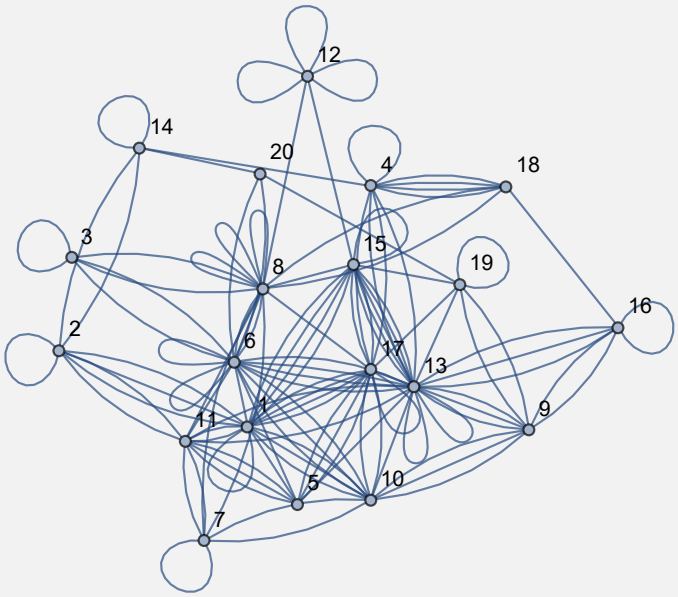
Genere un grafo seudoaleatorio conexo no simple de orden 20×50, aplique sobre él, el algoritmo *buscar primero a lo largo*.

Solución:

In[ ] :=

```
grafo = GrafoRandomConexo[20, 50, simple->False]
BuscarPrimeroLargo[grafo]
```

Out[ ] :=



{{1, 2}, {2, 11}, {11, 5}, {5, 6}, {6, 3}, {3, 8}, {8, 7}, {7, 10}, {10, 9}, {9, 13}, {13, 4}, {4, 14}, {14, 20}, {20, 19}, {19, 15}, {15, 12}, {15, 17}, {4, 18}, {18, 16}}

Ejemplo 8.66

En "Combinatorica", sea un grafo con lados: {{1, 2}, {1, 5}, {1, 6}, {2, 3}, {2, 7}, {3, 4}, {3, 8}, {4, 5}, {4, 9}, {5, 10}, {6, 11}, {6, 12}, {7, 11}, {7, 15}, {8, 14}, {8, 15}, {9, 13}, {9, 14}, {10, 12}, {10, 13}, {11, 16}, {12, 17}, {13, 18}, {14, 19}, {15, 20}, {16, 17}, {16, 20}, {17, 18}, {18, 19}, {19, 20}}. Mediante una animación, muestre el recorrido *buscar primero a lo largo*, tomando el orden en sus nodos: {20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1}.

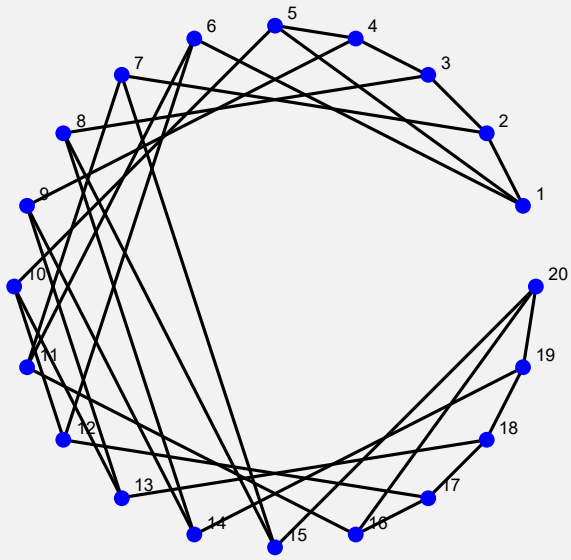
Solución:

En *Mathematica*:

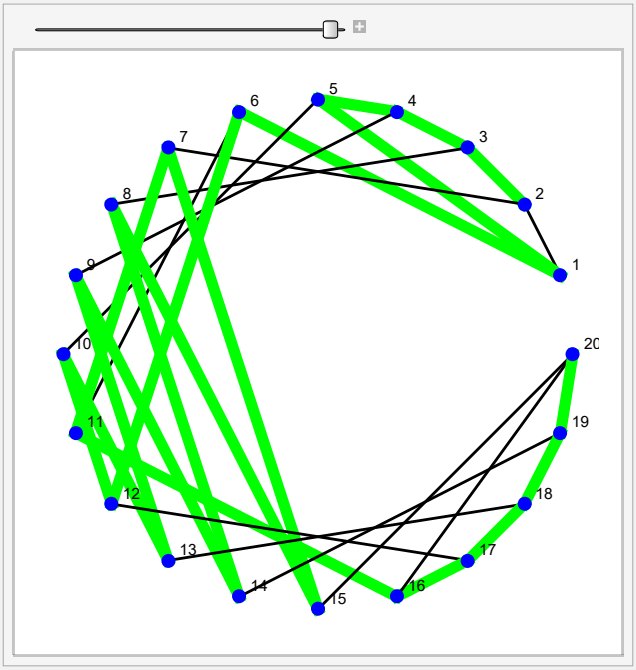
In[ ] :=

```
aristas = {{1, 2}, {1, 5}, {1, 6}, {2, 3}, {2, 7}, {3, 4}, {3, 8},
{4, 5}, {4, 9}, {5, 10}, {6, 11}, {6, 12}, {7, 11}, {7, 15}, {8, 14},
{8, 15}, {9, 13}, {9, 14}, {10, 12}, {10, 13}, {11, 16}, {12, 17},
{13, 18}, {14, 19}, {15, 20}, {16, 17}, {16, 20}, {17, 18}, {18, 19},
{19, 20}};
GrafoC[aristas]
ordennodos = Reverse[ListaVertices[G]]
BuscarPrimeroLargo[G, animacion -> True, orden -> ordennodos]
```

Out[ ] :=



{20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1}
{{20, 19}, {19, 18}, {18, 17}, {17, 16}, {16, 11}, {11, 7}, {7, 15}, {15, 8}, {8, 14}, {14, 9}, {9, 13}, {13, 10}, {10, 12}, {12, 6}, {6, 1}, {1, 5}, {5, 4}, {4, 3}, {3, 2}}



N El comando **Reverse** ha dado vuelta a la lista de vértices retornada por **ListaVertices**.

Explicación en video

34) **BuscarDatoAncho**: realiza una **búsqueda a lo ancho** de un **dato** "a" sobre un **grafo** "G" creado o no con el **paquete** "Combinatorica". Presenta la **opción** "**`orden->Lista`**" que **especifica** un **orden** de los **nodos** para el recorrido. El comando muestra **paso a paso** los **vértices visitados** hasta **encontrar** el dato "a". Si el dato **no es un vértice** de "G" recorre **todos** los **nodos** y muestra el **mensaje** "Dato no encontrado".

Sintaxis: **`BuscarDatoAncho[G, a]`**, o bien, **`BuscarDatoAncho[G, a, orden->Lista]`**. **No acepta** grafos **dirigidos**.

Ejemplo 8.67

Realice una *búsqueda a lo ancho* del dato 15 sobre **`GrafoRandomConexo[20, 50]`**.

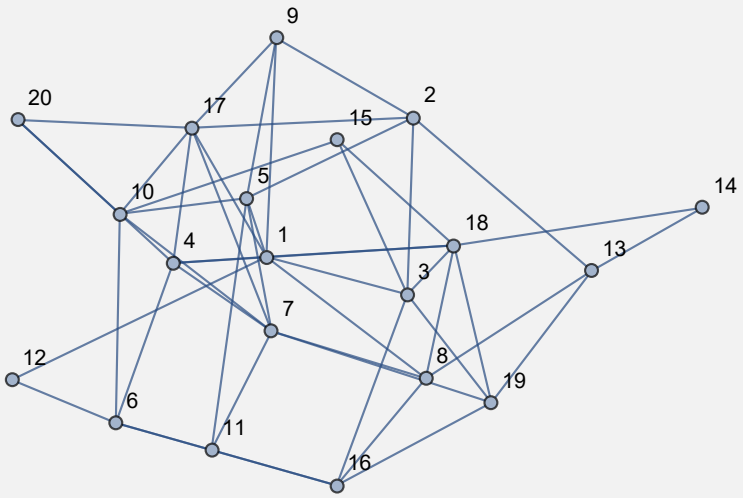
Solución:

Al emplear **`BuscarDatoAncho`**, se tiene:

In[ ] :=

```
grafo = GrafoRandomConexo[20, 50]
BuscarDatoAncho[grafo, 15]
```

Out[ ] :=



Se visitó: 1
Se visitó: 3
Se visitó: 4
Se visitó: 5
Se visitó: 8
Se visitó: 9
Se visitó: 12

Se visitó: 17
Se visitó: 18
Se visitó: 2
Se visitó: 15. Dato encontrado
$Aborted

N El alumno debe notar, que a pesar de formarse un grafo distinto cada vez que se ejecuta el *In* [ ], siempre se encuentra el dato 15, pues el grafo poseerá nodos consecutivos naturales del 1 al 20.

Ejemplo 8.68

Sea un grafo seudoaleatorio conexo no simple de orden 20×50. *Busque a lo ancho* sobre este grafo el dato 60 utilizando el orden: {20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1}.

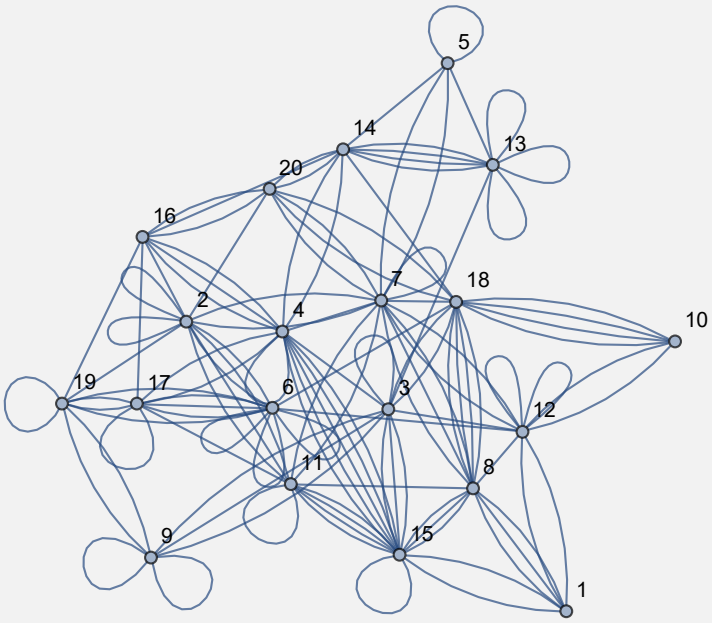
Solución:

En el software:

In[ ] :=

```
grafo = GrafoRandomConexo[20, 50, simple->False]
BuscarDatoAncho[grafo, 60, orden->Reverse[VertexList[grafo]]]
```

Out[ ] :=



Se visitó: 20
Se visitó: 18
Se visitó: 16
Se visitó: 14

```
Se visitó: 7
Se visitó: 2
Se visitó: 10
Se visitó: 8
Se visitó: 6
Se visitó: 3
Se visitó: 19
Se visitó: 17
Se visitó: 4
Se visitó: 13
Se visitó: 5
Se visitó: 12
Se visitó: 11
Se visitó: 15
Se visitó: 1
Se visitó: 9
Dato no encontrado
```

N Como era de esperarse, 60 no fue hallado en el grafo pues contiene nodos naturales consecutivos hasta el número 20. Por otra parte, no se muestra el $Aborted como en el ejemplo anterior, ya que en este ejercicio no se abortó el recorrido, al comparar el dato con todos los vértices del grafo.

Explicación en video

35) **BuscarDatoLargo**: realiza una **búsqueda a lo largo** de un **dato** "a" sobre un **grafo** "G" creado o no con el **paquete** "Combinatorica". Presenta la **opción** "**`orden->Lista`**" que **especifica** un **orden** de los **nodos** para el recorrido. El comando muestra **paso a paso** los **vértices visitados** hasta **encontrar** el dato "a". Si el dato **no es un vértice** de "G" recorre **todos** los **nodos** y muestra el **mensaje** "Dato no encontrado".

Sintaxis: **`BuscarDatoLargo[G, a]`**, o bien, **`BuscarDatoLargo[G, a, orden->Lista]`**. **No acepta** grafos **dirigidos**.

Ejemplo 8.69

A través de un *recorrido a lo largo*, busque el dato 15 en un grafo devuelto por **`GrafoRandomConexo[20, 50]`**.

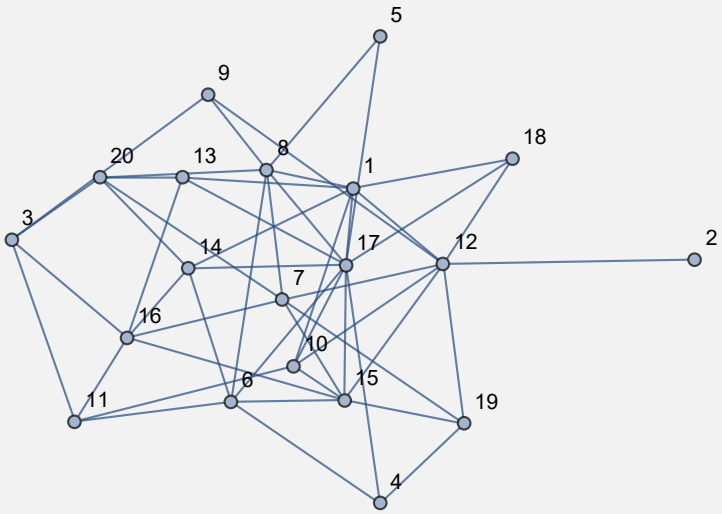
Solución:

En *Mathematica*:

In[ ] :=

```
grafo = GrafoRandomConexo[20, 50]
BuscarDatoLargo[grafo, 15]
```

Out[ ] :=



Se visitó: 1
Se visitó: 8
Se visitó: 5
Se visitó: 17
Se visitó: 4
Se visitó: 6
Se visitó: 11
Se visitó: 3
Se visitó: 9
Se visitó: 12
Se visitó: 2
Se visitó: 7
Se visitó: 15. Dato encontrado
$Aborted

Ejemplo 8.70

Considere el grafo de "Combinatorica" con aristas: {{1, 2}, {1, 5}, {1, 6}, {2, 3}, {2, 7}, {3, 4}, {3, 8}, {4, 5}, {4, 9}, {5, 10}, {6, 11}, {6, 12}, {7, 11}, {7, 15}, {8, 14}, {8, 15}, {9, 13}, {9, 14}, {10, 12}, {10, 13}, {11, 16}, {12, 17}, {13, 18}, {14, 19}, {15, 20}, {16, 17}, {16, 20}, {17, 18}, {18, 19}, {19, 20}}. *Busque a lo largo* el dato 60, tomando un orden descendente para el conjunto de vértices.

Solución:

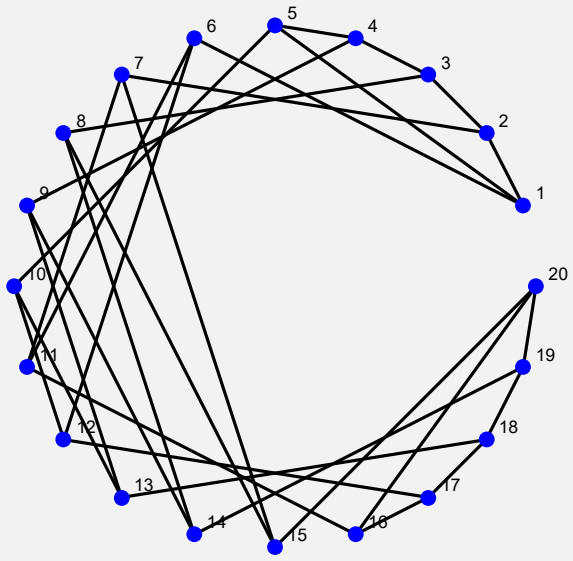
En el software se utiliza la opción "**orden -> Lista**" de **BuscarDatoLargo**:

In[ ] :=

```
aristas = {{1, 2}, {1, 5}, {1, 6}, {2, 3}, {2, 7}, {3, 4}, {3, 8},
```

```
{4, 5}, {4, 9}, {5, 10}, {6, 11}, {6, 12}, {7, 11}, {7, 15}, {8, 14},
{8, 15}, {9, 13}, {9, 14}, {10, 12}, {10, 13}, {11, 16}, {12, 17},
{13, 18}, {14, 19}, {15, 20}, {16, 17}, {16, 20}, {17, 18}, {18, 19},
{19, 20}};
GrafoC[aristas]
BuscarDatoLargo[G, 60, orden->Reverse[Table[i, {i, 20}]]]
```

Out[ ] :=



Se visitó: 20
Se visitó: 19
Se visitó: 18
Se visitó: 17
Se visitó: 16
Se visitó: 11
Se visitó: 7
Se visitó: 15
Se visitó: 8
Se visitó: 14
Se visitó: 9
Se visitó: 13
Se visitó: 10
Se visitó: 12
Se visitó: 6
Se visitó: 1
Se visitó: 5
Se visitó: 4
Se visitó: 3
Se visitó: 2
Dato no encontrado

Explicación en video

36) **Prim**: muestra **paso a paso** la ejecución del **algoritmo de Prim** sobre un **grafo** "G" **no dirigido**, **conexo** y **ponderado**, generando adicionalmente una **animación arista por arista** del árbol de **expansión final**. La instrucción **acepta únicamente** grafos construidos en el "Wolfram System" de *Mathematica*. Proporciona al usuario la **opción** "`orden -> Lista`" que **define** un **orden** para los **vértices** de acuerdo con el cual se **ejecutará el proceso**. Por defecto, **usa** el orden **devuelto** en "`Sort[VertexList[G]]`". **No procesa** grafos con **aristas múltiples** y **ningún peso** puede ser **igual a cero**.

Sintaxis: `Prim[G]`.

Ejemplo 8.71

Utilice el algoritmo de *Prim* para hallar un árbol de expansión de peso mínimo, sobre un grafo con pesos seudoaleatorios reales de uno a diez, cuyos lados vienen dados por: {{a, b}, {a, d}, {a, f}, {b, d}, {b, e}, {b, c}, {c, e}, {c, g}, {d, e}, {d, g}, {d, f}, {e, g}, {f, g}}.

Solución:

Al recurrir a la instrucción **`Prim`**, se obtiene:

In[ ] :=

```
grafo = Grafo[{{a, b}, {a, d}, {a, f}, {b, d}, {b, e}, {b,
c}, {c, e}, {c, g}, {d, e}, {d, g}, {d, f}, {e, g}, {f, g}},
pesos -> RandomReal[{1, 10}, 13], mostrarpesos -> True]
Prim[grafo]
```

Out[ ] :=

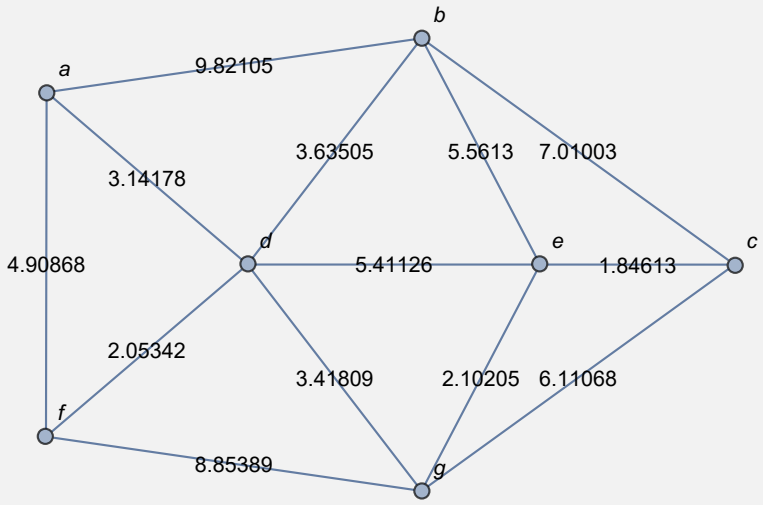


Tabla de pesos del grafo:

	a	b	c	d	e	f	g
a	0	9.82105	0	3.14178	0	4.90868	0
b	9.82105	0	7.01003	3.63505	5.5613	0	0
c	0	7.01003	0	0	1.84613	0	6.11068
d	3.14178	3.63505	0	0	5.41126	2.05342	3.41809
e	0	5.5613	1.84613	5.41126	0	0	2.10205
f	4.90868	0	0	2.05342	0	0	8.85389
g	0	0	6.11068	3.41809	2.10205	8.85389	0

El orden de los nodos es: {a, b, c, d, e, f, g}

*** Inicialización ***

Aristas seleccionables (AS) del árbol T a construir: {{{a, b}, 9.82105}, {{a, d}, 3.14178}, {{a, f}, 4.90868}}

*** Iteración 1 ***

Se agregó la arista: {a,d}, T={{a, d}}

Peso mínimo acumulado: 3.14178

AS={{{a, b}, 9.82105}, {{a, f}, 4.90868}, {{d, b}, 3.63505}, {{d, e}, 5.41126}, {{d, f}, 2.05342}, {{d, g}, 3.41809}}

*** Iteración 2 ***

Se agregó la arista: {d, f}, T={{a, d}, {d, f}}

Peso mínimo acumulado: 5.1952

AS={{{a, b}, 9.82105}, {{a, f}, 4.90868}, {{d, b}, 3.63505}, {{d, e}, 5.41126}, {{d, g}, 3.41809}, {{f, g}, 8.85389}}

*** Iteración 3 ***

Se agregó la arista: {d, g}, T={{a, d}, {d, f}, {d, g}}

Peso mínimo acumulado: 8.61329

AS={{{a, b}, 9.82105}, {{a, f}, 4.90868}, {{d, b}, 3.63505}, {{d, e}, 5.41126}, {{f, g}, 8.85389}, {{g, c}, 6.11068}, {{g, e}, 2.10205}}

*** Iteración 4 ***

Se agregó la arista: {g, e}, T={{a, d}, {d, f}, {d, g}, {g, e}}

Peso mínimo acumulado: 10.7153

AS={{{a, b}, 9.82105}, {{a, f}, 4.90868}, {{d, b}, 3.63505}, {{d, e}, 5.41126}, {{f, g}, 8.85389}, {{g, c}, 6.11068}, {{e, b}, 5.5613}, {{e, c}, 1.84613}}

*** Iteración 5 ***

Se agregó la arista: {e, c}, T={{a, d}, {d, f}, {d, g}, {g, e}, {e, c}}

Peso mínimo acumulado: 12.5615

AS={{{a, b}, 9.82105}, {{a, f}, 4.90868}, {{d, b}, 3.63505}, {{d, e}, 5.41126}, {{f, g}, 8.85389}, {{g, c}, 6.11068}, {{e, b}, 5.5613}, {{c, b}, 7.01003}}

*** Iteración 6 ***

Se agregó la arista: {d, b}, T={{a, d}, {d, f}, {d, g}, {g, e}, {e, c}, {d, b}}

Peso mínimo acumulado: 16.1965

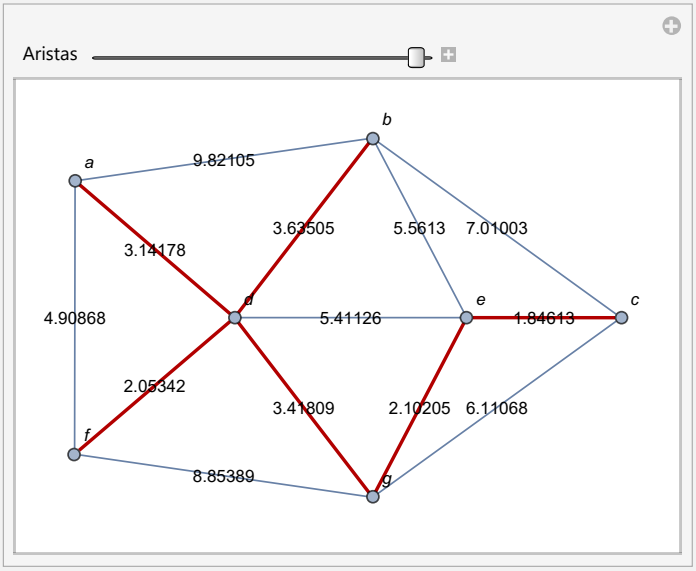
AS={{{a, b}, 9.82105}, {{a, f}, 4.90868}, {{d, e}, 5.41126}, {{f, g}, 8.85389}, {{g, c}, 6.11068}, {{e, b}, 5.5613}, {{c, b}, 7.01003}}

*** Finalmente ***

T={{a, d}, {d, f}, {d, g}, {g, e}, {e, c}, {d, b}} y es de peso: 16.1965

Animación del árbol generador T:

{{a, d}, {d, f}, {d, g}, {g, e}, {e, c}, {d, b}}



N Como no se especificó ningún orden en el conjunto de vértices, se debe recordar que **`Prim`** internamente usa **`Sort[VertexList[G]]`**, siendo "**`G`**" el grafo respectivo.

Ejemplo 8.72

Mediante el algoritmo de *Prim*, halle un árbol generador minimal sobre un grafo con pesos: {5, 6, 7, 5, 5, 4, 3, 3, 2, 2, 3, 1, 2}, correspondientes al conjunto de lados: {{a, b}, {a, d}, {a, f}, {b, c}, {b, d}, {b, e}, {c, e}, {c, g}, {d, e}, {d, f}, {d, g}, {e, g}, {f, g}}.

Solución:

In[ ] :=

```
grafo = Grafo[{{a, b}, {a, d}, {a, f}, {b, c}, {b, d}, {b, e}, {c,
e}, {c, g}, {d, e}, {d, f}, {d, g}, {e, g}, {f, g}}, pesos -> {5, 6,
7, 5, 5, 4, 3, 3, 2, 2, 3, 1, 2}, mostrarpesos -> True]
Prim[grafo]
```

Out[ ] :=

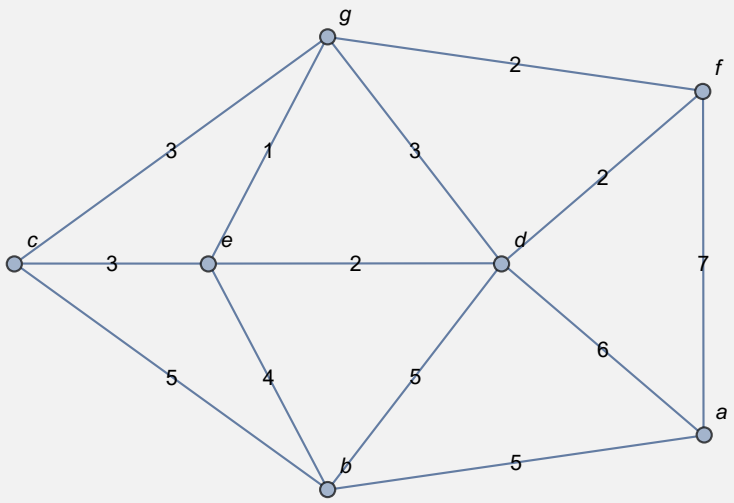


Tabla de pesos del grafo:

	a	b	c	d	e	f	g
a	0	5	0	6	0	7	0
b	5	0	5	5	4	0	0
c	0	5	0	0	3	0	3
d	6	5	0	0	2	2	3
e	0	4	3	2	0	0	1
f	7	0	0	2	0	0	2
g	0	0	3	3	1	2	0

El orden de los nodos es: {a, b, c, d, e, f, g}

*** Inicialización ***

Aristas seleccionables (AS) del árbol T a construir: {{{a, b}, 5}, {{a, d}, 6}, {{a, f}, 7}}

*** Iteración 1 ***

Se agregó la arista: {a, b}, T={{a, b}}

Peso mínimo acumulado: 5

AS={{{a, d}, 6}, {{a, f}, 7}, {{b, c}, 5}, {{b, d}, 5}, {{b, e}, 4}}

*** Iteración 2 ***

Se agregó la arista: {b, e}, T={{a, b}, {b, e}}

Peso mínimo acumulado: 9

AS={{{a, d}, 6}, {{a, f}, 7}, {{b, c}, 5}, {{b, d}, 5}, {{e, c}, 3}, {{e, d}, 2}, {{e, g}, 1}}

*** Iteración 3 ***

Se agregó la arista: {e, g}, T={{a, b}, {b, e}, {e, g}}

Peso mínimo acumulado: 10

AS={{{a, d}, 6}, {{a, f}, 7}, {{b, c}, 5}, {{b, d}, 5}, {{e, c}, 3}, {{e, d}, 2}, {{g, c}, 3}, {{g, d}, 3}, {{g, f}, 2}}

*** Iteración 4 ***

Se agregó la arista: {e, d}, T={{a, b}, {b, e}, {e, g}, {e, d}}

Peso mínimo acumulado: 12

AS={{{a, d}, 6}, {{a, f}, 7}, {{b, c}, 5}, {{b, d}, 5}, {{e, c}, 3}, {{g, c}, 3}, {{g, d}, 3}, {{g, f}, 2}, {{d, f}, 2}}

*** Iteración 5 ***

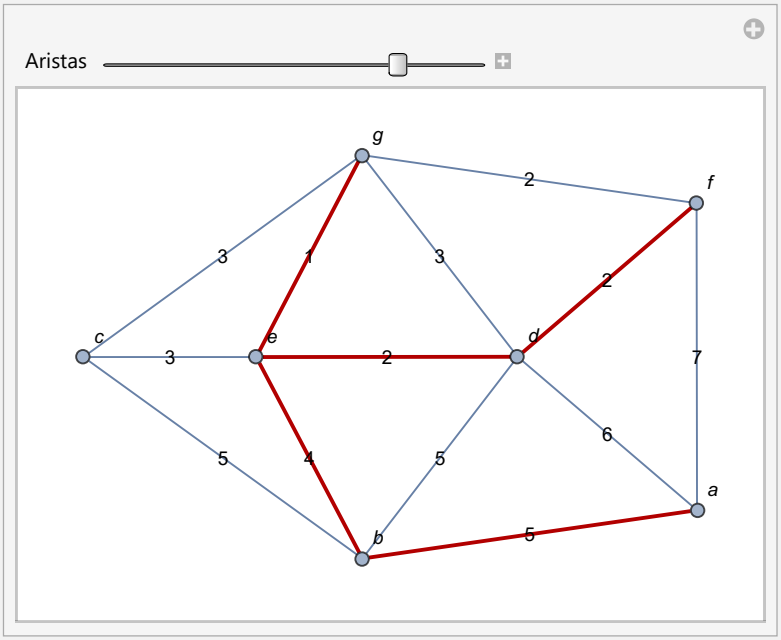
Se agregó la arista: {d, f}, T={{a, b}, {b, e}, {e, g}, {e, d}, {d, f}}

Peso mínimo acumulado: 14

AS={{{a, d}, 6}, {{a, f}, 7}, {{b, c}, 5}, {{b, d}, 5}, {{e, c}, 3}, {{g, c}, 3}, {{g, d}, 3}, {{g, f}, 2}}

*** Iteración 6 ***

Se agregó la arista: {e, c}, T={{a, b}, {b, e}, {e, g}, {e, d}, {d, f}, {e, c}}
Peso mínimo acumulado: 17
AS={{{a, d}, 6}, {{a, f}, 7}, {{b, c}, 5}, {{b, d}, 5}, {{g, c}, 3}, {{g, d}, 3}}
*** Finalmente ***
T={{a, b}, {b, e}, {e, g}, {e, d}, {d, f}, {e, c}} y es de peso: 17
Animación del árbol generador T:
{{a, b}, {b, e}, {e, g}, {e, d}, {d, f}, {e, c}}



Explicación en video

37) **Kruskal**: muestra **paso a paso** la ejecución del **algoritmo de Kruskal** sobre un **grafo** "G" **no dirigido**, **conexo** y **ponderado**, generando adicionalmente una **animación arista por arista** del árbol de **expansión final**. La instrucción **acepta únicamente** grafos construidos en el "Wolfram System" de *Mathematica*. **No procesa** grafos con **aristas múltiples** y **ningún peso** puede ser **igual a cero**.

Sintaxis: `Kruskal[G]`. **No siempre** genera el **mismo árbol** si se invoca **varias veces**.

Ejemplo 8.73

Aplique el algoritmo de *Kruskal* sobre un grafo con aristas: {{a, b}, {a, d}, {a, f}, {b, d}, {b, e}, {b, c}, {c, e}, {c, g}, {d, e}, {d, g}, {d, f}, {e, g}, {f, g}} y pesos seudoaleatorios reales de uno a diez.

Solución:

Al emplear el comando **`Kruskal`**, se obtiene:

In[ ] :=

```
grafo = Grafo[{{a, b}, {a, d}, {a, f}, {b, d}, {b, e}, {b,
c}, {c, e}, {c, g}, {d, e}, {d, g}, {d, f}, {e, g}, {f, g}},
pesos -> RandomReal[{1, 10}, 13], mostrarpesos -> True]
Kruskal[grafo]
```

Out[ ] :=

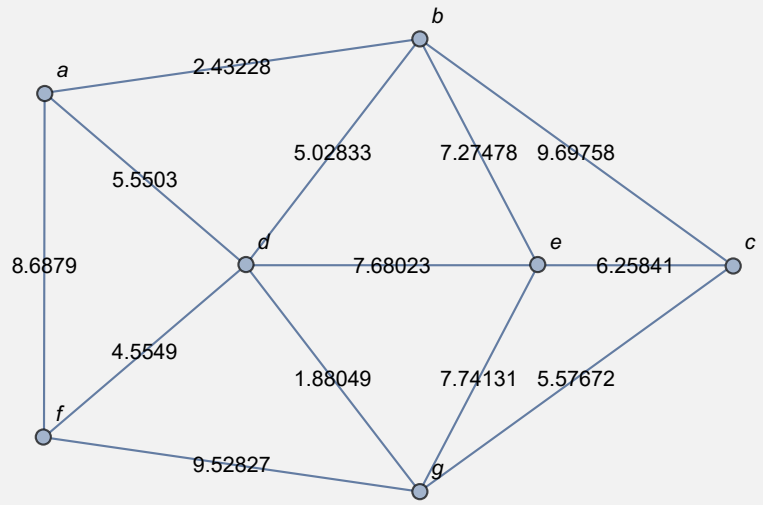


Tabla de pesos del grafo:

	a	b	c	d	e	f	g
a	0	2.43228	0	5.5503	0	8.6879	0
b	2.43228	0	9.69758	5.02833	7.27478	0	0
c	0	9.69758	0	0	6.25841	0	5.57672
d	5.5503	5.02833	0	0	7.68023	4.5549	1.88049
e	0	7.27478	6.25841	7.68023	0	0	7.74131
f	8.6879	0	0	4.5549	0	0	9.52827
g	0	0	5.57672	1.88049	7.74131	9.52827	0

*** Inicialización ***

Aristas seleccionables (AS) del árbol T a construir: {{a ●—● b, 2.43228}, {a ●—● d, 5.5503}, {a ●—● f, 8.6879}, {b ●—● d, 5.02833}, {b ●—● e, 7.27478}, {b ●—● c, 9.69758}, {c ●—● e, 6.25841}, {c ●—● g, 5.57672}, {d ●—● e, 7.68023}, {d ●—● g, 1.88049}, {d ●—● f, 4.5549}, {e ●—● g, 7.74131}, {f ●—● g, 9.52827}}

*** Iteración 1 ***

Se agregó la arista: d ●—● g, T={d ●—● g}

Peso mínimo acumulado: 1.88049

AS={{a ●—● b, 2.43228}, {a ●—● d, 5.5503}, {a ●—● f, 8.6879}, {b ●—● d, 5.02833}, {b ●—● e, 7.27478}, {b ●—● c, 9.69758}, {c ●—● e, 6.25841}, {c ●—● g, 5.57672}, {d ●—● e, 7.68023}, {d ●—● f, 4.5549}, {e ●—● g, 7.74131}, {f ●—● g, 9.52827}}

*** Iteración 2 ***

Se agregó la arista: a ●—● b, T={d ●—● g, a ●—● b}

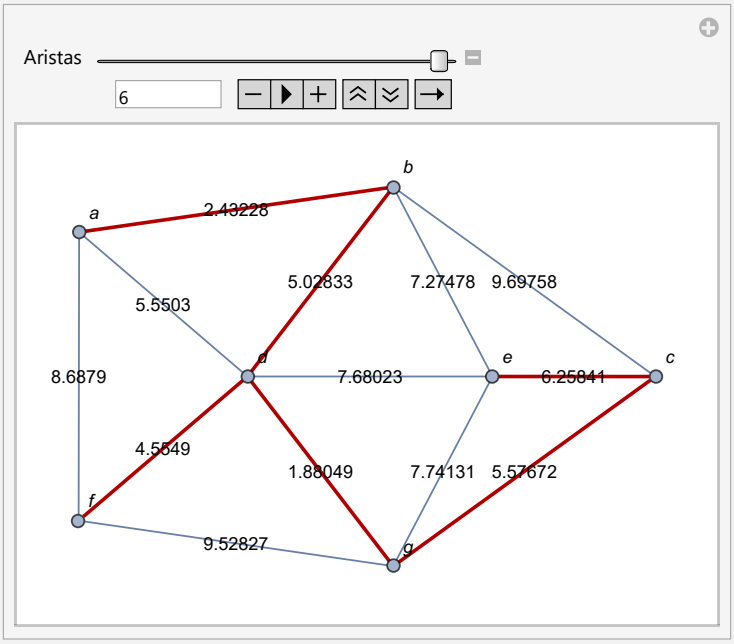
Peso mínimo acumulado: 4.31277

AS={{a ●—● d, 5.5503}, {a ●—● f, 8.6879}, {b ●—● d, 5.02833}, {b ●—● e, 7.27478}, {b ●—● c, 9.69758}, {c ●—● e, 6.25841}, {c ●—● g, 5.57672}, {d ●—● e, 7.68023}, {d ●—● f, 4.5549}, {e ●—● g, 7.74131}, {f ●—● g, 9.52827}}

*** Iteración 3 ***

Se agregó la arista: d ●—● f, T={d ●—● g, a ●—● b, d ●—● f}

Peso mínimo acumulado: 8.86767
AS={{a ●—● d, 5.5503}, {a ●—● f, 8.6879}, {b ●—● d, 5.02833}, {b ●—● e, 7.27478}, {b ●—● c, 9.69758}, {c ●—● e, 6.25841}, {c ●—● g, 5.57672}, {d ●—● e, 7.68023}, {e ●—● g, 7.74131}, {f ●—● g, 9.52827}}
*** Iteración 4 ***
Se agregó la arista: b ●—● d, T={d ●—● g, a ●—● b, d ●—● f, b ●—● d}
Peso mínimo acumulado: 13.896
AS={{a ●—● d, 5.5503}, {a ●—● f, 8.6879}, {b ●—● e, 7.27478}, {b ●—● c, 9.69758}, {c ●—● e, 6.25841}, {c ●—● g, 5.57672}, {d ●—● e, 7.68023}, {e ●—● g, 7.74131}, {f ●—● g, 9.52827}}
*** Iteración 5 ***
Se agregó la arista: c ●—● g, T={d ●—● g, a ●—● b, d ●—● f, b ●—● d, c ●—● g}
Peso mínimo acumulado: 19.4727
AS={{a ●—● f, 8.6879}, {b ●—● e, 7.27478}, {b ●—● c, 9.69758}, {c ●—● e, 6.25841}, {d ●—● e, 7.68023}, {e ●—● g, 7.74131}, {f ●—● g, 9.52827}}
*** Iteración 6 ***
Se agregó la arista: c ●—● e, T={d ●—● g, a ●—● b, d ●—● f, b ●—● d, c ●—● g, c ●—● e}
Peso mínimo acumulado: 25.7311
AS={{a ●—● f, 8.6879}, {b ●—● e, 7.27478}, {b ●—● c, 9.69758}, {d ●—● e, 7.68023}, {e ●—● g, 7.74131}, {f ●—● g, 9.52827}}
*** Finalmente ***
T={d ●—● g, a ●—● b, d ●—● f, b ●—● d, c ●—● g, c ●—● e} y es de peso: 25.7311
Animación del árbol generador T:



Ejemplo 8.74

Genere paso a paso las iteraciones del algoritmo de *Kruskal* sobre el grafo del ejercicio tras anterior.

Solución:

En el software:

In[ ] :=

```
grafo = Grafo[{{a, b}, {a, d}, {a, f}, {b, c}, {b, d}, {b, e}, {c,
e}, {c, g}, {d, e}, {d, f}, {d, g}, {e, g}, {f, g}}, pesos -> {5, 6,
7, 5, 5, 4, 3, 3, 2, 2, 3, 1, 2}, mostrarpesos -> True]
Kruskal[grafo]
```

Out[ ] :=

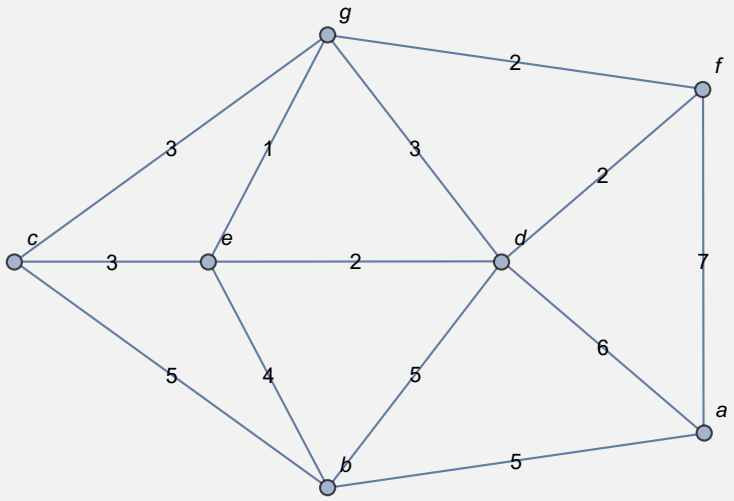


Tabla de pesos del grafo:

	a	b	c	d	e	f	g
a	0	5	0	6	0	7	0
b	5	0	5	5	4	0	0
c	0	5	0	0	3	0	3
d	6	5	0	0	2	2	3
e	0	4	3	2	0	0	1
f	7	0	0	2	0	0	2
g	0	0	3	3	1	2	0

*** Inicialización ***

Aristas seleccionables (AS) del árbol T a construir: {{a ●—● b, 5}, {a ●—● d, 6}, {a ●—● f, 7}, {b ●—● c, 5}, {b ●—● d, 5}, {b ●—● e, 4}, {c ●—● e, 3}, {c ●—● g, 3}, {d ●—● e, 2}, {d ●—● f, 2}, {d ●—● g, 3}, {e ●—● g, 1}, {f ●—● g, 2}}

*** Iteración 1 ***

Se agregó la arista: e ●—● g, T={e ●—● g}

Peso mínimo acumulado: 1

AS={{a ●—● b, 5}, {a ●—● d, 6}, {a ●—● f, 7}, {b ●—● c, 5}, {b ●—● d, 5}, {b ●—● e, 4}, {c ●—● e, 3}, {c ●—● g, 3}, {d ●—● e, 2}, {d ●—● f, 2}, {d ●—● g, 3}, {f ●—● g, 2}}

*** Iteración 2 ***

Se agregó la arista: d ●—● f, T={e ●—● g, d ●—● f}

Peso mínimo acumulado: 3

AS={{a ●—● b, 5}, {a ●—● d, 6}, {a ●—● f, 7}, {b ●—● c, 5}, {b ●—● d, 5}, {b ●—● e, 4}, {c ●—● e, 3}, {c ●—● g, 3}, {d ●—● e, 2}, {d ●—● g, 3}, {f ●—● g, 2}}

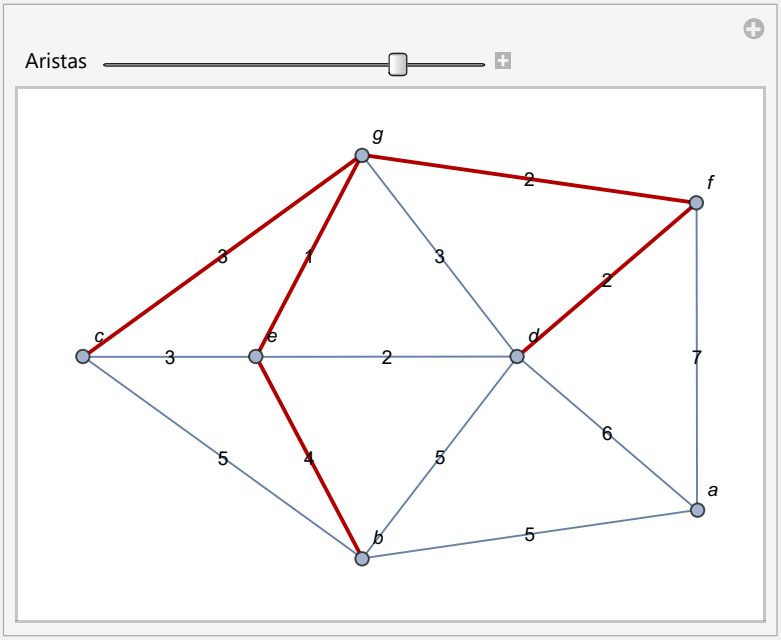
*** Iteración 3 ***

Se agregó la arista: f ●—● g, T={e ●—● g, d ●—● f, f ●—● g}

Peso mínimo acumulado: 5

AS={{a ●—● b, 5}, {a ●—● d, 6}, {a ●—● f, 7}, {b ●—● c, 5}, {b ●—● d, 5}, {b ●—● e, 4}, {c ●—● e, 3}, {c

●—● g, 3}, {d ●—● e, 2}, {d ●—● g, 3}}
*** Iteración 4 ***
Se agregó la arista: c ●—● g, T={e ●—● g, d ●—● f, f ●—● g, c ●—● g}
Peso mínimo acumulado: 8
AS={{a ●—● b, 5}, {a ●—● d, 6}, {a ●—● f, 7}, {b ●—● c, 5}, {b ●—● d, 5}, {b ●—● e, 4}, {c ●—● e, 3}, {d ●—● g, 3}}
*** Iteración 5 ***
Se agregó la arista: b ●—● e, T={e ●—● g, d ●—● f, f ●—● g, c ●—● g, b ●—● e}
Peso mínimo acumulado: 12
AS={{a ●—● b, 5}, {a ●—● d, 6}, {a ●—● f, 7}, {b ●—● c, 5}, {b ●—● d, 5}}
*** Iteración 6 ***
Se agregó la arista: a ●—● b, T={e ●—● g, d ●—● f, f ●—● g, c ●—● g, b ●—● e, a ●—● b}
Peso mínimo acumulado: 17
AS={{a ●—● d, 6}, {a ●—● f, 7}, {b ●—● c, 5}, {b ●—● d, 5}}
*** Finalmente ***
T={e ●—● g, d ●—● f, f ●—● g, c ●—● g, b ●—● e, a ●—● b} y es de peso: 17
Animación del árbol generador T:



Explicación en video

38) **ArbolGeneradorMaximo**: **función** que recibe como parámetro un **grafo conexo**, **no dirigido**, **ponderado** "G" y **retorna** un **árbol generador** (o de **expansión**) de **peso máximo**. El grafo pudo haber sido **creado** en el "Wolfram System" de *Mathematica*, o bien, mediante el uso del **paquete** "Combinatorica". Brinda la **opción** "`animacion->True`" que muestra una **animación** sobre "G", **arista por**

arista del árbol de expansión **encontrado**.

Sintaxis: **`ArbolGeneradorMaximo[G]`**, o bien, **`ArbolGeneradorMaximo[G, animacion -> True]`**. **No procesa** grafos con **aristas múltiples** y **ningún peso** puede ser **igual a cero**.

Ejemplo 8.75

Considerando el grafo del ejemplo anterior, obtenga haciendo uso de la instrucción **`ArbolGeneradorMaximo`**, un árbol de expansión de peso máximo y muéstrelo mediante una animación.

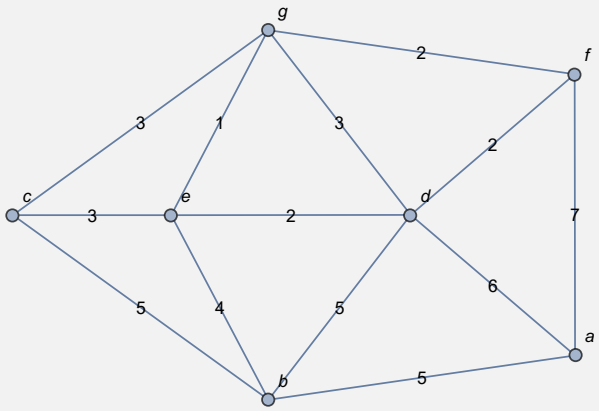
Solución:

Al recurrir a la opción "**`animacion -> True`**", se tiene:

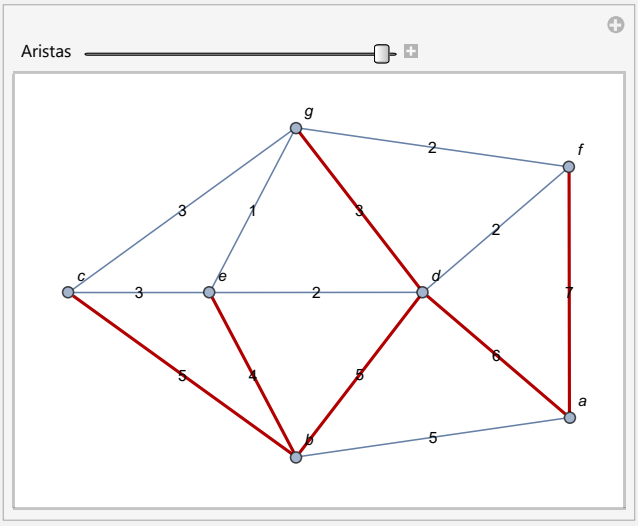
In[ ] :=

```
grafo = Grafo[{{a, b}, {a, d}, {a, f}, {b, c}, {b, d}, {b, e}, {c, e}, {c, g}, {d, e}, {d, f}, {d, g}, {e, g}, {f, g}}, pesos -> {5, 6, 7, 5, 5, 4, 3, 3, 2, 2, 3, 1, 2}, mostrarpesos -> True]
ArbolGeneradorMaximo[grafo, animacion -> True]
```

Out[ ] :=



Aristas: {{a, f}, {a, d}, {b, d}, {b, c}, {b, e}, {d, g}}
Peso máximo: 30



Ejemplo 8.76

A través de "Combinatorica", construya un grafo cuyas aristas correspondan a otro generado mediante **`GrafoRandomConexo[10, 10]`**. Asigne pesos seudoaleatorios enteros de uno a diez y encuentre un árbol de expansión de peso máximo sobre el grafo resultante. Visualice la salida en una animación.

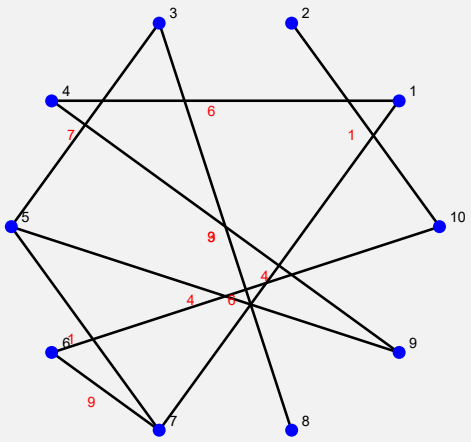
Solución:

En *Mathematica*:

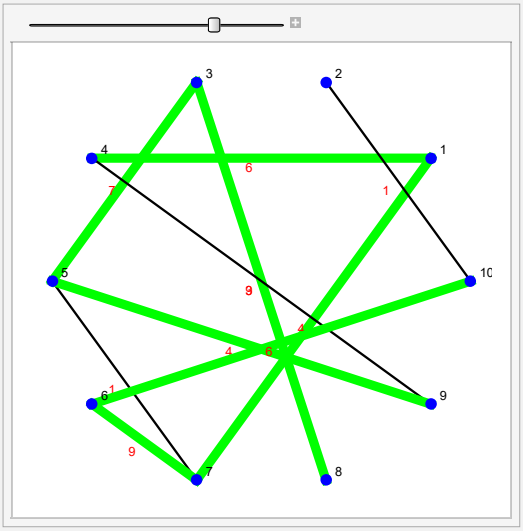
In[ ] :=

```
grafo = GrafoRandomConexo[10, 10];
aristas = AristasWolframSystemToCombinatorica[EdgeList[grafo]];
pesosaristas = RandomInteger[{1, 10}, Length[aristas]];
GrafoC[aristas, pesos->pesosaristas, mostrarpesos->True]
ArbolGeneradorMaximo[G, animacion->True]
```

Out[ ] :=



Aristas: {{3, 8}, {6, 7}, {3, 5}, {1, 4}, {6, 10}, {1, 7}, {5, 9}, {4, 9}, {2, 10}}
Peso máximo: 49



Ⓝ La animación de este ejercicio en comparación con el anterior, se muestra estéticamente un poco diferente. La razón de ello radica en que la actual, se creó en el ambiente provisto por el paquete "Combinatorica".

Explicación en video

39) **CDFArbolesGeneradores**: **construye** una **animación** dadas las **aristas** de un **grafo** "G", sus **pesos** y un **orden** en sus vértices, mostrando la **aplicación** de los **algoritmos**: "buscar primero a lo ancho", "buscar primero a lo largo", "Prim" y "Kruskal" sobre "G".

Sintaxis: `CDFArbolesGeneradores[lados, pesos, orden]`, siendo "lados" las **aristas** de "G" dadas como **pares ordenados**, "pesos" un **vector** de **ponderaciones** y "orden" la **lista ordenada** de los **vértices**, a través de la cual, se **aplicarán** los procedimientos.

Ejemplo 8.77

Ejecute la instrucción **`CDFArbolesGeneradores`** en un grafo con aristas: {{a, b}, {a, c}, {b, c}, {b, d}, {b, f}, {c, d}, {c, g}, {c, e}, {d, f}, {d, g}, {e, g}, {f, g}}, pesos: {5, 10, 15, 5, 6, 7, 12, 30, 9, 8, 20, 14} y de acuerdo al orden del abecedario para sus vértices.

Solución:

En el software:

In[ ] :=

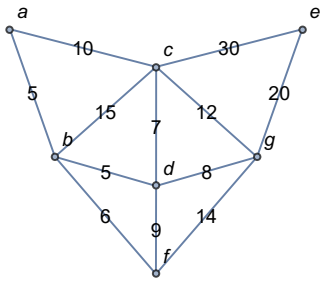
```
CDFArbolesGeneradores[{{a, b}, {a, c}, {b, c}, {b, d}, {b, f}, {c,
d}, {c, g}, {c, e}, {d, f}, {d, g}, {e, g}, {f, g}}, {5, 10, 15, 5,
6, 7, 12, 30, 9, 8, 20, 14}, {a, b, c, d, e, f, g}]
```

Out[ ] :=

Aristas del grafo {{a, b}, {a, c}, {b, c}, {b, d}, {b, f}, {c, d}, {c, g}, {c, e}, {d, f}, {d, g}, {e, g}, {f, g}}

Pesos de los lados {5, 10, 15, 5, 6, 7, 12, 30, 9, 8, 20, 14}

Orden de los nodos {a, b, c, d, e, f, g}




```
Buscar primero a lo ancho: {{a, b}, {a, c}, {b, d}, {b, f}, {c, e}, {c, g}}
Buscar primero a lo largo: {{a, b}, {b, c}, {c, d}, {d, f}, {f, g}, {g, e}}
Prim: {{{a, b}, {b, d}, {b, f}, {d, c}, {d, g}, {g, e}}, 51}
Kruskal: {{{a, b}, {b, d}, {b, f}, {c, d}, {d, g}, {e, g}}, 51}
```

Ejemplo 8.78

Considerando un grafo con lados: {{a, b}, {a, d}, {a, f}, {b, d}, {b, e}, {b, c}, {c, e}, {c, g}, {d, e}, {d, g}, {d, f}, {e, g}, {f, g}}, pesos seudoaleatorios reales de uno a diez y un orden en los nodos en función del abecedario, corra los algoritmos: *buscar primero a lo ancho*, *buscar primero a lo largo*, *Prim* y *Kruskal*, mediante el uso del comando **`CDFArbolesGeneradores`**.

Solución:

In[ ] :=

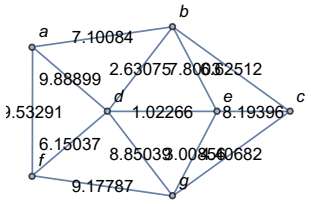
```
CDFArbolesGeneradores[{{a, b}, {a, d}, {a, f}, {b, d}, {b, e},
{b, c}, {c, e}, {c, g}, {d, e}, {d, g}, {d, f}, {e, g}, {f, g}},
RandomReal[{1, 10}, 13], {a, b, c, d, e, f, g}]
```

Out[ ] :=

Aristas del grafo {{a, b}, {a, d}, {a, f}, {b, d}, {b, e}, {b, c}, {c, e}, {c, g}, {d, e}, {d, g}, {d, f}, {e, g}, {f, g}}

Pesos de los lados {7.10084, 9.88899, 9.53291, 2.63075, 7.8003, 6.62512, 8.19396, 4.40682, 1.02266, 8.85039, 6.15037, 3.00856, 9.17787}

Orden de los nodos {a, b, c, d, e, f, g}




```
Buscar primero a lo ancho: {{a, b}, {a, d}, {a, f}, {b, c}, {b, e}, {d, g}}
Buscar primero a lo largo: {{a, b}, {b, c}, {c, e}, {e, d}, {d, f}, {f, g}}
Prim: {{{a, b}, {b, d}, {d, e}, {e, g}, {g, c}, {d, f}}, 24.32}
Kruskal: {{{d, e}, {b, d}, {e, g}, {c, g}, {d, f}, {a, b}}, 24.32}
```

Explicación en video

Aporte pedagógico

Ciertos comandos de *VilCretas* pueden ser claves en el apoyo didáctico para **interiorizar conceptos esenciales** de la teoría de árboles y sus aplicaciones. Por ejemplo, **`CDFKArbol`** y **`CDFArbolesGeneradores`**, son instrucciones que con mucha eficacia le permiten al estudiante **visualizar** definiciones como las señaladas con anterioridad y con rapidez, evidenciar el **uso** de **importantes algoritmos**, tales como: *Prim* y *Kruskal*.

Aporte de investigación

Algunos de los ejemplos compartidos en la sección, han constatado cómo es posible utilizar los comandos de *VilCretas* vinculados con la teoría de árboles, para la **construcción de conjeturas**. Ejemplo de ello, lo circunscriben los ejercicios expuestos en el uso de las instrucciones **`CantAristasKArbolCompletoNivel`**, **`CantNodosKArbolCompletoNivel`** y **`NHojasTotalBinario`**. Existen otros comandos de *VilCretas* que dentro de un planeamiento de clase oportuno, pueden favorecer **procesos de exploración** relacionados con las **aplicaciones** de la teoría de árboles.

Ejercicios

Resuelva los siguientes ejercicios utilizando como apoyo el paquete *VilCretas*.

1) Construya los árboles con el conjunto de aristas dado a continuación:

   a) A_1 : Aristas={{1, 2}, {1, 7}, {2, 5}, {3, 19}, {4, 8}, {4, 12}, {4, 18}, {5, 16}, {5, 17}, {6, 15}, {7, 10}, {9, 11}, {9, 15}, {10, 14}, {11, 14}, {13, 19}, {13, 20}, {14, 19}, {18, 20}}, con nodos hexagonales.

   b) A_2 : Aristas={{1, 2}, {1, 3}, {2, 4}, {2, 5}, {3, 6}, {3, 7}, {4, 8}, {4, 9}, {5, 10}, {5, 11}, {6, 12}, {6, 13}, {7, 14}, {7, 15}, {8, 16}, {8, 17}, {9, 18}, {9, 19}, {10, 20}}, dirigido y vértices con forma octagonal.

   c) A_3 : Aristas={{1, 3}, {1, 5}, {1, 7}, {2, 9}, {3, 9}, {3, 17}, {4, 6}, {6, 14}, {7, 16}, {7, 19}, {8, 10}, {8, 25}, {9, 14}, {10, 21}, {11, 24}, {12, 15}, {13, 21}, {15, 16}, {16, 24}, {17, 22}, {17, 25}, {18, 19}, {20, 23}, {20, 25}}, con pesos seudoaleatorios reales en el intervalo $[1,10]$ y vértices rectangulares.

   d) A_4 : Aristas={{1, 2}, {1, 3}, {2, 4}, {2, 5}, {3, 6}, {3, 7}, {4, 8}, {4, 9}, {5, 10}, {5, 11}, {6, 12}, {6, 13}, {7, 14}, {7, 15}, {8, 16}, {8, 17}, {9, 18}, {9, 19}, {10, 20}, {10, 21}, {11, 22}, {11, 23}, {12, 24}, {12, 25}}, dirigido, con pesos seudoaleatorios reales en el intervalo $[1,10]$ y nodos pentagonales.

   e) A_5 : Aristas={{1, 12}, {1, 17}, {1, 27}, {2, 8}, {3, 30}, {4, 12}, {4, 18}, {4, 35}, {5, 31}, {6, 10}, {6, 24}, {6, 35}, {7, 9}, {7, 11}, {8, 16}, {8, 30}, {10, 20}, {10, 28}, {11, 30}, {12, 26}, {13, 18}, {14, 30}, {15, 25}, {15, 32}, {19, 20}, {19, 21}, {20, 34}, {21, 29}, {22, 27}, {23, 31}, {23, 32}, {27, 31}, {27, 33}, {30, 35}}, con pesos seudoaleatorios reales en el intervalo $[1,10]$ y vértices triangulares.

2) Construya los árboles del ejercicio anterior asumiendo el nodo 12 como raíz.

3) Construya un árbol gráfico dirigido y no dirigido, tomando el conjunto de aristas dado en cada caso, en el ejercicio 1.

4) Resuelva el ejercicio 2 por medio de un árbol gráfico.

5) Convierta a un árbol gráfico, los árboles obtenidos como salida del ejercicio 1.

6) Transforme los árboles no dirigidos del ejercicio 3, a grafos en el "Wolfram System" de *Mathematica*. Sugerencia: recurra al uso de **`ArbolGraficoToArbol`**.

7) Use el comando **`ArbolBinarioQ`** del paquete *VilCretas*, para determinar si los árboles del ejercicio 1 son binarios.

8) Considere los grafos con el conjunto de aristas dado a continuación:

   a) G_1 : Aristas={{1, 19}, {1, 24}, {2, 5}, {2, 7}, {2, 14}, {2, 16}, {3, 6}, {3, 7}, {4, 26}, {5, 12}, {5, 21}, {6, 14}, {6, 22}, {7, 8}, {7, 9}, {7, 11}, {8, 17}, {8, 20}, {9, 14}, {9, 25}, {10, 12}, {10, 26}, {12, 13}, {13, 27}, {13, 29}, {15, 27}, {15, 30}, {16, 18}, {17, 19}, {17, 21}, {19, 26}, {21, 22}, {22, 23}, {24, 26}, {27, 28}}.

   b) G_2 : Aristas={{1, 7}, {1, 12}, {1, 28}, {2, 13}, {2, 14}, {2, 22}, {3, 4}, {3, 23}, {3, 29}, {3, 30}, {4, 24}, {4, 25}, {4, 30}, {5, 9}, {5, 16}, {5, 19}, {5, 28}, {6, 8}, {6, 11}, {6, 30}, {7, 9}, {7, 17}, {7, 19}, {7, 29}, {8, 17}, {9, 28}, {10, 12}, {10, 16}, {10, 23}, {11, 12}, {12, 14}, {12, 27}, {12, 29}, {14, 25}, {14, 26}, {15, 20}, {15, 21}, {16, 28}, {17, 21}, {17, 24}, {18, 20}, {19, 20}, {19, 22}, {19, 29}, {19, 30}, {20, 25}, {20, 28}, {22, 25}, {23, 28}, {24, 29}}.

c) G_3 : Aristas={{1, 2}, {1, 12}, {2, 8}, {2, 14}, {2, 16}, {3, 4}, {3, 6}, {3, 16}, {3, 18}, {3, 19}, {4, 5}, {4, 23}, {5, 8}, {5, 13}, {6, 24}, {6, 25}, {7, 9}, {7, 11}, {7, 13}, {7, 27}, {8, 24}, {8, 26}, {9, 28}, {10, 24}, {10, 30}, {11, 15}, {11, 20}, {11, 30}, {12, 15}, {12, 16}, {12, 25}, {12, 28}, {12, 29}, {13, 21}, {14, 30}, {15, 17}, {15, 18}, {15, 22}, {16, 17}, {16, 20}, {16, 27}, {16, 28}, {17, 20}, {17, 21}, {18, 20}, {18, 28}, {20, 21}, {20, 22}, {20, 24}, {20, 28}, {21, 28}, {22, 25}, {22, 26}, {23, 27}, {25, 30}}.

d) G_4 : Aristas={{1, 2}, {1, 3}, {2, 4}, {2, 5}, {3, 6}, {3, 7}, {4, 8}, {4, 9}, {5, 10}, {5, 11}, {6, 12}, {6, 13}, {7, 14}, {7, 15}, {8, 16}, {8, 17}, {9, 18}, {9, 19}, {10, 20}, {10, 21}, {11, 22}, {11, 23}, {12, 24}, {12, 25}, {13, 26}, {13, 27}, {14, 28}, {14, 29}, {15, 30}, {15, 31}, {16, 32}, {16, 33}, {17, 34}, {17, 35}, {18, 36}, {18, 37}, {19, 38}, {19, 39}, {20, 40}, {20, 41}, {21, 42}, {21, 43}, {22, 44}, {22, 45}, {23, 46}, {23, 47}, {24, 48}, {24, 49}, {25, 50}, {25, 51}, {26, 52}, {26, 53}, {27, 54}, {27, 55}, {28, 56}, {28, 57}, {29, 58}, {29, 59}, {30, 60}}.

Empleando los comandos **`ArbolQ`** y **`ArbolJQ`**, determine si los grafos son árboles.

9) Halle los nodos terminales y la altura de los árboles expuestos en el ejercicio 1. Sugerencia: recurra al uso de las instrucciones **`Hojas`** y **`Altura`** del paquete *VilCretas*.

10) Encuentre las hojas y la altura sobre los árboles: **`ArbolRandom[100]`**, **`KArbol[20, 100]`**, **`KArbolCompletoNivel[10, 5]`**, **`ArbolBinario[100]`**, **`ArbolBinarioCompleto-Nivel[10]`** y **`ArbolBinarioRandom[100]`**.

11) Construya utilizando los comandos **`ArbolHuffman`** y **`ArbolHuffmanN`**, un árbol de códigos de *Huffman* optimizado y no optimizado, respectivamente, para:

a) "tecnologias"

b) "matematicas discretas"

12) Construya un árbol que represente las siguientes expresiones algebraicas:

a) $((a+b)\,c) - f - \dfrac{f}{a}\,(c-f) + (b + c \cdot a)$

b) $\left((a-b)\,\dfrac{c}{g}\right)^2 - f\,(a - b + c \cdot h) + (c-f)^2 + d$

c) $(a+b+d)\,\sqrt[3]{\dfrac{c}{g}} - f\left((a-b)^2 + \sqrt{c \cdot h}\right) + \dfrac{(c+f)^2}{d}$

Sugerencia: use el comando **`ArbolExpAlgebraica`**.

13) Genere un árbol binario de búsqueda para almacenar:

a) {2, 15, 5, 13, 1, 8, 10, 19, 14, 9, 20, 18, 7, 11, 4, 12, 3, 17, 6}

b) {50, 32, 5, 49, 28, 6, 14, 33, 15, 7, 20, 10, 18, 45, 48, 30, 39, 44, 25, 36, 24, 17, 46, 16, 31, 26, 22, 41, 37, 40, 43, 29, 11, 21, 42, 2, 38, 4, 34, 23, 12, 47, 19, 13}

c) {22, 23, 41, 33, 6, 27, 5, 14, 11, 2, 18, 44, 20, 50, 45, 48, 10, 39, 43, 36, 15, 42, 16, 9, 29, 4, 3, 1, 35, 30, 47, 19, 17, 12, 38, 21, 8, 34, 13, 40, 26, 25, 37, 28, 32, 7, 46, 31}

14) Recorra en orden inicial y final los árboles obtenidos en el ejercicio anterior. Muestre mediante una animación.

15) Determine una notación polaca y polaca inversa para las expresiones algebraicas del ejercicio 12. Sugerencia: utilice los comandos **`Polaca`** y **`PolacaInversa`** del paquete *VilCretas*.

16) Encuentre haciendo uso del comando **`ArbolesGeneradores`**, árboles de expansión para los grafos compartidos en el ejercicio 8.

17) Aplique los algoritmos "buscar primero a lo ancho" y "buscar primero a lo largo" sobre los grafos del ejercicio 8. Muestre a través de una animación.

18) Busque el dato **`RandomInteger[{1, 100}]`** a lo ancho y a lo largo, sobre los grafos del ejercicio 8. Para ello utilice las instrucciones de *VilCretas*: **`BuscarDatoAncho`** y **`BuscarDatoLargo`**.

19) Aplique el algoritmo de *Prim* y de *Kruskal* sobre los grafos del ejercicio 8, asumiendo pesos seudoaleatorios reales en el intervalo $[1,10]$.

20) Encuentre por medio de una animación, un árbol generador de peso máximo, sobre los grafos del ejercicio 8, asumiendo pesos seudoaleatorios reales en el intervalo $[1,10]$. Sugerencia: use la opción "**`animacion ->`** **`True`**" de **`ArbolGeneradorMaximo`**.

Capítulo 9

Máquinas y autómatas de estado finito con *VilCretas*

El tema tratado en el presente capítulo, con frecuencia es catalogado como un área de conocimiento abstracta y en ocasiones difícil de explicar, más aún, cuando uno de los objetivos principales reside en modelar a la población estudiantil, algunas de sus diversas aplicaciones. En el paquete *VilCretas* se han diseñado **veintitrés** instrucciones, muchas de ellas orientadas a la creación de objetos de manipulación dinámica, que pretenden facilitar el abordaje didáctico y la representación conceptual. Las máquinas y los autómatas de estado finito ofrecen un área de conocimiento muy interesante, vinculada con problemas de computabilidad. *VilCretas* a este respecto, favorece un robustecimiento teórico y algorítmico, dando cabida a experiencias de aprendizaje de naturaleza experimental.

1) **MaquinaToDiagrama**: **genera** el **diagrama de transición** de una **máquina de estado finito** dadas las **seis componentes** que la definen: el **conjunto de estados**, el **conjunto de símbolos de entrada**, el **conjunto de símbolos de salida**, el **estado inicial**, la **función estado siguiente** y la **función de salida**. Las funciones de estado siguiente y de salida se escriben juntas, como una **matriz** con **cuatro columnas**, donde cada fila contiene en orden: el **par** "(estado, entrada)" y las **imágenes** en dicha dupla de la función estado siguiente y de salida, respectivamente. Brinda la **opción** "`padding->Valor`" que añade un **espacio** de **contorno** especificado en "Valor" (por defecto es igual a **diez**).

   Sintaxis: `MaquinaToDiagrama[Estados, Entradas, Salidas, Inicio, FEstadosSalidas]`, o bien,

   ```
   MaquinaToDiagrama[Estados, Entradas, Salidas, Inicio, FEstadosSalidas,
   padding->Valor]
   ```

   con "Inicio" el **estado inicial** y "FEstadosSalidas" las **funciones estado siguiente y de salida**.

Ejemplo 9.1

Devuelva el diagrama de transición de una máquina de estado finito con:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
- Símbolos de entrada: $\tau = \{a, b, c, d\}$
- Símbolos de salida: $\delta = \{0, 1, 2, 3, 4\}$
- Estado inicial: $\sigma^* = \sigma_3$
- Función de estado siguiente y de salida: $\{\{\sigma_0, a, \sigma_1, 3\}, \{\sigma_0, b, \sigma_1, 2\}, \{\sigma_0, c, \sigma_1, 1\}, \{\sigma_0, d, \sigma_1, 4\}, \{\sigma_1, a, \sigma_1, 0\}, \{\sigma_1, b, \sigma_1, 3\}, \{\sigma_1, c, \sigma_2, 2\}, \{\sigma_1, d, \sigma_1, 3\}, \{\sigma_2, a, \sigma_0, 1\}, \{\sigma_2, b, \sigma_1, 3\}, \{\sigma_2, c, \sigma_1, 2\}, \{\sigma_2, d, \sigma_0, 4\}, \{\sigma_3, a, \sigma_0, 0\}, \{\sigma_3, b, \sigma_2, 2\}, \{\sigma_3, c, \sigma_0, 1\}, \{\sigma_3, d, \sigma_0, 4\}\}$

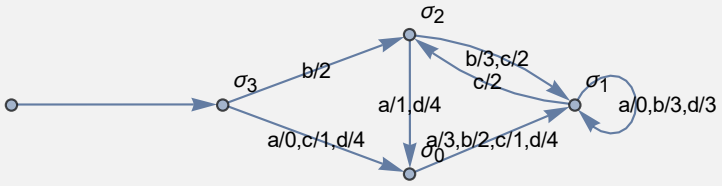
Solución:

Al recurrir al comando **MaquinaToDiagrama**, se tiene:

In[ ] :=

```
MaquinaToDiagrama[{σ0, σ1, σ2, σ3}, {a, b, c, d}, {0, 1, 2, 3, 4}, σ3,
{{σ0, a, σ1, 3}, {σ0, b, σ1, 2}, {σ0, c, σ1, 1}, {σ0, d, σ1, 4}, {σ1, a, σ1,
0}, {σ1, b, σ1, 3}, {σ1, c, σ2, 2}, {σ1, d, σ1, 3}, {σ2, a, σ0, 1}, {σ2,
b, σ1, 3}, {σ2, c, σ1, 2}, {σ2, d, σ0, 4}, {σ3, a, σ0, 0}, {σ3, b, σ2,
2}, {σ3, c, σ0, 1}, {σ3, d, σ0, 4}}, padding -> 20]
```

Out[ ] :=



N En este ejemplo se hace uso de la opción "**padding -> 20**", para evitar cortes en las etiquetas mostradas dentro del diagrama.

Ejemplo 9.2

Construya el diagrama de transición de la siguiente máquina de estado finito:

- Estados: $\sigma = \{\{\}, \{\sigma_0\}, \{\sigma_1\}, \{\sigma_2\}, \{\sigma_0, \sigma_1\}, \{\sigma_0, \sigma_2\}, \{\sigma_1, \sigma_2\}, \{\sigma_0, \sigma_1, \sigma_2\}\}$
- Símbolos de entrada: $\tau = \{a, b\}$
- Símbolos de salida: $\delta = \{0, 1, 2, 3, 4\}$
- Estado inicial: $\sigma^* = \{\sigma_0, \sigma_1, \sigma_2\}$
- Función de estado siguiente y de salida: $\{\{\{\}, a, \{\sigma_0, \sigma_1\}, 3\}, \{\{\}, b, \{\sigma_0, \sigma_1, \sigma_2\}, 2\}, \{\{\sigma_0\}, a, \{\}, 1\}, \{\{\sigma_0\}, b, \{\sigma_0\}, 4\}, \{\{\sigma_1\}, a, \{\sigma_0, \sigma_1\}, 0\}, \{\{\sigma_1\}, b, \{\sigma_0, \sigma_2\}, 3\}, \{\{\sigma_2\}, a, \{\sigma_0, \sigma_1, \sigma_2\}, 2\}, \{\{\sigma_2\}, b, \{\sigma_0, \sigma_2\}, 3\}, \{\{\sigma_0, \sigma_1\}, a, \{\sigma_1, \sigma_2\}, 1\}, \{\{\sigma_0, \sigma_1\}, b, \{\sigma_2\}, 3\}, \{\{\sigma_0, \sigma_2\}, a, \{\sigma_1\}, 2\}, \{\{\sigma_0, \sigma_2\}, b, \{\sigma_0, \sigma_1\}, 4\}, \{\{\sigma_1, \sigma_2\}, a, \{\}, 0\}, \{\{\sigma_1, \sigma_2\}, b, \{\}, 2\}, \{\{\sigma_0, \sigma_1, \sigma_2\}, a, \{\sigma_0\}, 1\}, \{\{\sigma_0, \sigma_1, \sigma_2\}, b, \{\sigma_0, \sigma_1, \sigma_2\}, 4\}\}$

Solución:

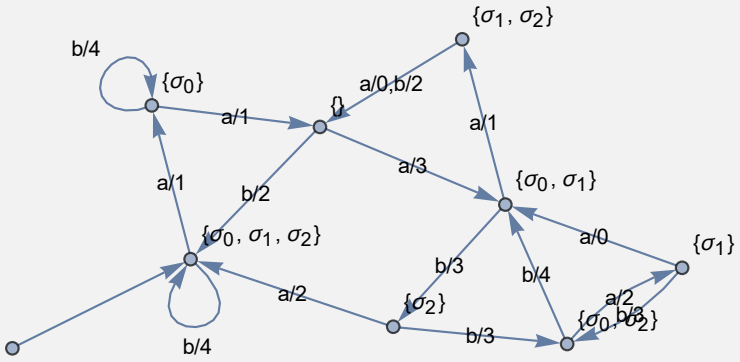
En el software:

In[ ] :=

```
MaquinaToDiagrama[{{}, {σ0}, {σ1}, {σ2}, {σ0, σ1}, {σ0, σ2}, {σ1, σ2},
{σ0, σ1, σ2}}, {a, b}, {0, 1, 2, 3, 4}, {σ0, σ1, σ2}, {{{}, a, {σ0, σ1},
3}, {{}, b, {σ0, σ1, σ2}, 2}, {{σ0}, a, {}, 1}, {{σ0}, b, {σ0}, 4},
{{σ1}, a, {σ0, σ1}, 0}, {{σ1}, b, {σ0, σ2}, 3}, {{σ2}, a, {σ0, σ1, σ2},
```

```
2}, {{σ2}, b, {σ0, σ2}, 3}, {{σ0, σ1}, a, {σ1, σ2}, 1}, {{σ0, σ1}, b,
{σ2}, 3}, {{σ0, σ2}, a, {σ1}, 2}, {{σ0, σ2}, b, {σ0, σ1}, 4}, {{σ1, σ2},
a, {}, 0}, {{σ1, σ2}, b, {}, 2}, {{σ0, σ1, σ2}, a, {σ0}, 1}, {{σ0, σ1,
σ2}, b, {σ0, σ1, σ2}, 4}}, padding->15]
```

Out[ ] :=



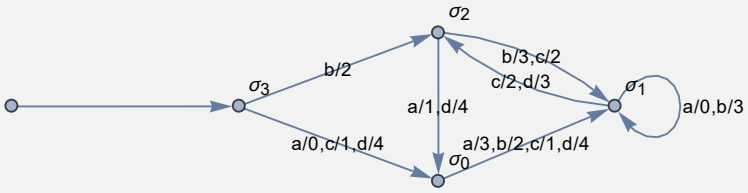
Explicación en video

2) **DiagramaToMaquina**: **construye** una **máquina de estado finito** con sus **seis componentes** respectivas, a través de su **diagrama de transición**.

Sintaxis: `DiagramaToMaquina[Maquina]`, donde "Maquina" corresponde al **diagrama**.

Ejemplo 9.3

Considere el diagrama de transición:



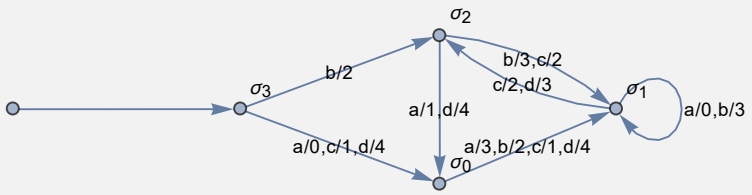
Obtenga cada una de las seis componentes que caracterizan a la máquina de estado finito.

Solución:

En *Mathematica*:

In[ ] :=

Maquina =



;

```
DiagramaToMaquina[Maquina]
```

Out[ ] :=

Conjunto de estados: $\{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$

Símbolos de entrada: {a, b, c, d}

Símbolos de salida: {0, 1, 2, 3, 4}

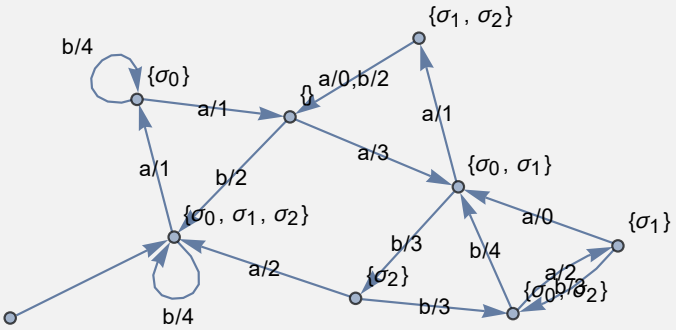
Estado inicial: σ_3

		Δ	Ω
σ_0	a	σ_1	3
σ_0	b	σ_1	2
σ_0	c	σ_1	1
σ_0	d	σ_1	4
σ_1	a	σ_1	0
σ_1	b	σ_1	3
σ_1	c	σ_2	2
σ_1	d	σ_2	3
σ_2	a	σ_0	1
σ_2	b	σ_1	3
σ_2	c	σ_1	2
σ_2	d	σ_0	4
σ_3	a	σ_0	0
σ_3	b	σ_2	2
σ_3	c	σ_0	1
σ_3	d	σ_0	4

N Los símbolos Δ y Ω representan las funciones "estado siguiente" y de "salida", respectivamente.

Ejemplo 9.4

Retorne las seis componentes que definen la máquina de estado finito representada por:

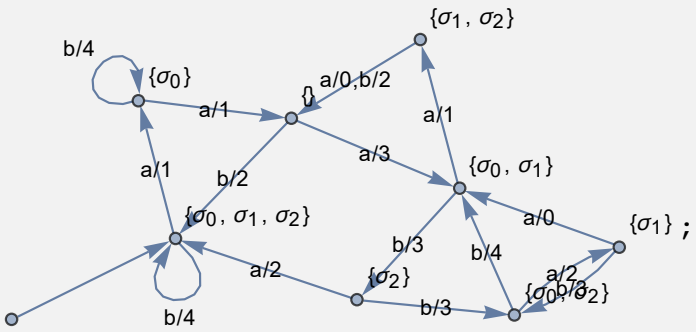


Solución:

Al emplear **`DiagramaToMaquina`**:

In[ ] :=

`Maquina =`



`DiagramaToMaquina[Maquina]`

Out[ ] :=

Conjunto de estados: {{}, {σ_0}, {σ_1}, {σ_2}, {σ_0, σ_1}, {σ_0, σ_2}, {σ_1, σ_2}, {σ_0, σ_1, σ_2}}

Símbolos de entrada: {a, b}

Símbolos de salida: {0, 1, 2, 3, 4}

Estado inicial: {σ_0, σ_1, σ_2}

		Δ	Ω
{ }	a	{σ0, σ1}	3
{ }	b	{σ0, σ1, σ2}	2
{σ0}	a	{ }	1
{σ0}	b	{σ0}	4
{σ1}	a	{σ0, σ1}	0
{σ1}	b	{σ0, σ2}	3
{σ2}	a	{σ0, σ1, σ2}	2
{σ2}	b	{σ0, σ2}	3
{σ0, σ1}	a	{σ1, σ2}	1
{σ0, σ1}	b	{σ2}	3
{σ0, σ2}	a	{σ1}	2
{σ0, σ2}	b	{σ0, σ1}	4
{σ1, σ2}	a	{ }	0
{σ1, σ2}	b	{ }	2
{σ0, σ1, σ2}	a	{σ0}	1
{σ0, σ1, σ2}	b	{σ0, σ1, σ2}	4

Explicación en video

3) **StringSalida**: **construye** la **hilera** de **símbolos de salida** que se obtiene al **procesar** una **hilera** de **símbolos de entrada** en una **máquina de estado finito**. La máquina se recibe por medio de sus **seis componentes**: el **conjunto de estados**, el **conjunto de símbolos de entrada**, el **conjunto de símbolos de salida**, el **estado inicial**, la **función estado siguiente** y la **función de salida**. Las funciones de

estado siguiente y de salida se escriben juntas, como una **matriz** con **cuatro columnas**, donde cada fila contiene en orden: el **par** "(estado, entrada)" y las **imágenes** en dicha dupla de la función estado siguiente y de salida, respectivamente. Presenta la **opción** "`trace->True`" que muestra, además, la **lista** de los **estados visitados** durante el recorrido y una **animación**.

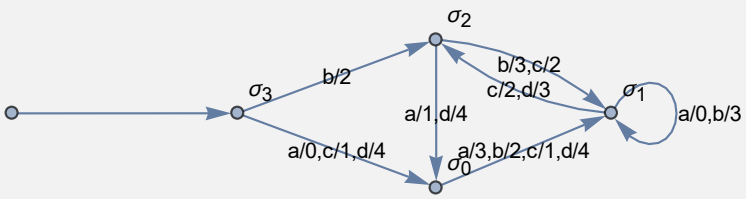
Sintaxis: `StringSalida[Estados, Entradas, Salidas, Inicio, FEstadosSalidas, string]`, o bien,

```
StringSalida[Estados, Entradas, Salidas, Inicio, FEstadosSalidas, string, trace->True]
```

con "string" un **vector** cuyas coordenadas son los **símbolos de entrada** a procesar.

Ejemplo 9.5

Sobre la máquina de estado finito:



halle el "string" de salida de una hilera seudoaleatoria de símbolos de entrada de longitud veinte. Muestre el resultado mediante una animación.

Solución:

Se creará la hilera seudoaleatoria de símbolos de entrada, utilizando el comando de *Mathematica* **`RandomChoice`**:

In[ ] :=

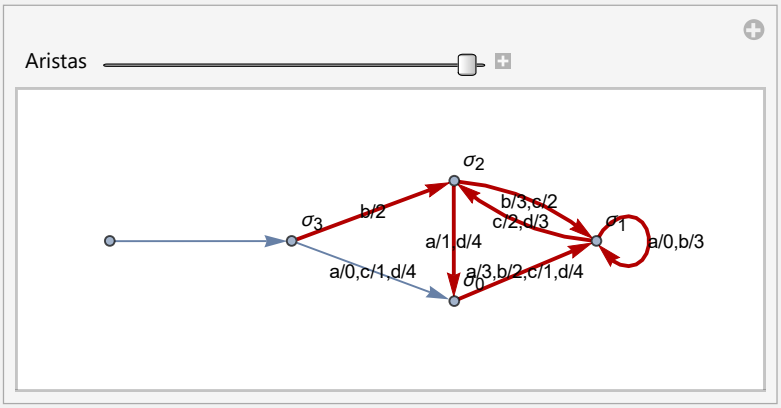
```
α = RandomChoice[{a, b, c, d}, 20]
StringSalida[{σ0, σ1, σ2, σ3}, {a, b, c, d}, {0, 1, 2, 3, 4}, σ3, {{σ0, a, σ1, 3}, {σ0, b, σ1, 2}, {σ0, c, σ1, 1}, {σ0, d, σ1, 4}, {σ1, a, σ1, 0}, {σ1, b, σ1, 3}, {σ1, c, σ2, 2}, {σ1, d, σ1, 3}, {σ2, a, σ0, 1}, {σ2, b, σ1, 3}, {σ2, c, σ1, 2}, {σ2, d, σ0, 4}, {σ3, a, σ0, 0}, {σ3, b, σ2, 2}, {σ3, c, σ0, 1}, {σ3, d, σ0, 4}}, α]
StringSalida[{σ0, σ1, σ2, σ3}, {a, b, c, d}, {0, 1, 2, 3, 4}, σ3, {{σ0, a, σ1, 3}, {σ0, b, σ1, 2}, {σ0, c, σ1, 1}, {σ0, d, σ1, 4}, {σ1, a, σ1, 0}, {σ1, b, σ1, 3}, {σ1, c, σ2, 2}, {σ1, d, σ1, 3}, {σ2, a, σ0, 1}, {σ2, b, σ1, 3}, {σ2, c, σ1, 2}, {σ2, d, σ0, 4}, {σ3, a, σ0, 0}, {σ3, b, σ2, 2}, {σ3, c, σ0, 1}, {σ3, d, σ0, 4}}, α, trace->True]
```

Out[ ] :=

{b, c, d, a, a, a, d, c, a, b, b, c, b, b, c, c, c, a, c, b}

{2, 2, 3, 1, 3, 0, 3, 2, 0, 3, 3, 2, 3, 3, 2, 2, 2, 1, 1, 3}

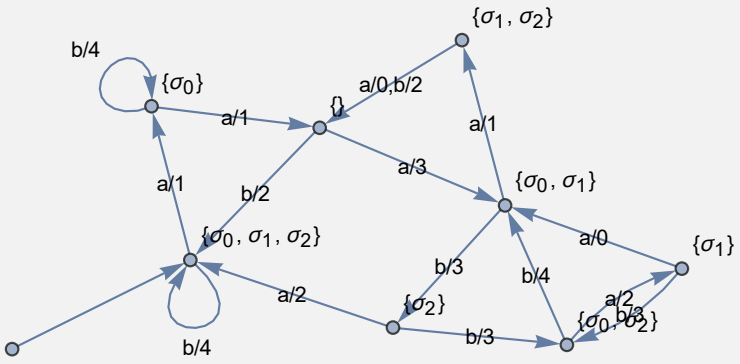
{{2, 2, 3, 1, 3, 0, 3, 2, 0, 3, 3, 2, 3, 3, 2, 2, 2, 1, 1, 3}, {σ_3, σ_2, σ_1, σ_2, σ_0, σ_1, σ_1, σ_2, σ_1, σ_1, σ_1, σ_1, σ_2, σ_1, σ_1, σ_2, σ_1, σ_2, σ_0, σ_1, σ_1}}



N **`RandomChoice`** en este ejercicio, selecciona de forma seudoaleatoria veinte elementos del conjunto **`{a, b, c, d}`**.

Ejemplo 9.6

Determine la hilera de salida de un "string" de símbolos de entrada seudoaleatorio de longitud veinte, en la máquina de estado finito:



Use la opción "**`trace -> True`**" de **`StringSalida`** para obtener el recorrido paso a paso.

Solución:

In[ ] :=

```
α = RandomChoice[{a, b}, 20]
StringSalida[{{}, {σ0}, {σ1}, {σ2}, {σ0, σ1}, {σ0, σ2}, {σ1, σ2}, {σ0, σ1,
σ2}}, {a, b}, {0, 1, 2, 3, 4}, {σ0, σ1, σ2}, {{{}, a, {σ0, σ1}, 3}, {{},
b, {σ0, σ1, σ2}, 2}, {{σ0}, a, {}, 1}, {{σ0}, b, {σ0}, 4}, {{σ1}, a, {σ0,
σ1}, 0}, {{σ1}, b, {σ0, σ2}, 3}, {{σ2}, a, {σ0, σ1, σ2}, 2}, {{σ2}, b,
{σ0, σ2}, 3}, {{σ0, σ1}, a, {σ1, σ2}, 1}, {{σ0, σ1}, b, {σ2}, 3}, {{σ0,
σ2}, a, {σ1}, 2}, {{σ0, σ2}, b, {σ0, σ1}, 4}, {{σ1, σ2}, a, {}, 0}, {{σ1,
σ2}, b, {}, 2}, {{σ0, σ1, σ2}, a, {σ0}, 1}, {{σ0, σ1, σ2}, b, {σ0, σ1,
σ2}, 4}}, α]
```

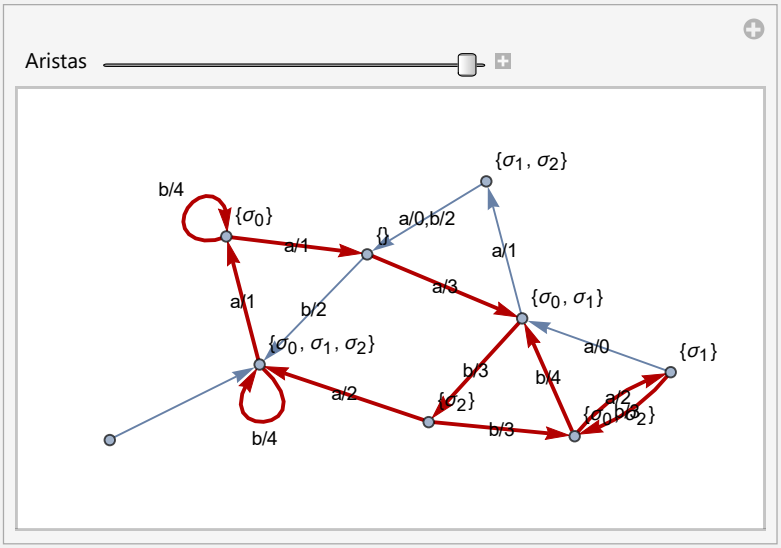
```
StringSalida[{{}, {σ0}, {σ1}, {σ2}, {σ0, σ1}, {σ0, σ2}, {σ1, σ2}, {σ0, σ1,
σ2}}, {a, b}, {0, 1, 2, 3, 4}, {σ0, σ1, σ2}, {{{}, a, {σ0, σ1}, 3}, {{},
b, {σ0, σ1, σ2}, 2}, {{σ0}, a, {}, 1}, {{σ0}, b, {σ0}, 4}, {{σ1}, a, {σ0,
σ1}, 0}, {{σ1}, b, {σ0, σ2}, 3}, {{σ2}, a, {σ0, σ1, σ2}, 2}, {{σ2}, b,
{σ0, σ2}, 3}, {{σ0, σ1}, a, {σ1, σ2}, 1}, {{σ0, σ1}, b, {σ2}, 3}, {{σ0,
σ2}, a, {σ1}, 2}, {{σ0, σ2}, b, {σ0, σ1}, 4}, {{σ1, σ2}, a, {}, 0}, {{σ1,
σ2}, b, {}, 2}, {{σ0, σ1, σ2}, a, {σ0}, 1}, {{σ0, σ1, σ2}, b, {σ0, σ1,
σ2}, 4}}, α, trace -> True]
```

Out[ ] :=

{b, a, b, b, a, a, b, b, a, b, b, b, a, a, a, a, b, b, a, b}

{4, 1, 4, 4, 1, 3, 3, 3, 2, 3, 4, 3, 2, 1, 1, 3, 3, 3, 2, 3}

{{4, 1, 4, 4, 1, 3, 3, 3, 2, 3, 4, 3, 2, 1, 1, 3, 3, 3, 2, 3}, {{σ0, σ1, σ2}, {σ0, σ1, σ2}, {σ0}, {σ0}, {σ0}, {}, {σ0, σ1}, {σ2}, {σ0, σ2}, {σ1}, {σ0, σ2}, {σ0, σ1}, {σ2}, {σ0, σ1, σ2}, {σ0}, {}, {σ0, σ1}, {σ2}, {σ0, σ2}, {σ1}, {σ0, σ2}}}



Explicación en video

4) **MaquinaSumadoraBinarios**: **muestra** el **diagrama de transición** de una máquina de estado finito cuya **función** es **sumar dos números binarios**.

Sintaxis: `MaquinaSumadoraBinarios[]`.

Ejemplo 9.7

Observe la salida de `MaquinaSumadoraBinarios[]`.

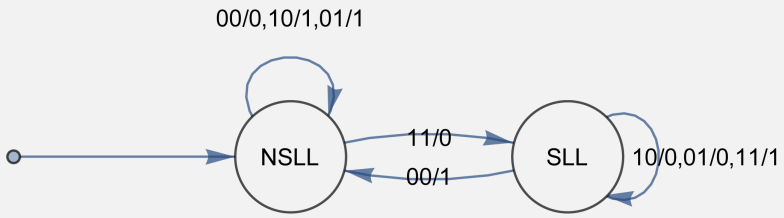
Solución:

En el software:

In[ ] :=

```
MaquinaSumadoraBinarios[]
```

Out[ ] :=



N Los nombres de los estados en esta máquina de estado finito, NSLL y SLL significan: "No se lleva" y "Se lleva", respectivamente, al pensar en los dígitos que se van sumando en la operación binaria. Si se desea recibir una explicación teórica sobre la construcción y funcionamiento de la máquina se puede consultar [25].

Ejemplo 9.8

Encuentre las seis componentes que definen a **`MaquinaSumadoraBinarios[]`**.

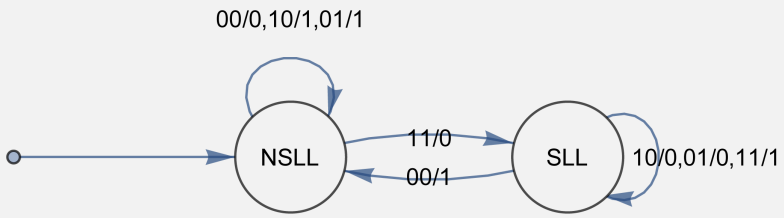
Solución:

Al combinar el uso de las instrucciones **`MaquinaSumadoraBinarios`** y **`DiagramaToMaquina`**, se resuelve lo solicitado:

In[ ] :=

```
Maquina = MaquinaSumadoraBinarios[]
DiagramaToMaquina[Maquina]
```

Out[ ] :=



Conjunto de estados: {NSLL, SLL}
Símbolos de entrada: {00, 01, 10, 11}
Símbolos de salida: {0, 1}
Estado inicial: NSLL

		Δ	Ω
NSLL	00	NSLL	0
NSLL	01	NSLL	1
NSLL	10	NSLL	1
NSLL	11	SLL	0
SLL	00	NSLL	1
SLL	01	SLL	0
SLL	10	SLL	0
SLL	11	SLL	1

Explicación en video

5) **SumaBinaria**: **realiza** la **suma** de **dos números binarios** "a" y "b" usando como **base** una máquina de estado finito. Presenta la **opción** "**steps -> True**" que **muestra** la **máquina** utilizada y el procedimiento **paso a paso**.

Sintaxis: **SumaBinaria[a, b]**, o bien, **SumaBinaria[a, b, steps -> True]**.

Ejemplo 9.9

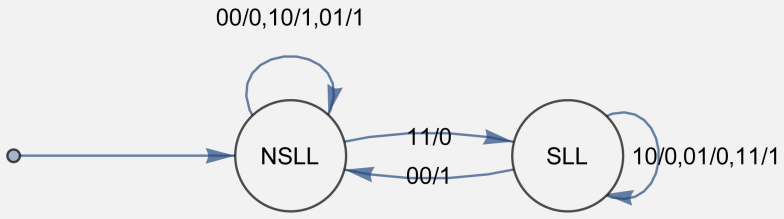
Sume mediante una máquina de estado finito, los números binarios 1101101 y 10010111. Exponga en detalle el procedimiento.

Solución:

Al recurrir a la opción "**steps -> True**":

In[ ] :=
```
SumaBinaria[1101101, 10010111, steps -> True]
```

Out[ ] :=



Hilera a procesar en la máquina (intercala los dígitos binarios): {11, 01, 11, 10, 01, 10, 10, 01}
Procesamiento: {{0, 0, 1, 0, 0, 0, 0, 0, 1}, {NSLL, SLL, SLL, SLL, SLL, SLL, SLL, SLL, SLL}}
Se toma la lista de símbolos de salida al revés: {1, 0, 0, 0, 0, 0, 1, 0, 0}
100000100

N ¡Se aclara al lector! que la suma binaria se efectúa, al construir un "string" de símbolos de entrada, tomando el primer dígito de cada binario, el segundo dígito y así sucesivamente, hasta recorrer el número de mayor longitud. Luego, se obtiene la hilera de salida y ésta escrita al revés, constituye la suma buscada.

Ejemplo 9.10

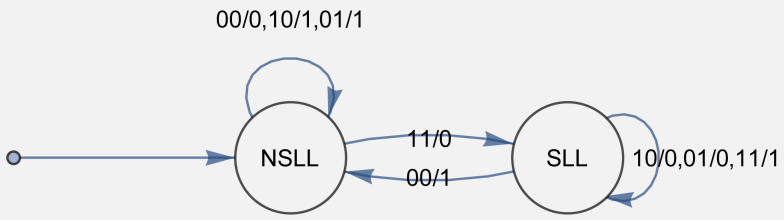
Sume los binarios 1101111111 y 10010001111011, a través de una máquina de estado finito y exhiba el procedimiento.

Solución:

In[ ] :=

```
SumaBinaria[1101111111, 10010001111011, steps -> True]
```

Out[ ] :=



Hilera a procesar en la máquina (intercala los dígitos binarios): {11, 11, 10, 11, 11, 11, 11, 00, 10, 10, 01, 00, 00, 01}
Procesamiento: {{0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1}, {NSLL, SLL, SLL, SLL, SLL, SLL, SLL, SLL, NSLL, NSLL, NSLL, NSLL, NSLL, NSLL, NSLL}}
Se toma la lista de símbolos de salida al revés: {1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0}
10011111111010

Explicación en video

6) **CDFMaquinaEF**: recibe una **máquina de estado finito** a través de las **seis componentes** que la definen: el **conjunto de estados**, el **conjunto de símbolos de entrada**, el **conjunto de símbolos de salida**, el **estado inicial**, la **función estado siguiente** y la **función de salida**. Las funciones de estado siguiente y de salida se escriben juntas, como una **matriz** con **cuatro columnas**, donde cada fila contiene en orden: el **par** "(estado, entrada)" y las **imágenes** en dicha dupla de la función estado siguiente y de salida, respectivamente. El comando muestra en una **animación** el diagrama de transición de la máquina y permite **procesar** cualquier **hilera** de **símbolos de entrada**.

Sintaxis: **`CDFMaquinaEF[Estados, Entradas, Salidas, Inicio, FEstadosSalidas, string]`**, con "FEstadosSalidas" las **funciones estado siguiente y de salida** y, "string" un **vector** cuyas coordenadas son los **símbolos de entrada** a procesar.

Ejemplo 9.11

Ejecute el comando **`CDFMaquinaEF`**, tomando la máquina de estado finito dada y una hilera de símbolos de entrada seudoaleatoria de longitud veinte.

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
- Símbolos de entrada: $\tau = \{a, b, c\}$
- Símbolos de salida: $\delta = \{0, 1, 2, 3\}$
- Estado inicial: $\sigma^* = \sigma_3$
- Función de estado siguiente y de salida: $\{\{\sigma_0, a, \sigma_1, 3\}, \{\sigma_0, b, \sigma_1, 2\}, \{\sigma_0, c, \sigma_0, 1\}, \{\sigma_1, a, \sigma_1, 0\}, \{\sigma_1, b, \sigma_1, 3\}, \{\sigma_1, c, \sigma_2, 2\}, \{\sigma_2, a, \sigma_0, 1\}, \{\sigma_2, b, \sigma_1, 3\}, \{\sigma_2, c, \sigma_1, 2\}, \{\sigma_3, a, \sigma_0, 0\}, \{\sigma_3, b, \sigma_2, 2\}, \{\sigma_3, c, \sigma_0, 1\}\}$

Solución:

In[ ] :=

```
CDFMaquinaEF[{σ0, σ1, σ2, σ3}, {a, b, c}, {0, 1, 2, 3}, σ3, {{σ0, a, σ1, 3}, {σ0, b, σ1, 2}, {σ0, c, σ0, 1}, {σ1, a, σ1, 0}, {σ1, b, σ1, 3}, {σ1, c, σ2, 2}, {σ2, a, σ0, 1}, {σ2, b, σ1, 3}, {σ2, c, σ1, 2}, {σ3, a, σ0, 0}, {σ3, b, σ2, 2}, {σ3, c, σ0, 1}}, RandomChoice[{a, b, c}, 20]]
```

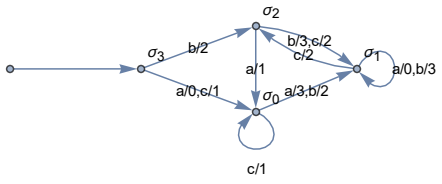
Out[ ] :=

Componentes de la máquina

Estados	$\{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
Símbolos de entrada	{a, b, c}
Símbolos de salida	{0, 1, 2, 3}
Estado inicial	σ_3
Función estado siguiente/salida	$\{\{\sigma_0, a, \sigma_1, 3\}, \{\sigma_0, b, \sigma_1, 2\}, \{\sigma_0, c, \sigma_0, 1\}, \{\sigma_1, a, \sigma_1, 0\}, \{\sigma_1, b, \sigma_1, 3\}, \{\sigma_1, c, \sigma_2, 2\}, \{\sigma_2, a, \sigma_0, 1\}, \{\sigma_2, b, \sigma_1, 3\}, \{\sigma_2, c, \sigma_1, 2\}, \{\sigma_3, a, \sigma_0, 0\}, \{\sigma_3, b, \sigma_2, 2\}, \{\sigma_3, c, \sigma_0, 1\}\}$

Hilera de símbolos de entrada

Vector de símbolos de entrada	{c, c, b, b, c, c, a, a, c, b, b, c, a, c, b, a, a, a, b, b}



Hilera de salida: {1, 1, 2, 3, 2, 2, 0, 0, 2, 3, 3, 2, 1, 1, 2, 0, 0, 0, 3, 3}

Recorrido: $\{\sigma_3, \sigma_0, \sigma_0, \sigma_1, \sigma_1, \sigma_2, \sigma_1, \sigma_1, \sigma_1, \sigma_2, \sigma_1, \sigma_1, \sigma_2, \sigma_0, \sigma_0, \sigma_1, \sigma_1, \sigma_1, \sigma_1, \sigma_1, \sigma_1\}$

Ejemplo 9.12

Utilizando **CDFMaquinaEF**, genere una animación que contenga la máquina de estado finito detallada a continuación y procese un "string" de símbolos de entrada seudoaleatorio de longitud veinte.

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
- Símbolos de entrada: $\tau = \{a, b, c, d\}$
- Símbolos de salida: $\delta = \{0, 1, 2, 3, 4\}$
- Estado inicial: $\sigma^* = \sigma_3$
- Función de estado siguiente y de salida: $\{\{\sigma_0, a, \sigma_1, 3\}, \{\sigma_0, b, \sigma_1, 2\}, \{\sigma_0, c, \sigma_1, 1\}, \{\sigma_0, d, \sigma_1, 4\}, \{\sigma_1, a, \sigma_1, 0\}, \{\sigma_1, b, \sigma_1, 3\}, \{\sigma_1, c, \sigma_2, 2\}, \{\sigma_1, d, \sigma_2, 3\}, \{\sigma_2, a, \sigma_0, 1\}, \{\sigma_2, b, \sigma_1, 3\}, \{\sigma_2, c, \sigma_1, 2\}, \{\sigma_2, d, \sigma_0, 4\}, \{\sigma_3, a, \sigma_0, 0\}, \{\sigma_3, b, \sigma_2, 2\}, \{\sigma_3, c, \sigma_0, 1\}, \{\sigma_3, d, \sigma_0, 4\}\}$

Solución:

Al ejecutar **CDFMaquinaEF** con los datos del ejemplo:

In[ ] :=

```
CDFMaquinaEF[{σ0, σ1, σ2, σ3}, {a, b, c, d}, {0, 1, 2, 3, 4}, σ3, {{σ0, a, σ1, 3}, {σ0, b, σ1, 2}, {σ0, c, σ1, 1}, {σ0, d, σ1, 4}, {σ1, a, σ1, 0}, {σ1, b, σ1, 3}, {σ1, c, σ2, 2}, {σ1, d, σ2, 3}, {σ2, a, σ0, 1}, {σ2, b, σ1, 3}, {σ2, c, σ1, 2}, {σ2, d, σ0, 4}, {σ3, a, σ0, 0}, {σ3, b, σ2, 2}, {σ3, c, σ0, 1}, {σ3, d, σ0, 4}}, RandomChoice[{a, b, c, d}, 20]]
```

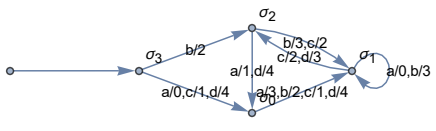
Out[ ] :=

Componentes de la máquina

Estados	$\{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
Símbolos de entrada	{a, b, c, d}
Símbolos de salida	{0, 1, 2, 3, 4}
Estado inicial	σ_3
Función estado siguiente/salida	$\{\{\sigma_0, a, \sigma_1, 3\}, \{\sigma_0, b, \sigma_1, 2\}, \{\sigma_0, c, \sigma_1, 1\}, \{\sigma_0, d, \sigma_1, 4\}, \{\sigma_1, a, \sigma_1, 0\}, \{\sigma_1, b, \sigma_1, 3\}, \{\sigma_1, c, \sigma_2, 2\}, \{\sigma_1, d, \sigma_2, 3\}, \{\sigma_2, a, \sigma_0, 1\}, \{\sigma_2, b, \sigma_1, 3\}, \{\sigma_2, c, \sigma_1, 2\}, \{\sigma_2, d, \sigma_0, 4\}, \{\sigma_3, a, \sigma_0, 0\}, \{\sigma_3, b, \sigma_2, 2\}, \{\sigma_3, c, \sigma_0, 1\}, \{\sigma_3, d, \sigma_0, 4\}\}$

Hilera de símbolos de entrada

Vector de símbolos de entrada	{d, c, c, d, c, b, c, c, b, c, b, c, c, a, d, b, c, a, d, d}




```
Hilera de salida: {4, 1, 2, 4, 1,
  3, 2, 2, 3, 2, 3, 2, 2, 0, 3, 3, 2, 1, 4, 3}
Recorrido: {σ3, σ0, σ1, σ2, σ0, σ1, σ1, σ2, σ1,
  σ1, σ2, σ1, σ2, σ1, σ1, σ2, σ1, σ2, σ0, σ1, σ2}
```

Explicación en video

7) **Automata**: **crea** un **autómata de estado finito** (**determinístico** (*DFA*) o **no determinístico** (*NDFA*)) dadas las **cinco componentes** que lo definen: el **conjunto de estados**, el **conjunto de símbolos de entrada**, el **estado inicial**, la **función estado siguiente** y el **conjunto de estados aceptados**. La función estado siguiente (o de transición de estados) se escribe como una **matriz** con **tres columnas**, donde cada fila contiene en orden: el **par** "(estado, entrada)" y la **imagen** en dicha dupla de la función estado siguiente.

Sintaxis: `Automata[Estados, Entradas, Inicio, FEstados, EstadosAceptados]`, con "Inicio" el **estado inicial** y "FEstados" la **función de transición de estados**.

Ejemplo 9.13

Construya en *Mathematica* el autómata de estado finito no determinístico, dado como sigue:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
- Símbolos de entrada: $\tau = \{a, b, c\}$
- Estado inicial: $\sigma^* = \sigma_3$
- Función de transición de estados: $\{\{\sigma_0, a, \{\sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_0, b, \{\sigma_2, \sigma_3\}\}, \{\sigma_0, c, \{\sigma_0\}\}, \{\sigma_1, a, \{\}\}, \{\sigma_1, b, \{\sigma_1, \sigma_3\}\}, \{\sigma_1, c, \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_2, a, \{\sigma_3\}\}, \{\sigma_2, b, \{\sigma_0, \sigma_2\}\}, \{\sigma_2, c, \{\sigma_2\}\}, \{\sigma_3, a, \{\sigma_3\}\}, \{\sigma_3, b, \{\sigma_0\}\}, \{\sigma_3, c, \{\sigma_0\}\}\}$
- Estados aceptados: $\{\sigma_1, \sigma_2\}$

Solución:

Al invocar el comando **`Automata`**:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3};
entradas = {a, b, c};
inicial = σ3;
trans = {{σ0, a, {σ1, σ2, σ3}}, {σ0, b, {σ2, σ3}}, {σ0, c, {σ0}}, {σ1, a,
{}}, {σ1, b, {σ1, σ3}}, {σ1, c, {σ0, σ1, σ2, σ3}}, {σ2, a, {σ3}}, {σ2, b,
{σ0, σ2}}, {σ2, c, {σ2}}, {σ3, a, {σ3}}, {σ3, b, {σ0}}, {σ3, c, {σ0}}};
aceptados = {σ1, σ2};
Automata[estados, entradas, inicial, trans, aceptados]
```

Out[ ] :=

- Automaton -

> **N** El mensaje de salida - Automaton - indica en el software, la creación correcta del autómata.

Ejemplo 9.14

A través del uso de software, construya el siguiente autómata de estado finito:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$
- Símbolos de entrada: $\tau = \{a, b, c\}$
- Estado inicial: $\sigma^* = \sigma_3$
- Función de transición de estados: $\{\{\sigma_0, a, \{\sigma_2, \sigma_3\}\}, \{\sigma_0, b, \{\sigma_1\}\}, \{\sigma_0, c, \{\sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_1, a, \{\sigma_2, \sigma_3, \sigma_4\}\}, \{\sigma_1, b, \{\sigma_1\}\}, \{\sigma_1, c, \{\}\}, \{\sigma_2, a, \{\sigma_4\}\}, \{\sigma_2, b, \{\sigma_1, \sigma_2\}\}, \{\sigma_2, c, \{\sigma_0, \sigma_1, \sigma_2\}\}, \{\sigma_3, a, \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_3, b, \{\sigma_3\}\}, \{\sigma_3, c, \{\sigma_4\}\}, \{\sigma_4, a, \{\sigma_0, \sigma_3\}\}, \{\sigma_4, b, \{\sigma_2\}\}, \{\sigma_4, c, \{\}\}\}$
- Estados aceptados: $\{\sigma_1, \sigma_2\}$

Solución:

En *Mathematica*:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3, σ4};
entradas = {a, b, c};
inicial = σ3;
trans = {{σ0, a, {σ2, σ3}}, {σ0, b, {σ1}}, {σ0, c, {σ1, σ2, σ3}}, {σ1,
a, {σ2, σ3, σ4}}, {σ1, b, {σ1}}, {σ1, c, {}}, {σ2, a, {σ4}}, {σ2, b, {σ1,
σ2}}, {σ2, c, {σ0, σ1, σ2}}, {σ3, a, {σ0, σ1, σ2, σ3}}, {σ3, b, {σ3}}, {σ3,
c, {σ4}}, {σ4, a, {σ0, σ3}}, {σ4, b, {σ2}}, {σ4, c, {}}};
aceptados = {σ1, σ2};
Automata[estados, entradas, inicial, trans, aceptados]
```

Out[ ] :=

- Automaton -

Explicación en video

8) **AutomataToDiagrama**: **genera** el **diagrama de transición** de un **autómata de estado finito** "A" recibido como parámetro (**determinístico** (*DFA*) o **no determinístico** (*NDFA*)). Brinda las **opciones**: "**`forma -> string`**" con "string" igual a "circular" o "rectangular" que especifica la **forma** de **colocación** de los estados, "**`colores -> True`**" que añade **color** a la representación y "**`reducido -> True`**"

que muestra el diagrama únicamente con los **estados** por los que es **posible pasar**.

Sintaxis: **`AutomataToDiagrama[A]`**, o bien,

```
AutomataToDiagrama[A, forma->string, colores->True, reducido->True]
```

pudiendo **prescindir** de cualquiera de las opciones.

Ejemplo 9.15

Realice con apoyo de software, el diagrama de transición del autómata de estado finito:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
- Símbolos de entrada: $\tau = \{a, b, c\}$
- Estado inicial: $\sigma^* = \sigma_3$
- Función de transición de estados: $\{\{\sigma_0, a, \{\sigma_1, \sigma_3\}\}, \{\sigma_0, b, \{\sigma_1, \sigma_3\}\}, \{\sigma_0, c, \{\sigma_0\}\}, \{\sigma_1, a, \{\}\}, \{\sigma_1, b, \{\sigma_1, \sigma_3\}\}, \{\sigma_1, c, \{\sigma_0, \sigma_1, \sigma_3\}\}, \{\sigma_2, a, \{\sigma_3\}\}, \{\sigma_2, b, \{\sigma_0, \sigma_2\}\}, \{\sigma_2, c, \{\sigma_2\}\}, \{\sigma_3, a, \{\sigma_3\}\}, \{\sigma_3, b, \{\sigma_0\}\}, \{\sigma_3, c, \{\sigma_0\}\}\}$
- Estados aceptados: $\{\sigma_1, \sigma_2\}$

Devuelva además, el diagrama reducido con una disposición rectangular en los nodos y con colores.

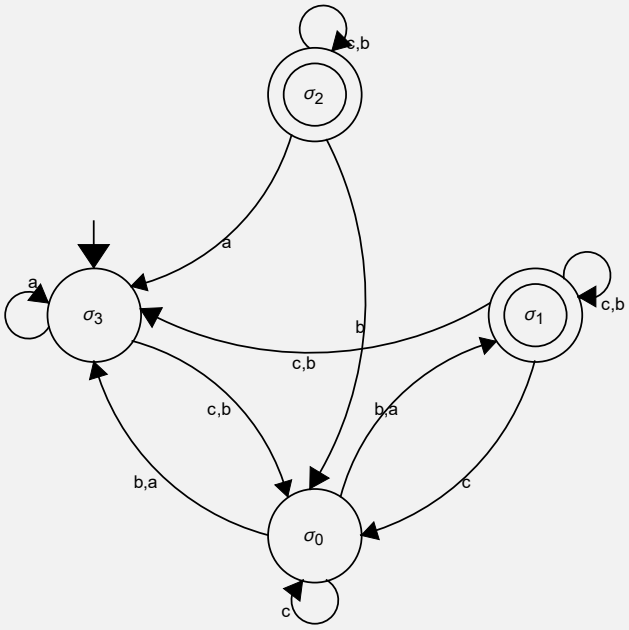
Solución:

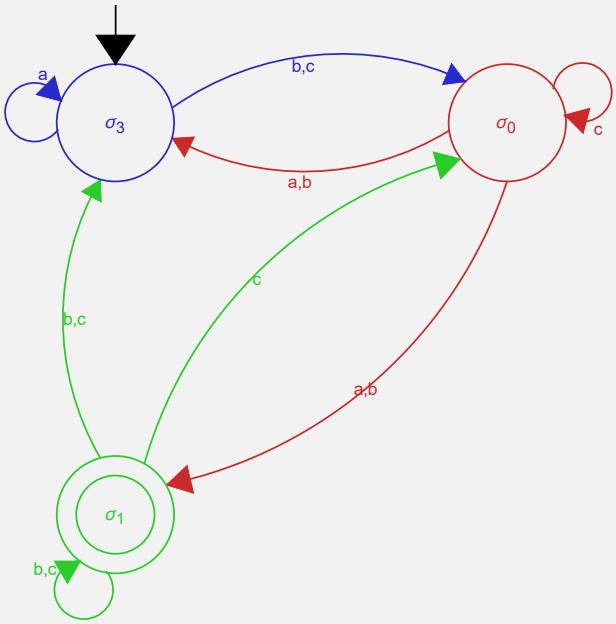
Al emplear el comando **`AutomataToDiagrama`**:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3};
entradas = {a, b, c};
inicial = σ3;
trans = {{σ0, a, {σ1, σ3}}, {σ0, b, {σ1, σ3}}, {σ0, c, {σ0}}, {σ1, a, {}}, {σ1, b, {σ1, σ3}}, {σ1, c, {σ0, σ1, σ3}}, {σ2, a, {σ3}}, {σ2, b, {σ0, σ2}}, {σ2, c, {σ2}}, {σ3, a, {σ3}}, {σ3, b, {σ0}}, {σ3, c, {σ0}}};
aceptados = {σ1, σ2};
automata = Automata[estados, entradas, inicial, trans, aceptados];
AutomataToDiagrama[automata]
AutomataToDiagrama[automata, forma -> "rectangular", colores -> True, reducido -> True]
```

Out[ ] :=





N El diagrama reducido del autómata, excluye aquellos estados por los que es imposible pasar, partiendo del estado inicial correspondiente. En este ejercicio, se ha eliminado el estado σ_2 y sus aristas salientes de la segunda gráfica, por las razones anteriormente citadas.

Ejemplo 9.16

Considerando el siguiente autómata de estado finito:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$
- Símbolos de entrada: $\tau = \{a, b, c\}$
- Estado inicial: $\sigma^* = \sigma_2$
- Función de transición de estados: $\{\{\sigma_0, a, \{\sigma_2, \sigma_4\}\}, \{\sigma_0, b, \{\sigma_4\}\}, \{\sigma_0, c, \{\sigma_0, \sigma_2, \sigma_4\}\}, \{\sigma_1, a, \{\sigma_2, \sigma_3, \sigma_4\}\}, \{\sigma_1, b, \{\sigma_1\}\}, \{\sigma_1, c, \{\}\}, \{\sigma_2, a, \{\sigma_4\}\}, \{\sigma_2, b, \{\sigma_4, \sigma_2\}\}, \{\sigma_2, c, \{\sigma_0, \sigma_2, \sigma_4\}\}, \{\sigma_3, a, \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_3, b, \{\sigma_3\}\}, \{\sigma_3, c, \{\sigma_4\}\}, \{\sigma_4, a, \{\sigma_0, \sigma_2\}\}, \{\sigma_4, b, \{\sigma_2\}\}, \{\sigma_4, c, \{\}\}\}$
- Estados aceptados: $\{\sigma_1, \sigma_2\}$

Retorne su diagrama de transición con colores y muestre el diagrama reducido.

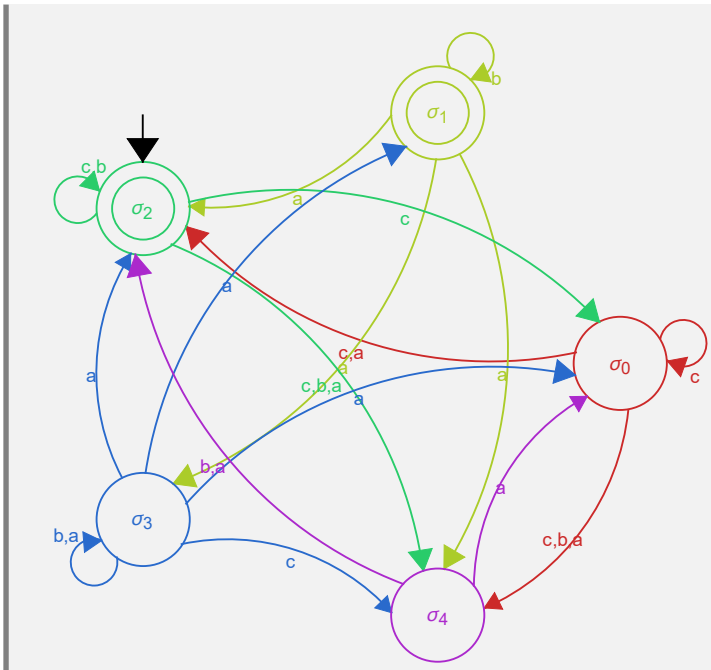
Solución:

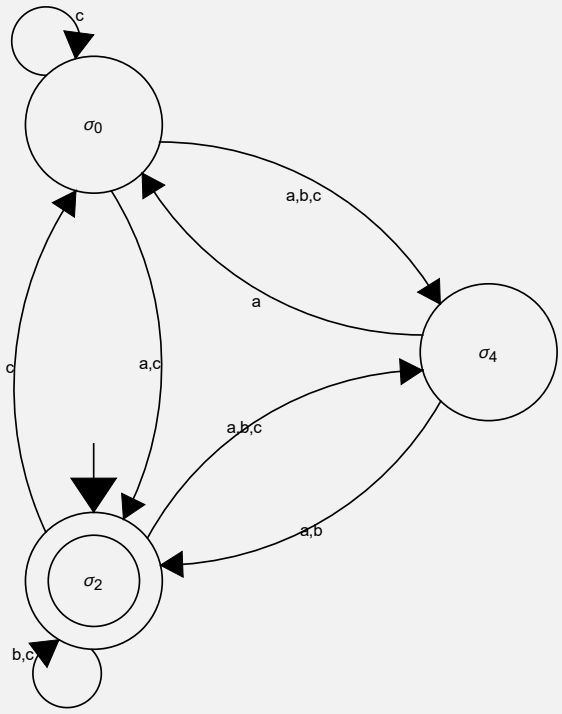
En el software:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3, σ4};
entradas = {a, b, c};
inicial = σ2;
trans = {{σ0, a, {σ2, σ4}}, {σ0, b, {σ4}}, {σ0, c, {σ0, σ2, σ4}}, {σ1,
a, {σ2, σ3, σ4}}, {σ1, b, {σ1}}, {σ1, c, {}}, {σ2, a, {σ4}}, {σ2, b, {σ4,
σ2}}, {σ2, c, {σ0, σ2, σ4}}, {σ3, a, {σ0, σ1, σ2, σ3}}, {σ3, b, {σ3}}, {σ3,
c, {σ4}}, {σ4, a, {σ0, σ2}}, {σ4, b, {σ2}}, {σ4, c, {}}};
aceptados = {σ1, σ2};
automata = Automata[estados, entradas, inicial, trans, aceptados];
AutomataToDiagrama[automata, colores -> True]
AutomataToDiagrama[automata, reducido -> True]
```

Out[ ] :=





Explicación en video

9) **AutomataQ**: retorna "`True`" si el argumento "A" recibido como parámetro es un **autómata de estado finito** (**determinístico** (*DFA*) o **no determinístico** (*NDFA*)) y "`False`", en caso contrario.

Sintaxis: `AutomataQ[A]`.

Ejemplo 9.17

Verifique el valor de verdad "`True`", al emplear la instrucción **`AutomataQ`** sobre el autómata:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
- Símbolos de entrada: $\tau = \{a, b, c\}$
- Estado inicial: $\sigma^* = \sigma_3$
- Función de transición de estados: $\{\{\sigma_0, a, \{\sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_0, b, \{\sigma_2, \sigma_3\}\}, \{\sigma_0, c, \{\sigma_0\}\}, \{\sigma_1, a, \{\}\}, \{\sigma_1, b, \{\sigma_1, \sigma_3\}\}, \{\sigma_1, c, \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_2, a, \{\sigma_3\}\}, \{\sigma_2, b, \{\sigma_0, \sigma_2\}\}, \{\sigma_2, c, \{\sigma_2\}\}, \{\sigma_3, a, \{\sigma_3\}\}, \{\sigma_3, b, \{\sigma_0\}\}, \{\sigma_3, c, \{\sigma_0\}\}\}$
- Estados aceptados: $\{\sigma_1, \sigma_2\}$

Solución:

En *Mathematica*:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3};
entradas = {a, b, c};
inicial = σ3;
trans = {{σ0, a, {σ1, σ2, σ3}}, {σ0, b, {σ2, σ3}}, {σ0, c, {σ0}}, {σ1, a, {}}, {σ1, b, {σ1, σ3}}, {σ1, c, {σ0, σ1, σ2, σ3}}, {σ2, a, {σ3}}, {σ2, b, {σ0, σ2}}, {σ2, c, {σ2}}, {σ3, a, {σ3}}, {σ3, b, {σ0}}, {σ3, c, {σ0}}};
aceptados = {σ1, σ2};
automata = Automata[estados, entradas, inicial, trans, aceptados];
AutomataQ[automata]
```

Out[ ] :=

True

Ejemplo 9.18

Ejecute la asignación **`automata = Automata[estados, entradas, inicial, trans, aceptados]`**, con:

- estados = $\{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$
- entradas = $\{a, b, c\}$
- inicial = σ_3
- trans = $\{\{\sigma_0, a, \{\sigma_2, \sigma_3\}\}, \{\sigma_0, b, \{\sigma_1\}\}, \{\sigma_0, c, \{\sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_1, a, \{\sigma_2, \sigma_3, \sigma_4\}\}, \{\sigma_1, b, \{\sigma_1\}\}, \{\sigma_1, c, \{\{\}\}\}, \{\sigma_2, a, \{\sigma_4\}\}, \{\sigma_2, b, \{\sigma_1, \sigma_2\}\}, \{\sigma_2, c, \{\sigma_0, \sigma_1, \sigma_2\}\}, \{\sigma_3, a, \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_3, b, \{\sigma_3\}\}, \{\sigma_3, c, \{\sigma_4\}\}, \{\sigma_4, a, \{\sigma_0, \sigma_3\}\}, \{\sigma_4, b, \{\sigma_2\}\}, \{\sigma_4, c, \{\}\}\}$
- aceptados = $\{\sigma_1, \sigma_2\}$

Compruebe que el comando **AutomataQ** retorna "**False**", ¿cuál es el motivo de este resultado?

Solución:

En el software:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3, σ4};
entradas = {a, b, c};
inicial = σ3;
trans = {{σ0, a, {σ2, σ3}}, {σ0, b, {σ1}}, {σ0, c, {σ1, σ2, σ3}}, {σ1, a,
{σ2, σ3, σ4}}, {σ1, b, {σ1}}, {σ1, c, {{}}}, {σ2, a, {σ4}}, {σ2, b, {σ1,
σ2}}, {σ2, c, {σ0, σ1, σ2}}, {σ3, a, {σ0, σ1, σ2, σ3}}, {σ3, b, {σ3}}, {σ3,
c, {σ4}}, {σ4, a, {σ0, σ3}}, {σ4, b, {σ2}}, {σ4, c, {}}};
aceptados = {σ1, σ2};
automata = Automata[estados, entradas, inicial, trans, aceptados];
AutomataQ[automata]
```

Out[ ] :=

False

N El contenido de la variable **automata** no es un autómata de estado finito, debido a que en la función de estado siguiente existe una inconsistencia, particularmente en: **{σ1, c, {{}}}**. De acuerdo con esto, la imagen del par (σ_1, c) es igual a $\{\emptyset\}$ y $\emptyset$ no es un estado, esto ocasiona en *Mathematica* que no se pueda crear el autómata, quedando almacenado en **automata** un valor nulo, de allí la respuesta mostrada en el *Out*[] igual a False.

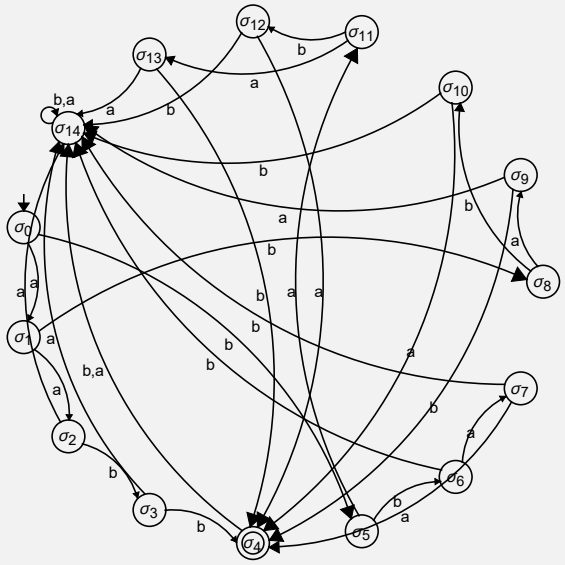
Explicación en video

10) **LenguajeFinitoQ**: retorna "**True**" si el argumento "A" recibido como parámetro es un **autómata determinístico** (*DFA*) que **acepta** un **número finito** de "strings" y "**False**", en caso contrario.

Sintaxis: **LenguajeFinitoQ[A]**.

Ejemplo 9.19

Sea el autómata:



¿Posee un lenguaje finito?

Solución:

Al recurrir a la instrucción **LenguajeFinitoQ** y crear el autómata en el programa, se tiene:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3, σ4, σ5, σ6, σ7, σ8, σ9, σ10, σ11, σ12, σ13, σ14};
entradas = {a, b};
inicial = σ0;
trans = {{σ0, a, σ1}, {σ1, a, σ2}, {σ2, b, σ3}, {σ2, a, σ14}, {σ3, b, σ4},
{σ3, a, σ14}, {σ0, b, σ5}, {σ5, b, σ6}, {σ6, a, σ7}, {σ6, b, σ14}, {σ7,
a, σ4}, {σ7, b, σ14}, {σ1, b, σ8}, {σ8, a, σ9}, {σ9, b, σ4}, {σ9, a, σ14},
{σ8, b, σ10}, {σ10, a, σ4}, {σ10, b, σ14}, {σ5, a, σ11}, {σ11, b, σ12}, {σ12,
a, σ4}, {σ12, b, σ14}, {σ11, a, σ13}, {σ13, b, σ4}, {σ13, a, σ14}, {σ4, a,
σ14}, {σ4, b, σ14}, {σ14, a, σ14}, {σ14, b, σ14}};
aceptados = {σ4};
automata = Automata[estados, entradas, inicial, trans, aceptados]
LenguajeFinitoQ[automata]
```

Out[ ] :=

```
- Automaton -
True
```

N El valor lógico de la salida es True, pues el lenguaje del autómata corresponde al conjunto finito: {{a, a, b, b}, {a, b, a, b}, {a, b, b, a}, {b, a, a, b}, {b, a, b, a}, {b, b, a, a}}.

Ejemplo 9.20

Determine si el siguiente autómata determinístico contiene un lenguaje finito:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
- Símbolos de entrada: $\tau = \{a, b, c\}$
- Estado inicial: $\sigma^* = \sigma_2$
- Función de transición de estados: $\{\{\sigma_0, a, \sigma_3\}, \{\sigma_0, b, \sigma_0\}, \{\sigma_0, c, \sigma_1\}, \{\sigma_1, a, \sigma_0\}, \{\sigma_1, b, \sigma_3\}, \{\sigma_1, c, \sigma_2\}, \{\sigma_2, a, \sigma_1\}, \{\sigma_2, b, \sigma_1\}, \{\sigma_2, c, \sigma_3\}, \{\sigma_3, a, \sigma_0\}, \{\sigma_3, b, \sigma_0\}, \{\sigma_3, c, \sigma_3\}\}$
- Estados aceptados: $\{\sigma_0, \sigma_2\}$

Muestre además, su diagrama de transición utilizando colores.

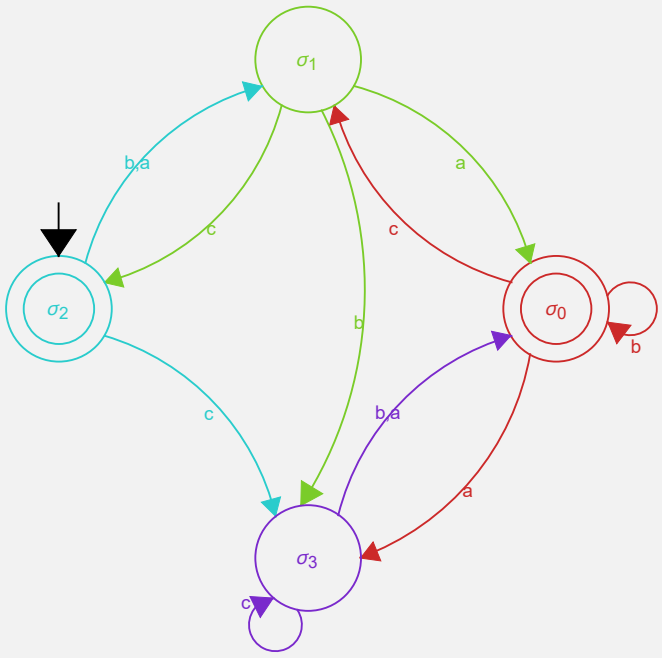
Solución:

En *Mathematica*:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3};
entradas = {a, b, c};
inicial = σ2;
trans = {{σ0, a, σ3}, {σ0, b, σ0}, {σ0, c, σ1}, {σ1, a, σ0}, {σ1, b, σ3},
{σ1, c, σ2}, {σ2, a, σ1}, {σ2, b, σ1}, {σ2, c, σ3}, {σ3, a, σ0}, {σ3, b,
σ0}, {σ3, c, σ3}};
aceptados = {σ0, σ2};
automata = Automata[estados, entradas, inicial, trans, aceptados];
AutomataToDiagrama[automata, colores -> True]
LenguajeFinitoQ[automata]
```

Out[ ] :=



False

Se concluye que el conjunto de hileras aceptadas por el autómata no es finito.

Explicación en video

11) **AutomataRandom**: **construye** de forma **seudoaleatoria** un **autómata de estado finito determinístico** con "n" **estados** y "m" **símbolos de entrada**. Por defecto, los **estados** y **símbolos de entrada** son **números naturales consecutivos** iniciando en **uno** y el autómata creado se almacena en una **variable denominada** "G".

Sintaxis: `AutomataRandom[n, m]`, con "n" y "m" **mayores o iguales** que **uno**. Presenta la **opción** "`colores -> True`" que añade **color** a la representación.

Ejemplo 9.21

Construya el diagrama de transición de un autómata de estado finito seudoaleatorio con cinco estados y tres símbolos de entrada, empleando la opción "`colores -> True`".

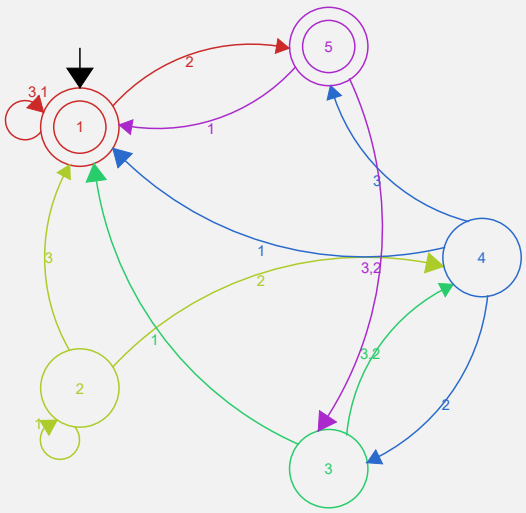
Solución:

Al usar el comando **`AutomataRandom`**:

In[ ] :=

```
AutomataRandom[5, 3, colores -> True]
```

Out[ ] :=



N La salida es seudoaleatoria, por lo que se genera cada vez, al correr el $In[]$, un autómata distinto 5×3.

Ejemplo 9.22

Geste haciendo uso de **`AutomataRandom`**, un autómata de estado finito determinístico de orden 6×7, añadiendo colores al diagrama.

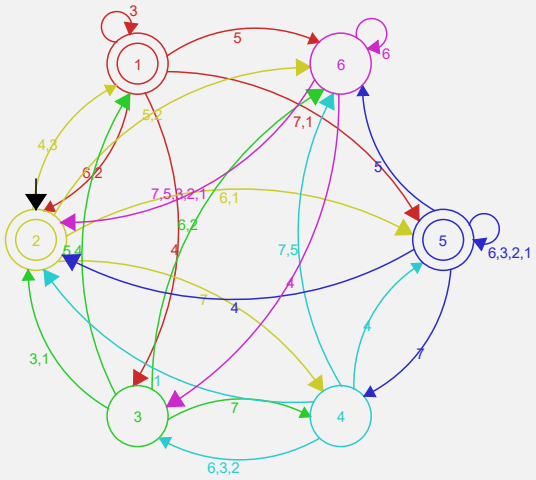
Solución:

En el software:

In[ ] :=

```
AutomataRandom[6, 7, colores -> True]
```

Out[ ] :=



Explicación en video

12) **LenguajeCantidad**: **devuelve** el **número** de **cadenas** de símbolos de entrada de **longitud** "n" que son **aceptadas** por un **autómata de estado finito** "A" recibido como parámetro (**determinístico** (*DFA*) o **no determinístico** (*NDFA*)). La instrucción provee la **opción** "`limite -> True`" que retorna la **cantidad** de **elementos** del **lenguaje del autómata**, de longitud **menor o igual** a "n".

Sintaxis: `LenguajeCantidad[A, n]`, o bien, `LenguajeCantidad[A, n, limite -> True]`.

Ejemplo 9.23

Considere el siguiente autómata de estado finito:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
- Símbolos de entrada: $\tau = \{a, b, c\}$
- Estado inicial: $\sigma^* = \sigma_2$
- Función de transición de estados: $\{\{\sigma_0, a, \sigma_3\}, \{\sigma_0, b, \sigma_0\}, \{\sigma_0, c, \sigma_1\}, \{\sigma_1, a, \sigma_0\}, \{\sigma_1, b, \sigma_3\}, \{\sigma_1, c, \sigma_2\}, \{\sigma_2, a, \sigma_1\}, \{\sigma_2, b, \sigma_1\}, \{\sigma_2, c, \sigma_3\}, \{\sigma_3, a, \sigma_0\}, \{\sigma_3, b, \sigma_0\}, \{\sigma_3, c, \sigma_3\}\}$
- Estados aceptados: $\{\sigma_0, \sigma_2\}$

Presente el diagrama de transición del autómata. Encuentre la cantidad de hileras aceptadas de longitud igual a diez y menor o igual a diez.

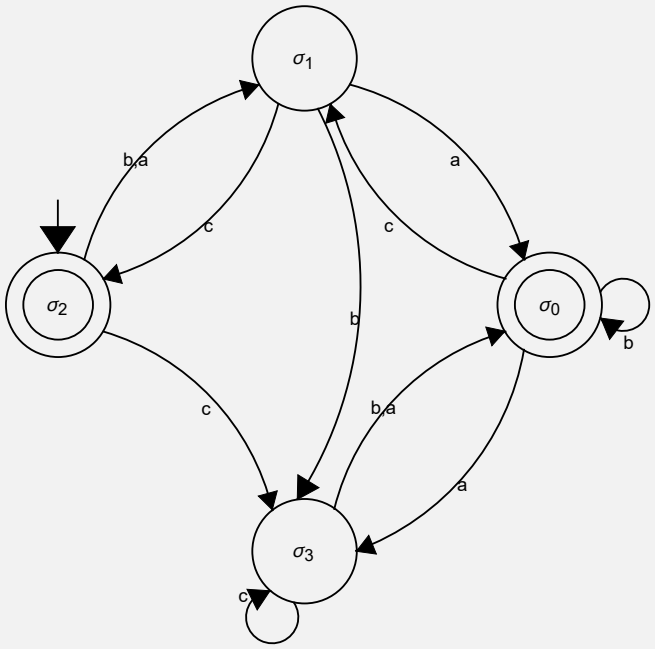
Solución:

Las instrucciones **Automata**, **AutomataToDiagrama** y **LenguajeCantidad** permiten resolver este ejemplo:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3};
entradas = {a, b, c};
inicial = σ2;
trans = {{σ0, a, σ3}, {σ0, b, σ0}, {σ0, c, σ1}, {σ1, a, σ0}, {σ1, b, σ3},
{σ1, c, σ2}, {σ2, a, σ1}, {σ2, b, σ1}, {σ2, c, σ3}, {σ3, a, σ0}, {σ3, b,
σ0}, {σ3, c, σ3}};
aceptados = {σ0, σ2};
automata = Automata[estados, entradas, inicial, trans, aceptados];
AutomataToDiagrama[automata]
LenguajeCantidad[automata, 10]
LenguajeCantidad[automata, 10, limite->True]
```

Out[ ] :=



28716
42997

Por lo tanto, el autómata acepta 28716 hileras de símbolos de entrada de tamaño diez y 42997 "strings" de longitud menor o igual a diez.

Ejemplo 9.24

Halle la cantidad de hileras de símbolos de entrada de longitud igual a cinco y menor o igual a cinco, aceptadas sobre un autómata de estado finito determinístico, obtenido al ejecutar **`AutomataRandom[6, 7, colores->True]`**.

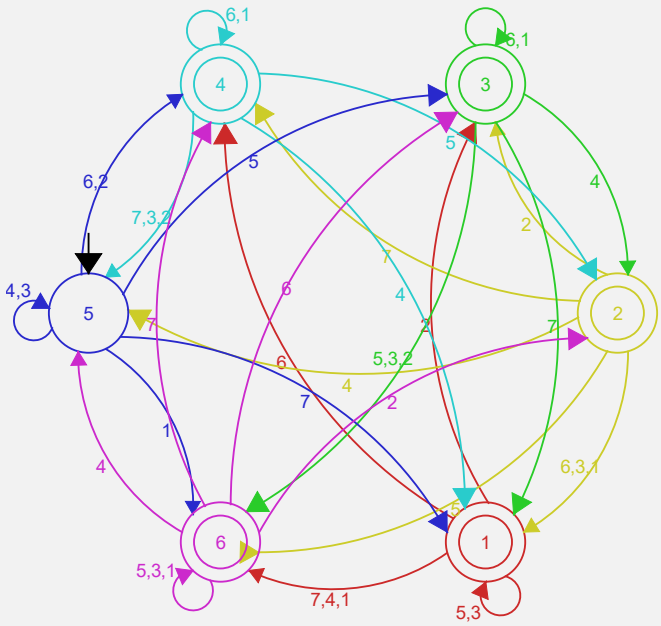
Solución:

En *Mathematica*:

In[ ] :=

```
AutomataRandom[6, 7, colores->True]
LenguajeCantidad[G, 5]
LenguajeCantidad[G, 5, limite->True]
```

Out[ ] :=



13795
16075

> **N** Se recuerda al alumno que la invocar **`AutomataRandom`**, el autómata queda almacenado automáticamente en una variable llamada "G". De allí, que se utilice **`G`** dentro de la instrucción **`LenguajeCantidad`**.

Explicación en video

13) **LenguajeStrings**: **devuelve** las **cadenas** de símbolos de entrada de **longitud** "n" que son **aceptadas** por un **autómata de estado finito** "A" recibido como parámetro (**determinístico** (*DFA*) o **no determinístico** (*NDFA*)). La instrucción provee la **opción** "**`limite -> True`**" que retorna los **elementos** del **lenguaje del autómata** de longitud **menor o igual** a "n".

Sintaxis: **`LenguajeStrings[A, n]`**, o bien, **`LenguajeStrings[A, n, limite -> True]`**.

Ejemplo 9.25

Determine las palabras del lenguaje del autómata de estado finito dado a continuación, de longitud diez y menor o igual a diez. Muestre su diagrama de transición.

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
- Símbolos de entrada: $\tau = \{a, b, c\}$

- Estado inicial: $\sigma^* = \sigma_3$
- Función de transición de estados: {{σ_0, a, {σ_1, σ_2, σ_3}}, {σ_0, b, {σ_2, σ_3}}, {σ_0, c, {σ_0}}, {σ_1, a, {}}, {σ_1, b, {σ_1, σ_3}}, {σ_1, c, {σ_0, σ_1, σ_2, σ_3}}, {σ_2, a, {σ_3}}, {σ_2, b, {σ_0, σ_2}}, {σ_2, c, {σ_2}}, {σ_3, a, {σ_3}}, {σ_3, b, {σ_0}}, {σ_3, c, {σ_0}}}
- Estados aceptados: {σ_1, σ_2}

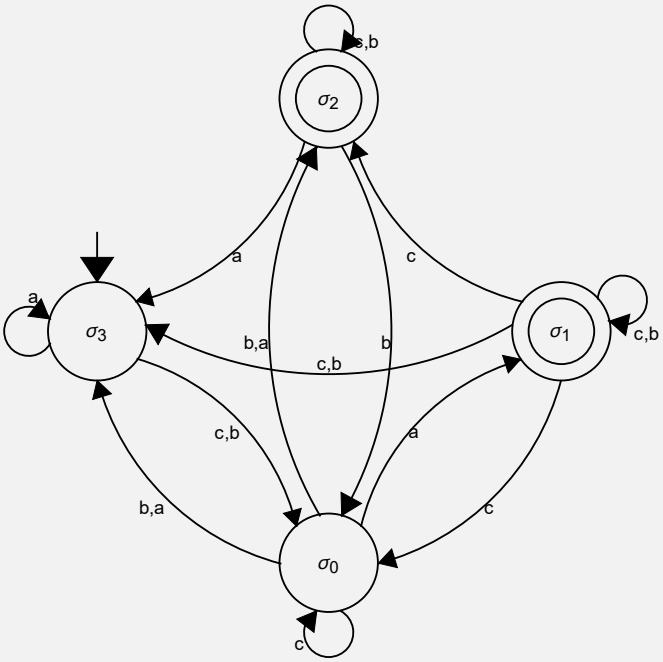
Solución:

Al utilizar **`LenguajeStrings`**:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3};
entradas = {a, b, c};
inicial = σ3;
trans = {{σ0, a, {σ1, σ2, σ3}}, {σ0, b, {σ2, σ3}}, {σ0, c, {σ0}}, {σ1, a, {}}, {σ1, b, {σ1, σ3}}, {σ1, c, {σ0, σ1, σ2, σ3}}, {σ2, a, {σ3}}, {σ2, b, {σ0, σ2}}, {σ2, c, {σ2}}, {σ3, a, {σ3}}, {σ3, b, {σ0}}, {σ3, c, {σ0}}};
aceptados = {σ1, σ2};
automata = Automata[estados, entradas, inicial, trans, aceptados];
AutomataToDiagrama[automata]
LenguajeStrings[automata, 10]
LenguajeStrings[automata, 10, limite -> True]
```

Out[ ] :=



{{a, a, a, a, a, a, a, a, b, a}, {a, a, a, a, a, a, a, a, b, b},
{a, a, a, a, a, a, a, a, c, a}, {a, a, a, a, a, a, a, a, c, b}, {a, a, a, a, a, a, a, b, a, b},
··· 44 082 ···, {c, c, c, c, c, c, c, c, a, c}, {c, c, c, c, c, c, c, c, b, b},
{c, c, c, c, c, c, c, c, b, c}, {c, c, c, c, c, c, c, c, c, a}, {c, c, c, c, c, c, c, c, c, b}}

salida grande | Mostrar menos | Mostrar más | Mostrar salida completa | Establecer límite de tamaño...

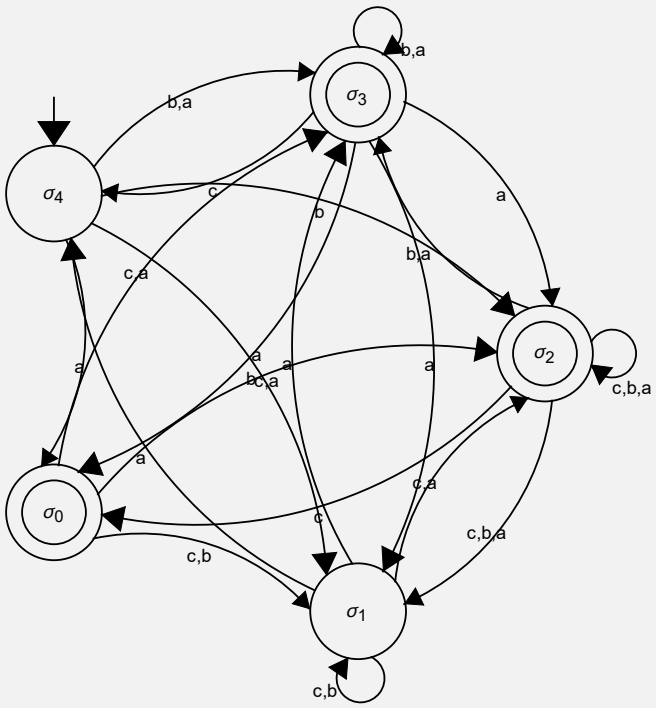
```
{{b, a}, {b, b}, {c, a}, {c, b}, {a, b, a}, {a, b, b}, {a, c, a},
 {a, c, b}, ··· 65 941 ···, {c, c, c, c, c, c, c, b, c, c}, {c, c, c, c, c, c, c, c, a, b},
 {c, c, c, c, c, c, c, c, a, c}, {c, c, c, c, c, c, c, c, b, b},
 {c, c, c, c, c, c, c, c, b, c}, {c, c, c, c, c, c, c, c, c, a}, {c, c, c, c, c, c, c, c, c, b}}
```

salida grande | **Mostrar menos** | **Mostrar más** | **Mostrar salida completa** | **Establecer límite de tamaño...**

La salida no aparece de forma completa por sus dimensiones.

Ejemplo 9.26

Dado el autómata:



Halle las hileras de símbolos de entrada aceptadas, de longitud igual a diez.

Solución:

En el software se crea primero el autómata y posteriormente se usa el comando **LenguajeStrings**:

```
In[ ] :=
estados = {σ0, σ1, σ2, σ3, σ4};
entradas = {a, b, c};
inicial = σ4;
trans = {{σ0, a, {σ2, σ3}}, {σ0, b, {σ1}}, {σ0, c, {σ1, σ2, σ3}}, {σ1, a,
{σ2, σ3, σ4}}, {σ1, b, {σ1}}, {σ1, c, {σ1, σ2}}, {σ2, a, {σ1, σ2, σ3}},
{σ2, b, {σ1, σ2, σ3}}, {σ2, c, {σ0, σ1, σ2}}, {σ3, a, {σ0, σ1, σ2, σ3}},
{σ3, b, {σ3}}, {σ3, c, {σ4}}, {σ4, a, {σ0, σ3}}, {σ4, b, {σ1, σ2, σ3}},
{σ4, c, {}}};
```

```
aceptados = {σ0, σ2, σ3};
automata = Automata[estados, entradas, inicial, trans, aceptados];
LenguajeStrings[automata, 10]
```

Out[ ] :=

```
{{a, a, a, a, a, a, a, a, a, a}, {a, a, a, a, a, a, a, a, a, b},
 {a, a, a, a, a, a, a, a, a, c}, {a, a, a, a, a, a, a, a, b, a}, {a, a, a, a, a, a, a, a, b, b},
 ··· 39 356 ···, {b, c, c, c, c, c, c, c, b, b}, {b, c, c, c, c, c, c, c, b, c},
 {b, c, c, c, c, c, c, c, c, a}, {b, c, c, c, c, c, c, c, c, b}, {b, c, c, c, c, c, c, c, c, c}}
```

salida grande | Mostrar menos | Mostrar más | Mostrar salida completa | Establecer límite de tamaño...

Como ya se ha mencionado en otros ejemplos, si el estudiante desea visualizar la salida completa, puede presionar el botón "Mostrar salida completa".

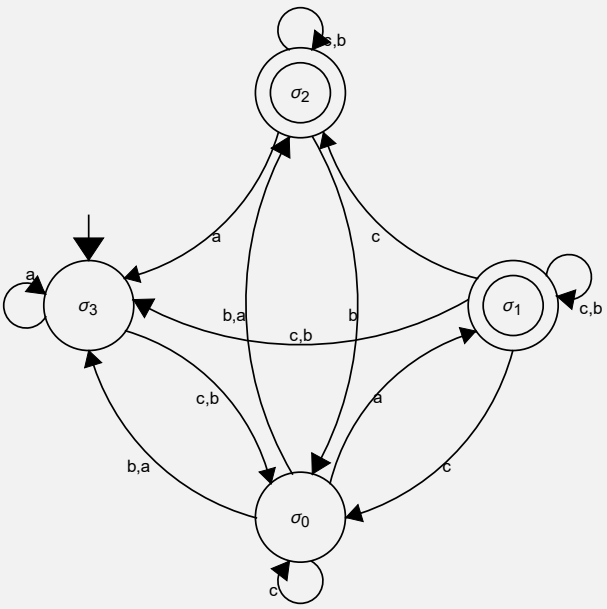
Explicación en video

14) **ComponentesAutomata**: **retorna** las **cinco componentes** que definen a un **autómata de estado finito** "A" recibido como parámetro (**determinístico** (*DFA*) o **no determinístico** (*NDFA*)).

Sintaxis: `ComponentesAutomata[A]`.

Ejemplo 9.27

Determine las cinco componentes que caracterizan el autómata de estado finito:



Solución:

Al emplear el comando **ComponentesAutomata**, se tiene:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3};
entradas = {a, b, c};
inicial = σ3;
trans = {{σ0, a, {σ1, σ2, σ3}}, {σ0, b, {σ2, σ3}}, {σ0, c, {σ0}}, {σ1, a,
{}}, {σ1, b, {σ1, σ3}}, {σ1, c, {σ0, σ1, σ2, σ3}}, {σ2, a, {σ3}}, {σ2, b,
{σ0, σ2}}, {σ2, c, {σ2}}, {σ3, a, {σ3}}, {σ3, b, {σ0}}, {σ3, c, {σ0}}};
aceptados = {σ1, σ2};
automata = Automata[estados, entradas, inicial, trans, aceptados];
ComponentesAutomata[automata]
```

Out[ ] :=

Estados: {σ_0, σ_1, σ_2, σ_3}

Símbolos de entrada: {a, b, c}

Estado inicial: σ_3

Estados aceptados: {σ_1, σ_2}

	a	b	c
σ_0	$\{\sigma_1,\sigma_2,\sigma_3\}$	$\{\sigma_2,\sigma_3\}$	$\{\sigma_0\}$
σ_1	$\{\}$	$\{\sigma_1,\sigma_3\}$	$\{\sigma_0,\sigma_1,\sigma_2,\sigma_3\}$
σ_2	$\{\sigma_3\}$	$\{\sigma_0,\sigma_2\}$	$\{\sigma_2\}$
σ_3	$\{\sigma_3\}$	$\{\sigma_0\}$	$\{\sigma_0\}$

Función de transición de estados con otro formato: {{σ_0, a, {σ_1, σ_2, σ_3}}, {σ_0, b, {σ_2, σ_3}}, {σ_0, c, {σ_0}}, {σ_1, a, {}}, {σ_1, b, {σ_1, σ_3}}, {σ_1, c, {σ_0, σ_1, σ_2, σ_3}}, {σ_2, a, {σ_3}}, {σ_2, b, {σ_0, σ_2}}, {σ_2, c, {σ_2}}, {σ_3, a, {σ_3}}, {σ_3, b, {σ_0}}, {σ_3, c, {σ_0}}}

Ejemplo 9.28

Encuentre las cinco componentes del autómata generado al correr **AutomataRandom[6, 7, colores -> True]**.

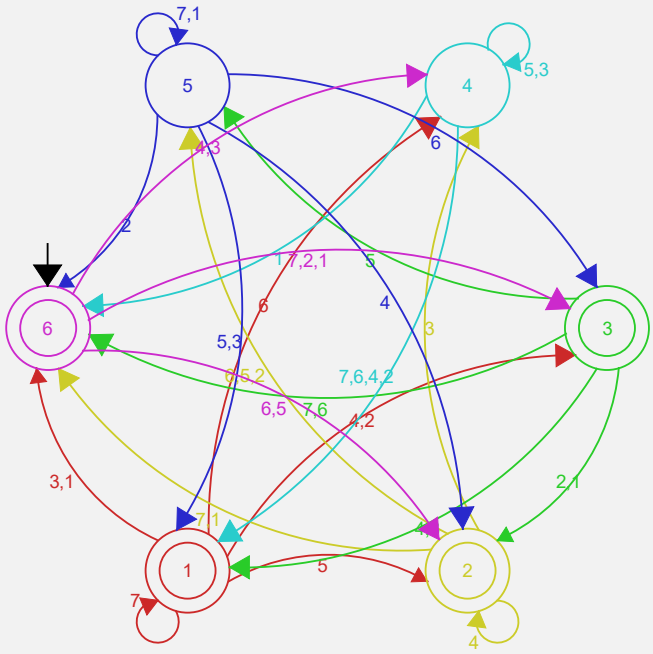
Solución:

En el software:

In[ ] :=

```
AutomataRandom[6, 7, colores -> True]
ComponentesAutomata[G]
```

Out[ ] :=



Estados: {1, 2, 3, 4, 5, 6}
Símbolos de entrada: {1, 2, 3, 4, 5, 6, 7}
Estado inicial: 6
Estados aceptados: {1, 2, 3, 6}

	1	2	3	4	5	6	7
1	6	3	6	3	2	4	1
2	6	5	4	2	5	5	6
3	2	2	1	1	5	6	6
4	6	1	4	1	4	1	1
5	5	6	1	2	1	3	5
6	3	3	4	4	2	2	3

Función de transición de estados con otro formato: {{1, 1, 6}, {1, 2, 3}, {1, 3, 6}, {1, 4, 3}, {1, 5, 2}, {1, 6, 4}, {1, 7, 1}, {2, 1, 6}, {2, 2, 5}, {2, 3, 4}, {2, 4, 2}, {2, 5, 5}, {2, 6, 5}, {2, 7, 6}, {3, 1, 2}, {3, 2, 2}, {3, 3, 1}, {3, 4, 1}, {3, 5, 5}, {3, 6, 6}, {3, 7, 6}, {4, 1, 6}, {4, 2, 1}, {4, 3, 4}, {4, 4, 1}, {4, 5, 4}, {4, 6, 1}, {4, 7, 1}, {5, 1, 5}, {5, 2, 6}, {5, 3, 1}, {5, 4, 2}, {5, 5, 1}, {5, 6, 3}, {5, 7, 5}, {6, 1, 3}, {6, 2, 3}, {6, 3, 4}, {6, 4, 4}, {6, 5, 2}, {6, 6, 2}, {6, 7, 3}}

Explicación en video

15) **StringAceptadaQ**: retorna "`True`" si una **hilera de símbolos de entrada** es **aceptada** por un **autómata de estado finito** "A" recibido como parámetro (**determinístico** (*DFA*) o **no determinístico** (*NDFA*)), o "`False`", en caso contrario. Presenta la **opción** "`trace -> True`" que muestra, además, la **lista** de los **estados visitados** o **posibles** (si el autómata es *NDFA*) durante el recorrido.

Sintaxis: `StringAceptadaQ[A, string]`, o bien, `StringAceptadaQ[A, string, trace->True]`, con "string" un **vector** cuyas coordenadas son los **símbolos de entrada** a procesar.

Ejemplo 9.29

Mediante una hilera de símbolos de entrada seudoaleatoria de longitud veinte, establezca si es aceptada o no, observando el recorrido de los estados, sobre el autómata:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
- Símbolos de entrada: $\tau = \{a, b\}$
- Estado inicial: $\sigma^* = \sigma_0$
- Función de transición de estados: $\{\{\sigma_0, a, \sigma_0\}, \{\sigma_0, b, \sigma_1\}, \{\sigma_1, a, \sigma_2\}, \{\sigma_1, b, \sigma_1\}, \{\sigma_2, a, \sigma_3\}, \{\sigma_2, b, \sigma_1\}, \{\sigma_3, a, \sigma_0\}, \{\sigma_3, b, \sigma_1\}\}$
- Estados aceptados: $\{\sigma_3\}$

Despliegue su diagrama de transición.

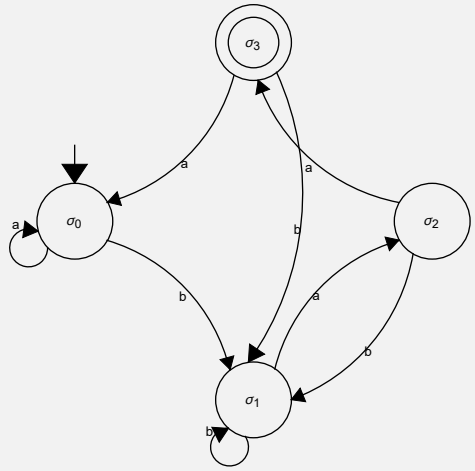
Solución:

Al utilizar el comando **`StringAceptadaQ`** y su opción "**`trace->True`**":

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3};
entradas = {a, b};
inicial = σ0;
trans = {{σ0, a, σ0}, {σ0, b, σ1}, {σ1, a, σ2}, {σ1, b, σ1}, {σ2, a, σ3},
{σ2, b, σ1}, {σ3, a, σ0}, {σ3, b, σ1}};
aceptados = {σ3};
automata = Automata[estados, entradas, inicial, trans, aceptados];
AutomataToDiagrama[automata]
α = RandomChoice[alphabet[automata], 20]
StringAceptadaQ[automata, α, trace->True]
```

Out[ ] :=



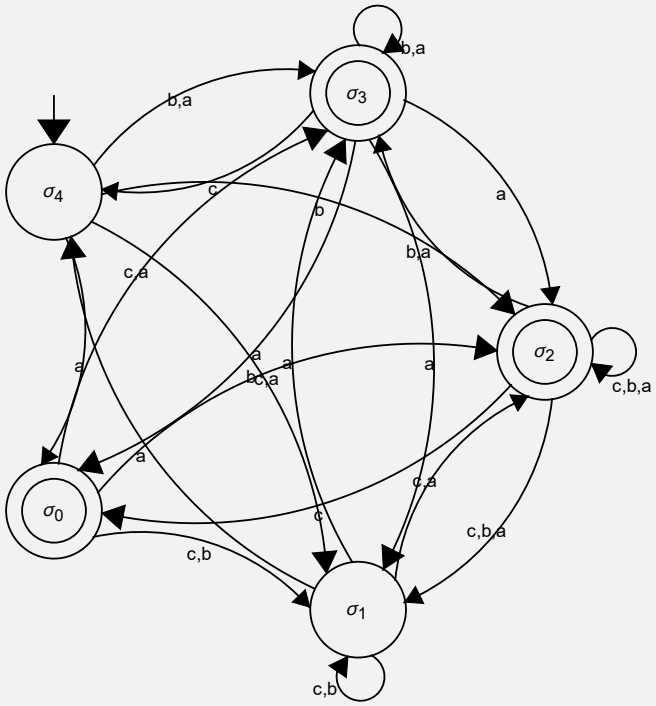
{b, b, b, a, b, a, a, b, b, a, b, b, a, b, a, a, b, b, a, a}

{True, {σ_0, σ_1, σ_1, σ_1, σ_2, σ_1, σ_2, σ_3, σ_1, σ_1, σ_2, σ_1, σ_1, σ_2, σ_1, σ_2, σ_3, σ_1, σ_1, σ_2, σ_3}}

N Se concluye que la hilera es aceptada pues en el recorrido, se finaliza en el estado aceptado σ_3. Es importante mencionar que la instrucción **`alphabet`** utilizada en el *In*[ ], devuelve los símbolos de entrada del autómata. Es un comando de autoría de *Alon Levy*.

Ejemplo 9.30

Considerando un "string" de símbolos de entrada seudoaleatorio de longitud veinte y el autómata:



use la instrucción **`StringAceptadaQ`**, para determinar si la hilera resultante es aceptada o no. Retorne el recorrido paso a paso.

Solución:

En *Mathematica*:

In[ ] :=

```
estados = {σ0, σ1, σ2, σ3, σ4};
entradas = {a, b, c};
inicial = σ4;
trans = {{σ0, a, {σ2, σ3}}, {σ0, b, {σ1}}, {σ0, c, {σ1, σ2, σ3}}, {σ1, a,
{σ2, σ3, σ4}}, {σ1, b, {σ1}}, {σ1, c, {σ1, σ2}}, {σ2, a, {σ1, σ2, σ3}},
{σ2, b, {σ1, σ2, σ3}}, {σ2, c, {σ0, σ1, σ2}}, {σ3, a, {σ0, σ1, σ2, σ3}},
{σ3, b, {σ3}}, {σ3, c, {σ4}}, {σ4, a, {σ0, σ3}}, {σ4, b, {σ1, σ2, σ3}},
{σ4, c, {}}};
```

```
aceptados = {σ0, σ2, σ3};
automata = Automata[estados, entradas, inicial, trans, aceptados];
α = RandomChoice[alphabet[automata], 20]
StringAceptadaQ[automata, α]
StringAceptadaQ[automata, α, trace -> True]
```

Out[ ] :=

{a, c, b, c, a, c, a, c, b, b, a, c, c, a, a, b, b, c, a, c}

True

{True, {{σ_4}, {σ_0, σ_3}, {σ_1, σ_2, σ_3, σ_4}, {σ_1, σ_2, σ_3}, {σ_0, σ_1, σ_2, σ_4}, {σ_0, σ_1, σ_2, σ_3, σ_4}, {σ_0, σ_1, σ_2, σ_3, σ_4}, {σ_0, σ_1, σ_2, σ_3, σ_4}, {σ_0, σ_1, σ_2, σ_3, σ_4}, {σ_1, σ_2, σ_3}, {σ_1, σ_2, σ_3}, {σ_0, σ_1, σ_2, σ_3, σ_4}, {σ_0, σ_1, σ_2, σ_3, σ_4}, {σ_0, σ_1, σ_2, σ_3, σ_4}, {σ_0, σ_1, σ_2, σ_3, σ_4}, {σ_0, σ_1, σ_2, σ_3, σ_4}, {σ_1, σ_2, σ_3}, {σ_1, σ_2, σ_3}, {σ_0, σ_1, σ_2, σ_4}, {σ_0, σ_1, σ_2, σ_3, σ_4}, {σ_0, σ_1, σ_2, σ_3, σ_4}}}

N La hilera es aceptada. Al visualizar el "trace" se infiere que el recorrido puede terminar en cualquiera de los estados aceptados: σ_0, σ_2 o σ_3. Además, en este ejemplo, como α es seudoaleatoria, podría ocurrir que al volver a ejecutar el $In[\,]$, se forme otro "string" no aceptado.

Explicación en video

16) **AutomatasEquivalentes**: **retorna** los **diagramas de transición** de **dos autómatas seudoaleatorios equivalentes** con "n" **estados** y "m" **símbolos de entrada**. Por defecto los autómatas se almacenan en **dos variables denominadas** "G1" y "G2", respectivamente. Brinda la **opción** "**colores -> True**" que añade **color** a la representación. El comando realiza **a lo sumo** "k" búsquedas con la intención de ofrecer una salida adecuada, si posterior a ello, no se logró construir los autómatas, **retorna** "NaD". La instrucción tiene **fines didácticos** para mostrar el **concepto** de **autómatas equivalentes** (autómatas que **aceptan** las **mismas** hileras del **alfabeto**).

Sintaxis: **AutomatasEquivalentes[n, m, k]**, o bien, **AutomatasEquivalentes[n, m, k, colores -> True]**.

Ejemplo 9.31

Construya dos autómatas equivalentes seudoaleatorios con tres estados y tres símbolos de entrada, realizando 100 búsquedas como máximo y añadiendo color a los diagramas de transición.

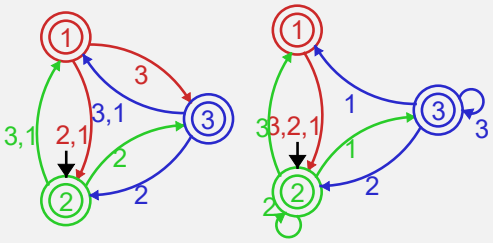
Solución:

En el software:

In[ ] :=

```
AutomatasEquivalentes[3, 3, 100, colores -> True]
```

Out[ ] :=



Ejemplo 9.32

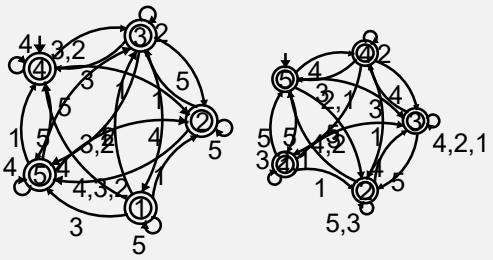
Genere el diagrama de transición de dos autómatas de estado finito equivalentes seudoaleatorios de orden 5×5.

Solución:

In[ ] :=

```
AutomatasEquivalentes[5, 5, 1000]
```

Out[ ] :=



N El número máximo de pruebas 1000 se escogió en este caso, ya que en valores inferiores, la instrucción retorna "NaD". Desde luego, el estudiante debe analizar que entre mayor sea la cantidad de pruebas realizadas en *Mathematica*, mayor será el tiempo consumido por el software para producir una salida satisfactoria.

Explicación en video

17) **AutomataExactamente**: **construye** un **autómata de estado finito determinístico** que **acepta** hileras con **exactamente** cierta cantidad de "a's", "b's", o cualquier **otro carácter** ingresado por el usuario. Presenta la **opción** "`colores -> True`" que añade **color** a la representación.

Sintaxis: `AutomataExactamente[M]`, o bien, `AutomataExactamente[M, colores -> True]`,

con "M" una **matriz** en **dos columnas**, donde cada fila especifica en la **primera entrada** el **carácter** y en la **segunda**, el **número** de **veces** que se **repetirá** dentro de la hilera. Por defecto, el autómata se almacena en una **variable denominada** "G".

Ejemplo 9.33

Construya el diagrama de transición de un autómata que acepte únicamente "strings" con exactamente dos "a" y dos "b".

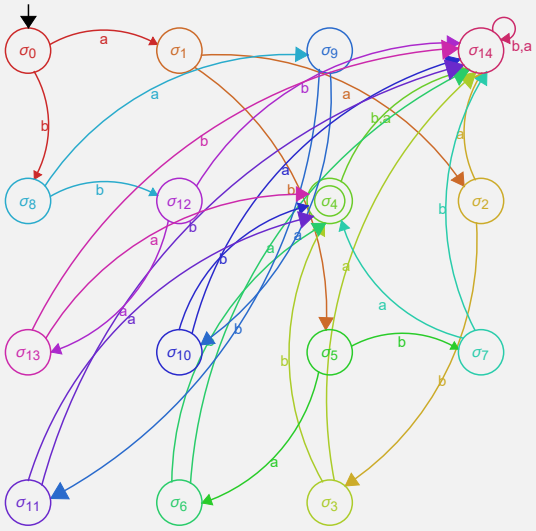
Solución:

Al recurrir al comando **`AutomataExactamente`**, se tiene:

In[ ] :=

```
AutomataExactamente[{{a, 2}, {b, 2}}, colores -> True]
```

Out[ ] :=



Se utilizó la opción "**`colores -> True`**", para dar un aspecto más agradable al diagrama devuelto.

Ejemplo 9.34

Determine el diagrama de transición de un autómata de estado finito determístico, que posea como lenguaje hileras que contengan exactamente cuatro "a" y tres "b". Verifique haciendo uso de **`LenguajeStrings`** el lenguaje del automáta.

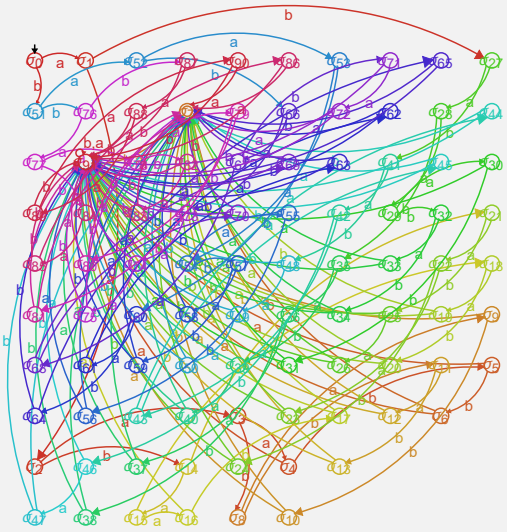
Solución:

En *Mathematica*:

In[ ] :=

```
AutomataExactamente[{{a, 4}, {b, 3}}, colores -> True]
LenguajeStrings[G, 8, limite -> True]
```

Out[ ] :=



{{a, a, a, a, b, b, b}, {a, a, a, b, a, b, b}, {a, a, a, b, b, a, b}, {a, a, a, b, b, b, a}, {a, a, b, a, a, b, b}, {a, a, b, a, b, a, b}, {a, a, b, a, b, b, a}, {a, a, b, b, a, a, b}, {a, a, b, b, a, b, a}, {a, a, b, b, b, a, a}, {a, b, a, a, a, b, b}, {a, b, a, a, b, a, b}, {a, b, a, a, b, b, a}, {a, b, a, b, a, a, b}, {a, b, a, b, a, b, a}, {a, b, a, b, b, a, a}, {a, b, b, a, a, b, a}, {a, b, b, a, b, a, a}, {a, b, b, b, a, a, a}, {b, a, a, a, a, b, b}, {b, a, a, a, b, a, b}, {b, a, a, a, b, b, a}, {b, a, a, b, a, a, b}, {b, a, a, b, a, b, a}, {b, a, a, b, b, a, a}, {b, a, b, a, a, a, b}, {b, a, b, a, a, b, a}, {b, a, b, b, a, a, a}, {b, b, a, a, a, a, b}, {b, b, a, a, a, b, a}, {b, b, a, a, b, a, a}, {b, b, a, b, a, a, a}, {b, b, b, a, a, a, a}}

N El lenguaje retornado por **`LenguajeStrings`** coincide con todas las permutaciones posibles para el conjunto: $\{a,a,a,a,b,b,b\}$. En el software, estas permutaciones se calculan al ejecutar **`Permutations[{a, a, a, a, b, b, b}]`**.

Explicación en video

18) **AutomataDeterministicoEquivalente**: **construye** un **autómata de estado finito determinístico** (*DFA*) **equivalente** a otro **no determinístico** "A" (*NDFA*), ingresado como parámetro. Brinda la **opción** "**`colores->True`**" que añade **color** a los diagramas de transición mostrados. Por defecto, el autómata *DFA* queda almacenado en una **variable denominada** "G".

Sintaxis: **`AutomataDeterministicoEquivalente[A]`**, o bien,

```
AutomataDeterministicoEquivalente[A, colores->True]
```

Ejemplo 9.35

Sea el autómata no determinístico:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$
- Símbolos de entrada: $\tau = \{a, b\}$
- Estado inicial: $\sigma^* = \sigma_0$
- Función de transición de estados: $\{\{\sigma_0, a, \{\sigma_1\}\}, \{\sigma_0, b, \{\sigma_3\}\}, \{\sigma_1, a, \{\sigma_1, \sigma_2\}\}, \{\sigma_1, b, \{\sigma_3\}\}, \{\sigma_2, a, \{\}\}, \{\sigma_2, b, \{\sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_3, a, \{\}\}, \{\sigma_3, b, \{\}\}\}$
- Estados aceptados: $\{\sigma_1\}$

Encuentre un autómata determinístico equivalente.

Solución:

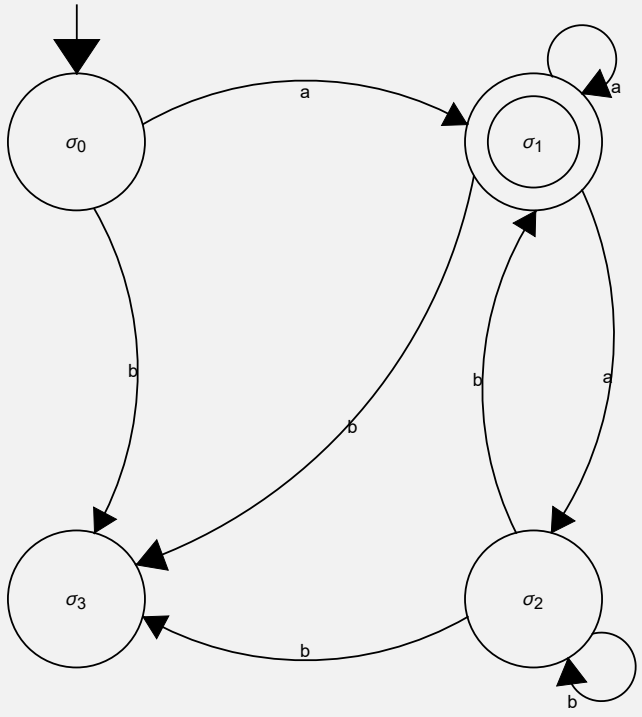
La instrucción **AutomataDeterministicoEquivalente** automatiza todo el proceso requerido para solucionar el ejemplo:

In[ ] :=

```
automata = Automata[{σ0, σ1, σ2, σ3}, {a, b}, σ0, {{σ0, a, {σ1}}, {σ0, b, {σ3}}, {σ1, a, {σ1, σ2}}, {σ1, b, {σ3}}, {σ2, a, {}}, {σ2, b, {σ1, σ2, σ3}}, {σ3, a, {}}, {σ3, b, {}}}, {σ1}]
AutomataDeterministicoEquivalente[automata]
```

Out[ ] :=

Diagrama de transición del autómata no determinístico:



Estados del nuevo autómata:

μ_0 = {}

$\mu_1 = \{\sigma_0\}$
$\mu_2 = \{\sigma_1\}$
$\mu_3 = \{\sigma_2\}$
$\mu_4 = \{\sigma_3\}$
$\mu_5 = \{\sigma_0, \sigma_1\}$
$\mu_6 = \{\sigma_0, \sigma_2\}$
$\mu_7 = \{\sigma_0, \sigma_3\}$
$\mu_8 = \{\sigma_1, \sigma_2\}$
$\mu_9 = \{\sigma_1, \sigma_3\}$
$\mu_{10} = \{\sigma_2, \sigma_3\}$
$\mu_{11} = \{\sigma_0, \sigma_1, \sigma_2\}$
$\mu_{12} = \{\sigma_0, \sigma_1, \sigma_3\}$
$\mu_{13} = \{\sigma_0, \sigma_2, \sigma_3\}$
$\mu_{14} = \{\sigma_1, \sigma_2, \sigma_3\}$
$\mu_{15} = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$

Estado inicial: μ_1

Estados aceptados: $\{\mu_2, \mu_5, \mu_8, \mu_9, \mu_{11}, \mu_{12}, \mu_{14}, \mu_{15}\}$

	a	b
μ_0	μ_0	μ_0
μ_1	μ_2	μ_4
μ_2	μ_8	μ_4
μ_3	μ_0	μ_{14}
μ_4	μ_0	μ_0
μ_5	μ_8	μ_4
μ_6	μ_2	μ_{14}
μ_7	μ_2	μ_4
μ_8	μ_8	μ_{14}
μ_9	μ_8	μ_4
μ_{10}	μ_0	μ_{14}
μ_{11}	μ_8	μ_{14}
μ_{12}	μ_8	μ_4
μ_{13}	μ_2	μ_{14}
μ_{14}	μ_8	μ_{14}
μ_{15}	μ_8	μ_{14}

Diagrama de transición del nuevo autómata:

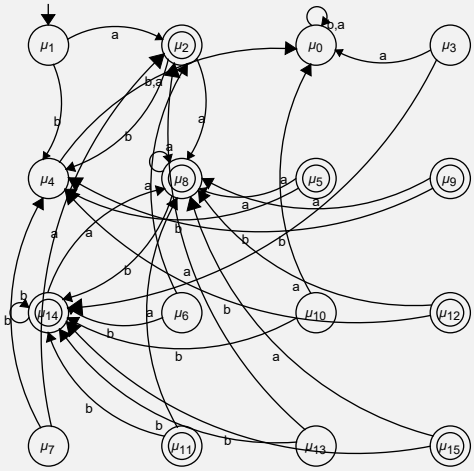
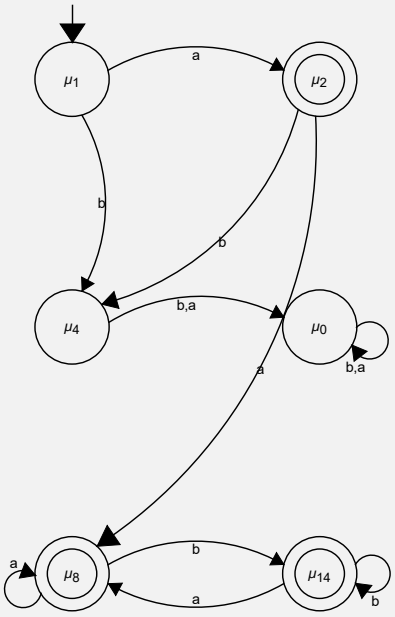


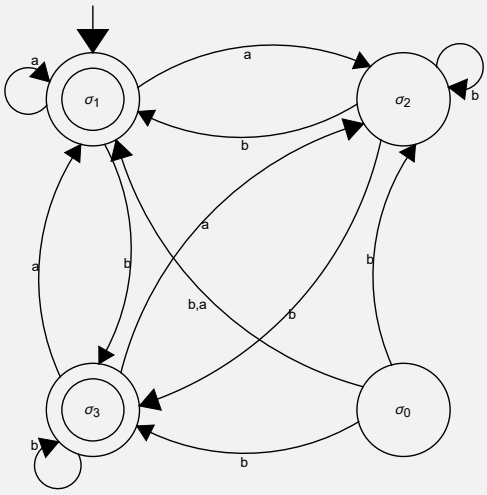
Diagrama de transición reducido del nuevo autómata:



N Como se aprecia el comando **`AutomataDeterministicoEquivalente`** realiza en detalle todo el procedimiento, que de acuerdo con la teoría de autómatas, se necesita para generar el autómata determinístico equivalente al original. Es una instrucción muy propicia con el objetivo de que el estudiante revise por cuenta personal, lo que ha resuelto previamente a mano, sin requerir la asistencia directa del docente.

Ejemplo 9.36

Encuentre un autómata determinístico equivalente a:



Solución:

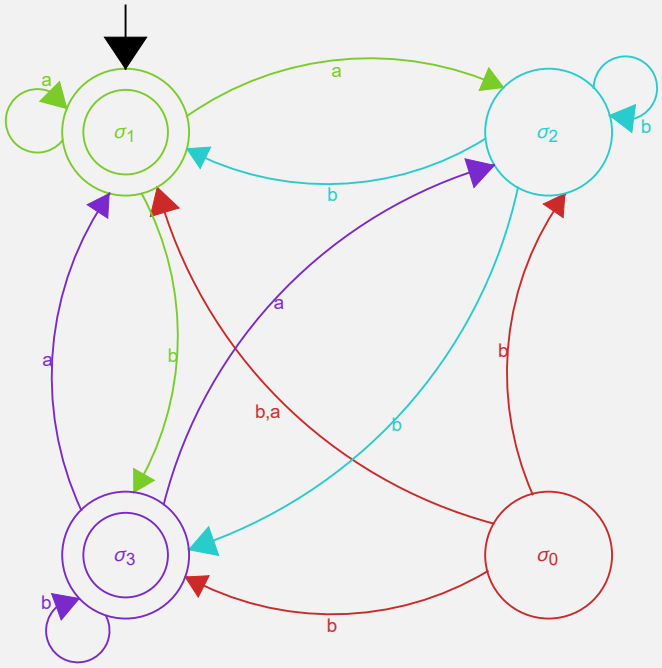
En el software:

In[ ] :=

```
automata = Automata[{σ0, σ1, σ2, σ3}, {a, b}, σ1, {{σ0, a, {σ1}}, {σ0, b, {σ3, σ1, σ2}}, {σ1, a, {σ1, σ2}}, {σ1, b, {σ3}}, {σ2, a, {}}, {σ2, b, {σ1, σ2, σ3}}, {σ3, a, {σ1, σ2}}, {σ3, b, {σ3}}}, {σ1, σ3}]
AutomataDeterministicoEquivalente[automata, colores -> True]
```

Out[ ] :=

Diagrama de transición del autómata no determinístico:



Estados del nuevo autómata:

μ_0 = {}
μ_1 = $\{\sigma_0\}$
μ_2 = $\{\sigma_1\}$
μ_3 = $\{\sigma_2\}$
μ_4 = $\{\sigma_3\}$
μ_5 = $\{\sigma_0, \sigma_1\}$
μ_6 = $\{\sigma_0, \sigma_2\}$
μ_7 = $\{\sigma_0, \sigma_3\}$
μ_8 = $\{\sigma_1, \sigma_2\}$
μ_9 = $\{\sigma_1, \sigma_3\}$
μ_{10} = $\{\sigma_2, \sigma_3\}$
μ_{11} = $\{\sigma_0, \sigma_1, \sigma_2\}$
μ_{12} = $\{\sigma_0, \sigma_1, \sigma_3\}$
μ_{13} = $\{\sigma_0, \sigma_2, \sigma_3\}$
μ_{14} = $\{\sigma_1, \sigma_2, \sigma_3\}$
μ_{15} = $\{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$

Estado inicial: μ_2

Estados aceptados: $\{\mu_2, \mu_4, \mu_5, \mu_7, \mu_8, \mu_9, \mu_{10}, \mu_{11}, \mu_{12}, \mu_{13}, \mu_{14}, \mu_{15}\}$

	a	b
μ_0	μ_0	μ_0
μ_1	μ_2	μ_{14}
μ_2	μ_8	μ_4
μ_3	μ_0	μ_{14}
μ_4	μ_8	μ_4
μ_5	μ_8	μ_{14}
μ_6	μ_2	μ_{14}
μ_7	μ_8	μ_{14}
μ_8	μ_8	μ_{14}
μ_9	μ_8	μ_4
μ_{10}	μ_8	μ_{14}
μ_{11}	μ_8	μ_{14}
μ_{12}	μ_8	μ_{14}
μ_{13}	μ_8	μ_{14}
μ_{14}	μ_8	μ_{14}
μ_{15}	μ_8	μ_{14}

Diagrama de transición del nuevo autómata:

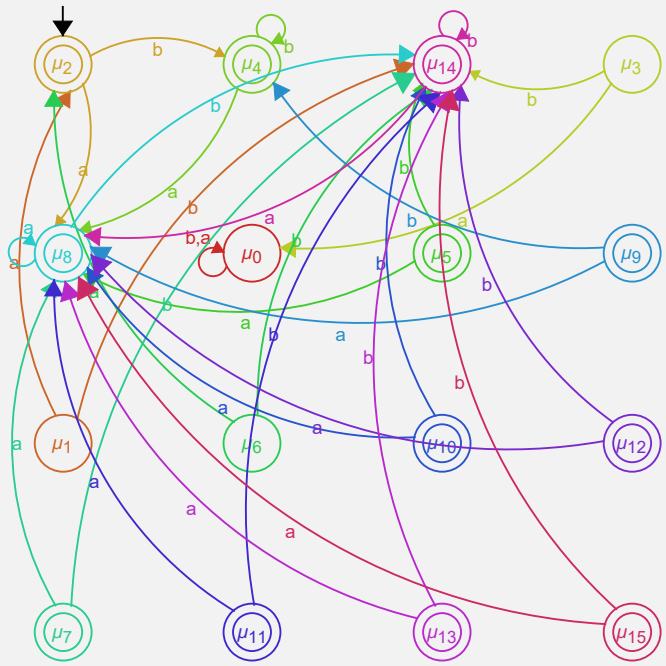
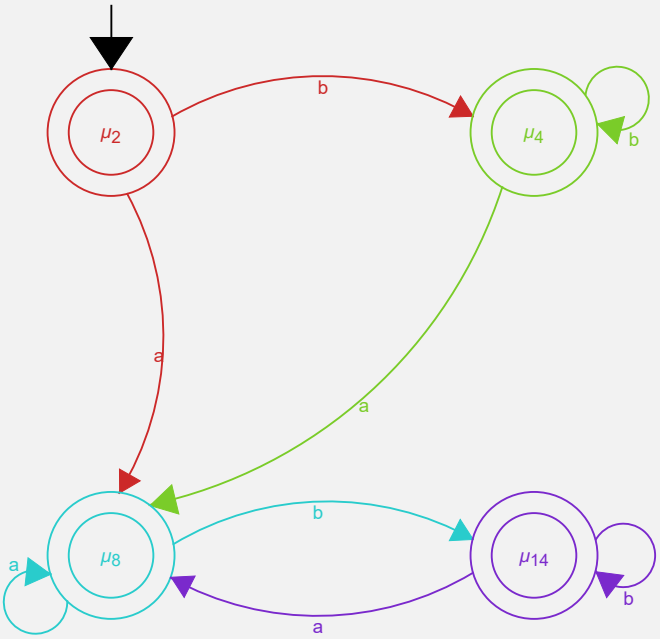


Diagrama de transición reducido del nuevo autómata:



Como un detalle estético en la resolución del ejercicio, se añadió color a todos los diagramas de transición, usando la opción "`colores -> True`".

Explicación en video

19) **MaquinaTuring**: **recorre** una **máquina de Turing** dadas las **siete componentes** que la definen: el **conjunto de estados**, el **conjunto de símbolos de entrada**, el **estado inicial**, la **función de transición**, el **conjunto de estados aceptados**, el **conjunto de símbolos de cinta** y el **símbolo espacio en blanco**. La función de transición se escribe como una **matriz** con **cinco columnas**, donde cada fila contiene en orden: el **par** "(estado, scinta)" y la **imagen** en dicha dupla como el **triplete** "(estado, socinta, movimiento)", siendo "scinta" y "socinta" **símbolos de cinta** y **movimiento** un valor igual a "1" si la cabeza lectora se mueve a la **derecha**, "-1" si se mueve a la **izquierda** y "0" si el movimiento es **nulo** (en cuyo caso la máquina se **detiene**). Los **estados aceptados** tienen que tener **movimiento nulo**. El comando, además, **recibe** una **hilera de símbolos de entrada** a procesar en la cinta y la **cantidad de pasos** a **ejecutar** (por defecto, se **inicia** en **cero**).

Sintaxis: `MaquinaTuring[Estados, Entradas, Inicio, FTrans, EstadosAceptados, SCinta, Blanco, Hilera, n]`, con "Inicio" el **estado inicial**, "FTrans" la **función de transición**, "SCinta" los **símbolos de cinta**, "Blanco" el **símbolo de cinta** que corresponde al **espacio en blanco**, "Hilera" un **vector** de **símbolos de entrada** a **recorrer** y "n" el **número de pasos**. La salida es una **matriz** donde cada fila muestra el **contenido de la cinta** y la última entrada el **estado actual**.

Ejemplo 9.37

Procese la hilera {1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1} con seis pasos, en la siguiente máquina de *Turing*:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2\}$
- Símbolos de entrada: $\tau = \{1\}$
- Estado inicial: $\sigma^* = \sigma_0$
- Función de transición: $\{\{\sigma_0, 0, \sigma_0, 0, 1\}, \{\sigma_0, 1, \sigma_1, 0, 1\}, \{\sigma_1, 0, \sigma_2, 1, 0\}, \{\sigma_1, 1, \sigma_1, 1, 1\}\}$
- Estados aceptados: $\{\sigma_2\}$
- Símbolos de cinta: $\Gamma = \{0, 1\}$
- Símbolo espacio en blanco: 0

Solución:

Al utilizar el comando **`MaquinaTuring`**, se tiene:

In[ ] :=

```
MaquinaTuring[{σ0, σ1, σ2}, {1}, σ0, {{σ0, 0, σ0, 0, 1}, {σ0, 1, σ1, 0,
1}, {σ1, 0, σ2, 1, 0}, {σ1, 1, σ1, 1, 1}}, {σ2}, {0, 1}, 0, {1, 1, 1,
1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, 6]
```

Out[ ] :=

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_0 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_2
\end{pmatrix}$$

Como σ_2 es aceptado, se concluye que la hilera {1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1} es aceptada por la máquina, observando la última entrada de la matriz anterior.

Ejemplo 9.38

Sea la máquina de *Turing*:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$
- Símbolos de entrada: $\tau = \{0, 1\}$
- Estado inicial: $\sigma^* = \sigma_0$

- Función de transición: {{σ_0, 0, σ_1, X, 1}, {σ_0, Y, σ_3, Y, 1}, {σ_1, 0, σ_1, 0, 1}, {σ_1, 1, σ_2, Y, -1}, {σ_1, Y, σ_1, Y, 1}, {σ_2, 0, σ_2, 0, -1}, {σ_2, X, σ_0, X, 1}, {σ_2, Y, σ_2, Y, -1}, {σ_3, Y, σ_3, Y, 1}, {σ_3, B, σ_4, B, 0}}
- Estados aceptados: {σ_4}
- Símbolos de cinta: $\Gamma = \{0, 1, X, Y, B\}$
- Símbolo espacio en blanco: B

Procese la hilera {0, 0, 1, 1} en trece pasos, ¿se puede inferir que es aceptada?

Solución:

En el software:

In[ ] :=

```
MaquinaTuring[{σ0, σ1, σ2, σ3, σ4}, {0, 1}, σ0, {{σ0, 0, σ1, X, 1}, {σ0, Y, σ3, Y, 1}, {σ1, 0, σ1, 0, 1}, {σ1, 1, σ2, Y, -1}, {σ1, Y, σ1, Y, 1}, {σ2, 0, σ2, 0, -1}, {σ2, X, σ0, X, 1}, {σ2, Y, σ2, Y, -1}, {σ3, Y, σ3, Y, 1}, {σ3, B, σ4, B, 0}}, {σ4}, {0, 1, X, Y, B}, B, {0, 0, 1, 1}, 13]
```

Out[ ] :=

$$\begin{pmatrix} 0 & 0 & 1 & 1 & B & \sigma_0 \\ X & 0 & 1 & 1 & B & \sigma_1 \\ X & 0 & 1 & 1 & B & \sigma_1 \\ X & 0 & Y & 1 & B & \sigma_2 \\ X & 0 & Y & 1 & B & \sigma_2 \\ X & 0 & Y & 1 & B & \sigma_0 \\ X & X & Y & 1 & B & \sigma_1 \\ X & X & Y & 1 & B & \sigma_1 \\ X & X & Y & Y & B & \sigma_2 \\ X & X & Y & Y & B & \sigma_2 \\ X & X & Y & Y & B & \sigma_0 \\ X & X & Y & Y & B & \sigma_3 \\ X & X & Y & Y & B & \sigma_3 \\ X & X & Y & Y & B & \sigma_4 \end{pmatrix}$$

Se deduce que la hilera es acepta pues se finaliza el recorrido en el estado aceptado σ_4.

Explicación en video

20) **MTStringAceptadaQ**: devuelve el valor lógico "`True`" si una **hilera de símbolos de entrada** es **aceptada** por una **máquina de Turing**, o "`False`", en caso contrario. Por defecto, el comando **ejecuta cien pasos** y con relación a ese recorrido **retorna** el **valor lógico**. Brinda al usuario las **opciones**: "`pasos -> Valor`" si se desea **ampliar** o **reducir** el número de avances y "`trace -> True`" que muestra, además, los **cambios** de la **cinta** y el **estado actual** en **cada iteración**.

Sintaxis:

```
MTStringAceptadaQ[Estados, Entradas, Inicio, FTrans, EstadosAceptados,
SCinta, Blanco, Hilera]
```

O bien,

```
MTStringAceptadaQ[Estados, Entradas, Inicio, FTrans, EstadosAceptados,
SCinta, Blanco, Hilera, pasos->Valor, trace->True]
```

pudiendo **prescindir** de cualquiera de las opciones.

Ejemplo 9.39

Considere la siguiente máquina de *Turing*:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2\}$
- Símbolos de entrada: $\tau = \{1\}$
- Estado inicial: $\sigma^* = \sigma_0$
- Función de transición: $\{\{\sigma_0, 0, \sigma_0, 0, 1\}, \{\sigma_0, 1, \sigma_1, 0, 1\}, \{\sigma_1, 0, \sigma_2, 1, 0\}, \{\sigma_1, 1, \sigma_1, 1, 1\}\}$
- Estados aceptados: $\{\sigma_2\}$
- Símbolos de cinta: $\Gamma = \{0, 1\}$
- Símbolo espacio en blanco: 0

¿Es aceptada la hilera de símbolos de entrada {1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}?

Solución:

En *Mathematica*, al emplear **MTStringAceptadaQ**:

```
In[ ] :=
MTStringAceptadaQ[{σ0, σ1, σ2}, {1}, σ0, {{σ0, 0, σ0, 0, 1}, {σ0, 1, σ1,
0, 1}, {σ1, 0, σ2, 1, 0}, {σ1, 1, σ1, 1, 1}}, {σ2}, {0, 1}, 0, {1, 1,
1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}]
```

```
Out[ ] :=
True
```

La instrucción facilita la opción "**trace -> True**", para observar el proceso de recorrido en la cinta iteración por iteración:

```
In[ ] :=
MTStringAceptadaQ[{σ0, σ1, σ2}, {1}, σ0, {{σ0, 0, σ0, 0, 1}, {σ0, 1, σ1,
0, 1}, {σ1, 0, σ2, 1, 0}, {σ1, 1, σ1, 1, 1}}, {σ2}, {0, 1}, 0, {1, 1,
1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, trace -> True, pasos -> 6]
```

Out[ ] :=

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_2 \end{pmatrix}$$

True

Ⓝ No se deja por defecto la cantidad de pasos mostrados a través de **`MTStringAceptadaQ`**, por el hecho de corresponder a cien, ocasionando un tamaño demasiado grande en la matriz que detalla el recorrido. En este ejercicio, a partir de 6 el valor lógico es True, es decir, por ejemplo, con la opción "**`pasos -> 5`**" el retorno es False, sin embargo, la hilera sí es aceptada. En este sentido, el estudiante debe ser cuidadoso corriendo una cantidad de pruebas relevante, que le garantice la correctitud de la respuesta.

Ejemplo 9.40

Haciendo uso del comando **`MTStringAceptadaQ`**, verifique que la hilera $\{0, 0, 1, 1\}$ es aceptada en la máquina de *Turing* del ejemplo tras anterior.

Solución:

En el software:

In[ ] :=

```
MTStringAceptadaQ[{σ0, σ1, σ2, σ3, σ4}, {0, 1}, σ0, {{σ0, 0, σ1, X, 1},
{σ0, Y, σ3, Y, 1}, {σ1, 0, σ1, 0, 1}, {σ1, 1, σ2, Y, -1}, {σ1, Y, σ1, Y,
1}, {σ2, 0, σ2, 0, -1}, {σ2, X, σ0, X, 1}, {σ2, Y, σ2, Y, -1}, {σ3, Y,
σ3, Y, 1}, {σ3, B, σ4, B, 0}}, {σ4}, {0, 1, X, Y, B}, B, {0, 0, 1, 1},
trace -> True, pasos -> 13]
```

Out[ ] :=

$$\begin{pmatrix} 0 & 0 & 1 & 1 & B & \sigma_0 \\ X & 0 & 1 & 1 & B & \sigma_1 \\ X & 0 & 1 & 1 & B & \sigma_1 \\ X & 0 & Y & 1 & B & \sigma_2 \\ X & 0 & Y & 1 & B & \sigma_2 \\ X & 0 & Y & 1 & B & \sigma_0 \\ X & X & Y & 1 & B & \sigma_1 \\ X & X & Y & 1 & B & \sigma_1 \\ X & X & Y & Y & B & \sigma_2 \\ X & X & Y & Y & B & \sigma_2 \\ X & X & Y & Y & B & \sigma_0 \\ X & X & Y & Y & B & \sigma_3 \\ X & X & Y & Y & B & \sigma_3 \\ X & X & Y & Y & B & \sigma_4 \end{pmatrix}$$

True

Explicación en video

21) **MTSumaEnteros**: **suma dos números enteros positivos** "n" y "m" recibidos como parámetros, mediante una **máquina de Turing**. Brinda la **opción** "**`mostrar -> True`**" que despliega las **siete componentes** de la máquina, además, de los **datos necesarios** para efectuar la **suma entera**.

Sintaxis: **`MTSumaEnteros[n, m]`**, o bien, **`MTSumaEnteros[n, m, mostrar -> True]`**.

Ejemplo 9.41

A través de una máquina de *Turing* sume: $3 + 4$. Despliegue la sucesión de pasos en detalle.

Solución:

Al recurrir a **`MTSumaEnteros`**:

In[ ] :=

```
MTSumaEnteros[3, 4, mostrar -> True]
```

Out[ ] :=

** Máquina de Turing **
Estados: $\{\sigma_0, \sigma_1, \sigma_2\}$
Símbolos de entrada: {1}
Estado inicial: σ_0
Función de transición: $\{\{\sigma_0, 0, \sigma_0, 0, 1\}, \{\sigma_0, 1, \sigma_1, 0, 1\}, \{\sigma_1, 0, \sigma_2, 1, 0\}, \{\sigma_1, 1, \sigma_1, 1, 1\}\}$
Estados aceptados: $\{\sigma_2\}$
Símbolos de cinta: {0, 1}
Símbolo espacio en blanco: 0
** Datos **

Primer entero positivo: {1, 1, 1}
Segundo entero positivo: {1, 1, 1, 1}
Hilera a procesar en la máquina de Turing: {1, 1, 1, 0, 1, 1, 1, 1}
Número de pasos: 4

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & \sigma_0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_2 \end{pmatrix}$$

Suma o número de unos de la última fila: 7

Ejemplo 9.42

Sume $6+7$ usando la máquina de *Turing* de la instrucción **`MTSumaEnteros`**.

Solución:

En *Mathematica*:

In[ ] :=

```
MTSumaEnteros[6, 7]
```

Out[ ] :=
** Datos **
Primer entero positivo: {1,1,1,1,1,1}
Segundo entero positivo: {1,1,1,1,1,1,1}
Hilera a procesar en la máquina de Turing: {1,1,1,1,1,1,0,1,1,1,1,1,1,1}
Número de pasos: 7

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \sigma_2 \end{pmatrix}$$

Suma o número de unos de la última fila: 13

Explicación en video

22) **MTCopiado**: **copia una expresión** "exp" un **número** "n" de **veces**, mediante el empleo de una **máquina de Turing**. Brinda la **opción** "`mostrar -> True`" que despliega las **siete componentes** de la máquina, además, de los **datos necesarios** para efectuar el **copiado**.

Sintaxis: `MTCopiado[{exp, n}]`, o bien, `MTCopiado[{exp, n}, mostrar -> True]`.

Ejemplo 9.43

Realice el copiado del carácter "1" tres veces usando una máquina de *Turing*.

Solución:

El comando **MTCopiado**, resuelve el ejercicio:

In[ ] :=

```
MTCopiado[{1, 3}]
```

Out[ ] :=

** Datos **

Hilera a procesar en la máquina de Turing: {1, 1, 1}

Número de pasos: 27

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & \sigma_0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & \sigma_1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & \sigma_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & \sigma_4 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & \sigma_0 \end{pmatrix}$$

La matriz posee 28 filas, por lo que no se muestra en su totalidad.

Ejemplo 9.44

Copie mediante la máquina de la instrucción **MTCopiado**, cuatro veces el carácter "a". Muestre la máquina de *Turing* respectiva.

Solución:

Al usar la opción "**mostrar -> True**":

In[ ] :=

```
MTCopiado[{a, 4}, mostrar -> True]
```

Out[ ] :=

** Máquina de Turing **

Estados: {σ_0, σ_1, σ_2, σ_3, σ_4}

Símbolos de entrada: {a}

Estado inicial: σ_0

Función de transición: {{σ_0, a, σ_1, 0, 1}, {σ_1, 0, σ_2, 0, 1}, {σ_1, a, σ_1, a, 1}, {σ_2, 0, σ_3, a, -1}, {σ_2, a, σ_2, a, 1}, {σ_3, 0, σ_4, 0, -1}, {σ_3, a, σ_3, a, -1}, {σ_4, 0, σ_0, a, 1}, {σ_4, a, σ_4, a, -1}}

Estados aceptados: {}

Símbolos de cinta: {0, a}

Símbolo espacio en blanco: 0

** Datos **

Hilera a procesar en la máquina de Turing: {a, a, a, a}

Número de pasos: 44

$$\begin{pmatrix} a & a & a & a & 0 & 0 & 0 & 0 & 0 & \sigma_0 \\ 0 & a & a & a & 0 & 0 & 0 & 0 & 0 & \sigma_1 \\ 0 & a & a & a & 0 & 0 & 0 & 0 & 0 & \sigma_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a & a & a & 0 & 0 & a & a & a & a & \sigma_4 \\ a & a & a & a & 0 & a & a & a & a & \sigma_0 \end{pmatrix}$$

La matriz de la salida contiene 45 filas, por lo que no se muestra en detalle.

Explicación en video

23) **MTEncuentraHilerasAceptadas**: **retorna** un **máximo** de "n" **hileras de símbolos de entrada** de **longitud** "m", **aceptadas** por una **máquina de Turing** recibida como parámetro a través de sus **siete componentes**. Presenta las **opciones**: "**`limite->Valor`**" que especifica la **cantidad máxima** de **pruebas** que realiza el comando, para **generar** cada una de las **hileras de forma seudoaleatoria** (por defecto, "**`Valor=10`**") y "**`all->True`**" que **muestra** una serie de **listas** con un **máximo** de "n" **hileras aceptadas** de **longitud menor o igual** a "m". La instrucción busca **estudiar** el lenguaje de la máquina.

Sintaxis:

```
MTEncuentraHilerasAceptadas[Estados, Entradas, Inicio, FTrans,
EstadosAceptados, SCinta, Blanco, n, m]
```

O bien,

```
MTEncuentraHilerasAceptadas[Estados, Entradas, Inicio, FTrans,
EstadosAceptados, SCinta, Blanco, n, m, limite->Valor, all->True]
```

pudiendo **prescindir** de cualquiera de las opciones y donde "Inicio" es el **estado inicial**, "FTrans" la **función de transición**, "SCinta" el conjunto de **símbolos de cinta** y "Blanco" el **símbolo de cinta** que corresponde al **espacio en blanco**.

Ejemplo 9.45

Sobre la máquina de *Turing*:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$
- Símbolos de entrada: $\tau = \{0, 1\}$
- Estado inicial: $\sigma^* = \sigma_0$

- Función de transición: {{σ_0, 0, σ_1, X, 1}, {σ_0, Y, σ_3, Y, 1}, {σ_1, 0, σ_1, 0, 1}, {σ_1, 1, σ_2, Y, -1}, {σ_1, Y, σ_1, Y, 1}, {σ_2, 0, σ_2, 0, -1}, {σ_2, X, σ_0, X, 1}, {σ_2, Y, σ_2, Y, -1}, {σ_3, Y, σ_3, Y, 1}, {σ_3, B, σ_4, B, 0}}
- Estados aceptados: {σ_4}
- Símbolos de cinta: $\Gamma = \{0, 1, X, Y, B\}$
- Símbolo espacio en blanco: B

Determine si es posible cuatro hileras aceptadas de longitud cuatro.

Solución:

En el software:

In[ ] :=

```
MTEncuentraHilerasAceptadas[{σ0, σ1, σ2, σ3, σ4}, {0, 1}, σ0, {{σ0, 0, σ1, X, 1}, {σ0, Y, σ3, Y, 1}, {σ1, 0, σ1, 0, 1}, {σ1, 1, σ2, Y, -1}, {σ1, Y, σ1, Y, 1}, {σ2, 0, σ2, 0, -1}, {σ2, X, σ0, X, 1}, {σ2, Y, σ2, Y, -1}, {σ3, Y, σ3, Y, 1}, {σ3, B, σ4, B, 0}}, {σ4}, {0, 1, X, Y, B}, B, 4, 4]
```

Out[ ] :=

```
{{0, 0, 1, 1}}
```

(N) En este ejemplo, la instrucción **MTEncuentraHilerasAceptadas** solo fue capaz de retornar una hilera aceptada y no cuatro. Ocasionalmente el comando tiene este comportamiento.

Ejemplo 9.46

En la máquina de *Turing* siguiente:

- Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6\}$
- Símbolos de entrada: $\tau = \{0, 1\}$
- Estado inicial: $\sigma^* = \sigma_0$
- Función de transición: {{σ_0, 0, σ_1, B, 1}, {σ_0, 1, σ_5, B, 1}, {σ_1, 0, σ_1, 0, 1}, {σ_1, 1, σ_2, 1, 1}, {σ_2, 0, σ_3, 1, -1}, {σ_2, 1, σ_2, 1, 1}, {σ_2, B, σ_4, B, -1}, {σ_3, 0, σ_3, 0, -1}, {σ_3, 1, σ_3, 1, -1}, {σ_3, B, σ_0, B, 1}, {σ_4, 0, σ_4, 0, -1}, {σ_4, 1, σ_4, B, -1}, {σ_4, B, σ_6, 0, 0}, {σ_5, 0, σ_5, B, 1}, {σ_5, 1, σ_6, B, 0}}
- Estados aceptados: {σ_6}
- Símbolos de cinta: $\Gamma = \{0, 1, B\}$
- Símbolo espacio en blanco: B

Halle de ser posible, diez hileras aceptadas de longitudes: 1, 2, ..., 20.

Solución:

En este caso, se recurre a la opción "**all->True**" del comando **MTEncuentraHilerasAceptadas**:

In[ ] :=

```
MTEncuentraHilerasAceptadas[{σ0, σ1, σ2, σ3, σ4, σ5, σ6}, {0, 1}, σ0,
{{σ0, 0, σ1, B, 1}, {σ0, 1, σ5, B, 1}, {σ1, 0, σ1, 0, 1}, {σ1, 1, σ2, 1,
1}, {σ2, 0, σ3, 1, -1}, {σ2, 1, σ2, 1, 1}, {σ2, B, σ4, B, -1}, {σ3, 0,
σ3, 0, -1}, {σ3, 1, σ3, 1, -1}, {σ3, B, σ0, B, 1}, {σ4, 0, σ4, 0, -1},
{σ4, 1, σ4, B, -1}, {σ4, B, σ6, 0, 0}, {σ5, 0, σ5, B, 1}, {σ5, 1, σ6, B,
0}}, {σ6}, {0, 1, B}, B, 10, 20, all->True]
```

Out[ ] :=

{1 -> {}, 2 -> {{1, 1}, {0, 1}}, 3 -> {{0, 0, 1}, {0, 1, 0}, {1, 1, 0}, {1, 1, 1}}, 4 -> {{0, 1, 0, 1}, {1, 0, 0, 1}, {0, 1, 0, 0}, {1, 0, 1, 1}, {0, 0, 1, 0}, {1, 0, 1, 0}}, 5 -> {{1, 1, 1, 0, 0}, {0, 0, 1, 1, 0}, {0, 0, 0, 0, 1}, {0, 1, 1, 1, 0}, {0, 0, 1, 0, 0}, {0, 0, 1, 1, 1}, {1, 0, 1, 0, 1}, {1, 0, 1, 1, 0}, {0, 1, 0, 0, 0}}, ..., 19 -> {{0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1}, {0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0}, {0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0}, {0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1}, {0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1}, {1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0}, {1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0}, {1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0}, {1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1}, {1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0}}, 20 -> {{0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0}, {0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1}, {1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0}, {1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1}, {1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0}, {0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1}, {1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1}, {1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0}, {0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0}, {0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0}}}

N El *Out* [ ] reporta veinte listas, cada una con un máximo de diez hileras aceptadas, de longitud igual a: 1, 2, 3, 4, ..., 20. No se muestra la salida completa por su tamaño.

Explicación en video

Aporte pedagógico

Varios comandos incluidos en este apartado, tienen aplicaciones netamente didácticas para **profundizar conceptos** relacionados con máquinas de estado finito, autómatas y máquinas de *Turing*. Por ejemplo, **CDFMaquinaEF** falicita la observación del **procesamiento** de una **hilera de símbolos de entrada** en una máquina ingresada por el usuario, **AutomatasEquivalentes** pretende analizar los **conceptos** de **hilera aceptada**, **lenguaje de un autómata** y **equivalencia** entre autómatas, **MTStringAceptadaQ** permite la **exploración** del **recorrido** en la **cinta** correspondiente a una máquina de *Turing*, en donde el alumno puede visualizar su comportamiento **paso a paso**. Iniciativas como éstas, adecuadamente empleadas, aportan ingredientes de aprendizajes **más significativos**, po-

tencializando el nivel de **comprensión** de la población estudiantil.

Aporte de investigación

Estudiar el **lenguaje** asociado a un autómata de estado finito e inclusive a una máquina de *Turing*, constituye una **tarea no trivial** que abre un abanico de posibilidades de **análisis** e **investigación** individual o colectiva. A este respecto, las instrucciones de *VilCretas* **`LenguajeStrings`**, **`StringAceptadaQ`**, **`AutomataDeterministicoEquivalente`** y **`MTEncuentraHilerasAceptadas`** proporcionan **recursos** de **indagación** y **visualización**, que podrían ser aprovechados en la inferencia de las hileras de **símbolos de entrada aceptadas** en autómatas o máquinas de *Turing* **no convencionales**, o bien, con un **tamaño relevante**, donde el uso de software se convierta en un **medio natural** y **transparente** para la resolución de problemas.

Ejercicios

Resuelva los siguientes ejercicios utilizando como apoyo el paquete *VilCretas*.

1) Retorne el diagrama de transición de las siguientes máquinas de estado finito:

   a) M_1 : Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$, símbolos de entrada: $\tau = \{a, b, c, d\}$, símbolos de salida: $\delta = \{0, 1, 2, 3, 4\}$, estado inicial: $\sigma^* = \sigma_3$, función de estado siguiente y de salida: $\{\{\sigma_0, a, \sigma_0, 4\}, \{\sigma_0, b, \sigma_3, 0\}, \{\sigma_0, c, \sigma_2, 0\}, \{\sigma_0, d, \sigma_3, 1\}, \{\sigma_1, a, \sigma_2, 0\}, \{\sigma_1, b, \sigma_0, 1\}, \{\sigma_1, c, \sigma_2, 2\}, \{\sigma_1, d, \sigma_0, 1\}, \{\sigma_2, a, \sigma_0, 1\}, \{\sigma_2, b, \sigma_2, 3\}, \{\sigma_2, c, \sigma_3, 2\}, \{\sigma_2, d, \sigma_1, 4\}, \{\sigma_3, a, \sigma_3, 0\}, \{\sigma_3, b, \sigma_2, 2\}, \{\sigma_3, c, \sigma_0, 1\}, \{\sigma_3, d, \sigma_0, 3\}\}$.

   b) M_2 : Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$, símbolos de entrada: $\tau = \{a, b, c, d\}$, símbolos de salida: $\delta = \{0, 1, 2, 3, 4\}$, estado inicial: $\sigma^* = \sigma_1$, función de estado siguiente y de salida: $\{\{\sigma_0, a, \sigma_1, 1\}, \{\sigma_0, b, \sigma_2, 2\}, \{\sigma_0, c, \sigma_3, 3\}, \{\sigma_0, d, \sigma_0, 4\}, \{\sigma_1, a, \sigma_1, 1\}, \{\sigma_1, b, \sigma_2, 2\}, \{\sigma_1, c, \sigma_3, 3\}, \{\sigma_1, d, \sigma_0, 4\}, \{\sigma_2, a, \sigma_0, 0\}, \{\sigma_2, b, \sigma_1, 1\}, \{\sigma_2, c, \sigma_2, 2\}, \{\sigma_2, d, \sigma_3, 3\}, \{\sigma_3, a, \sigma_0, 4\}, \{\sigma_3, b, \sigma_0, 0\}, \{\sigma_3, c, \sigma_1, 1\}, \{\sigma_3, d, \sigma_2, 3\}\}$.

   c) M_3 : Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$, símbolos de entrada: $\tau = \{a, b, c, d\}$, símbolos de salida: $\delta = \{0, 1\}$, estado inicial: $\sigma^* = \sigma_2$, función de estado siguiente y de salida: $\{\{\sigma_0, a, \sigma_2, 0\}, \{\sigma_0, b, \sigma_1, 1\}, \{\sigma_0, c, \sigma_1, 1\}, \{\sigma_0, d, \sigma_2, 1\}, \{\sigma_1, a, \sigma_3, 0\}, \{\sigma_1, b, \sigma_1, 1\}, \{\sigma_1, c, \sigma_0, 0\}, \{\sigma_1, d, \sigma_3, 1\}, \{\sigma_2, a, \sigma_0, 1\}, \{\sigma_2, b, \sigma_3, 1\}, \{\sigma_2, c, \sigma_0, 0\}, \{\sigma_2, d, \sigma_2, 1\}, \{\sigma_3, a, \sigma_1, 0\}, \{\sigma_3, b, \sigma_3, 0\}, \{\sigma_3, c, \sigma_0, 1\}, \{\sigma_3, d, \sigma_2, 1\}\}$.

   d) M_4 : Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$, símbolos de entrada: $\tau = \{a, b, c, d\}$, símbolos de salida: $\delta = \{0, 1\}$, estado inicial: $\sigma^* = \sigma_2$, función de estado siguiente y de salida: $\{\{\sigma_0, a, \sigma_2, 0\}, \{\sigma_0, b, \sigma_1, 1\}, \{\sigma_0, c, \sigma_1, 1\}, \{\sigma_0, d, \sigma_2, 0\}, \{\sigma_1, a, \sigma_3, 0\}, \{\sigma_1, b, \sigma_1, 1\}, \{\sigma_1, c, \sigma_0, 0\}, \{\sigma_1, d, \sigma_3, 0\}, \{\sigma_2, a, \sigma_0, 0\}, \{\sigma_2, b, \sigma_3, 0\}, \{\sigma_2, c, \sigma_0, 0\}, \{\sigma_2, d, \sigma_2, 0\}, \{\sigma_3, a, \sigma_1, 1\}, \{\sigma_3, b, \sigma_3, 0\}, \{\sigma_3, c, \sigma_0, 0\}, \{\sigma_3, d, \sigma_2, 0\}\}$.

2) Recurriendo a los diagramas obtenidos en el ejercicio 1, devuelva las seis componentes de cada una de las máquinas de estado finito, empleando la instrucción **`DiagramaToMaquina`** de *VilCretas*.

3) Use el comando **`StringSalida`** para hallar la hilera de símbolos de salida sobre las máquinas del ejercicio 1, considerando la hilera de símbolos de entrada: **`RandomChoice[{a, b, c, d}, 20]`**. Retorne el recorrido a través de la opción "**`trace->True`**".

4) Resuelva paso a paso, por medio de una máquina de estado finito, las sumas:

   a) $11001111001 + 01111110101$

   b) $101010101110101 + 1010001010111$

   c) $101010110111110111 + 1011111101111000111$

   Sugerencia: emplee las sentencias **`MaquinaSumadoraBinarios`** y **`SumaBinaria`**.

5) Genere el diagrama de transición normal y reducido, con y sin color, de los siguientes autómatas de estado finito:

   a) A_1 : Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}$, símbolos de entrada: $\tau = \{a, b, c, d\}$, estado inicial: $\sigma^* = \sigma_3$, función de transición de estados: $\{\{\sigma_0, a, \sigma_2\}, \{\sigma_0, b, \sigma_1\}, \{\sigma_0, c, \sigma_1\}, \{\sigma_0, d, \sigma_2\}, \{\sigma_1, a, \sigma_3\}, \{\sigma_1, b, \sigma_1\}, \{\sigma_1, c, \sigma_0\}, \{\sigma_1, d, \sigma_3\}, \{\sigma_2, a, \sigma_0\}, \{\sigma_2, b, \sigma_3\}, \{\sigma_2, c, \sigma_0\}, \{\sigma_2, d, \sigma_2\}, \{\sigma_3, a, \sigma_1\}, \{\sigma_3, b, \sigma_3\}, \{\sigma_3, c, \sigma_0\}, \{\sigma_3, d, \sigma_2\}\}$ y estados aceptados: $\{\sigma_1\}$.

   b) A_2 : Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, símbolos de entrada: $\tau = \{a, b, c, d\}$, estado inicial: $\sigma^* = \sigma_4$, función de transición de estados: $\{\{\sigma_0, a, \sigma_0\}, \{\sigma_0, b, \sigma_1\}, \{\sigma_0, c, \sigma_2\}, \{\sigma_0, d, \sigma_3\}, \{\sigma_1, a, \sigma_4\}, \{\sigma_1, b, \sigma_0\}, \{\sigma_1, c, \sigma_4\}, \{\sigma_1, d, \sigma_3\}, \{\sigma_2, a, \sigma_2\}, \{\sigma_2, b, \sigma_1\}, \{\sigma_2, c, \sigma_0\}, \{\sigma_2, d, \sigma_3\}, \{\sigma_3, a, \sigma_4\}, \{\sigma_3, b, \sigma_1\}, \{\sigma_3, c, \sigma_0\}, \{\sigma_3, d, \sigma_2\}, \{\sigma_4, a, \sigma_1\}, \{\sigma_4, b, \sigma_1\}, \{\sigma_4, c, \sigma_0\}, \{\sigma_4, d, \sigma_4\}\}$ y estados aceptados: $\{\sigma_2, \sigma_4\}$.

   c) A_3 : Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7, \sigma_8, \sigma_9, \sigma_{10}, \sigma_{11}, \sigma_{12}, \sigma_{13}, \sigma_{14}, \sigma_{15}, \sigma_{16}, \sigma_{17}, \sigma_{18}, \sigma_{19}, \sigma_{20}, \sigma_{21}, \sigma_{22}, \sigma_{23}, \sigma_{24}\}$, símbolos de entrada: $\tau = \{a, b, c\}$, estado inicial: $\sigma^* = \sigma_0$, función de transición de estados: $\{\{\sigma_0, a, \sigma_1\}, \{\sigma_0, b, \sigma_8\}, \{\sigma_0, c, \sigma_{18}\}, \{\sigma_1, a, \sigma_{24}\}, \{\sigma_1, b, \sigma_2\}, \{\sigma_1, c, \sigma_6\}, \{\sigma_2, a, \sigma_{24}\}, \{\sigma_2, b, \sigma_3\}, \{\sigma_2, c, \sigma_5\}, \{\sigma_3, a, \sigma_{24}\}, \{\sigma_3, b, \sigma_{24}\}, \{\sigma_3, c, \sigma_4\}, \{\sigma_4, a, \sigma_{24}\}, \{\sigma_4, b, \sigma_{24}\}, \{\sigma_4, c, \sigma_{24}\}, \{\sigma_5, a, \sigma_{24}\}, \{\sigma_5, b, \sigma_4\}, \{\sigma_5, c, \sigma_{24}\}, \{\sigma_6, a, \sigma_{24}\}, \{\sigma_6, b, \sigma_7\}, \{\sigma_6, c, \sigma_{24}\}, \{\sigma_7, a, \sigma_{24}\}, \{\sigma_7, b, \sigma_4\}, \{\sigma_7, c, \sigma_{24}\}, \{\sigma_8, a, \sigma_9\}, \{\sigma_8, b, \sigma_{12}\}, \{\sigma_8, c, \sigma_{15}\}, \{\sigma_9, a, \sigma_{24}\}, \{\sigma_9, b, \sigma_{10}\}, \{\sigma_9, c, \sigma_{11}\}, \{\sigma_{10}, a, \sigma_{24}\}, \{\sigma_{10}, b, \sigma_{24}\}, \{\sigma_{10}, c, \sigma_4\}, \{\sigma_{11}, a, \sigma_{24}\}, \{\sigma_{11}, b, \sigma_4\}, \{\sigma_{11}, c, \sigma_{24}\}, \{\sigma_{12}, a, \sigma_{13}\}, \{\sigma_{12}, b, \sigma_{24}\}, \{\sigma_{12}, c, \sigma_{14}\}, \{\sigma_{13}, a, \sigma_{24}\}, \{\sigma_{13}, b, \sigma_{24}\}, \{\sigma_{13}, c, \sigma_4\}, \{\sigma_{14}, a, \sigma_4\}, \{\sigma_{14}, b, \sigma_{24}\}, \{\sigma_{14}, c, \sigma_{24}\}, \{\sigma_{15}, a, \sigma_{16}\}, \{\sigma_{15}, b, \sigma_{17}\}, \{\sigma_{15}, c, \sigma_{24}\}, \{\sigma_{16}, a, \sigma_{24}\}, \{\sigma_{16}, b, \sigma_4\}, \{\sigma_{16}, c, \sigma_{24}\}, \{\sigma_{17}, a, \sigma_4\}, \{\sigma_{17}, b, \sigma_{24}\}, \{\sigma_{17}, c, \sigma_{24}\}, \{\sigma_{18}, a, \sigma_{19}\}, \{\sigma_{18}, b, \sigma_{21}\}, \{\sigma_{18}, c, \sigma_{24}\}, \{\sigma_{19}, a, \sigma_{24}\}, \{\sigma_{19}, b, \sigma_{20}\}, \{\sigma_{19}, c, \sigma_{24}\}, \{\sigma_{20}, a, \sigma_{24}\}, \{\sigma_{20}, b, \sigma_4\}, \{\sigma_{20}, c, \sigma_{24}\}, \{\sigma_{21}, a, \sigma_{22}\}, \{\sigma_{21}, b, \sigma_{23}\}, \{\sigma_{21}, c, \sigma_{24}\}, \{\sigma_{22}, a, \sigma_{24}\}, \{\sigma_{22}, b, \sigma_4\}, \{\sigma_{22}, c, \sigma_{24}\}, \{\sigma_{23}, a, \sigma_4\}, \{\sigma_{23}, b, \sigma_{24}\}, \{\sigma_{23}, c, \sigma_{24}\}, \{\sigma_{24}, a, \sigma_{24}\}, \{\sigma_{24}, b, \sigma_{24}\}, \{\sigma_{24}, c, \sigma_{24}\}\}$ y estados aceptados: $\{\sigma_4\}$.

   d) A_4 : Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, símbolos de entrada: $\tau = \{a, b, c\}$, estado inicial: $\sigma^* = \sigma_2$, función de transición de estados: $\{\{\sigma_0, a, \{\sigma_0, \sigma_2\}\}, \{\sigma_0, b, \{\sigma_3, \sigma_4\}\}, \{\sigma_0, c, \{\sigma_0, \sigma_4\}\}, \{\sigma_1, a, \{\sigma_2, \sigma_3\}\}, \{\sigma_1, b, \{\sigma_1, \sigma_4\}\}, \{\sigma_1, c, \{\}\}, \{\sigma_2, a, \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_2, b, \{\}\}, \{\sigma_2, c, \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_3, a, \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_3, b, \{\sigma_1\}\}, \{\sigma_3, c, \{\sigma_1, \sigma_4\}\}, \{\sigma_4, a, \{\sigma_0, \sigma_2, \sigma_3\}\}, \{\sigma_4, b, \{\sigma_0, \sigma_2\}\}, \{\sigma_4, c, \{\}\}\}$ y estados aceptados: $\{\sigma_1, \sigma_2, \sigma_3\}$.

   e) A_5 : Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, símbolos de entrada: $\tau = \{a, b, c, d\}$, estado inicial: $\sigma^* = \sigma_2$, función de transición de estados: $\{\{\sigma_0, a, \{\sigma_2\}\}, \{\sigma_0, b, \{\}\}, \{\sigma_0, c, \{\sigma_0, \sigma_4\}\}, \{\sigma_0, d, \{\sigma_0, \sigma_4\}\}, \{\sigma_1, a, \{\sigma_2, \sigma_4\}\}, \{\sigma_1, b, \{\}\}, \{\sigma_1, c, \{\sigma_1\}\}, \{\sigma_1, d, \{\sigma_1\}\}, \{\sigma_2, a, \{\sigma_0\}\}, \{\sigma_2, b, \{\sigma_1, \sigma_2\}\}, \{\sigma_2, c, \{\sigma_0, \sigma_4\}\}, \{\sigma_2, d, \{\sigma_0, \sigma_4\}\}, \{\sigma_3, a, \{\sigma_0, \sigma_1, \sigma_2, \sigma_3\}\}, \{\sigma_3, b, \{\}\}, \{\sigma_3, c, \{\}\}, \{\sigma_3, d, \{\sigma_4\}\}, \{\sigma_4, a, \{\sigma_0, \sigma_2\}\}, \{\sigma_4, b, \{\sigma_2\}\}, \{\sigma_4, c, \{\}\}, \{\sigma_4, d, \{\}\}\}$ y estados aceptados: $\{\sigma_1, \sigma_3\}$.

   f) A_6 : Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, símbolos de entrada: $\tau = \{a, b, c, d\}$, estado inicial: $\sigma^* = \sigma_1$, función de transición de estados: $\{\{\sigma_0, a, \{\sigma_2, \sigma_4\}\}, \{\sigma_0, b, \{\sigma_1, \sigma_4\}\}, \{\sigma_0, c, \{\sigma_0, \sigma_2\}\}, \{\sigma_0, d, \{\sigma_4\}\}, \{\sigma_1, a, \{\sigma_2\}\}, \{\sigma_1, b, \{\sigma_1\}\}, \{\sigma_1, c, \{\sigma_2\}\}, \{\sigma_1, d, \{\sigma_1, \sigma_2\}\}, \{\sigma_2, a, \{\sigma_2, \sigma_4\}\}, \{\sigma_2, b, \{\sigma_0, \sigma_2\}\}, \{\sigma_2,$

c, $\{\sigma_1, \sigma_2, \sigma_4\}$}, {$\sigma_2$, d, {}}, {$\sigma_3$, a, $\{\sigma_0, \sigma_2, \sigma_3\}$}, {$\sigma_3$, b, $\{\sigma_0, \sigma_4\}$}, {σ_3, c, {}}, {σ_3, d, $\{\sigma_0, \sigma_1\}$}, {σ_4, a, {}}, {σ_4, b, $\{\sigma_4\}$}, {σ_4, c, $\{\sigma_4\}$}, {σ_4, d, $\{\sigma_4\}$}} y estados aceptados: $\{\sigma_1, \sigma_2\}$.

Cambie a "rectangular" la colocación de los estados en los diagramas anteriores. Sugerencia: utilice las instrucciones **`Automata`** y **`AutomataToDiagrama`** del paquete *VilCretas*.

6) Los autómatas del ejercicio anterior, ¿tienen un lenguaje finito? Sugerencia: use el comando **`LenguajeFinitoQ`**.

7) Determine el número de hileras de símbolos de entrada de longitud menor o igual a cinco, aceptadas por cada uno de los autómatas del ejercicio 5. Sugerencia: emplee la opción "**`limite -> True`**" de la sentencia **`LenguajeCantidad`**.

8) Devuelva elementos del lenguaje de cada uno de los autómatas compartidos en el ejercicio 5, de longitud menor o igual a cinco. Sugerencia: use la opción "**`limite -> True`**" del comando **`LenguajeStrings`**.

9) Mediante la sentencia **`ComponentesAutomata`** retorne las cinco componentes de cada uno de los autómatas de estado finito expuestos en el ejercicio 5.

10) La hilera de símbolos de entrada **`RandomChoice[alphabet[`**A_j**`], 5]`** ¿es aceptada?, con $j \in \{1, 2, 3, 4, 5, 6\}$. Sugerencia: emplee la instrucción **`StringAceptadaQ`** del paquete *VilCretas*.

11) Construya recurriendo al comando **`AutomataExactamente`**, un autómata que acepte hileras con exactamente:

   a) una "a", dos "b" y una "c".

   b) dos "a", dos "b" y dos "c".

   Halle sus cinco componentes.

12) Encuentre a través de la sentencia **`AutomataDeterministicoEquivalente`**, un autómata de estado finito determinístico equivalente a A_j, con $j \in \{4, 5, 6\}$.

13) Procese la hilera **`RandomChoice[{0, 1}, 20]`** con veinte pasos, en las siguientes máquinas de *Turing*:

   a) Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2\}$, símbolos de entrada: $\tau = \{1\}$, estado inicial: $\sigma^* = \sigma_1$, función de transición: {$\{\sigma_0, 0, \sigma_1, 0, -1\}$, $\{\sigma_0, 1, \sigma_2, 0, 0\}$, $\{\sigma_1, 0, \sigma_1, 1, -1\}$, $\{\sigma_1, 1, \sigma_1, 1, 1\}$, $\{\sigma_2, 0, \sigma_1, 1, 1\}$, $\{\sigma_2, 1, \sigma_0, 1, 0\}$}, estados aceptados: $\{\sigma_2\}$, símbolos de cinta: $\Gamma = \{0, 1\}$ y el símbolo espacio en blanco: 0.

   b) Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4\}$, símbolos de entrada: $\tau = \{0, 1\}$, estado inicial: $\sigma^* = \sigma_0$, función de transición: {$\{\sigma_0, 0, \sigma_1, X, -1\}$, $\{\sigma_0, Y, \sigma_3, Y, 0\}$, $\{\sigma_1, 0, \sigma_2, 0, -1\}$, $\{\sigma_1, 1, \sigma_2, Y, 1\}$, $\{\sigma_1, Y, \sigma_1, Y, 1\}$, $\{\sigma_2, 0, \sigma_0, 0, 1\}$, $\{\sigma_2, X, \sigma_0, X, -1\}$, $\{\sigma_2, Y, \sigma_1, Y, -1\}$, $\{\sigma_3, Y, \sigma_3, Y, 0\}$, $\{\sigma_3, B, \sigma_2, B, 0\}$}, estados aceptados: $\{\sigma_3\}$, símbolos de cinta: $\Gamma = \{0, 1, X, Y, B\}$ y el símbolo espacio en blanco: B.

14) Determine si la hilera **`RandomChoice[{0, 1}, 20]`** es aceptada sobre las máquinas de *Turing* del ejercicio anterior. Sugerencia: utilice **`MTStringAceptadaQ`**.

15) Realice las sumas siguientes mediante una máquina de *Turing*:

   a) $16 + 50$
   b) $15 + 100$

16) Copie a través de una máquina de *Turing*:

   a) "matemática" cinco veces.
   b) "discreta" seis veces.

17) Sea la máquina de *Turing*:

   - Estados: $\sigma = \{\sigma_0, \sigma_1, \sigma_2\}$
   - Símbolos de entrada: $\tau = \{1\}$
   - Estado inicial: $\sigma^* = \sigma_1$
   - Función de transición: {{σ_0, 0, σ_1, 0, -1}, {σ_0, 1, σ_2, 0, 0}, {σ_1, 0, σ_1, 1, -1}, {σ_1, 1, σ_0, 1, 1}, {σ_2, 0, σ_1, 1, 1}, {σ_2, 1, σ_0, 1, 0}}
   - Estados aceptados: {σ_2}
   - Símbolos de cinta: $\Gamma = \{0, 1\}$
   - Símbolo espacio en blanco: 0

   Halle recurriendo a la instrucción **`MTEncuentraHilerasAceptadas`** del paquete *VilCretas*, una serie de listas con un máximo de cinco hileras aceptadas de longitud menor o igual a cien.

Capítulo 10

Gramáticas y lenguajes con *VilCretas*

En este capítulo se comparten **ocho** comandos del paquete *VilCretas*, cada uno desarrollado para emprender procesos de enseñanza y aprendizaje en el tema de lenguajes y gramáticas libres del contexto y regulares. Se brindan instrucciones de distinta índole, abarcando conceptos básicos, notaciones destinadas a representar gramáticas, medios para la derivación de lenguajes y la relación existente entre los autómatas de estado finito y las gramáticas regulares. Algunos de los comandos incorporados en *VilCretas*, tienen como objetivo ayudar al alumno a mejorar su desarrollo cognitivo, dada la complejidad abstracta de este tema y su difícil abordaje didáctico con recursos tradicionales.

1) **GramaticaLibreContextoQ**: retorna "`True`" si al recibir como parámetro una **gramática** "G" ésta es **libre del contexto**, o "`False`", en caso contrario. La gramática "G" se escribe como un **vector** de "strings" donde cada hilera es de la **forma** "A -> B", siendo la expresión anterior una **regla de composición o producción**. Si existen "n" composiciones que inician con "A", **se expresan así**: "A -> B1|B2|...|Bn". Por defecto, los **símbolos no terminales** se asumen como **letras mayúsculas** del abecedario, los **símbolos terminales** como **cualquier otro carácter** y el **símbolo no terminal inicial** es **igual** a "S", además, "ε" representa la **hilera vacía**.

Sintaxis: `GramaticaLibreContextoQ[G]`.

Ejemplo 10.1

Determine si la gramática dada a continuación es libre del contexto.

- Símbolos no terminales: $\sigma = \{S, A, B, C, D\}$
- Símbolos terminales: $\tau = \{a, b, c\}$
- Producciones o reglas de composición: $P = \{S \to aAB, S \to aBBB, A \to aAC, A \to CCaC, A \to aaaCC, B \to CCDc, AC \to CCb\}$
- Símbolo no terminal inicial: $\sigma^* = S$

Solución:

En *Mathematica*:

```
In[ ] :=
GramaticaLibreContextoQ[{"S -> aAB|aBBB", "A -> aAC|CCaC|aaaCC",
"B -> CCDc", "AC→CCb"}]
Out[ ] :=
False
```

N La gramática no es libre del contexto. En el software cada producción se puede escribir con el símbolo " `->` ", o bien, "→".

Ejemplo 10.2

Considere la gramática:

- Símbolos no terminales: $\sigma = \{S, A, B, C\}$
- Símbolos terminales: $\tau = \{a, b\}$
- Producciones o reglas de composición: $P = \{S \rightarrow a, S \rightarrow aB, A \rightarrow aA, A \rightarrow aC, A \rightarrow aB, B \rightarrow \varepsilon, C \rightarrow b\}$
- Símbolo no terminal inicial: $\sigma^* = S$

¿Es libre del contexto?

Solución:

In[ ] :=
```
GramaticaLibreContextoQ[{"S->a|aB", "A->aA|aC|aB", "B->ε", "C→b"}]
```

Out[ ] :=
True

N Sí es una gramática libre del contexto. Se recuerda al lector que el símbolo "ε" corresponde a la hilera vacía.

Explicación en video

2) **GramaticaRegularQ**: retorna "`True`" si al recibir como parámetro una **gramática** "G" ésta es **regular**, o "`False`", en caso contrario. La gramática "G" se escribe como un **vector** de "strings" donde cada hilera es de la **forma** "A -> B", siendo la expresión anterior una **regla de composición o producción**. Si existen "n" composiciones que inician con "A", **se expresan así**: "A -> B1|B2|...|Bn". Por defecto, los **símbolos no terminales** se asumen como **letras mayúsculas** del abecedario, los **símbolos terminales** como **cualquier otro carácter** y el **símbolo no terminal inicial** es **igual** a "S", además, "ε" representa la **hilera vacía**.

Sintaxis: `GramaticaRegularQ[G]`.

Ejemplo 10.3

¿La gramática adjunta es regular?

- Símbolos no terminales: $\sigma = \{S, A, B, C, D\}$

- Símbolos terminales: $\tau = \{a, b, c\}$
- Producciones o reglas de composición: $P = \{S \to aA, S \to aB, A \to aA, A \to aC, A \to a, B \to cD, C \to bC\}$
- Símbolo no terminal inicial: $\sigma^* = S$

Solución:

En el software al recurrir al comando **`GramaticaRegularQ`**, se tiene:

In[ ] :=

```
GramaticaRegularQ[{"S -> aA|aB", "A -> aA|aC|a", "B -> cD", "C→bC"}]
```

Out[ ] :=

True

Sí es una gramática regular.

Ejemplo 10.4

¿Es regular la gramática detallada a continuación?

- Símbolos no terminales: $\sigma = \{S, A, B, C\}$
- Símbolos terminales: $\tau = \{a, b, c\}$
- Producciones o reglas de composición: $P = \{S \to aB, S \to aA, A \to aA, A \to a, A \to aC, A \to Ab, B \to c\}$
- Símbolo no terminal inicial: $\sigma^* = S$

Solución:

En *Mathematica*:

In[ ] :=

```
GramaticaRegularQ[{"S -> aB|aA", "A -> aA|a|aC|Ab", "B -> c"}]
```

Out[ ] :=

False

No es una gramática regular.

Explicación en video

3) **ElementoLenguajeQ**: función booleana encargada de **determinar** si una **hilera de símbolos terminales** "string" en una **gramática libre del contexto** "G", es un **elemento del lenguaje**. Presenta la **opción** "**derivaciones -> True**" que muestra **paso a paso** las **derivaciones** necesarias para retornar el valor lógico. La gramática "G" se escribe como un **vector** de "strings" donde cada hilera es de la **forma** "A -> B", siendo la expresión anterior una **regla de composición o producción**. Si existen "n" composiciones que inician con "A", **se expresan así**: "A -> B1|B2|...|Bn". Por defecto, los **símbolos no terminales** se asumen como **letras mayúsculas** del abecedario, los **símbolos terminales** como **cualquier otro carácter** y el **símbolo no terminal inicial** es **igual** a "S", además, "ε" representa la **hilera vacía**.

Sintaxis: `ElementoLenguajeQ[G, string]`, o bien,

```
ElementoLenguajeQ[G, string, derivaciones->True]
```

Ejemplo 10.5

Establezca sobre un "string" de símbolos terminales seudoaleatorio de longitud diez, si es un elemento del lenguaje derivado por la gramática:

- Símbolos no terminales: $\sigma = \{S, A, B, C, D\}$
- Símbolos terminales: $\tau = \{a, b, c\}$
- Producciones o reglas de composición: $P = \{S \to aAB, S \to aB, A \to aAC, A \to aCb, A \to aC, B \to Dc, C \to b\}$
- Símbolo no terminal inicial: $\sigma^* = S$

Muestre paso a paso el proceso.

Solución:

Para crear el "string" se utilizarán las instrucciones **RandomChoice**, **ToString** y **StringJoin**:

In[ ] :=
```
α = StringJoin[RandomChoice[ToString/@{a, b, c}, 10]]
ElementoLenguajeQ[{"S->aAB|aB", "A->aAC|aCb|aC", "B->Dc", "C->b"},
α]
ElementoLenguajeQ[{"S->aAB|aB", "A->aAC|aCb|aC", "B->Dc", "C->b"},
α, derivaciones->True]
```

Out[ ] :=
```
bbcbcbbcca
False
1 {bbcbcbCcca, bbcbcCbcca, bbcCcbbcca, bCcbcbbcca, Cbcbcbbcca}
2 {bbcbcCCcca, bbcCcbCcca, bbcCcCbcca, bCcbcbCcca, bCcbcCbcca, bCcCcbbcca, CbcbcbCcca,
CbcbcCbcca, CbcCcbbcca, CCcbcbbcca}
3 {bbcCcCCcca, bCcbcCCcca, bCcCcbCcca, bCcCcCbcca, CbcbcCCcca, CbcCcbCcca, CbcCcCbcca,
CCcbcbCcca, CCcbcCbcca, CCcCcbbcca}
4 {bCcCcCCcca, CbcCcCCcca, CCcbcCCcca, CCcCcbCcca, CCcCcCbcca}
```

5 {CCcCcCCcca}
6 {}
False

N **ToString/@** convierte a "string" cada uno de los elementos del conjunto $\{a,b,c\}$, **RandomChoice** selecciona diez términos de manera seudoaleatoria y **StringJoin** forma una sola hilera, con cada una de las componentes de la lista (en este ejemplo, corresponde a: bbcbcbbcca). Se deduce que el "string" no es aceptado. Además, la opción "**derivaciones -> True**" ha permitido visualizar iteración por iteración, las derivaciones necesarias involucradas en la respuesta final.

Ejemplo 10.6

A través de una hilera seudoaleatoria de longitud diez de símbolos terminales, determine si pertenece al lenguaje definido por la gramática:

- Símbolos no terminales: $\sigma = \{S, A, B, C, D\}$
- Símbolos terminales: $\tau = \{a, b, c\}$
- Producciones o reglas de composición: $P = \{S \to aAB, S \to aB, A \to aAC, A \to aC, A \to a, A \to Ab, B \to Dc, C \to b, C \to CE, D \to Dc\}$
- Símbolo no terminal inicial: $\sigma^* = S$

Observe el procedimiento paso a paso.

Solución:

En el software:

In[ ] :=

```
α = StringJoin[RandomChoice[ToString/@{a, b, c}, 10]]
ElementoLenguajeQ[{"S -> aAB|aB", "A -> aAC|aC|a|Ab", "B -> Dc", "C -> b|CE",
"D -> Dc"}, α]
ElementoLenguajeQ[{"S -> aAB|aB", "A -> aAC|aC|a|Ab", "B -> Dc", "C -> b|CE",
"D -> Dc"}, α, derivaciones -> True]
```

Out[ ] :=

abccacbccb
False
1 {abccAcbccb, Abccacbccb, abccacbccC, abccacCccb, aCccacbccb}
2 {AbccAcbccb, abccAcbccC, Abccacbccc, abccAcCccb, AbccacCccb, abccacCccC, Accacbccb, aCccAcbccb, ACccacbccb, aCccacbccC, aCccacCccb}
3 {AbccAcbccC, AbccAcCccb, abccAcCccC, AbccacCccC, AccAcbccb, AccacbccC, AccacCccb, ACccAcbccb, aCccAcbccC, ACccacbccC, aCccAcCccb, ACccacCccb, aCccacCccC}
4 {AbccAcCccC, AccAcbccC, AccAcCccb, AccacCccC, ACccAcbccC, ACccAcCccb, aCccAcCccC, ACccacCccC}
5 {AccAcCccC, ACccAcCccC}

6 {}
False

Se concluye que abccacbccb no forma parte del lenguaje vinculado con la gramática libre del contexto.

Explicación en video

4) **NotationBNF**: **recibe** una **gramática libre del contexto** "G" y la **retorna** en **notación** de **Backus-Naur**. La gramática "G" se escribe como un **vector** de "strings" donde cada hilera es de la **forma** "A -> B", siendo la expresión anterior una **regla de composición o producción**. Si existen "n" composiciones que inician con "A", **se expresan así**: "A -> B1|B2|...|Bn". Por defecto, los **símbolos no terminales** se asumen como **letras mayúsculas** del abecedario, los **símbolos terminales** como **cualquier otro carácter** y el **símbolo no terminal inicial** es **igual** a "S", además, "ε" representa la **hilera vacía**.

Sintaxis: `NotationBNF[G]`.

Ejemplo 10.7

Exprese en notación de *Backus-Naur* la gramática:

- Símbolos no terminales: $\sigma = \{S, A, C, J, P, V\}$
- Símbolos terminales: $\tau = \{e, f, m, p, r, s, x\}$
- Producciones o reglas de composición: $P = \{S \to JP, J \to r, J \to m, P \to VCA, V \to e, V \to x, C \to p, A \to s, A \to f\}$
- Símbolo no terminal inicial: $\sigma^* = S$

Solución:

Usando el comando **`NotationBNF`**, se obtiene:

```
In[ ] :=
NotationBNF[{"S→JP", "J -> r|m", "P→VCA", "V→e|x", "C→p", "A→s|f"}]

Out[ ] :=
<S>::=<J><P>
<J>::=r|m
<P>::=<V><C><A>
<V>::=e|x
<C>::=p
<A>::=s|f
σ*=S
```

Ejemplo 10.8

Represente en notación *BNF*, la gramática libre del contexto:

- Símbolos no terminales: $\sigma = \{S, C, D, E, G, N, P\}$
- Símbolos terminales: $\tau = \{-, ., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Producciones o reglas de composición: $P = \{S \to G, S \to N, G \to -P, N \to P, P \to E, P \to D, P \to ED, D \to .E, E \to C, E \to CE, C \to 0, C \to 1, C \to 2, C \to 3, C \to 4, C \to 5, C \to 6, C \to 7, C \to 8, C \to 9\}$
- Símbolo no terminal inicial: $\sigma^* = S$

Solución:

En *Mathematica*:

```
In[ ] :=
NotationBNF[{"S→G|N", "G→-P", "N→P", "P→E|D|ED", "D→.E", "E→C|CE",
"C→0|1|2|3|4|5|6|7|8|9"}]

Out[ ] :=
<S>::=<G>|<N>
<G>::=-<P>
<N>::=<P>
<P>::=<E>|<D>|<E><D>
<D>::=.<E>
<E>::=<C>|<C><E>
<C>::=0|1|2|3|4|5|6|7|8|9
σ*=S
```

Explicación en video

5) **NotationFNormal**: **recibe** una **gramática libre del contexto** "G" y la **retorna** en **notación** "forma normal". La gramática "G" se escribe como un **vector** de "strings" donde cada hilera tiene la **estructura** "A -> B", siendo la expresión anterior una **regla de composición o producción**. Si existen "n" composiciones que inician con "A", **se expresan así**: "A -> B1|B2|...|Bn". Por defecto, los **símbolos no terminales** se asumen como **letras mayúsculas** del abecedario, los **símbolos terminales** como **cualquier otro carácter** y el **símbolo no terminal inicial** es **igual** a "S", además, "ε" representa la **hilera vacía**.

Sintaxis: `NotationFNormal[G]`.

Ejemplo 10.9

Considere la gramática:

- Símbolos no terminales: $\sigma = \{S, A, C, J, P, V\}$
- Símbolos terminales: $\tau = \{e, f, m, p, r, s, x\}$
- Producciones o reglas de composición: $P = \{S \to JP, J \to r, J \to m, P \to VCA, V \to e, V \to x, C \to p, A \to s, A \to f\}$
- Símbolo no terminal inicial: $\sigma^* = S$

obtenga su representación en *forma normal*.

Solución:

En el software:

In[ ] :=

```
NotationFNormal[{"S→JP", "J→r|m", "P→VCA", "V→e|x", "C→p",
"A→s|f"}]
```

Out[ ] :=

Símbolos no terminales: {A, C, J, P, S, V}
Símbolos terminales: {e, f, m, p, r, s, x}
Producciones o reglas de composición: {A→f, A→s, C→p, J→m, J→r, P→VCA, S→JP, V→e, V→x}
σ^*=S

Ejemplo 10.10

Halle la notación *forma normal* de la gramática del ejemplo tras anterior.

Solución:

In[ ] :=

```
NotationFNormal[{"S→G|N", "G→-P", "N→P", "P→E|D|ED", "D→.E",
"E→C|CE", "C→0|1|2|3|4|5|6|7|8|9"}]
```

Out[ ] :=

Símbolos no terminales: {C, D, E, G, N, P, S}
Símbolos terminales: {-, ., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Producciones o reglas de composición: {C→0, C→1, C→2, C→3, C→4, C→5, C→6, C→7, C→8, C→9, D→.E, E→C, E→CE, G→-P, N→P, P→D, P→E, P→ED, S→G, S→N}
σ^*=S

Explicación en video

6) **ElementosLenguaje**: **recibe** una **gramática libre del contexto** "G" y **retorna todas** las **palabras del lenguaje** de **longitud máxima** "m" **obtenidas** por "n" o **menos** derivaciones. Brinda la **opción** "**derivaciones -> True**" que muestra adicionalmente, **todas** las **derivaciones necesarias** durante el **proceso**. La gramática "G" se escribe como un **vector** de "strings" donde cada hilera tiene la **estructura** "A -> B", siendo la expresión anterior una **regla de composición o producción**. Si existen "k" composiciones que inician con "A", **se expresan así**: "A -> B1|B2|...|Bk". Por defecto, los **símbolos no terminales** se asumen como **letras mayúsculas** del abecedario, los **símbolos terminales** como **cualquier otro carácter** y el **símbolo no terminal inicial** es **igual** a "S", además, "ε" representa la **hilera vacía**.

Sintaxis: **ElementosLenguaje[G, n, m]**, o bien, **ElementosLenguaje[G, n, m, derivaciones -> True]**.

Ejemplo 10.11

Encuentre por medio de seis o menos derivaciones, palabras del lenguaje de longitud máxima seis, en la gramática libre del contexto:

- Símbolos no terminales: $\sigma = \{S, A, C, J, P, V\}$
- Símbolos terminales: $\tau = \{e, f, m, p, r, s, x\}$
- Producciones o reglas de composición: $P = \{S \to JP, J \to r, J \to m, P \to VCA, V \to e, V \to x, C \to p, A \to s, A \to f\}$
- Símbolo no terminal inicial: $\sigma^* = S$

Genere las derivaciones iteración por iteración.

Solución:

Al invocar la instrucción **ElementosLenguaje** y su opción "**derivaciones -> True**":

In[ ] :=

```
ElementosLenguaje[{"S→JP", "J→r|m", "P→VCA", "V→e|x", "C→p",
"A→s|f"}, 6, 6]
ElementosLenguaje[{"S→JP", "J -> r|m", "P→VCA", "V→e|x", "C→p",
"A→s|f"}, 6, 6, derivaciones -> True]
```

Out[ ] :=

{mepf, meps, mxpf, mxps, repf, reps, rxpf, rxps}
{{JP}, {JVCA, mP, rP}, {JeCA, JVCf, JVCs, JVpA, JxCA, mVCA, rVCA}, {mepf, meps, mxpf, mxps, repf, reps, rxpf, rxps}, {JeCf, JeCs, JepA, JVpf, JVps, JxCf, JxCs, JxpA, meCA, mVCf, mVCs, mVpA, mxCA, reCA, rVCf, rVCs, rVpA, rxCA}, {Jepf, Jeps, Jxpf, Jxps, meCf, meCs, mepA, mVpf, mVps, mxCf, mxCs, mxpA, reCf, reCs, repA, rVpf, rVps, rxCf, rxCs, rxpA}}
{mepf, meps, mxpf, mxps, repf, reps, rxpf, rxps}

Los elementos del lenguaje de la gramática retornados son: {mepf, meps, mxpf, mxps, repf, reps, rxpf, rxps}.

Ejemplo 10.12

Sea la gramática:

- Símbolos no terminales: $\sigma = \{S, C, D, E, G, N, P\}$
- Símbolos terminales: $\tau = \{-, ., 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Producciones o reglas de composición: $P = \{S \to G, S \to N, G \to -P, N \to P, P \to E, P \to D, P \to ED, D \to .E, E \to C, E \to CE, C \to 0, C \to 1, C \to 2, C \to 3, C \to 4, C \to 5, C \to 6, C \to 7, C \to 8, C \to 9\}$
- Símbolo no terminal inicial: $\sigma^* = S$

determine con un máximo de diez derivaciones, términos del lenguaje de longitud a lo sumo diez. Muestre el proceso paso a paso.

Solución:

En *Mathematica*:

In[ ] :=

```
ElementosLenguaje[{"S→G|N", "G→-P", "N→P", "P→E|D|ED", "D→.E",
"E→C|CE", "C→0|1|2|3|4|5|6|7|8|9"}, 10, 10]
ElementosLenguaje[{"S→G|N", "G→-P", "N→P", "P→E|D|ED", "D→.E",
"E→C|CE", "C→0|1|2|3|4|5|6|7|8|9"}, 10, 10, derivaciones->True]
```

Out[ ] :=

{-.0, -0, .0, 0, -.00, -0.0, -00, .00, 0.0, 00, -.000, -0.00, -00.0, -000, .000, 0.00, 00.0, 000, -.001, -0.01, -00.1, -001, .001, 0.01, 00.1, 001, -.002, -0.02, -00.2, -002, .002, 0.02, 00.2, 002, -.003, -0.03, -00.3, -003, .003, 0.03, 00.3, 003, -.004, -0.04, -00.4, -004, .004, 0.04, 00.4, 004, -.005, -0.05, -00.5, -005, .005, 0.05, 00.5, 005, -.006, -0.06, -00.6, -006, .006, 0.06, 00.6, 006, -.007, -0.07, -00.7, -007, .007, 0.07, 00.7, 007, -.008, -0.08, -00.8, -008, .008, 0.08, 00.8, 008, -.009, -0.09, -00.9, -009, .009, 0.09, 00.9, 009, -.01, -0.1, -01, .01, 0.1, 01, -.010, -0.10, -01.0, -010, .010, 0.10, 01.0, 010, -.011, -0.11, -01.1, -011, .011, 0.11, 01.1, 011, -.012, ..., -9.95, -99.5, -995, .995, 9.95, 99.5, 995, -.996, -9.96, -99.6, -996, .996, 9.96, 99.6, 996, -.997, -9.97, -99.7, -997, .997, 9.97, 99.7, 997, -.998, -9.98, -99.8, -998, .998, 9.98, 99.8, 998, -.999, -9.99, -99.9, -999, .999, 9.99, 99.9, 999}

{{G, N}, {-P, P}, {-D, D, -E, E, -ED, ED}, {-C, C, -CD, CD, -CE, CE, -CED, CED, -.E, .E, -E.E, E.E}, {-0, 0, -0D, 0D, -0E, 0E, -0ED, 0ED, -1, 1, -1D, 1D, -1E, 1E, -1ED, 1ED, -2, 2, -2D, 2D, -2E, 2E, -2ED, 2ED, -3, 3, -3D, 3D, -3E, 3E, -3ED, 3ED, -4, 4, -4D, 4D, -4E, 4E, -4ED, 4ED, -5, 5, -5D, 5D, -5E, 5E, -5ED, 5ED, -6, 6, -6D, 6D, -6E, 6E, -6ED, 6ED, -7, 7, -7D, 7D, -7E, 7E, -7ED, 7ED, -8, 8, -8D, 8D, -8E, 8E, -8ED, 8ED, -9, 9, -9D, 9D, -9E, 9E, -9ED, 9ED, -.C, .C, -CC, CC, -CCD, CCD, -CCE, CCE, -CCED, CCED, -.CE, -C.E, .CE, C.E, -CE.E, CE.E, -E.C, E.C, -E.CE, E.CE}, ..., E.CCC3CE, -E.CCC4C, E.CCC4C, -E.CCC4CE, E.CCC4CE, -E.CCC5C, E.CCC5C, -E.CCC5CE, E.CCC5CE, -E.CCC6C, E.CCC6C, -E.CCC6CE, E.CCC6CE, -E.CCC7C, E.CCC7C, -E.CCC7CE, E.CCC7CE, -E.CCC8C, E.CCC8C, -E.CCC8CE, E.CCC8CE, -E.CCC9C, E.CCC9C, -E.CCC9CE, E.CCC9CE, -E.CCCC0, E.CCCC0, -E.CCCC0E, E.CCCC0E, -E.CCCC1, E.CCCC1, -E.CCCC1E, E.CCCC1E, -E.CCCC2, E.CCCC2, -E.CCCC2E, E.CCCC2E, -E.CCCC3, E.CCCC3, -E.CCCC3E, E.CCCC3E, -E.CCCC4, E.CCCC4, -E.CCCC4E, E.CCCC4E, -E.CCCC5, E.CCCC5, -E.CCCC5E, E.CCCC5E, -E.CCCC6, E.CCCC6, -E.CCCC6E, E.CCCC6E, -E.CCCC7, E.CCCC7, -E.CCCC7E, E.CCCC7E, -E.CCCC8, E.CCCC8, -E.CCCC8E, E.CCCC8E, -E.CCCC9, E.CCCC9, -E.CCCC9E, E.CCCC9E, -E.CCCCCC, E.CCCCCC, -E.CCCCCCE, E.CCCCCCE}}

{-.0, -0, .0, 0, -.00, -0.0, -00, .00, 0.0, 00, -.000, -0.00, -00.0, -000, .000, 0.00, 00.0, 000, -.001, -0.01, -00.1, -001, .001, 0.01, 00.1, 001, -.002, -0.02, -00.2, -002, .002, 0.02, 00.2, 002, -.003, -0.03, -00.3, -003, .003, 0.03, 00.3, 003, -.004, -0.04, -00.4, -004, .004, 0.04, 00.4, 004, -.005, -0.05, -00.5, -005, .005, 0.05, 00.5, 005, -.006, -0.06, -00.6, -006, .006, 0.06, 00.6, 006, -.007, -0.07, -00.7, -007, .007, 0.07, 00.7, 007, -.008, -0.08, -00.8, -008, .008, 0.08, 00.8, 008, -.009, -0.09, -00.9, -009, .009, 0.09, 00.9, 009, -.01, -0.1, -01, .01, 0.1, 01, -.010, -0.10, -01.0, -010, .010, 0.10, 01.0, 010, -.011, -0.11, -01.1, -011, .011, 0.11, 01.1, 011, -.012, ..., -9.95, -99.5, -995, .995, 9.95, 99.5, 995, -.996, -9.96, -99.6, -996, .996, 9.96, 99.6, 996, -.997, -9.97, -99.7, -997, .997, 9.97, 99.7, 997, -.998, -9.98, -99.8, -998, .998, 9.98, 99.8, 998, -.999, -9.99, -99.9, -999, .999, 9.99, 99.9, 999}

La salida no se muestra completa por su tamaño. Solo el conjunto de elementos del lenguaje tiene una cardinalidad igual a 8640.

Explicación en video

7) **RegularToAutomata**: **recibe** una **gramática regular** "G" y a través de ella **construye** un **autómata no determinístico** con su **mismo lenguaje**. Presenta las **opciones**: "**`dfa -> True`**" que **genera**, además, un **autómata determinístico equivalente** y, "**`colores -> True`**" que añade **color** a los diagramas de transición mostrados. La gramática "G" se escribe como un **vector** de "strings" donde cada hilera tiene la **estructura** "A -> B", siendo la expresión anterior una **regla de composición o producción**. Si existen "n" composiciones que inician con "A", **se expresan así**: "A -> B1|B2|...|Bn". Por defecto, los **símbolos no terminales** se asumen como **letras mayúsculas** del abecedario, los **símbolos terminales** como **cualquier otro carácter** y el **símbolo no terminal inicial** es **igual** a "S", además, "ε" representa la **hilera vacía**.

Sintaxis: **`RegularToAutomata[G]`**, o bien, **`RegularToAutomata[G, dfa -> True, colores -> True]`**, pudiendo **prescindir** de cualquiera de las opciones.

Ejemplo 10.13

Halle un autómata de estado finito no determinístico equivalente a la gramática regular:

- Símbolos no terminales: $\sigma = \{S, A, B, C, D\}$
- Símbolos terminales: $\tau = \{a, b, c\}$
- Producciones o reglas de composición: $P = \{S \to aA, S \to aB, A \to aA, A \to aC, A \to a, B \to cD, C \to bC\}$
- Símbolo no terminal inicial: $\sigma^* = S$

Solución:

El comando **`RegularToAutomata`**, resuelve lo solicitado en este ejemplo:

In[ ] :=

```
RegularToAutomata[{"S→aA|aB", "A→aA|aC|a", "B→cD", "C→bC"}]
```

Out[ ] :=

Estados: {A, B, C, D, EA, S}

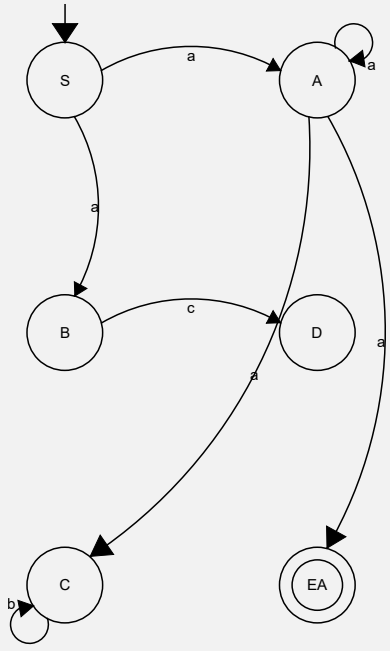
Símbolos de entrada: {a, b, c}

Estado inicial: S

Estados aceptados: {EA}

	a	b	c
A	{A, C, EA}	{}	{}
B	{}	{}	{D}
C	{}	{C}	{}
D	{}	{}	{}
EA	{}	{}	{}
S	{A, B}	{}	{}

Función de transición de estados con otro formato: {{A, a, {A, C, EA}}, {A, b, {}}, {A, c, {}}, {B, a, {}}, {B, b, {}}, {B, c, {D}}, {C, a, {}}, {C, b, {C}}, {C, c, {}}, {D, a, {}}, {D, b, {}}, {D, c, {}}, {EA, a, {}}, {EA, b, {}}, {EA, c, {}}, {S, a, {A, B}}, {S, b, {}}, {S, c, {}}}



Ejemplo 10.14

Dada la siguiente gramática regular:

- Símbolos no terminales: $\sigma = \{S, A, B, C\}$
- Símbolos terminales: $\tau = \{a, b, c\}$
- Producciones o reglas de composición: $P = \{S \to aB, S \to aA, S \to b, S \to cA, A \to aA, A \to a, A \to aC, A \to b, B \to c, C \to a, C \to bA\}$
- Símbolo no terminal inicial: $\sigma^* = S$

determine un autómata de estado finito determinístico equivalente a la gramática y añada color a los diagramas de transición.

Solución:

Al utilizar las opciones "**dfa -> True**" y "**colores -> True**" de **RegularToAutomata**:

In[ ] :=

```
RegularToAutomata[{"S→aB|aA|b|cA", "A→aA|a|aC|b", "B→c", "C→a|bA"},
dfa -> True, colores -> True]
```

Out[ ] :=

Estados: {A, B, C, EA, S}

Símbolos de entrada: {a, b, c}

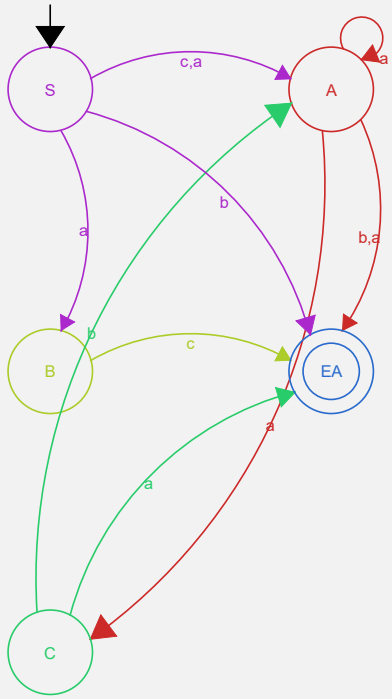
Estado inicial: S

Estados aceptados: {EA}

	a	b	c
A	{A,C,EA}	{EA}	{}
B	{}	{}	{EA}
C	{EA}	{A}	{}
EA	{}	{}	{}
S	{A,B}	{EA}	{A}

Función de transición de estados con otro formato: {{A, a, {A, C, EA}}, {A, b, {EA}}, {A, c, {}}, {B, a, {}}, {B, b, {}}, {B, c, {EA}}, {C, a, {EA}}, {C, b, {A}}, {C, c, {}}, {EA, a, {}}, {EA, b, {}}, {EA, c, {}}, {S, a, {A, B}}, {S, b, {EA}}, {S, c, {A}}}

Diagrama de transición del autómata no determinístico:



Estados del nuevo autómata:

μ_0 = {}

μ_1 = {A}

μ_2 = {B}

μ_3 = {C}

μ_4 = {EA}

μ_5 = {S}

μ_6 = {A, B}

μ_7 = {A, C}

μ_8 = {A, EA}

μ_9 = {A, S}

μ_{10} = {B, C}

μ_{11} = {B, EA}

μ_{12} = {B, S}

μ_{13} = {C, EA}

μ_{14} = {C, S}

μ_{15} = {EA, S}

μ_{16} = {A, B, C}

μ_{17} = {A, B, EA}

μ_{18} = {A, B, S}

μ_{19} = {A, C, EA}

μ_{20} = {A, C, S}

μ_{21} = {A, EA, S}

μ_{22} = {B, C, EA}

μ_{23} = {B, C, S}

μ_{24} = {B, EA, S}

μ_{25} = {C, EA, S}

μ_{26} = {A, B, C, EA}
μ_{27} = {A, B, C, S}
μ_{28} = {A, B, EA, S}
μ_{29} = {A, C, EA, S}
μ_{30} = {B, C, EA, S}
μ_{31} = {A, B, C, EA, S}
Estado inicial: μ_5
Estados aceptados: $\{\mu_4, \mu_8, \mu_{11}, \mu_{13}, \mu_{15}, \mu_{17}, \mu_{19}, \mu_{21}, \mu_{22}, \mu_{24}, \mu_{25}, \mu_{26}, \mu_{28}, \mu_{29}, \mu_{30}, \mu_{31}\}$

	a	b	c
μ_0	μ_0	μ_0	μ_0
μ_1	μ_{19}	μ_4	μ_0
μ_2	μ_0	μ_0	μ_4
μ_3	μ_4	μ_1	μ_0
μ_4	μ_0	μ_0	μ_0
μ_5	μ_6	μ_4	μ_1
μ_6	μ_{19}	μ_4	μ_4
μ_7	μ_{19}	μ_8	μ_0
μ_8	μ_{19}	μ_4	μ_0
μ_9	μ_{26}	μ_4	μ_1
μ_{10}	μ_4	μ_1	μ_4
μ_{11}	μ_0	μ_0	μ_4
μ_{12}	μ_6	μ_4	μ_8
μ_{13}	μ_4	μ_1	μ_0
μ_{14}	μ_{17}	μ_8	μ_1
μ_{15}	μ_6	μ_4	μ_1
μ_{16}	μ_{19}	μ_8	μ_4
μ_{17}	μ_{19}	μ_4	μ_4
μ_{18}	μ_{26}	μ_4	μ_8
μ_{19}	μ_{19}	μ_8	μ_0
μ_{20}	μ_{26}	μ_8	μ_1
μ_{21}	μ_{26}	μ_4	μ_1
μ_{22}	μ_4	μ_1	μ_4
μ_{23}	μ_{17}	μ_8	μ_8
μ_{24}	μ_6	μ_4	μ_8
μ_{25}	μ_{17}	μ_8	μ_1
μ_{26}	μ_{19}	μ_8	μ_4
μ_{27}	μ_{26}	μ_8	μ_8
μ_{28}	μ_{26}	μ_4	μ_8
μ_{29}	μ_{26}	μ_8	μ_1
μ_{30}	μ_{17}	μ_8	μ_8
μ_{31}	μ_{26}	μ_8	μ_8

Diagrama de transición del nuevo autómata:

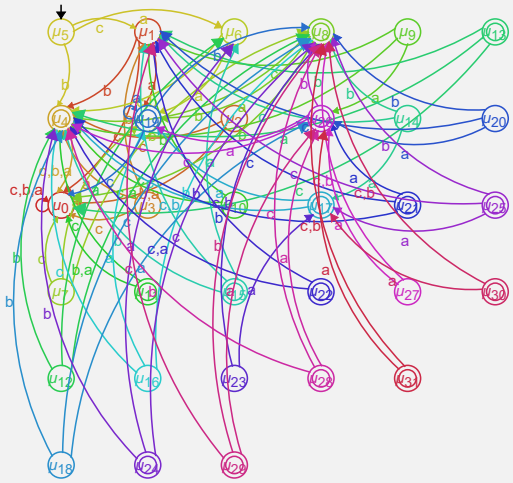
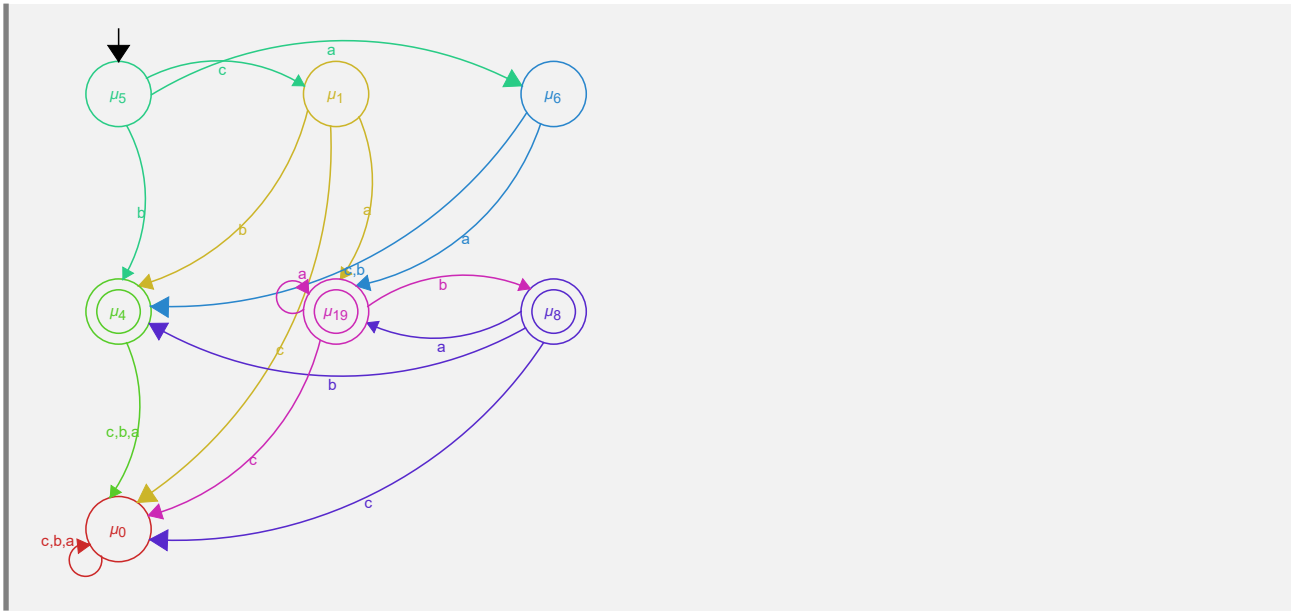


Diagrama de transición reducido del nuevo autómata:



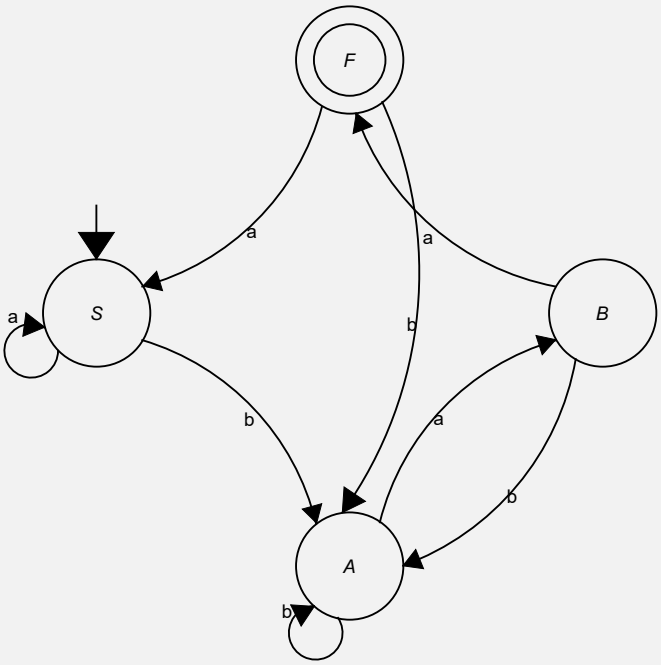
Explicación en video

8) **AutomataToRegular**: **construye** una **gramática regular** con el **mismo lenguaje**, de un **autómata de estado finito determinístico** "A" recibido como parámetro. Retorna la gramática en **notación** "forma normal". Por defecto, los **estados** se asumen como **letras mayúsculas** del abecedario, los **símbolos de entrada** como **cualquier otro carácter** exceptuando "ε" y el **estado inicial** es **igual** a "S".

Sintaxis: `AutomataToRegular[A]`.

Ejemplo 10.15

Construya una gramática regular equivalente al autómata de estado finito determinístico:



Solución:

En *Mathematica,* se creará primero el autómata para emplear después la instrucción **AutomataToRegular**:

In[ ] :=

```
estados = {S, A, B, F};
entradas = {a, b};
inicial = S;
trans = {{S, a, S}, {S, b, A}, {A, a, B}, {A, b, A}, {B, a, F}, {B,
b, A}, {F, a, S}, {F, b, A}};
aceptados = {F};
automata = Automata[estados, entradas, inicial, trans, aceptados];
AutomataToRegular[automata]
```

Out[ ] :=

Símbolos no terminales: {S, A, B, F}

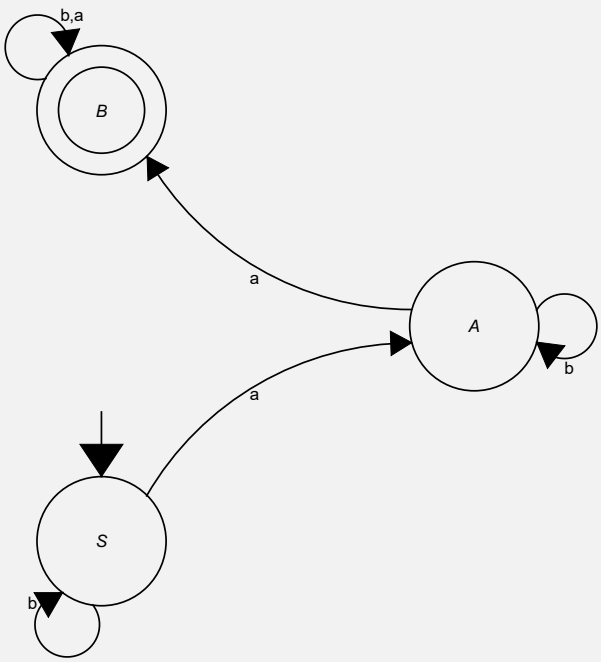
Símbolos terminales: {a, b}

Producciones o reglas de composición: {A→aB, A→bA, B→a, B→aF, B→bA, F→aS, F→bA, S→aS, S→bA}

σ^*=S

Ejemplo 10.16

Considerando el autómata:



encuentre una gramática regular que tenga su mismo lenguaje.

Solución:

En el software:

In[ ] :=

```
automata = Automata[{S, A, B}, {a, b}, S, {{S, a, A}, {S, b, S}, {A,
a, B}, {A, b, A}, {B, a, B}, {B, b, B}}, {B}];
AutomataToRegular[automata]
```

Out[ ] :=

Símbolos no terminales: {S, A, B}
Símbolos terminales: {a, b}
Producciones o reglas de composición: {A→a, A→aB, A→bA, B→a, B→aB, B→b, B→bB, S→aA, S→bS}
σ^*=S

Explicación en video

Aporte pedagógico

Dentro del grupo de comandos socializados en este tema, **destacan** las instrucciones de *VilCretas* **`RegularToAutomata`** y **`AutomataToRegular`**. Ellas componen recursos de **autoaprendizaje** donde el educando **sin necesidad** de consultar al profesor, tiene la alternativa de **revisar** de manera **autónoma** lo realizado como práctica en sus horas de **estudio independiente**. Como se ha insistido en otros apartados, comandos similares a éstos, hacen de *VilCretas* una librería que gesta los medios necesarios, para adquirir conocimientos de matemática discreta sin la forzosa presencia del docente.

Aporte de investigación

Obtener el **lenguaje** asociado a una **gramática libre del contexto o regular** es un ejercicio analítico muy interesante que podría exigir a la población estudiantil, el **estímulo** de habilidades relacionadas con la **observación**, la **comparación**, la **clasificación** y la **descripción**, todas ellas en directa concordancia con el desarrollo de los **procesos del pensamiento**. En este sentido, las instrucciones **`ElementoLenguajeQ`** y **`ElementosLenguaje`** abren posibilidades de creación de **escenarios de aprendizaje**, donde el o la estudiante por **descubrimiento**, **conjeture** el **lenguaje** de ciertas gramáticas elegidas de forma apropiada por el profesor.

Ejercicios

Resuelva los siguientes ejercicios utilizando como apoyo el paquete *VilCretas*.

1) Determine si las siguientes gramáticas son libres del contexto o regulares, haciendo uso de los comandos **`GramaticaLibreContextoQ`** y **`GramaticaRegularQ`**, respectivamente:

   a) $G_1 : \sigma = \{S, A, B, C, D\}$, $\tau = \{a, b, c\}$, $P = \{S \to aA, S \to aB, S \to aD, A \to aD, A \to aC, A \to a, D \to cD, D \to c, C \to bC, C \to b\}$ y $\sigma^* = S$.

   b) $G_2 : \sigma = \{S, A, B, C, D\}$, $\tau = \{a, b, c\}$, $P = \{S \to aD, S \to aC, C \to aC, C \to aD, C \to aB, D \to aA, B \to c, A \to bC, A \to b, C \to c, B \to a\}$ y $\sigma^* = S$.

   c) $G_3 : \sigma = \{S, A, B, C, D\}$, $\tau = \{a, b, c\}$, $P = \{S \to c, S \to aB, A \to aC, C \to aC, A \to aB, B \to c, A \to b, C \to bD, C \to a, C \to b, D \to bD, D \to a\}$ y $\sigma^* = S$.

   d) $G_4 : \sigma = \{S, A, B, C, D\}$, $\tau = \{a, b, c\}$, $P = \{S \to bS, S \to aA, A \to aC, A \to aB, C \to bC, C \to b, B \to bB, B \to c, C \to bD, D \to bA, A \to c, D \to a, D \to bD, D \to b\}$ y $\sigma^* = S$.

   e) $G_5 : \sigma = \{S, A, B, C, D\}$, $\tau = \{a, b, c\}$, $P = \{S \to aB, S \to bB, S \to aD, B \to aA, A \to cC, B \to cD, D \to bC, C \to c, C \to a, C \to b, A \to cB, B \to aB, B \to a, D \to b, A \to c, D \to a, D \to bS, S \to cS, S \to a\}$ y $\sigma^* = S$.

   f) $G_6 : \sigma = \{S, A, B, C, D\}$, $\tau = \{a, b, c\}$, $P = \{S \to aBA, S \to aBC, A \to aAC, A \to aC, A \to Ca, B \to DCc, D \to CCbD, A \to CaDC, D \to a, A \to CaB, C \to CaC, C \to b\}$ y $\sigma^* = S$.

   Con σ el conjunto de símbolos no terminales, τ el conjunto de símbolos terminales, P las reglas de composición de la gramática y σ^* el símbolo no terminal inicial.

2) Usando la instrucción **`ElementoLenguajeQ`** determine si la hilera $\alpha =$**`StringJoin[RandomChoice[ToString /@ {a, b, c}, 5]]`** es un elemento del lenguaje asociado a cada una de las gramáticas del ejercicio anterior.

3) Retorne en notación de *Backus-Naur* y en notación forma normal, cada una de las gramáticas del ejercicio 1. Sugerencia: emplee las instrucciones **`NotationBNF`** y **`NotationFNormal`** del paquete *VilCretas*.

4) Halle palabras del lenguaje de longitud máxima diez, en cada una de las gramáticas del ejercicio 1, obtenidas por diez o menos derivaciones. Muestre las derivaciones. Sugerencia: recurra al uso del comando **`ElementosLenguaje`** y su opción "**`derivaciones ->`** **`True`**".

5) Encuentre un autómata de estado finito no determinístico equivalente a la gramática G_j, con $j \in \{1,2,3,4,5\}$. Halle un autómata determinístico equivalente. Sugerencia: use la instrucción **`RegularToAutomata`** y su opción "**`dfa ->`** **`True`**".

6) Determine una gramática regular cuyo lenguaje se detalla a continuación:

   a) $L(G) =$ {hileras que empiezan con "ab" y finalizan con "ac"}

   b) $L(G) =$ {hileras que poseen la cadena "abacab" en cualquier posición}

   Sugerencia: construya un autómata de estado finito determinístico con el lenguaje $L(G)$ y halle una gramática regular equivalente utilizando la instrucción **`AutomataToRegular`**.

Bibliografía

[1] Ávila, J. (2005). *Estructuras de matemática discreta para computación*. Costa Rica: UNA.

[2] Bournissen, J. (2012). Tecnologías de la información y comunicación en la educación. *Revista Cognición*, *8*(37), 1-15. Recuperado de: http://www.cognicion.net/images/articulos/Cog37/37-2-tecnologias-de-la-informacion-y-la-comunicacion-en-la-educacion.pdf.

[3] Brassard, G. y Bratley, P. (2000). *Fundamentos de algoritmia*. España: Pearson Prentice Hall.

[4] Caballero, M., Migallón, V. y Penadés, J. (2001). *Prácticas de matemática discreta con MAGRADA*. España: Publicaciones Universidad de Alicante.

[5] Cañizares, R., Febles, J. y Estrada, V. (2012). Los objetos de aprendizaje, una tecnología necesaria para las instituciones de la educación superior en Cuba. *ACIMED*, *23*(2), 102-115.

[6] Couturejuzón, L. (2003). Cumplimiento de los principios didácticos en la utilización de un software educativo para la educación superior. *Educación Médica Superior*, *17*(1), 53-57.

[7] Grassman, W. y Tremblay, J. (1997). *Matemática discreta y lógica*. España: Prentice-Hall.

[8] Gutiérrez, A. y Tyner, K. (2012). Educación para los medios, alfabetización mediática y competencia digital. *Comunicar: Revista científica iberoamericana de comunicación y educación* (38), 31-39.

[9] Hastings, K. (2006). *Introduction to the mathematics of operations research with Mathematica*. USA: Chapman & Hall/CRC.

[10] Hopcroft, J., Motwani, R. y Ullman, J. (2002). *Introducción a la teoría de autómatas, lenguajes y computación*. España: Pearson Educación.

[11] Johnsonbaugh, R. (2005). *Matemáticas discretas*. México: Pearson Prentice Hall.

[12] Kolman, B., Busby, R. y Ross, S. (1995). *Estructuras de matemáticas discretas para computación*. México: Prentice-Hall Hispanoamericana.

[13] Lipschutz, S. y Lipson, M. (2004). *2000 problemas resueltos de matemática discreta*. España: Mc. Graw-Hill.

[14] Maeder, R. (2000). *Computer science with Mathematica: theory and practice for science, mathematics, and engineering*. USA: Cambridge University Press.

[15] Mesa, F. (2012). Las tecnologías de la información y la comunicación en la universidad colombiana: evolución y prospectiva. *Revista Historia de la Educación Latinoamericana* (19), 71-90.

[16] Pemmaraju, S. y Skiena, S. (2009). *Computational discrete mathematics: combinatorics and graph theory with Mathematica*. USA: Cambridge University Press.

[17] Peña, I. (2010). Contextualizando la brecha digital en la Educación Superior. *RUSC. Universities and Knowledge Society Journal*, *7*(1), 2-6.

[18] Rodríguez, M. (2013). La educación matemática en la conformación del ciudadano. *Telos: Revista de Estudios Interdisciplinarios en Ciencias Sociales*, *15*(2), 215-230.

[19] Rosen, K. (2007). *Discrete mathematics and its applications*. USA: Mc. Graw-Hill.

[20] Serrano, J. y Narváez, P. (2010). Uso de Software Libre para el Desarrollo de Contenidos Educativos. *Formación universitaria*, *3*(6), 41-50. doi: 10.4067/S0718-50062010000600006.

[21] Shiflet, A y Shiflet, G. (2006). *Introduction to Computational Science: Modeling and Simulation for the Sciences*. USA: Princeton University Press.

[22] Universidad Nacional de Costa Rica (s.a.). *Modelo Pedagógico*. Costa Rica: Universidad Nacional.

[23] Vílchez, E. y González, E. (2014). Percepción estudiantil sobre una metodología asistida por computadora en las áreas cognitivas del álgebra lineal y la matemática discreta. *Revista Digital Matemática, Eduación e Internet*, *14*(1), 1-16.

[24] Vílchez, E. (2015a). Resolución de relaciones de recurrencia con apoyo de *Wolfram Mathematica*. *Revista UNICIENCIA*, *29*(1), 16-41.

[25] Vílchez, E. (2015b). *Estructuras discretas con Mathematica*. México: Alfaomega.

[26] Wolfram *Mathematica* 11: Documentation Center (2016). *Mathematica functions and tutorials.* Obtenido el 19 de setiembre del 2016: http://reference.wolfram.com/language.