TEMA 3

Relaciones y grafos

3.1. Relaciones binarias

Las *relaciones* en matemáticas formalizan las conexiones entre elementos de varios conjuntos. Son importantes en matemáticas y en computación, tanto por sus aplicaciones teóricas (relaciones de orden, equivalencias,...), como por su aplicaciones más prácticas (bases de datos, redes sociales,...). Constantemente encontramos y manejamos relaciones: ordenación de números, congruencias, divisibilidad, relaciones de parentesco, guías telefónicas, directorios de personal,...

En el tema anterior definimos las relaciones binarias como subconjuntos del producto cartesiano de dos conjuntos. Pero es posible establecer relaciones entre elementos de varios conjuntos, es decir, subconjuntos del producto cartesiano de varios conjuntos:

Definición 3.1.1 (Producto cartesiano de varios conjuntos) *El* producto cartesiano *de los conjuntos $A_1, A_2, \ldots, A_n$, denotado $A_1 \times A_2 \times \cdots A_n$, es el conjunto de todas las n-*tuplas ordenadas*:*

$$A_1 \times A_2 \times \cdots \times A_n = \big\{(x_1, x_2, \ldots, x_n); \quad x_j \in A_j, \quad j \in \{1, 2, \ldots, n\}\big\}$$

La denominación n-tupla es genérica, para números concretos, utilizaremos la denominación adecuada: terna, cuádrupla, quíntupla,...

Por otra parte, si los conjuntos del producto son iguales, se puede utilizar una notación abreviada:

$$A \times A = A^2, \qquad A \overset{(n}{\ldots} \times A = A^n$$

Definición 3.1.2 (Relación n-aria) *Una* relación n-aria *en los conjuntos A_1, A_2,..., A_n es cualquier subconjunto $\mathcal{R}$ del producto cartesiano $A_1 \times A_2 \times \cdots A_n$*

$$\mathcal{R} \subseteq A_1 \times A_2 \times \cdots A_n$$

El número n se denomina igualmente grado *o* aridad *de la relación.*

La denominación n-aria es genérica, en cada caso particular utilizaremos la denominación específica, como *binaria* para grado dos o *ternaria* para grado tres, aunque para más de tres elementos, es preferible utilizar la palabra grado: grado cuatro, grado cinco,...

En este curso, nos vamos a centrar en relaciones binarias.

Ejemplo 3.1.3

- Si E es el conjunto de los españoles, podemos considerar la relación $\mathcal{M}$ que contiene a todos los matrimonios: $\mathcal{M}$ es una relación binaria.
- Para $A = \mathbb{R}^+$ y $B = \mathbb{R}$, podemos considerar la relación

$$\mathcal{R} = \{(x, y) \in \mathbb{R}^+ \times \mathbb{R} \mid x = y^2\}$$

- Para $A = \mathbb{R}$ y $B = \mathbb{R}$, podemos considerar la relación

$$\mathcal{E} = \left\{(x, y) \in \mathbb{R} \times \mathbb{R} \middle| \frac{x^2}{9} + \frac{y^2}{4} = 1\right\}$$ □

Para las relaciones binarias, lo más habitual es utilizar la notación *infija* para presentar los pares relacionados: si $\mathcal{R} \subseteq A \times B$ y $(x, y) \in \mathcal{R}$, escribiremos $x\mathcal{R}y$, que se lee x está relacionado (mediante $\mathcal{R}$) con y.

Ejemplo 3.1.4 Si $A = \{1, 2, 3\}$, $B = \{a, b, c\}$, podemos definir la relación $\mathcal{R}$ escribiendo:

$$1\mathcal{R}c, \quad 2\mathcal{R}a, \quad 3\mathcal{R}b.$$

Es decir, $\mathcal{R} = \{(1, c), (2, a), (3, b)\}$. □

Ya conocemos y hemos trabajo con distintas relaciones representadas por símbolos conocidos. Por ejemplo, $x \leq y$ indica que el par (x, y) pertenece a la siguiente relación en cualquier conjunto de números:

$$\{(x, y) \mid x \text{ es menor o igual que } y\}$$

En el primer tema, definimos las relaciones de congruencia, que se denotan con el símbolo $\equiv$. También definimos la relación de divisibilidad entre números enteros, que se denota por el símbolo $|$.

Definición 3.1.5 (Dominio y Rango) *Sea $\mathcal{R}$ una relación binaria de A en B. Llamamos* dominio *de $\mathcal{R}$ al conjunto*

$$\mathrm{Dom}(\mathcal{R}) = \{x \in A \mid \textit{existe } y \in B \textit{ tal que } (x, y) \in \mathcal{R}\}$$

Naturalmente, $\mathrm{Dom}(\mathcal{R}) \subseteq A$

Llamamos rango *de* $\mathcal{R}$ *al conjunto*

$$\mathrm{Ran}(\mathcal{R}) = \{x \in B \mid \textit{existe } y \in A \textit{ tal que } (y, x) \in \mathcal{R}\}$$

Naturalmente, $\mathrm{Ran}(\mathcal{R}) \subseteq B$*. El rango se puede denominar igualmente* Imagen*, y denotarse* $\mathrm{Im}(\mathcal{R})$*.*

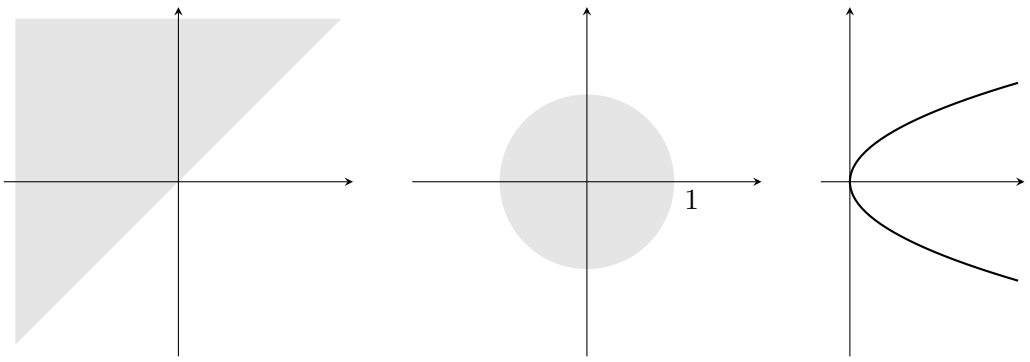
Ejemplo 3.1.6 Para $\mathcal{R} = \{(1,2),(2,3),(3,2),(4,2),(4,5),(5,1)\} \subset \mathbb{N} \times \mathbb{N}$:

$$\mathrm{Dom}(\mathcal{R}) = \{1,2,3,4,5\}, \qquad \mathrm{Ran}(\mathcal{R}) = \{1,2,3,5\}$$

Como vemos, tanto el dominio como el rango pueden estar estrictamente contenidos en los conjuntos en los que se define la relación. □

Representación de relaciones binarias La representación que podamos hacer de una relación depende de los conjuntos entre los que se defina. Por ejemplo, las relaciones entre números pueden representarse gráficamente con regiones del plano.

Ejemplo 3.1.7 Las regiones sombreadas en los siguientes gráficos representan las relaciones de $\mathbb{R}$ en $\mathbb{R}$ que se indican. En el tercer ejemplo, los pares de la relación son los puntos de la parábola.



$x \leq y$ $\qquad$ $x\mathcal{R}y \Leftrightarrow x^2 + y^2 \leq 1$ $\qquad$ $x\mathcal{S}y \Leftrightarrow x = y^2$

En el tema anterior hemos visto que las relaciones y funciones entre conjuntos finitos se pueden representar mediante las *matrices de adyacencia*. Si $\mathcal{R}$ es una relación binaria entre dos conjuntos finitos A y B y $[x_1, x_2, \ldots, x_n]$, $[y_1, y_2, \ldots, y_m]$ son ordenaciones de los elementos de A y B respectivamente, entonces la matriz (de adyacencia) asociada a $\mathcal{R}$ es la matriz $\mathcal{M}_{\mathcal{R}} = (m_{ij})$, de tamaño $n \times m$ (es decir, con n filas y m columnas) dada por:

$$m_{ij} = \begin{cases} 1 & \text{si } (x_i, y_j) \in \mathcal{R} \\ 0 & \text{si } (x_i, y_j) \notin \mathcal{R} \end{cases}$$

Ejemplo 3.1.8 Consideremos la relación $\mathcal{R} \subseteq \{1,2,3,4\} \times \{a,b,c,d\}$:

$$\mathcal{R} = \{(1,d),(3,d),(3,b),(2,a),(4,c)\}$$

Esta relación se puede representar con la tabla

	a	b	c	d
1				×
2	×			
3		×		×
4			×	

La matriz de adyacencia construida utilizando el orden en el que aparecen escritos los elementos en la tabla de arriba es:

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

□

Ejemplo con Maxima 3.1.9 Dado que una relación es un conjunto, podemos utilizar los operadores de `Maxima`, que ya conocemos, para definir nuevas relaciones. Por ejemplo, el operador `is` permite definir relaciones usando predicados binarios:

```
(%i1)  R(x,y) := is(remainder(x-y,3)=0)$
```

$R(x,y)$ es verdadero o "x está relacionado con y por R" si 3 divide a $x-y$.

```
(%i2)  R(-1,3);
```

```
false
```

```
(%i3)  R(2,5);
```

```
true
```

También podemos definir las relaciones como un conjunto de pares.

```
(%i4)  S: {[1,2],[2,4],[3,2],[4,1]}$
```

El operador `elementp` permite convertir la definición anterior en un predicado:

```
(%i5)  Sp(x,y):= elementp([x,y],S)$

(%i6)  Sp(2,4);
```

```
true
```

```
(%i7)  Sp(1,4);
```

```
false
```

E incluso construir la matriz de adyacencia:

```
(%i8)  M_S: genmatrix(lambda([i,j],
            if elementp([i,j],S) then 1 else 0),
            4,4);
```

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

□

3.1.1. Operaciones entre relaciones binarias

Dado que las relaciones son conjuntos, podemos combinar dos o más relaciones con las distintas operaciones entre conjuntos que ya conocemos: unión, intersección, diferencia, complementario y diferencia simétrica.

Ejemplo 3.1.10 Supongamos que A es el conjunto de alumnos de un centro universitario y B el conjunto de libros de la biblioteca. Consideremos la relación $\mathcal{R}$ definida por:

$x\mathcal{R}y$ si y solo si "al estudiante x se le recomienda el libro y"

Y la relación $\mathcal{S}$ definida por:

$x\mathcal{S}s$ si y solo si "el estudiante x utiliza el libro y"

A partir de ellas, construimos las siguientes relaciones.

- $x(R \cup S)y$: "el estudiante x tiene recomendado o utiliza el libro y".
- $x(R \cap S)y$. "el estudiante x tiene recomendado y utiliza el libro y".
- $x(R - S)y$: "el estudiante x tiene recomendo el libro y pero no lo consulta"
- $x(S - R)y$: "el estudiante x utiliza el libro y aunque no lo tiene por recomendado".
- $x\overline{R}y$: "el estudiante x no tiene recomendado el libro y".
- $x\overline{S}y$: "el estudiante x no utiliza el libro y". □

Teorema 3.1.11 *Si $\mathcal{R}$ y $\mathcal{S}$ son relaciones entre los conjuntos finitos A y B con matrices $\mathcal{M}_{\mathcal{R}} = (m_{ij})$ y $\mathcal{M}_{\mathcal{S}} = (m_{ij})$, entonces*

$$\mathcal{M}_{\mathcal{R}\cap\mathcal{S}} = \mathcal{M}_{\mathcal{R}} \wedge \mathcal{M}_{\mathcal{S}} = (m_{ij} \wedge m_{ij})$$
$$\mathcal{M}_{\mathcal{R}\cup\mathcal{S}} = \mathcal{M}_{\mathcal{R}} \vee \mathcal{M}_{\mathcal{S}} = (m_{ij} \vee m_{ij})$$

Es decir, para calcular la matriz de la intersección y unión de relaciones, multiplicamos y sumamos las matrices respectivamente, considerando el producto y suma booleanos elemento a elemento.

Aparte de estas operaciones conjuntistas podemos definir otras operaciones específicas sobre relaciones.

Definición 3.1.12 (Relación inversa) *Sea $\mathcal{R}$ una relación de A en B. Se llama* relación inversa *de $\mathcal{R}$ a la relación de B en A definida por:*

$$\mathcal{R}^{-1} = \{(x, y) \in B \times A \mid (y, x) \in \mathcal{R}\}$$

Obsérvese que $\mathrm{Dom}(\mathcal{R}^{-1}) = \mathrm{Ran}(\mathcal{R})$ y $\mathrm{Ran}(\mathcal{R}^{-1}) = \mathrm{Dom}(\mathcal{R})$. Además, si $\mathcal{R}$ es una relación entre conjuntos finitos, podemos calcular fácilmente la matriz de adyacencia de su inversa según establece el siguiente resultado.

Teorema 3.1.13 *Si $\mathcal{R}$ es una relación entre los conjuntos finitos A y B con matriz de adyacencia $\mathcal{M}_{\mathcal{R}} = (m_{ij})$, entonces la matriz de la relación inversa es la transpuesta de $\mathcal{M}_{\mathcal{R}}$:*

$$\mathcal{M}_{\mathcal{R}^{-1}} = \mathcal{M}_{\mathcal{R}}^t$$

Ejemplo 3.1.14 La relación inversa de

$$\mathcal{R} = \{(a, 1), (a, 2), (c, 4), (d, 3)\} \subset \{a, b, c, d\} \times \{1, 2, 3, 4\}$$

es

$$\mathcal{R}^{-1} = \{(1, a), (2, a), (4, c), (3, d)\} \subset \{1, 2, 3, 4\} \times \{a, b, c, d\}$$

Observamos además que

$$\mathcal{M}_{\mathcal{R}} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}; \qquad \mathcal{M}_{\mathcal{R}^{-1}} = \mathcal{M}_{\mathcal{R}}^t = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \square$$

Definición 3.1.15 (Composición de relaciones) *Sean los conjuntos A, B y C y las relaciones binarias*

$$\mathcal{R} \subseteq A \times B \quad y \quad \mathcal{S} \subseteq B \times C$$

La composición *de las relaciones $\mathcal{R}$ y $\mathcal{S}$ es la relación de A en C definida por:*

$$\mathcal{R} \circ \mathcal{S} = \{(x, y) \in A \times C \mid \textit{existe } z \in B \textit{ tal que } (x, z) \in \mathcal{R}, (z, y) \in \mathcal{S}\}$$

Es decir,

$$x(\mathcal{R}\circ\mathcal{S})y \iff \text{Existe } z\in B \text{ tal que } x\mathcal{R}z,\quad z\mathcal{S}y$$

TEOREMA 3.1.16 *Si A, B y C son conjuntos finitos, $\mathcal{R}$ es una relación de A en B y $\mathcal{S}$ una relación de B en C, entonces*

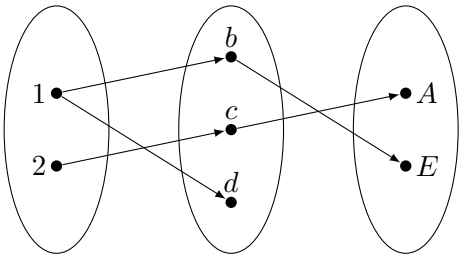
$$\mathcal{M}_{\mathcal{R}\circ\mathcal{S}} = \mathcal{M}_{\mathcal{R}} \odot \mathcal{M}_{\mathcal{S}}$$

en donde $(m_{ij})\odot(n_{ij}) = \Big(\bigvee_k (m_{ik}\wedge n_{kj})\Big)$.

EJEMPLO 3.1.17 Consideremos las relaciones

$$\mathcal{R} = \{(1,b),(1,d),(2,c)\} \subseteq \{1,2\}\times\{b,c,d\}$$
$$\mathcal{S} = \{(b,E),(c,A)\} \subseteq \{b,c,d\}\times\{A,E\}$$

La representación de estas relaciones con diagramas de Venn es la siguiente:



Las conexiones entre el primer y el tercer conjunto determinan la relación de composición, es decir, 1 está relacionado con E y 2 está relacionado con A. Vamos a calcular esta composición utilizando sus matrices

$$\mathcal{M}_{\mathcal{R}\circ\mathcal{S}} = \mathcal{M}_{\mathcal{R}} \odot \mathcal{M}_{\mathcal{S}} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Por lo tanto,

$$\mathcal{R}\circ\mathcal{S} = \{(1,E),(2,A)\}$$ □

EJEMPLO CON MAXIMA 3.1.18 En `Maxima`, podemos trabajar con las constantes booleanas `true` y `false`, y el sistema dispone de varios operadores sobre estos valores. Sin embargo, es más simple y práctico trabajar con los números 0 y 1 y definir las operaciones booleanas sobre ellos. Solo necesitamos definir la "suma booleana", ya que el "producto booleano" coincide con el producto numérico. Vamos a utilizar la secuencia "+b" escrita de forma infija para representar la suma booleana:

```
(%i1) infix("+b")$
(%i2) m +b n := m+n-m*n$
```

De esta forma, podremos usar el operador "+b" también para operar matrices booleanas elemento a elemento.

```
(%i3) 1 +b 1;
```

$$1$$

```
(%i4) matrix([1,0],[0,1]) +b matrix([1,0],[1,0]);
```

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

También podemos redefinir el producto estándar de matrices numéricas para que opere con la suma y producto booleano tal y como hemos visto anteriormente. Para ello, `Maxima` dispone de dos constantes con las que podemos establecer cual debe ser la suma y cual el producto en el producto de matrices: con `matrix_element_add` podemos establecer la suma y con `matrix_element_mult` el producto. En este caso, solo necesitamos cambiar la suma, puesto que sí estamos usando el producto numérico.

```
(%i5) matrix_element_add:
        lambda ([[x]], lreduce ("+b", x))$
```

A partir de aquí, el operador producto matricial (representado por un punto bajo) se comportará como el producto de matrices booleanas. Obsérvese que hemos tenido que recurrir al operador `lreduce`, puesto que "+b" se ha definido para dos argumentos y en el producto matricial se aplicará a una cantidad arbitraria de sumandos.

```
(%i6) matrix([1,0,1],[0,1,0]).
matrix([0,1],[1,0],[0,0]);
```

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

3.1.2. Relaciones binarias en un conjunto

En el caso particular de relaciones definidas de un conjunto en sí mismo, es importante estudiar las propiedades básicas que tales relaciones pueden verificar.

DEFINICIÓN 3.1.19 *Sea $\mathcal{R}$ una relación binaria en A (es decir, de A en A).*

- *$\mathcal{R}$ se dice* reflexiva *si $x\mathcal{R}x$ para todo $x \in A$.*
- *$\mathcal{R}$ se dice* simétrica *si para todo $x, y \in A$: si $x\mathcal{R}y$, entonces $y\mathcal{R}x$*
- *$\mathcal{R}$ se dice* transitiva *si para todo $x, y, z \in A$: si $x\mathcal{R}y$, $y\mathcal{R}z$, entonces $x\mathcal{R}z$.*

- $\mathcal{R}$ *se dice* antisimétrica *si para todo* $x, y \in A$*: si* $x\mathcal{R}y$*,* $y\mathcal{R}x$*, entonces* $x = y$*. Equivalentemente,* $\mathcal{R}$ *es* antisimétrica *si para todo* $x, y \in A$*: si* $x \neq y$*, entonces o bien* $x\not\mathcal{R}y$*, o bien* $y\not\mathcal{R}x$
- $\mathcal{R}$ *se dice* conexa *si para todo* $x, y \in A$*, o bien* $x\mathcal{R}y$*, o bien* $y\mathcal{R}x$

Estas definiciones pueden enunciarse mediante operaciones conjuntistas.

TEOREMA 3.1.20 *Sea* $\mathcal{R}$ *una relación binaria en* A *(es decir, de* A *en* A*).*

1. $\mathcal{R}$ *es reflexiva si y solo* $\mathcal{I}_A \subseteq \mathcal{R}$*, en donde* $\mathcal{I}_A$ *es la* relación identidad*:*

$$\mathcal{I}_A = \{(x, x); x \in A\}$$

2. $\mathcal{R}$ *es simétrica si y solo si* $\mathcal{R}^{-1} = \mathcal{R}$
3. $\mathcal{R}$ *es transitiva si y solo si* $\mathcal{R}^2 \subseteq \mathcal{R}$*.*
4. $\mathcal{R}$ *es antisimétrica si y solo si* $\mathcal{R} \cap \mathcal{R}^{-1} \subseteq \mathcal{I}_A$
5. $\mathcal{R}$ *es conexa si y solo si* $\mathcal{R} \cup \mathcal{R}^{-1} = A \times A$*.*

Para relaciones entre conjuntos finitos, podemos estudiar estas propiedades utilizando sus matrices de adyacencia. En el siguiente resultado, utilizamos la notación $\mathcal{I}_n$ para representar la matriz $n \times n$ en la cual los elementos de la diagonal principal son iguales a 1 y el resto son iguales a 0. También vamos a utilizar la siguiente relación entre matrices: $\mathcal{M} = (m_{ij})$ es menor o igual que $\mathcal{N} = (n_{ij})$ si $m_{ij} \leq n_{ij}$ para cada posición ij de las matrices, y lo denotamos por $\mathcal{M} \leq \mathcal{N}$.

TEOREMA 3.1.21 *Sea* $\mathcal{R}$ *una relación en un conjunto finito* A *y* $\mathcal{M}_\mathcal{R} = (m_{ij})_{n\times n}$ *su matriz de adyacencia. Entonces:*

1. $\mathcal{R}$ *es reflexiva si y solo si* $m_{ii} = 1$ *para todo* i*, es decir, si todos los elementos de la diagonal principal son iguales a 1.*
2. $\mathcal{R}$ *es simétrica si y solo si* $m_{ij} = m_{ji}$ *para todo* i, j*, es decir, si la matriz es simétrica:* $\mathcal{M}_\mathcal{R} = \mathcal{M}_\mathcal{R}^{-1}$*.*
3. *Si* $\mathcal{M}_{\mathcal{R}^2} = (n_{ij})$*, entonces* $\mathcal{R}$ *es transitiva si y solo si* $n_{ij} \leq m_{ij}$ *para todo* i, j*; es decir, si* $\mathcal{M}_\mathcal{R}^2 \leq \mathcal{M}_\mathcal{R}$*.*
4. $\mathcal{R}$ *es antisimétrica si y solo si* $m_{ij} \wedge m_{ji} = 0$ *para todo* i, j *tales que* $i \neq j$*, es decir, si en cada pareja de elementos simétricos, hay al menos un cero. Matricialmente: es antisimétrica si y solo si* $\mathcal{M}_R \wedge \mathcal{M}_R^t \leq \mathcal{I}_n$
5. $\mathcal{R}$ *es conexa si y solo si* $m_{ij} \vee m_{ji} = 1$ *para todo* i, j*. Matricialmente: es conexa si y solo si* $\mathcal{M}_R \vee \mathcal{M}_R^t = (1)_{n\times n}$

La siguiente tabla resume las definiciones de las propiedades y sus caracterizaciones en forma matricial.

	Definición	Matricialmente (Solo finitas)
Reflexiva	$x\mathcal{R}x$	$\mathcal{I} \leq \mathcal{M}_R$
Simétrica	$x\mathcal{R}y \implies y\mathcal{R}x$	$\mathcal{M}_R^t = \mathcal{M}_R$
Transitiva	$(x\mathcal{R}y, y\mathcal{R}z) \implies x\mathcal{R}z$	$\mathcal{M}_R^2 \leq \mathcal{M}_R$
Antisimétrica	$(x\mathcal{R}y, y\mathcal{R}x) \implies x = y$	$\mathcal{M}_R \wedge \mathcal{M}_R^t \leq \mathcal{I}$
Conexa	$x\not\mathcal{R}y \implies y\mathcal{R}x$	$\mathcal{M}_R \vee \mathcal{M}_R^t = (1)_{n\times n}$

Ejemplo con Maxima 3.1.22 Vamos a estudiar las propiedades de la siguiente relación.

```
(%i1)  S: {[1,2],[2,4],[3,2],[4,1]}$
(%i2)  MS: genmatrix(lambda([i,j],
                  if elementp([i,j],S) then 1 else 0),
                  4,4);
```

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

Por ejemplo, la relación no es simétrica, puesto que no lo es la matriz:

```
(%i3)  is(MS=transpose(MS));
```

```
false
```

Como hemos visto anteriormente, para estudiar algunas propiedades, necesitamos algunas matrices auxiliares. Por ejemplo, la matriz identidad, cuyos elementos en la diagonal principal son iguales a 1 y el resto son iguales a 0:

```
(%i4)  ident(4);
```

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

También necesitamos las matrices cuadradas con todos lo elementos iguales a 1, que podemos definir fácilmente:

```
(%i5)  unos(n) := genmatrix(lambda([i,j],1),n,n)$
(%i6)  unos(4);
```

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

La relación S es antisimétrica, ya que el producto booleano elemento a elemento de $\mathcal{M}_S$ por $\mathcal{M}_S^t$, es menor que la matriz diagonal.

```
(%i7) MS*transpose(MS);
```

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

```
(%i8) is(%=%*ident(4));
```

```
true
```

Para comprobar la propiedad transitiva, tenemos que usar el producto matricial considerando la suma y el producto booleano.

```
(%i9) infix("+b")$
(%i10) m +b n := m+n-m*n$
(%i11) matrix_element_add:
         lambda ([[x]], lreduce ("+b", x))$
(%i12) MS.MS;
```

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

Por lo tanto, la relación S no es transitiva.

```
(%i13) is(%=%*MS);
```

```
false
```

En estas comprobaciones hemos tenido en cuenta que el producto booleano coincide con la función mínimo y que $\min(a, b) = a$ si y solamente si $a \leq b$.

Finalmente, también podemos comprobar que la relación no es conexa.

```
(%i14) is (MS +b transpose(MS)=unos(4));
```

```
false
```

□

Definición 3.1.23 (Relación de orden parcial) *Una relación binaria $\mathcal{R}$ en un conjunto no vacío A se dice que es una relación de* orden parcial *si es reflexiva, antisimétrica y transitiva. Habitualmente, lo escribimos diciendo que el par $(A, \mathcal{R})$ es un* conjunto parcialmente ordenado.

Si una relación solo tiene las propiedades reflexiva y transitiva, se dice que es un *preorden*. Si una relación de orden es además conexa, se dice que la relación es de *orden total*. Las relaciones de orden sirven comparar magnitudes y establecer preferencias.

La relación de orden habitual entre números es efectivamente una relación de orden y además es total. En el tema anterior hemos estudiado la relación de divisibilidad, que es también una relación de orden parcial entre números naturales. La relación de inclusión entre conjuntos, es otro ejemplo de relación de orden parcial.

Habitualmente, las relaciones de orden se representan por símbolos que recuerdan a la relación de orden entre números, como por ejemplo $\preceq$ o $\sqsubseteq$.

3.1.3. Relaciones de equivalencia

Definición 3.1.24 *Una relación $\mathcal{R}$ en A se dice que es una* relación de equivalencia *si es reflexiva, simétrica y transitiva.*

Las relaciones de equivalencia sirven para introducir nociones de igualdad basada en determinadas características. Por ejemplo, las relaciones de congruencia que hemos estudiado en el tema anterior son relaciones de equivalencia. Otros ejemplo de relación de equivalencia es la semejanza de triángulos.

Habitualmente, las relaciones de equivalencia se representan por símbolos que recuerdan el símbolo de igualdad, como por ejemplo $\equiv$, $\sim$, $\approx$.

La noción de clase de equivalencia se ha introducido en el tema anterior para el ejemplo de las relaciones de congruencia y la extendemos ahora a cualquier relación de equivalencia.

Definición 3.1.25 (Clases de equivalencia) *Sea $\sim$ una relación de equivalencia definida en un conjunto A y sea $x \in A$.*

- *El subconjunto de A formado por los elementos equivalentes a x se llama* clase de equivalencia *de x y se denota $[x]_\sim$.*

$$[x]_\sim = \{y \in A \mid y \sim x\}$$

- *El conjunto formado por las clases de equivalencia de $\sim$ se denomina* conjunto cociente *y se denota $A_{/\sim}$:*

$$A_{/\sim} = \{[x]_\sim; x \in A\}$$

Cada elemento $y \in [x]_\sim$ puede ser considerado como *representante* de esta clase de equivalencia. Por ejemplo, para la relación de congruencia módulo 3 en $\mathbb{Z}$

$$x \equiv_3 y \iff x - y \text{ es multiplo de 3,}$$

el conjunto cociente está formado por las tres clases de congruencia

$$\mathbb{Z}_3 = \mathbb{Z}_{/\equiv_3} = \{[0]_3, [1]_3, [2]_3\}$$

Según vimos en el tema anterior, las clases de congruencia determinan una *partición* del conjunto de números enteros; esto ocurre en cualquier conjunto y para cualquier relación de equivalencia.

Definición 3.1.26 *Una* partición *de un conjunto S es una familia de subconjuntos, $\mathcal{P} = \{S_1, ..., S_k\}$ tales que:*

- *$S_i \cap S_j = \varnothing$ para todo i, j (es decir, son* disjuntos dos a dos*).*
- *$S_1 \cup S_2 \cup \cdots \cup S_k = S$*

Ejemplo 3.1.27

- $\mathcal{P} = \{\{1,3\}, \{2,5\}, \{4\}\}$ es una partición del conjunto $S = \{1,2,3,4,5\}$.
- $\mathcal{P} = \{\{x \in \mathbb{N} \mid x \text{ es par }\},\ \{x \in \mathbb{N} \mid x \text{ es impar }\}\}$ es una partición de $\mathbb{N}$. □

Teorema 3.1.28 *Si $\sim$ es una relación de equivalencia en A, entonces el conjunto cociente, $A_{/\sim}$ es una partición de A.*

Teorema 3.1.29 *Si $\mathcal{P}$ es un partición de A, entonces existe una relación de equivalencia en A tal que $A_{/\sim} = \mathcal{P}$.*

La demostración de estos resultados es bastante simple. Por ejemplo, para el segundo de ellos, basta probar que la relación

$$x \sim_{\mathcal{P}} y \iff \text{Existe } S \in \mathcal{P} \text{ tal que } x, y \in S$$

tiene como conjunto cociente la partición $\mathcal{P}$. La relación $\sim_{\mathcal{P}}$ se denomina *relación inducida* por $\mathcal{P}$.

Ejemplo 3.1.30 La relación en $A = \{1,2,3,4,5,6\}$ inducida por la partición

$$A_1 = \{1,3,5\},\ A_2 = \{4,6\},\ A_3 = \{2\}$$

es la siguiente

$$\begin{aligned}\mathcal{R}_{\mathcal{P}} = \{&(1,1),(1,3),(1,5),(3,1),(3,3),(3,5),(5,1),(5,3),(5,5),\\ &(4,4),(4,6),(6,4),(6,6),\\ &(2,2)\}\end{aligned}$$ □

Ejemplo con Maxima 3.1.31 El operador `equiv_classes`(A,R) determina las clases de equivalencia en el conjunto A dadas por la relación R definida con un predicado.

```
(%i1) conj: setify(makelist(i,i,-10,10));
```

$$\{-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10\}$$

```
(%i2) R(x,y) := is(remainder(x-y,3)=0)$
(%i3) equiv_classes(conj,R);
```

$$\{\{-10,-7,-4,-1,2,5,8\},\{-9,-6,-3,0,3,6,9\},\{-8,-5,-2,1,4,7,10\}\}$$ □

3.1.4. Cierres de relaciones

Dada una relación $\mathcal{R}$ buscamos la mínima relación que la contiene y que verifica una o varias propiedades básicas.

Definición 3.1.32 *Sea $\mathcal{R}$ una relación binaria en A. El* cierre reflexivo *de $\mathcal{R}$ denotado por* $\mathrm{r}(\mathcal{R})$ *es la menor relación reflexiva que contiene a $\mathcal{R}$. Es decir, es la única relación reflexiva que verifica que $\mathcal{R}\subseteq \mathrm{r}(\mathcal{R})$ y $\mathrm{r}(\mathcal{R})\subseteq\mathcal{S}$ para toda relación reflexiva $\mathcal{S}\supseteq\mathcal{R}$.*

Obviamente, una relación es reflexiva si y solo si coincide con su cierre reflexivo. Por otra parte, es muy sencillo determinar el cierre reflexivo de una relación.

Teorema 3.1.33 *Si $\mathcal{R}$ una relación binaria en A, entonces* $\mathrm{r}(\mathcal{R})=\mathcal{R}\cup\mathcal{I}_A$, *en donde* $\mathcal{I}_A=\{(x,x);x\in A\}$.

Definición 3.1.34 *Sea $\mathcal{R}$ una relación binaria en A. El* cierre simétrico *de $\mathcal{R}$, denotado por* $\mathrm{s}(\mathcal{R})$, *es la menor relación simétrica que contiene a $\mathcal{R}$. Es decir,* $\mathrm{s}(\mathcal{R})$ *es la única relación simétrica tal que $\mathcal{R}\subseteq\mathrm{s}(\mathcal{R})$ y $\mathrm{s}(\mathcal{R})\subseteq\mathcal{S}$ para toda relación simétrica $\mathcal{S}\supseteq\mathcal{R}$.*

Obviamente, una relación es simétrica si y solo si coincide con su cierre simétrico. Por otra parte, es muy sencillo determinar el cierre simétrico de una relación.

Teorema 3.1.35 *Si $\mathcal{R}$ una relación binaria en A, entonces* $\mathrm{s}(\mathcal{R})=\mathcal{R}\cup\mathcal{R}^{-1}$.

Ejemplo 3.1.36 Sea el conjunto $A=\{1,2,3\}$ y $\mathcal{R}=\{(1,2),(2,3)\}$ Entonces

- $\mathrm{r}(\mathcal{R})=\{(1,2),(2,3),(1,1),(2,2),(3,3)\}$. Ya que solo necesitamos añadir los pares $(1,1)$, $(2,2)$ y $(3,3)$ para obtener una relación reflexiva.

- $\mathrm{s}(\mathcal{R}) = \{(1,2),(2,3),(2,1),(3,2)\}$. Ya que solo necesitamos añadir los pares opuestos de los que forman $\mathcal{R}$. □

DEFINICIÓN 3.1.37 *Sea $\mathcal{R}$ una relación binaria en A. El* cierre transitivo *de $\mathcal{R}$, denotado por $\mathrm{t}(\mathcal{R})$, es la menor relación transitiva que contiene a $\mathcal{R}$. Es decir, $\mathrm{t}(\mathcal{R})$ es la única relación transitiva tal que $\mathcal{R} \subseteq \mathrm{t}(\mathcal{R})$ y $\mathrm{t}(\mathcal{R}) \subseteq \mathcal{S}$ para toda relación transitiva $\mathcal{S} \supseteq \mathcal{R}$.*

Obviamente, una relación es transitiva si y solo si coincide con su cierre transitivo. El siguiente resultado establece un método para determinar el cierre transitivo de una relación.

TEOREMA 3.1.38 *Sea $\mathcal{R}$ una relación binaria en A. Entonces*

$$\mathrm{t}(\mathcal{R}) = \mathcal{R} \cup \mathcal{R}^2 \cup \mathcal{R}^3 \cup \cdots = \bigcup_{i=1}^{\infty} \mathcal{R}^i$$

COROLARIO 3.1.39 *Si $\mathcal{R}$ es una relación binaria en un conjunto finito A, entonces*

$$\mathrm{t}(\mathcal{R}) = \bigcup_{i=1}^{n} \mathcal{R}^i$$

siendo n el número de elementos de A.

EJEMPLO 3.1.40 Consideremos el conjunto $A = \{a,b,c,d\}$ y la relación

$$\mathcal{R} = \{(a,a),(a,d),(b,d),(c,a),(d,b)\}$$

Vamos a calcular el cierre transitivo de $\mathcal{R}$.

Utilizando la ordenación $[a,b,c,d]$ de los elementos de A, la matriz de adyacencia de $\mathcal{R}$ es

$$\mathcal{M}_{\mathcal{R}} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

De esta forma, podemos calcular fácilmente las matrices de $\mathcal{R}^2$, $\mathcal{R}^3$ y $\mathcal{R}^4$.

$$\mathcal{M}_{\mathcal{R}^2} = \mathcal{M}_{\mathcal{R}}^2 = \begin{pmatrix} 1&0&0&1\\0&0&0&1\\1&0&0&0\\0&1&0&0 \end{pmatrix} \begin{pmatrix} 1&0&0&1\\0&0&0&1\\1&0&0&0\\0&1&0&0 \end{pmatrix} = \begin{pmatrix} 1&1&0&1\\0&1&0&0\\1&0&0&1\\0&0&0&1 \end{pmatrix}$$

$$\mathcal{M}_{\mathcal{R}^3} = \mathcal{M}_{\mathcal{R}}^3 = \begin{pmatrix} 1&1&0&1\\0&1&0&0\\1&0&0&1\\0&0&0&1 \end{pmatrix} \begin{pmatrix} 1&0&0&1\\0&0&0&1\\1&0&0&0\\0&1&0&0 \end{pmatrix} = \begin{pmatrix} 1&1&0&1\\0&0&0&1\\1&1&0&1\\0&1&0&0 \end{pmatrix}$$

$$\mathcal{M}_{\mathcal{R}^4} = \mathcal{M}_{\mathcal{R}}^4 = \begin{pmatrix} 1&1&0&1\\0&0&0&1\\1&1&0&1\\0&1&0&0 \end{pmatrix} \begin{pmatrix} 1&0&0&1\\0&0&0&1\\1&0&0&0\\0&1&0&0 \end{pmatrix} = \begin{pmatrix} 1&1&0&1\\0&1&0&0\\1&1&0&1\\0&0&0&1 \end{pmatrix}$$

Por lo tanto,

$$\mathcal{M}_{\mathrm{t}(\mathcal{R})} = \mathcal{M}_{\mathcal{R}} \vee \mathcal{M}_{\mathcal{R}^2} \vee \mathcal{M}_{\mathcal{R}^3} \vee \mathcal{M}_{\mathcal{R}^4} =$$

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

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

Es decir, $\mathrm{t}(\mathcal{R}) = \{(a,a), (a,b), (a,d), (b,b), (b,d), (c,a), (c,b), (c,d), (d,b), (d,d)\}$. □

Algoritmo de Warshall. Para calcular cada elemento de las matrices $\mathcal{M}_{\mathcal{R}^i}$ en el método del teorema anterior, tenemos que realizar $2n-1$ operaciones booleanas. De esta forma, para calcular cada elemento de $\mathcal{M}_{\mathrm{t}(\mathcal{R})}$ tenemos que realizar $2n(n-1)$ operaciones. Por lo tanto, el cálculo de $\mathcal{M}_{\mathrm{t}(\mathcal{R})}$ requiere $2n^3(n-1)$ operaciones.

El teorema siguiente describe el *Algoritmo de Warshall* para calcular el cierre transitivo de una relación. Este algoritmo reduce el número de operaciones necesarias.

Teorema 3.1.41 (Algoritmo de Warshall) *Dado un conjunto A con n elementos y una relación $\mathcal{R}$ en A, determinamos el cierre transitivo de $\mathcal{R}$ calculando una secuencia de matrices*

$$\mathcal{W}_0, \mathcal{W}_1, \cdots, \mathcal{W}_k, \cdots, \mathcal{W}_n$$

de forma que $\mathcal{W}_0 = \mathcal{M}_{\mathcal{R}}$, $\mathcal{W}_n = \mathcal{M}_{\mathrm{t}(\mathcal{R})}$ y si $\mathcal{W}_k = (w_{ij}^{(k)})$ entonces

$$w_{ij}^{(k)} = w_{ij}^{(k-1)} \vee (w_{ik}^{(k-1)} \wedge w_{kj}^{(k-1)})$$

Para calcular $w_{ij}^{(k)}$ a partir de matriz $\mathcal{W}_{k-1}$ necesitamos 2 operaciones booleanas y por lo tanto, para hallar $\mathcal{W}_k$ a partir de $\mathcal{W}_{k-1}$ necesitamos $2n^2$ operaciones. Por lo tanto, dado que calculamos n matrices, necesitamos $2n^3$ operaciones.

A la hora de calcular la secuencia de matrices en el algoritmo de Warshall, es conveniente tener en cuenta las siguientes observaciones:

- $w_{ij}^{(k-1)} \leq w_{ij}^{(k)}$ y por lo tanto, los posiciones iguales a 1 en $\mathcal{W}_{k-1}$ se mantienen en $\mathcal{W}_k$.
- Teniendo en cuenta las siguientes igualdades

$$w_{kj}^{(k)} = w_{kj}^{(k-1)} \vee (w_{kk}^{(k-1)} \wedge w_{kj}^{(k-1)}) = w_{kj}^{(k-1)}$$
$$w_{ik}^{(k)} = w_{ik}^{(k-1)} \vee (w_{ik}^{(k-1)} \wedge w_{kk}^{(k-1)}) = w_{ik}^{(k-1)}$$

  deducimos que la fila k-ésima y la columna k-ésima de $\mathcal{W}_k$ son iguales a las de $\mathcal{W}_{k-1}$.

Ejemplo 3.1.42 Consideremos el conjunto $A = \{a, b, c, d\}$ y la relación

$$\mathcal{R} = \{(a, a), (a, d), (b, d), (c, a), (d, b)\}$$

Vamos a calcular el cierre transitivo de $\mathcal{R}$ utilizando el algoritmo de Warshall.

$$\mathcal{W}_0 = \mathcal{M}_{\mathcal{R}} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$\mathcal{W}_1 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & \boxed{1} \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

En la matriz $\mathcal{W}_1$ copiamos la primera fila y la primera columna de $\mathcal{W}_0$. Recorremos sus elementos y vemos que $w_{31}^{(1)} = 1$ y $w_{14}^{(1)} = 1$, por lo que $w_{34}^{(1)} = 1$. El resto de elementos de la matriz se copian de la matriz anterior.

$$\mathcal{W}_2 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & \boxed{1} \end{pmatrix}$$

En la matriz $\mathcal{W}_2$ copiamos la segunda fila y la segunda columna de $\mathcal{W}_1$. Recorremos sus elementos y vemos que $w_{42}^{(2)} = 1$ y $w_{24}^{(2)} = 1$, por lo que $w_{44}^{(2)} = 1$. El resto de elementos de la matriz se copian de la matriz anterior.

$$\mathcal{W}_3 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

En la matriz $\mathcal{W}_3$ copiamos la tercera fila y la tercera columna de $\mathcal{W}_2$. Dado que la columna tercera esta compuesta solamente por ceros, en este paso no añadimos ningún uno más, por lo que el resto de elementos de la matriz se copian de la matriz anterior.

$$\mathcal{W}_4 = \mathcal{M}_{\mathrm{t}(\mathcal{R})} = \begin{pmatrix} 1 & \boxed{1} & 0 & 1 \\ 0 & \boxed{1} & 0 & 1 \\ 1 & \boxed{1} & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

En la matriz $\mathcal{W}_4$ copiamos la cuarta fila y la cuarta columna de $\mathcal{W}_3$. Recorremos sus elementos y vemos que $w_{14}^{(4)} = 1$ y $w_{42}^{(4)} = 1$, por lo que $w_{12}^{(2)} = 1$; $w_{24}^{(4)} = 1$ y $w_{42}^{(4)} = 1$, por lo que $w_{22}^{(2)} = 1$; $w_{34}^{(4)} = 1$ y $w_{42}^{(4)} = 1$, por lo que $w_{32}^{(2)} = 1$. El resto de elementos de la matriz se copian de la matriz anterior. □

Ejemplo con Maxima 3.1.43 Aunque `Maxima` no incluye ningún operador o paquete específico para el algoritmo de Warshall, no es difícil escribir una función recursiva a partir de la descripción del algoritmo.

```
(%i1) warshall(m):= warshall_aux(m,length(m),1)$
(%i2) warshall_aux(m,n,k):= if k=n+1 then m else
          warshall_aux (genmatrix(lambda([i,j],
                        max(m[i][j],min(m[i][k],m[k][j]))),
                        n,n),n,k+1)$
```

Para utilizar recursión de cola, recurrimos a un operador auxiliar con dos argumentos adicionales, el tamaño de la matriz y un contador para las iteraciones del algoritmo. Hemos utilizado el operador `genmatrix`$(a[i,j],k,m)$, que construye una matriz de tamaño $k \times m$ con el elemento $a[i,j]$ en la posición (i,j). Para describir los elementos de la matriz, recurrimos al operador `lambda`$([i,j],<$ expresión en i, j $>)$, que permite dar la expresión que corresponde a cada posición sin necesidad de asignarle un nombre. De esta forma, es fácil observar que si $m = \mathcal{M} = \mathcal{W}_0$, entonces `warshall_aux`$(m,,k) = \mathcal{W}_k$.

Vamos a verificar el ejemplo que hemos hecho más arriba, "trazando" el operador auxiliar para verificar las etapas intermedias.

```
(%i3) M0: matrix([1,0,0,1],
                 [0,0,0,1],
                 [1,0,0,0],
                 [0,1,0,0])$
(%i4) trace(warshall_aux)$
(%i15)  warshall(M0);
1 Introducir warshall_aux [matrix(
                  [1,     0,     0,     1],
                  [0,     0,     0,     1],
```

```
                [1,     0,     0,     0],
                [0,     1,     0,     0]
        ),4,1]
.2 Introducir warshall_aux [matrix(
                [1,     0,     0,     1],
                [0,     0,     0,     1],
                [1,     0,     0,     1],
                [0,     1,     0,     0]
        ),4,2]
..3 Introducir warshall_aux [matrix(
                [1,     0,     0,     1],
                [0,     0,     0,     1],
                [1,     0,     0,     1],
                [0,     1,     0,     1]
        ),4,3]
...4 Introducir warshall_aux [matrix(
                [1,     0,     0,     1],
                [0,     0,     0,     1],
                [1,     0,     0,     1],
                [0,     1,     0,     1]
        ),4,4]

....5 Introducir warshall_aux [matrix(
                [1,     1,     0,     1],
                [0,     1,     0,     1],
                [1,     1,     0,     1],
                [0,     1,     0,     1]
        ),4,5]
....5 Salir  warshall_aux matrix(
                [1,     1,     0,     1],
                [0,     1,     0,     1],
                [1,     1,     0,     1],
                [0,     1,     0,     1]
..............
        )
```

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

□

Para calcular el cierre de equivalencia basta realizar los tres cierres correspondientes a las propiedades que lo definen, pero hay que tener en cuenta que el cierre transitivo siempre debe ser el último.

TEOREMA 3.1.44 *Sea $\mathcal{R}$ una relación binaria definida en un conjunto A. Entonces, la mínima relación de equivalencia que contiene a $\mathcal{R}$ es* $\mathrm{t}(\mathrm{s}(\mathrm{r}(\mathcal{R})))$. *Es decir,*

$$\mathrm{t}(\mathcal{R} \cup \mathcal{R}^{-1} \cup \mathcal{I}_A)$$

3.2. Grafos

Los **grafos** son modelos matemáticos utilizados para representar relaciones entre objetos de un conjunto, y que permiten generalizar el modelo de las relaciones binarias que estudiamos en la lección anterior. Utilizamos grafos para estudiar conexiones entre objetos, datos o fuentes de información. Un ejemplo de grafo lo encontramos en las redes de carreteras, en donde las poblaciones se enlazan o conectan con una o varias carreteras. Otros ejemplos son las redes de comunicaciones, redes eléctricas, redes de carreteras, pero también redes sociales y árboles genealógicos.

Algunos problemas que habitualmente se estudian y resuelven desde la teoría de grafos son: calcular el número de combinaciones diferentes de vuelos entre dos ciudades de una red aérea; determinar las posibles rutas entre dos localizaciones de una ciudad o región; encontrar el camino más corto entre dos ciudades en una red de transporte; programar actividades en un calendario; asignar canales a las emisoras de televisión; determinar si se puede crear un circuito en una placa de una sola capa; construir modelos para redes informáticas, determinar si dos ordenadores están conectados entre sí.

3.2.1. Grafos simples

DEFINICIÓN 3.2.1 *Un **grafo simple** es un par $G = \langle V, E\rangle$ formado por un conjunto V, cuyos elementos denominaremos **vértices** de G, y*

$$E \subseteq \Big\{\{u, v\}; u, v \in V, u \neq v\Big\};$$

*los elementos de E se denominan **aristas** de G.*

Los vértices de un grafo pueden ser objetos (ordenadores, ciudades, personas,...) o datos (fechas, direcciones, módulos de un programa, identificadores,...) y las aristas pueden ser conexiones físicas (carreteras, conexiones de red,...) o conexiones virtuales (dependencias, herencias, conflictos,...).

EJEMPLO 3.2.2 Consideremos el grafo $G = \langle V, E\rangle$ determinado por:

$$V = \{a, b, c, d\}, \qquad E = \Big\{\{a, b\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}\Big\}.$$

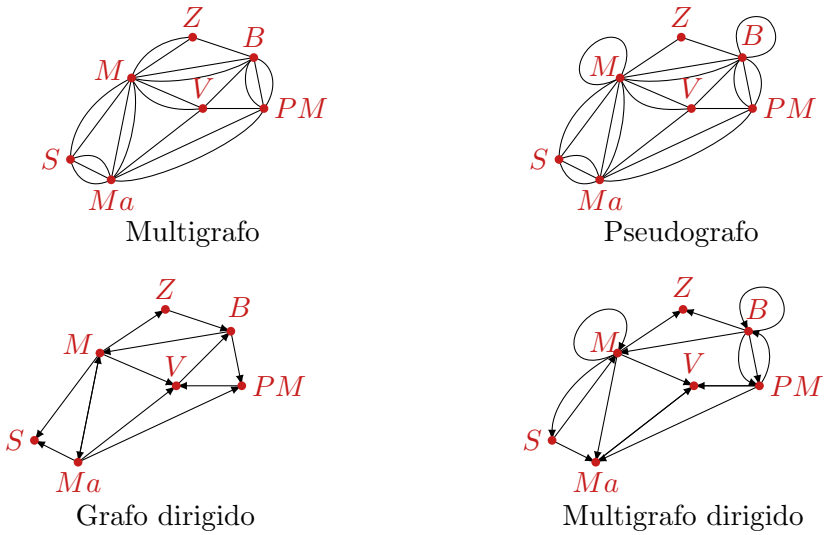
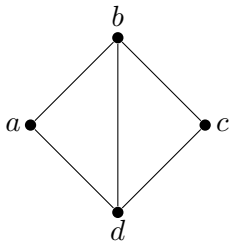


Figura 3.1: Tipos de grafos que no son simples.

Más adelante, aprenderemos a trabajar con diferentes representaciones formales de los grafos, pero en grafos con pocos vértices es conveniente utilizar igualmente su representación gráfica.



Los vértices se representan como puntos en el plano y las aristas se representan por líneas uniendo vértices. □

Los grafos simples son el tipo más sencillo de grafo y es un modelo formal que coincide con las *relaciones binarias simétricas y antirreflexivas* (es decir, ningún elemento está relacionado consigo mismo) en un conjunto. Existen varias posibles generalizaciones de la noción de grafo simple (ver figura 3.1). En los **multigrafos**, se admiten múltiples aristas conectando dos vértices; en los **pseudografos**, se permite conectar un vértice consigo mismo; en los **grafos dirigidos**, los pares que determinan las aristas son ordenados; en los **multigrafos dirigidos**, las aristas son pares ordenados y puede haber multiples aristas entre vértices; en los **grafos ponderados**, las conexiones determinadas por las aristas tiene asignadas pesos.

3.2.2. Conceptos y resultados básicos

DEFINICIÓN 3.2.3 *Si $G = (V, E)$ es un grafo simple, $u, v \in V$, $e = \{u, v\} \in E$, decimos que*

- *los vértices u y v son **adyacentes**,*
- *que la arista e es **incidente** con los vértices u y v,*
- *que los vértices u y v son **extremos** de la arista e,*
- *que la arista e **conecta** a los vértices u y v.*

DEFINICIÓN 3.2.4 *Sea $G = \langle V, E\rangle$ un grafo simple. Definimos la función **grado**, $\delta\colon V \to \mathbb{N}$, como: $\delta(v)$ es el número de aristas incidentes en v. Si $\delta(v) = 0$, decimos que v es un vértice **aislado**; si $\delta(v) = 1$, decimos que v es una **hoja**. Decimos que G es **regular** si todos sus vértices tienen el mismo grado.*

Enunciamos a continuación el teorema de Euler, en el que utilizamos las siguiente notación: si A es un conjunto finito, $|A|$ denota el número de elementos de A.

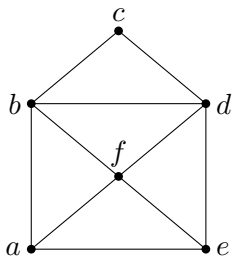
TEOREMA 3.2.5 (EULER) *Si $G = \langle V, E\rangle$ es un grafo simple, entonces*

$$\sum_{v \in V} \delta(v) = 2|E|.$$

En particular, la suma de los grados de todos los vértices de un grafo simple es un número par, de donde se deduce el siguiente resultado.

COROLARIO 3.2.6 *Todo grafo simple tiene un número par de vértices de grado impar.*

EJEMPLO 3.2.7 Consideremos el grafo



Entonces:

$$\delta(a) = 3, \quad \delta(b) = 4, \quad \delta(c) = 2, \quad \delta(d) = 4, \quad \delta(e) = 3, \quad \delta(f) = 4$$

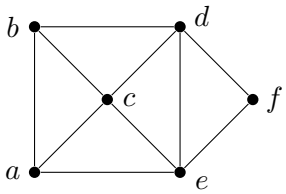
Por lo que efectivamente se verifica el teorema de Euler en este grafo:

$$\sum_{v\in V}\delta(v) = \delta(a)+\delta(b)+\delta(c)+\delta(d)+\delta(e)+\delta(f) =$$
$$= 3+4+2+4+3+4 = 20 = 2\cdot 10 = 2|E|$$

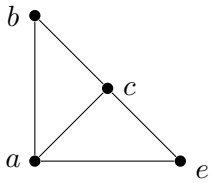
Además, hay exactamente dos vértices, a y e, que tienen grados impares. □

DEFINICIÓN 3.2.8 (SUBGRAFOS) *Se dice que el grafo $H = \langle V_H, E_H\rangle$ es un **subgrafo** del grafo $G = \langle V_G, E_G\rangle$ si $V_H \subseteq V_G$ y $E_H \subseteq E_G$. Decimos que es **subgrafo propio** si, o bien $V_H \neq V_G$, o bien $E_H \neq E_G$. Si $V_H = V_G$, se dice que H es un subgrafo **generador** (en inglés, se denomina **spanning subgraph**). del grafo G.*

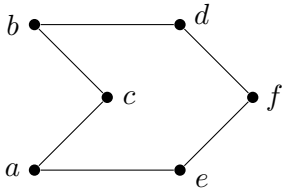
EJEMPLO 3.2.9 Los grafos que aparecen abajo verifican que tanto H_1 como H_2 son subgrafos propios de G. En H_1 hay menos vértices, y por lo tanto, menos aristas. Los vértices de H_2 son los mismo que los de G, pero en H_2 hay menos aristas, es decir, H_2 es un subgrafo generador de G.



G



H_1



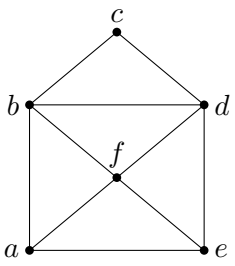
H_2

□

3.2.3. Representación de grafos simples

La representación gráfica que hemos usado en ejemplos anteriores ayuda a entender el concepto de grafo y a visualizar propiedades y operaciones sobre ellos. Si el número de vértices o el número de aristas es muy grande, esta representación carecerá de utilidad práctica. Por eso son necesarias otras formas de representación que permitan la descripción fácil de un grafo, con independencia de su tamaño, y sobre las cuales podamos realizar transformaciones y estudiar propiedades de forma efectiva y algorítmica.

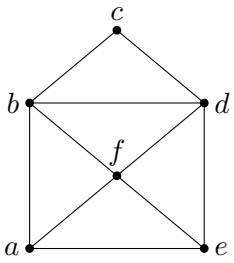
Dado que un grafo simple es una relación binaria, podemos representarlo por las matrices de adyacencia que estudiamos en el tema anterior y como vemos en el siguiente ejemplo.



$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Recordemos que es necesario fijar previamente un orden en el conjunto de vértices, por lo que no será necesario especificar la correspondencia de filas y columnas con los vértices. Por otra parte, dado que el grafo simple es una relación simétrica, estas matrices siempre serán simétricas. De la misma forma, dado que el grafo simple es una relación antirreflexiva, la diagonal principal solo contiene ceros.

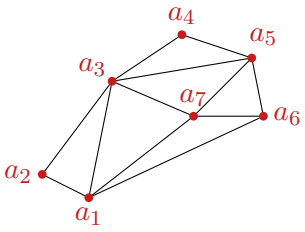
Otra forma de describir un grafo es mediante las **listas de adyacencia**. Estas listas son en realidad una tabla en cuya primera fila se disponen todos los vértices del grafo y, por debajo de cada vértice formando una columna, los vértices adyacentes a él.



a	b	c	d	e	f
b	a	b	b	a	a
e	c	d	c	d	b
f	d		e	f	d
	f		f		e

Aunque no es necesario, es conveniente establecer previamente un orden dentro del conjunto de vértices (igual que hacemos en las matrices de adyacencia) y utilizarlo al disponer los vértices en la primera fila y en cada columna.

Ejemplo 3.2.10 Mostramos a continuación un grafo dado por su representación gráfica, por su lista de adyacencia y por su matriz de adyacencia, en la cual, se ha elegido el orden de los vértices dado por sus subíndices.



a_1	a_2	a_3	a_4	a_5	a_6	a_7
a_2	a_1	a_1	a_3	a_3	a_1	a_1
a_3	a_3	a_2	a_5	a_4	a_5	a_3
a_6		a_4		a_6	a_7	a_5
a_7		a_5		a_7		a_6
		a_7				

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

□

Ejemplo con Maxima 3.2.11 `Maxima` dispone de un paquete con operadores específicos para trabajar con grafos. Podemos trabajar con grafos simples, dirigidos o

no dirigidos, y también con grafos ponderados. Internamente los grafos son representados por sus listas de adyacencia, aunque también podemos trabajar con la matriz de adyacencia. Los vértices son identificados con números naturales y las aristas son representadas por listas de longitud dos. Se pueden asignar etiquetas a vértices y se pueden asignar pesos a las aristas para definir grafos ponderados. El operador `create_graph` sirve para definir una grafo a partir de su lista de vértices y su lista de aristas. Este operador admite varias sintaxis; en la más simple, escribimos un primer argumento con un número positivo n, que será el número de vértices y los vértices serán identificados con los números 0, 1, 2,..., $n-1$. El segundo argumento es una lista de listas de longitud 2 que definen las aristas del grafo.

```
(%i1) load(graphs)$
(%i2) g1: create_graph(5,[[1,2],[1,3],[2,3],[0,4]]);
(%o2) GRAPH(5 vertices, 4 edges)
```

Como vemos, la salida no muestra nada, solo nos da la información del grafo que hemos definido, concretamente el número de vértices y el número de aristas. Para ver su representación como lista de adyacencia usamos `print_graph`:

```
(%i3) print_graph(g1);
Graph on 5 vertices with 4 edges.
Adjacencies:
  4 :  0
  3 :  2  1
  2 :  3  1
  1 :  3  2
  0 :  4
(%o3)   done
```

Obsérvese que la muestra como columna, no como fila. También podemos obtener la representación mediante la matriz de adyacencia del grafo:

```
(%i4) mg1: adjacency_matrix(g1);
```

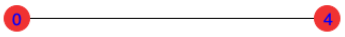
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

A lo largo del tema iremos viendo distintos operadores para estudiar los grafos y obtener información sobre ellos. Por ejemplo, en la sección anterior, hemos definido la noción de grado de un vértice y la lista de adyacencia no permite visualizar ese grado, pero un elemento importante del grafo es lo que se llama la *secuencia gráfica*, la lista de los grados de todos los vértices ordenada de forma creciente:

```
(%i5) degree_sequence(g1);
(%o15)   [1,1,2,2,2]
```

Finalmente, el operador `draw_graph` implementa un algoritmo para construir una representación gráfica.

```
(%i6) draw_graph(g1, vertex_size=4,show_id=true);
```



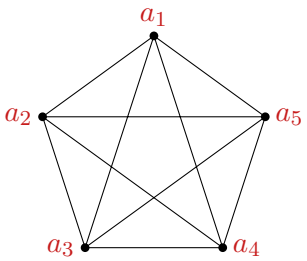
El operador `draw_graph` dispone de varias opciones para configurar y mejorar la representación gráfica. En este caso, hemos utilizado "show_id=true" para que se muestre la etiqueta de cada vértice y "vertex_size= 4" para aumentar el tamaño del círculo que los encierra.

También podemos crear un grafo a partir de su matriz de adyacencia.

```
(%i7) mat: matrix([0,0,1,1,1],[0,0,1,0,1],[1,1,0,0,0],
                  [1,0,0,0,0],[1,1,0,0,0])$
(%i7) g2: from_adjacency_matrix(mat);
(%o7) GRAPH(5 vertices, 5 edges)
(%i8) print_graph(g2);
Graph on 5 vertices with 5 edges.
Adjacencies:
  4 :  1  0
  3 :  0
  2 :  1  0
  1 :  4  2
  0 :  4  3  2
(%o8) done
```

Como vemos, los vértices se etiquetan con los números consecutivos desde el 0, siguiendo el orden dado por las filas y columnas. □

EJEMPLO 3.2.12 El grafo **completo** K_n es el grafo de n vértices en el que cualquier par de vértices está conectado por una arista. Por ejemplo, el grafo K_5 es el que mostramos a continuación:



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

Obsérvese que todos los elementos de la matriz son iguales a 1 excepto los de la diagonal principal.

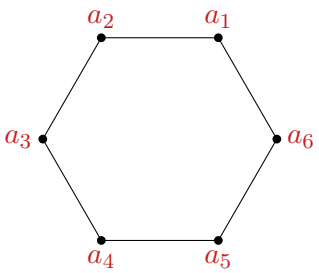
Por otra parte, para hacer el dibujo del grafo, hemos necesitado que las aristas "se corten" en puntos que no corresponden a vértices. Como veremos más adelante, esto es inevitable en algunos grafos y por eso es conveniente remarcar claramente los puntos del dibujo que corresponden con vértices.

En `Maxima`, estos grafos están predefinidos:

```
(%i1) load(graphs)$
(%i2) k5: complete_graph(5)$
(%i3) draw_graph(k5);
```

Nos muestra el grafo completo de 5 vértices. □

EJEMPLO 3.2.13 El **ciclo** C_n es un grafo de n vértices que están conectados formando una cadena cerrada. Vemos a continuación el ciclo C_6:



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

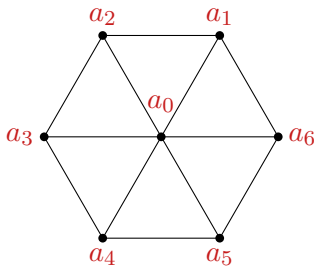
Obsérvese que hemos obtenido esa matriz de adyacencia porque los vértices se han ordenado siguiendo el recorrido circular sobre el ciclo.

En `Maxima`, estos grafos están predefinidos:

```
(%i1) load(graphs)$
(%i2) c6: cycle_graph(6)$
(%i3) draw_graph(c6);
```

Nos muestra el ciclo de 6 vértices. □

EJEMPLO 3.2.14 La **rueda** W_n es un grafo de $n+1$ que se obtiene añadiendo un vértice al ciclo C_n que está conectado con todos sus vértices. Mostramos a continuación el ejemplo W_6



$$\left(\begin{array}{c|cccccc} 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}\right)$$

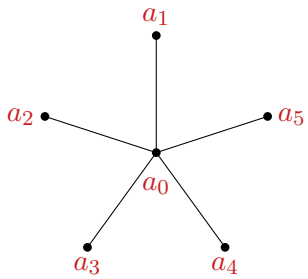
Hemos añadido una primera fila y una primera columna a la matriz de adyacencia de C_6 que corresponden al vértice central de la rueda.

En `Maxima`, estos grafos están predefinidos:

```
(%i1) load(graphs)$
(%i2) w6: wheel_graph(6)$
(%i3) draw_graph(w6);
```

Nos muestra la rueda de 7 vértices. □

EJEMPLO 3.2.15 La **estrella** S_n es un grafo de $n+1$ vértices que se obtiene eliminando del grafo W_n las aristas correspondientes al ciclo C_n. Vemos a continuación el ejemplo S_5:

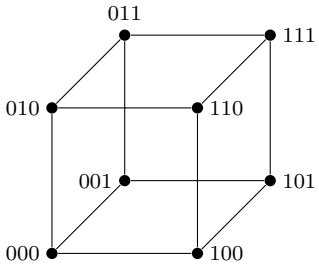


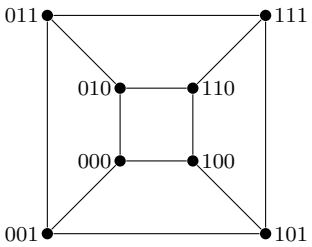
$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

□

EJEMPLO 3.2.16 El n**-cubo**, es el grafo simple $Q_n = (V_n, E_n)$ definido como sigue. Los vértices son las cadenas binarias de longitud n, es decir, $V = \{0,1\}^n$, que está formado por 2^n vértices. Las aristas son pares de cadenas que se diferencian en un solo elemento; por ejemplo, $\{010, 110\} \in E_3$, pero $\{010, 111\} \notin E_3$.

Representamos a continuación el grafo Q_3 con dos dibujos. Uno con la forma de cubo que justifica el nombre de este tipo de grafos y otro en el que las aristas se trazan sin cortarse.





En `Maxima`, estos grafos están predefinidos:

```
(%i1) load(graphs)$
(%i2) cube4: cube_graph(4)$
(%i3) draw_graph(cube4);
```

Nos muestra un cubo de dimensión 4, que se forma a partir de listas de longitud cuatro, tiene 16 vértices y 32 aristas. □

DEFINICIÓN 3.2.17 (GRAFOS BIPARTITOS) *Un grafo $G = \langle V, E\rangle$ se dice que es **bipartito** si $V = V_1 \cup V_2$, $V_1 \cap V_2 = \varnothing$, ningún par de vertices en V_1 está conectado y ningún par de vertices en V_2 está conectado.*

EJEMPLO CON MAXIMA 3.2.18 En `Maxima` disponemos de un predicado que analiza si un grafo es bipartito, `is_bipartite` y un operador que determina las dos partes de un grafo si es bipartito, `bipartition`.

- El grafo C_6 (ciclo de longitud 6) es bipartito:

$$V_1 = \{a_1, a_3, a_5\}, \qquad V_2 = \{a_2, a_4, a_6\}$$

```
(%i1) load(graphs)$
(%i2) is_bipartite(cycle_graph(6));
(%o2) true
(%i3) bipartition(cycle_graph(6));
(%o3) [[4,2,0],[5,3,1]]
```

- El grafo S_5 (estrella de 5 puntas) también es bipartito:

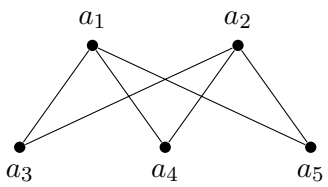
$$V_1 = \{a_0\}, \qquad V_2 = \{a_1, a_2, a_3, a_4, a_5\}$$

- Sin embargo, el grafo W_6 (rueda de 7 vértices) no es bipartito.

```
(%i5) is_bipartite(wheel_graph(6));
(%o5) false
(%i6) bipartition(wheel_graph(6));
(%o6) []
```

□

Ejemplo con Maxima 3.2.19 El grafo **bipartito completo** $K_{m,n}$ es el grafo con $m+n$ vértices en donde el conjunto de vértices está partido en dos conjuntos V_1 con m vértices y V_2 con n vértices, de tal forma que cada vértice de V_1 está conectado con cada vértice de V_2. Vemos abajo el grafo bipartito completo $K_{3,2}$:



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

Para escribir la matriz de adyacencia, hemos ordenado los vértices según su subíndice, los dos primeros corresponden a una parte de los vértices y los tres últimos a la otra parte. De esta forma, la matriz ha quedado formada por cuatro cajas que contienen o bien unos o bien ceros.

El operador `complete_bipartite_graph` define en `Maxima` operadores bipartitos completos del tamaño que deseemos. La siguiente línea siguiente mostrará el grafo del ejemplo anterior.

```
(%i1) load(graphs)$
(%i2) draw_graph(complete_bipartite_graph(3,2),
          vertex_size=3,show_id=true);
```

□

3.2.4. Conexión en grafos

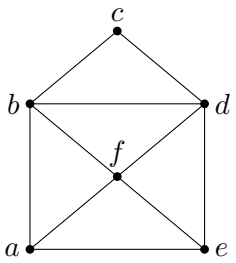
En esta sección vamos a introducir distintos conceptos y propiedades sobre la conexión de vértices en un grafo que se establece recorriendo una o varias aristas.

Definición 3.2.20 *Un **camino** (en inglés, se dice **walk**) entre los vértices v_0 y v_k de un grafo $G = (V, E)$, es una secuencia finita de vértices, no necesariamente distintos,*

$$C = v_0 v_1 v_2 \dots v_k$$

tal que $e_i = \{v_{i-1}, v_i\} \in E$ para todo $i = 1 \dots k$. En este caso, decimos que el camino C ***recorre*** *las aristas e_1, e_2,...,e_k y pasa por los vértices v_0, v_1,... v_k. El vértice v_0 se denomina* ***vértice inicial*** *de C y v_k se denomina* ***vértice final*** *de C. El número natural k es la longitud de C, es decir, el número de aristas que recorre.*

Ejemplo 3.2.21 La secuencia $C = abfdefbc$ es un camino del grafo

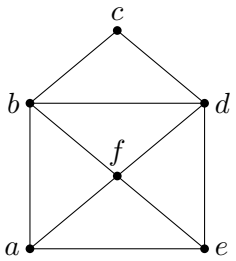


□

Definición 3.2.22 *Sea $C = v_0v_1v_2 \dots v_k$ un camino en un grafo $G = (V, E)$ y sea $e_i = \{v_{i-1}, v_i\}$ para cada $i = 1 \dots k$.*

- *C se dice que es* ***simple*** *(en inglés, se denomina* ***trail****) si $e_i \neq e_j$ para cada cada $i \neq j$; es decir, cada arista se recorre una única vez.*
- *C se dice que es* ***elemental*** *(en inglés, un camino elemental se denomina* ***path****), si $v_i \neq v_j$, para cada $i \neq j$; es decir por cada vértice se pasa a lo sumo una vez.*
- *C es un camino* ***cerrado*** *si $v_0 = v_k$, es decir, si empieza y termina en el mismo vértice.*
- *El camino C se dice que es un* ***circuito*** *si es un camino cerrado que no repite aristas.*
- *El camino C se dice que es un* ***ciclo*** *si es un camino cerrado en el que todos los vértices son distintos.*

Ejemplo 3.2.23 Consideramos el siguiente grafo simple



- El camino $C_1 = abdcbfe$ es simple.

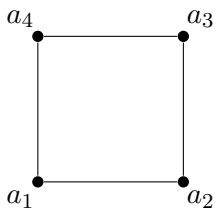
- El camino $C_2 = abfde$ es elemental
- El camino $C_3 = abdcba$ es cerrado.
- El camino $C_4 = abfdbcdea$ es un circuito.
- El camino $C_5 = abfdea$ es un ciclo. □

DEFINICIÓN 3.2.24 *Un grafo $G = (V, E)$ se dice* ***conexo*** *si hay un camino entre cada par de vértices distintos del grafo.*

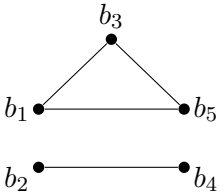
EJEMPLO 3.2.25 Consideremos los grafos G y H dados por las siguientes matrices de adyacencia:

$$M_G = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \qquad M_H = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Mientras que G es un grafo conexo, H no lo es. Evidentemente, no es fácil deducir estas afirmaciones a partir de las matrices, pero puede serlo si observamos sus representaciones gráficas.



G



H

En ambos casos, la ordenación de los vértices para la construcción de las matrices está dado por por los subíndices. □

En el ejemplo anterior, observamos que el grafo H, que no es conexo, es unión de dos subgrafos conexos y tales que no tienen aristas entre los vértices de uno y los vértices del otro. Estos subgrafos conexos disjuntos se denominan **componentes conexas** del grafo. En particular, un grafo es conexo si y solo si tiene exactamente una componente conexa.

EJEMPLO CON MAXIMA 3.2.26 En `Maxima`, el operador `is_conneceted` analiza si un grafo es o no conexo y `connected_components` nos devuelve las componentes conexas.

```
(%i1) load(graphs)$
(%i2) g1: create_graph(5,[[1,2],[1,3],[2,3],[0,4]])$
(%i3) is_connected(g1);
(%o3) false
(%i4) connected_components(g1);
(%o4) [[2,1,3],[0,4]]
(%i5) g2: create_graph([1,2,3,4,5,6,7,8], [[1,2], [1,6],
        [1,8],[1,3], [2,4],[2,5],[2,7],[3,4],[3,5],[3,7],
        [4,6],[4,8],[5,6],[5,8],[6,7],[7,8]])$
(%i6) is_connected(g2);
(%o6) true
(%i7) connected_components(g2);
(%o7) [[3,8,6,7,5,4,2,1]]
```

□

El siguiente resultado nos dice como calcular el número de caminos de una determinada longitud entre dos vértices, y nos dará un método para estudiar si un grafo es conexo a partir de su matriz de adyacencia.

TEOREMA 3.2.27 *Sea G un grafo con matriz de adyacencia M_G y sea $M_G^k = (m_{ij})$ la potencia k-ésima, calculada utilizando la suma y el producto en $\mathbb{N}$, de la matriz M_G. Entonces, m_{ij} es el número de caminos de longitud k entre los vértices v_i y v_j.*

EJEMPLO 3.2.28 Volvamos a considerar los grafos del ejemplo 3.2.25:

$$M_G = \begin{pmatrix} 0&1&0&1\\ 1&0&1&0\\ 0&1&0&1\\ 1&0&1&0 \end{pmatrix} \qquad M_H = \begin{pmatrix} 0&0&1&0&1\\ 0&0&0&1&0\\ 1&0&0&0&1\\ 0&1&0&0&0\\ 1&0&1&0&0 \end{pmatrix}$$

Empezamos estudiando G y calculamos M_G^2:

$$M_G^2 = \begin{pmatrix} 0&1&0&1\\ 1&0&1&0\\ 0&1&0&1\\ 1&0&1&0 \end{pmatrix} \begin{pmatrix} 0&1&0&1\\ 1&0&1&0\\ 0&1&0&1\\ 1&0&1&0 \end{pmatrix} = \begin{pmatrix} 2&0&2&0\\ 0&2&0&2\\ 2&0&2&0\\ 0&2&0&2 \end{pmatrix}$$

El número 2 de la posición $(1,3)$ en la matriz M_G^2, significa que hay dos caminos de longitud 2 que conectan los vértices a_1 y a_3; estos caminos son $C_1 = a_1a_2a_3$ y $C_2 = a_1a_4a_3$. Dado que cada una de las posiciones en una matriz 4×4 es distinta de cero en la matriz M_G^2 o en la matriz M_G, podemos deducir que el grafo es conexo, ya que cada par de vértices (determinado por cada posición en la matriz) están conectados por al menos un camino de longitud menor o igual que 2.

Calculemos ahora las potencias sucesivas de M_H.

$$M_H = \begin{pmatrix} 0&0&1&0&1\\ 0&0&0&1&0\\ 1&0&0&0&1\\ 0&1&0&0&0\\ 1&0&1&0&0 \end{pmatrix}; \qquad M_H^2 = \begin{pmatrix} 2&0&1&0&1\\ 0&1&0&0&0\\ 1&0&2&0&1\\ 0&0&0&1&0\\ 1&0&1&0&2 \end{pmatrix};$$

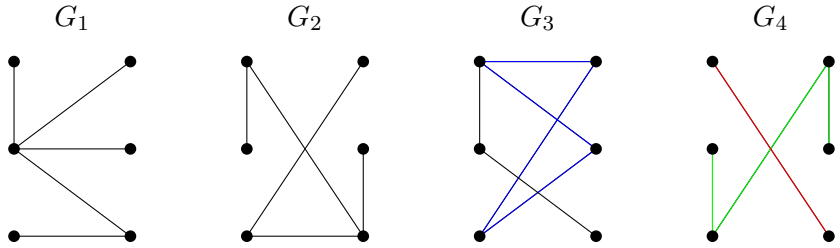
$$M_H^3 = \begin{pmatrix} 2&0&3&0&3\\ 0&0&0&1&0\\ 3&0&2&0&3\\ 0&1&0&0&0\\ 3&0&3&0&2 \end{pmatrix}; \qquad M_H^4 = \begin{pmatrix} 6&0&5&0&5\\ 0&1&0&0&0\\ 5&0&6&0&5\\ 0&0&0&1&0\\ 5&0&5&0&6 \end{pmatrix}$$

Observamos entonces que hay doce posiciones que son nulas en las cuatro matrices, las que determinan el número de caminos entre los vértices b_1 y b_2, entre los vértices b_1 y b_4, entre los vértices b_2 y b_5, entre los vértices b_4 y b_5 y entre los vértices b_3 y b_4. Podemos deducir que H no es conexo, ya que cuatro es el número de aristas del grafo y si un grafo es conexo, se debería poder conectar cada par de vértices con un número menor o igual que el total de las aristas. □

3.2.5. Árboles

DEFINICIÓN 3.2.29 *Un **árbol** es un grafo conexo sin ciclos.*

Por ejemplo, los grafos G_1 y G_2 son árboles, pero G_3 y G_4 no:



El grafo G_3 no es un árbol porque contiene un ciclo, mientras que G_4 no lo es porque no es conexo.

El matemático Arthur Cayley fue quien usó árboles por primera vez en 1857 para representar ciertos tipos de componentes químicos. En ciencias de la computación los árboles nos sirven para: almacenar y recuperar información; construir algoritmos eficientes para hacer búsquedas; construir códigos eficientes para almacenar y transmitir datos; modelar procedimientos que se realizan usando una secuencia de decisiones.

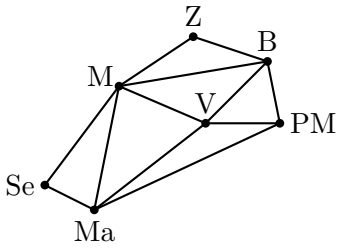
TEOREMA 3.2.30 *Sea $G = (V, E)$ un grafo simple. Los siguientes enunciados son equivalentes:*

1. *G es un árbol.*
2. *G es conexo y $|E| = |V| - 1$.*
3. *G no tiene ciclos y $|E| = |V| - 1$.*
4. *En G hay exactamente un camino entre cada par de vértices.*
5. *G es conexo, pero si eliminamos una arista cualquiera, el grafo resultante no es conexo.*

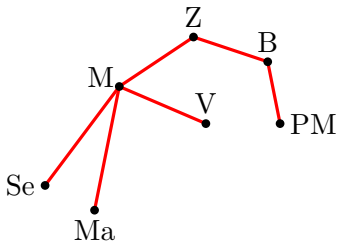
Los árboles tienen propiedades de minimalidad, en el sentido de que son los grafos conexos con el menor número posible de aristas. Por esa razón, nos sirven para modelizar redes de comunicación con un mínimo número de conexiones entre nodos.

DEFINICIÓN 3.2.31 *Un **árbol generador** (en inglés, se dice **spanning tree**) de un grafo simple G es un subgrafo generador de G que además es un árbol; es decir, es un subgrafo que es árbol y contiene todos los vértices de G.*

EJEMPLO 3.2.32 El grafo de la derecha en un árbol generador del grafo G de la izquierda:

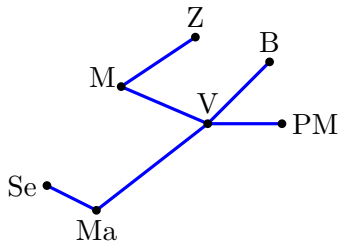


G



Árbol generador de G

Un grafo puede tener más de un árbol generador; por ejemplo, abajo aparece otro árbol generador del mismo grafo G.



□

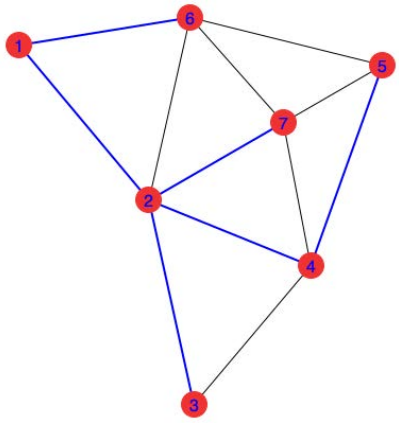
Los árboles generadores se pueden usar para estudiar propiedades de los grafos y definir sobre ellos álgoritmos y funciones.

Ejemplo con Maxima 3.2.33 En `Maxima`, el operador `is_tree` analiza si un grafo es o no árbol.

```
(%i1) load(graphs)$
(%i2) is_tree(cube_graph(3));
(%o2) false
(%i3) arb: create_graph(8,[[0,5],[0,6],[1,3],[2,4],[2,6],[2,7],[3,6]])$
(%i4) is_tree(arb);
(%o4) true
```

Podemos determinar árboles generadores de un grafo, concretamente aquellos que contienen el menor número de aristas; para ello usamos el operador `minimun_spanning_tree`.

```
(%i5) gt: create_graph([1,2,3,4,5,6,7],
        [[1,2],[1,6],[2,3],[2,4],[2,6],[2,7],[3,4],
        [4,5],[4,7],[5,6],[5,7],[6,7]])$
(%i6) sgt: minimum_spanning_tree(gt)$
(%i7) draw_graph(gt, vertex_size=4,
        show_id=true, show_edges=edges(sgt));
(%o8) done
```



En el código anterior hemos usado la opción `show_edges=edges(sgt)` para que se resalten las aristas del subgrafo `sgt`, es decir, del subárbol generador minimal. □

Teorema 3.2.34 *Un grafo simple G es conexo si y sólo si tiene un árbol generador T.*

Demostración: Por definición, un árbol es conexo y si es subárbol de otro grafo, necesariamente este también será conexo. Para demostrar que un grafo conexo contiene un árbol generador, vamos a probar que podemos construirlo siguiendo el siguiente proceso. Si G no es un árbol, entonces contiene un ciclo; si eliminamos una

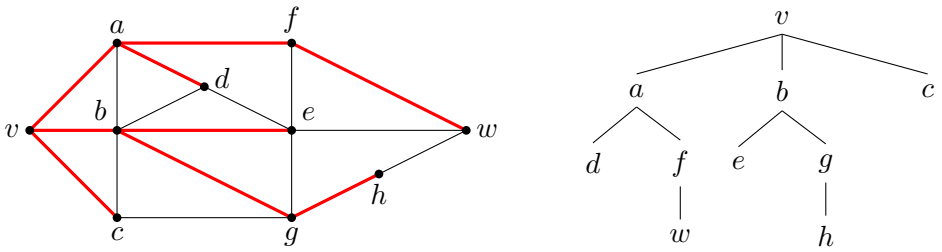
arista de este ciclo, estamos construyendo un subgrafo generador. En este subgrafo repetimos el análisis; si no es un árbol, entonces contiene un ciclo y eliminando una de sus aristas construimos un nuevo subgrafo. Para construir el árbol generador, solo tendremos que repetir el proceso eliminando tantas aristas como sea necesario. □

El algoritmo que hemos utilizado en la demostración anterior nos permite determinar fácilmente un árbol generador. Vemos a continuación otros métodos de generación de estos árboles mediante lo que se denominan árboles de búsqueda.

Búsqueda en anchura. El siguiente algoritmo construye un árbol generador siguiendo un recorrido en el grafo mediante **búsqueda en anchura**. La búsqueda en anchura es adecuada para resolver problemas de optimización, en los que se deba elegir la mejor solución entre varias posibles. También los usamos, por ejemplo, para colorear grafos bipartitos o para encontrar el camino más corto en un grafo entre dos vértices.

1. Elegimos un vértice arbitrario v_0, que llamaremos **raíz**.
2. Añadimos todas las aristas incidentes en v_0.
3. Para cada uno de los vértices conectados con los vértices añadido en el paso anterior, añadimos todas las aristas que inciden en ellos si el otro vértice que conectan no se había añadido ya al árbol.
4. Con los nuevos vértices, procedemos de la misma forma hasta añadir todos los vértices de árbol.

Ejemplo 3.2.35 Empezando en el vértice v, vamos a construir un árbol generador del siguiente grafo usando búsqueda en anchura.



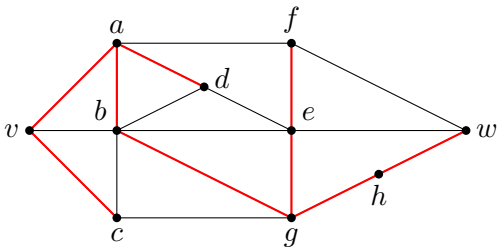
A la derecha mostramos el árbol generador dibujado con la forma habitual de un árbol. □

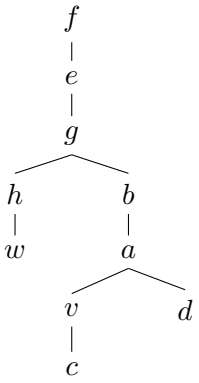
Búsqueda en profundidad. El siguiente algoritmo construye un árbol generador siguiendo un recorrido en el grafo mediante **búsqueda en profundidad**. Esta

búsqueda y los árboles así construidos se usan, por ejemplo, para encontrar las componentes conexas o para comprobar si un grafo es acíclico (es decir, no contiene ciclos).

- Elegimos un vértice arbitrario v_0, que llamamo raíz.
- Construimos un camino que comenzando en v_0 y añadiendo sucesivamente aristas y vértices mientras sea posible sin utilizar vértices ya añadidos al camino.
- Si el camino así construido pasa por todos los vértices del grafo, entonces el árbol generador es dicho camino. En caso contrario, retrocedemos al penúltimo vértice del camino y, si es posible, formamos un nuevo camino que empiece en este vértice y que pase por vértices no visitados.
- Si esto no se puede hacer, lo intentamos retrocediendo al vértice anterior.
- Repetimos el proceso hasta añadir todos los vértices.

Ejemplo 3.2.36 Empezando en el vértice f, hemos construido el árbol generador del siguiente grafo mediante búsqueda en profundidad.



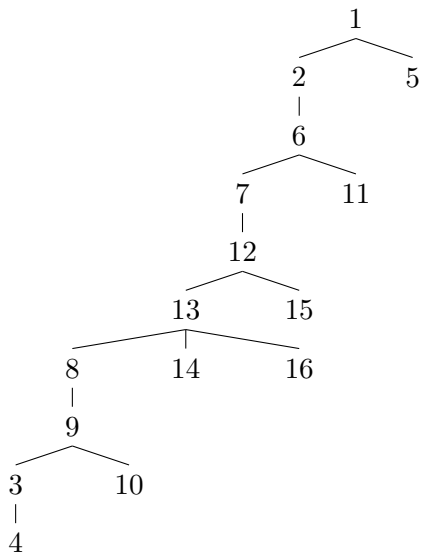


A la derecha, hemos dibujado el árbol con su forma habitual. □

Ejemplo 3.2.37 Considera el grafo dado por la siguiente lista de adyacencia

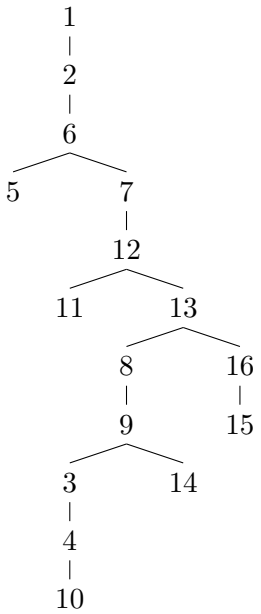
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	1	4	3	1	2	6	9	3	4	6	7	8	9	12	13
5	6	9	10	6	5	12	13	8	9	12	11	12	13	16	15
					7			10			13	14			
					11			14			15	16			

Un árbol generador construido mediante búsqueda en anchura es el siguiente



Aunque este dibujo no representa a todo el grafo, ya que no incluye todas las aristas, sí nos puede ayudar a estudiar sus propiedades; por ejemplo, podemos deducir que el grafo es conexo, ya que incluye todos los vértices.

A continuación, mostramos otro árbol generador construido mediante búsqueda en profundidad. Para construir los caminos, elegimos los vértices según el orden numérico.



Igual que en el árbol anterior, debemos tener en cuenta que este dibujo no representa a todo el grafo, ya que no incluye todas las aristas. □

3.2.6. Isomorfismo de grafos

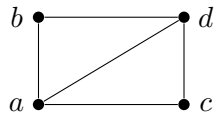
DEFINICIÓN 3.2.38 *Se dice que dos grafos simples $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$ son **isomorfos** si existe una función biyectiva $\phi\colon V_1 \to V_2$ tal que, para todo $u, v \in V_1$,*

$$\{u, v\} \in E_1 \quad \textit{si y solo si} \quad \{\phi(u), \phi(v)\} \in E_2$$

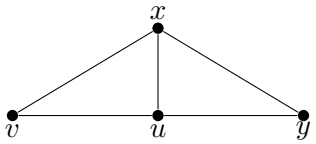
*Decimos también que la función ϕ es un **isomorfismo de grafos**.*

Es decir, dos grafos son isomorfos si entre sus vértices existe una función biyectiva que conserva las adyacencias en los dos sentidos. Matricialmente, esto significa que existe una reordenación de los vértices de G_1 y G_2 de manera que las matrices de adyacencia son exactamente iguales.

EJEMPLO 3.2.39 Los siguientes grafos son isomorfos



$G_1 = (V_1, E_1)$



$G_2 = (V_2, E_2)$

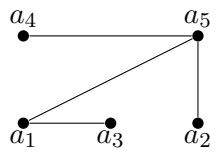
Y el isomorfismo está dado por la siguiente biyección:

$$\begin{array}{rcl} \phi : V_1 & \to & V_2 \\ a & \mapsto & u \\ b & \mapsto & v \\ c & \mapsto & y \\ d & \mapsto & x \end{array} \qquad \left\{ \begin{array}{l} \{a,b\} \in E_1 \iff \phi(\{a,b\}) = \{u,v\} \in E_2 \\ \{a,c\} \in E_1 \iff \phi(\{a,c\}) = \{u,y\} \in E_2 \\ \{a,d\} \in E_1 \iff \phi(\{a,d\}) = \{x,u\} \in E_2 \\ \{b,d\} \in E_1 \iff \phi(\{b,d\}) = \{v,x\} \in E_2 \\ \{c,d\} \in E_1 \iff \phi(\{c,d\}) = \{x,y\} \in E_2 \end{array} \right.$$

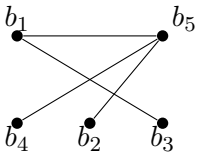
□

La relación de isomorfía entre grafos es una relación de equivalencia. Cada clase de equivalencia definida por esta relación es un conjunto de grafos isomorfos que se denomina **grafo no etiquetado**.

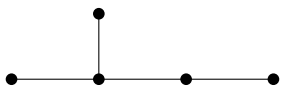
EJEMPLO 3.2.40 Los grafos G_1 y G_2 de la siguiente figura son isomorfos, basta tomar la función dada por $\phi(a_i) = b_i$.



G_1



G_2



G_3

Ambos, como grafos isomorfos, pertenecen a la clase representada por un grafo no etiquetado que podemos dibujar, por ejemplo, como G_3. □

Por lo general, verificar si dos grafos son o no isomorfos es un problema bastante complejo cuando trabajamos con grafos con gran número de vértices. Para abordar el problema, en primer lugar se analizan características necesarias para que dos grafos sean o no isomorfos y cuyo análisis tenga poco coste computacional. Como último recurso, se procedería a la búsqueda sistemática de un isomorfismo.

Por ejemplo, en primer lugar podríamos obtener y analizar lo que se llama **invariantes** de un grafo, es decir, magnitudes que se mantienen bajo isomorfismo. Si dos grafos difieren en alguna de estas invariantes, no pueden ser isomorfos. Algunas invariantes son:

1. Número de vértices.
2. Número de aristas.
3. *Sucesión gráfica*, es decir la lista de grados de los vértices, ordenados en orden decreciente.
4. Número de componentes conexas.
5. Número de ciclos o circuitos de determinada longitud incluidos en el grafo.

Y otros características que iremos estudiando a lo largo del tema, como coloración, planaridad,...

Ejemplo 3.2.41 Los grafos G_1 y G_2 no son isomorfos, aunque tiene el mismo número de aristas y vértices y la secuencia gráfica es la misma en ambos grafos, $(3, 3, 2, 2, 2)$.

G_1 G_2

Para demostrar que no son isomorfos, basta observar el grafo G_2 contiene un ciclo de longitud 3, mientras que G_1 no contiene ningún ciclo de longitud 3. □

Ejemplo 3.2.42 Los grafos dados por las siguientes listas de adyacencia no son isomorfos.

G_1

1	2	3	4	5	6	7	8	9
2	1	2	3	4	1	5	6	3
6	3	4	5	7	7	6		
		9			8			

G_2

1	2	3	4	5	6	7	8	9
2	1	4	3	2	1	3	9	8
6	5	7	7	6	2	4		
	6				5			

El número de vértices y arístas es el mismo, y la sucesión gráfica también es la misma en ambos grafos: $(3, 3, 2, 2, 2, 2, 1, 1)$. Sin embargo, el grafo G_1 es conexo y solo tiene una componente conexa, mientras que el grafo G_2 tiene tres, las formadas por los siguientes conjuntos de vértices: $\{1, 2, 5, 6\}, \{3, 4, 7\}, \{8, 9\}$. □

Ejemplo 3.2.43 Los siguientes grafo no son isomorfos.

G_1 G_2

Vemos que el número de vértices y aristas es el mismo, y también lo es la sucesión gráfica. Además, ninguno de los dos contiene ciclos ni circuitos. En este caso, nos podemos fijar en los dos vértices de grado 3 de los: en G_1, estos dos vértices están conectados, mientras que en G_2, no lo están. □

Ejemplo con Maxima 3.2.44 En `Maxima`, disponemos del operador `isomorphism` que analiza si dos grafos son isomorfos y, en tal, caso determina un isomorfismo. Los tres ejemplos que mostramos a continuación tienen la misma secuencia gráfica, pero solo dos de ellos son isomorfos.

```
(%i1) load(graphs)$
(%i2) g4: create_graph([1,2,3,4,5],[[1,4],[1,5],[2,3],
        [2,4],[2,5],[3,4]])$
      degree_sequence(g4);
(%o2) [2,2,2,3,3]
(%i3) g5: create_graph([1,2,3,4,5],[[1,2],[1,3],[1,5],
        [2,3],[3,4],[4,5]])$
      degree_sequence(g5);
(%o3) [2,2,2,3,3]
(%i4) g6: create_graph([1,2,3,4,5],[[1,4],[1,5],[2,4],
        [2,5],[3,4],[3,5]])$
```

```
          degree_sequence(g6);
(%o4)  [2,2,2,3,3]
(%i5)  isomorphism(g4,g5);
(%o5)  [4->3,5->5,3->2,1->4,2->1]
 (%i6)  isomorphism(g4,g6);
(%o6)  []
```

□

3.2.7. Grafos Eulerianos

Los grafos **eulerianos** deben su nombre a Leonhard Paul Euler, que es considerado el padre de la teoría de grafos. Esto se debe a que en 1736, resolvió matemáticamente un entretenimiento habitual en la ciudad de Königsberg (nombre de la ciudad rusa actualmente conocida como Kaliningrado y que entonces pertenecía a Prusia Oriental). Los habitantes de la ciudad se preguntaban si era posible recorrer los siete puentes que en aquel momento permitían cruzar el río Pregel, pero pasando solamente una vez por cada puente (ver figura 3.2). Euler demostró que no era posible hacerlo, ya que el número de puentes que se podían tomar era impar en más de dos zonas o islas.

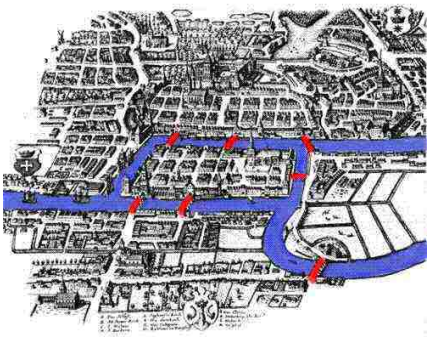
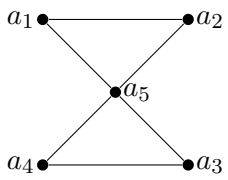


Figura 3.2: Mapa de la ciudad de Köningsber en el siglo XVIII.[1]

DEFINICIÓN 3.2.45 *Un **camino de Euler** o **euleriano** entre dos vértices distinos u y v de un grafo, es un camino que recorre cada arista del grafo exactamente una vez, es decir, un camino simple que recorre todas las aristas. Un **circuito de Euler** o **euleriano** es un camino cerrado que recorre cada arista del grafo exactamente una vez; si existe un circuito de Euler, decimos que el grafo es **euleriano**.*

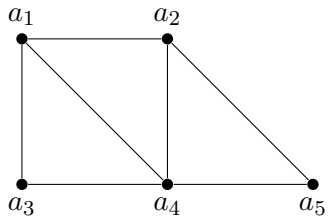
[1] Imagen de Bogdan Giuşcă, CC BY-SA 3.0, tomada de Wikipedia. http://commons.wikimedia.org/wiki/File:Konigsberg_bridges.png

Ejemplo 3.2.46 El grafo de la figura es euleriano, ya que todas sus aristas se pueden recorrer con un circuito de euler.



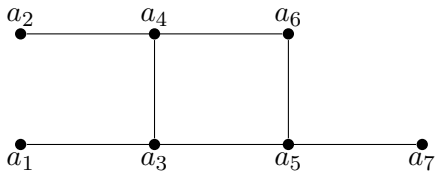
Un posible circuito de Euler es $C = a_1\ a_5\ a_3\ a_4\ a_5\ a_2\ a_1$. □

Ejemplo 3.2.47 Para el grafo de la figura siguiente, no es posible encontrar un circuito de Euler, pero sí es posible recorrer todas sus aristas con un camino de Euler.



Los caminos de Euler de este grafo deben conectar los vértices a_1 y a_2 y uno de ellos es: $C = a_1\ a_3\ a_4\ a_5\ a_2\ a_1\ a_4\ a_2$. □

Ejemplo 3.2.48 El siguiente grafo no contiene ni caminos ni circuitos de Euler



Los resultados siguientes fueron demostrados por Euler para justificar la imposibilidad de realizar el recorrido por todos los puentes de Köningsberg y justifican las afirmaciones hechas en el estudio de los ejemplos anteriores.

Teorema 3.2.49 *Un grafo simple tiene un circuito de Euler si y solo si es conexo y todos los vértices son de grado par.*

Corolario 3.2.50 *Un grafo simple tiene un camino de Euler entre los vértices u, v si y solo si es conexo y esos vértices son los únicos con grado impar.*

Los dos resultados anteriores son válidos también sobre multigrafos. De hecho, el grafo que representa el pasatiempo de lo puentes de Köningsber es un multigrafo.

Hallar caminos y circuitos de Euler es bastante simple y disponemos de varios algoritmos para ello. Los más conocidos son el **algoritmo de Fleury** y el **algoritmo de Hierholzer**; este segundo es el más eficiente y lo describimos a continuación.

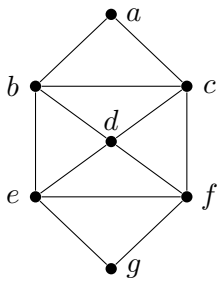
Algoritmo de Hierholzer. Se lo aplicamos a un grafo simple conexo cuyos vértices tienen grado par o que contiene exactamente dos vértices de grado impar.

1. Si el grafo tiene dos vértices de grado impar, elegimos uno de ellos; si todos los vértices tienen grado par podemos elegir cualquier vértice.
2. A partir de ese vértice empezamos a recorrer aristas, sin repetir ninguna hasta que no podamos continuar más.

   Si hemos empezado en un vértice de grado impar, esto ocurrirá necesariamente construyendo un camino que termina en el otro vértice de grado impar. Si el grafo solo tiene vértices de grado par, esto ocurrirá necesariamente construyendo un circuito que termina en el mismo vértice en el que lo hemos iniciado.
3. Si el camino o circuito contiene a todas las aristas, ya hemos terminado. En caso contrario, elegimos un vértice del camino o circuito que hemos construido, y en el que incidan aristas fuera del camino o circuito parcial construido.
4. A partir de ese vértice, construimos otro circuito recorriendo nuevas aristas hasta que no podamos continuar más. Este circuito terminará necesariamente en el mismo vértice en el que hemos empezado. Este segundo circuito se puede insertar en el camino o circuito que ya habíamos construido.
5. Si el camino o circuito contiene a todas las aristas, ya hemos terminado. En caso contrario, repetimos el proceso descrito en el punto anterior hasta conseguir incluir todas las aristas.

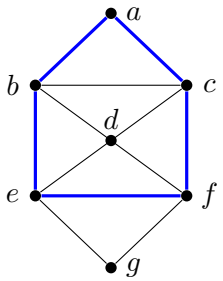
El algoritmo anterior es *no determinista*, ya que la elección de los vértices y la construcción de los circuitos parciales se hace de manera arbitraria.

Ejemplo 3.2.51 Vamos a determinar una circuito de Euler en el siguiente grafo



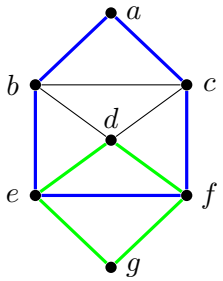
Empezamos eligiendo un vértice y construyendo un circuito que empiece y termine en ese vértice. Por ejemplo, desde el vértice a podemos construir el circuito.

$$a \blacktriangleright b \blacktriangleright e \blacktriangleright f \blacktriangleright c \blacktriangleright a$$



Excepto el vértice a, los demás vértices tienen aristas incidentes que no hemos incluido en el circuito. Elegimos uno cualquiera, por ejemplo e, y construimos un circuito que empiece y termine en e y utilizando aristas que todavía no hemos recorrido:

$$e \blacktriangleright d \blacktriangleright f \blacktriangleright g \blacktriangleright e$$



Insertamos entonces el circuito "verde" en el circuito "azul",

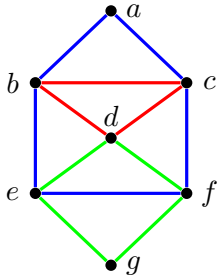
$$\overbrace{a \blacktriangleright b \blacktriangleright e \blacktriangleright f \blacktriangleright c \blacktriangleright a}^{e \blacktriangleright d \blacktriangleright f \blacktriangleright g \blacktriangleright e}$$

y obtenemos un circuito que recorre más aristas del grafo inicial

$$a \blacktriangleright b \blacktriangleright e \blacktriangleright d \blacktriangleright f \blacktriangleright g \blacktriangleright e \blacktriangleright f \blacktriangleright c \blacktriangleright a$$

Las aristas $\{b, c\}$, $\{c, d\}$, y $\{d, b\}$ no están en el circuito parcial, elegimos una de ellas para construir el siguiente. Por ejemplo, elegimos el vértice d y la arista $\{d, b\}$ para construir el siguiente circuito parcial:

$$d \blacktriangleright b \blacktriangleright c \blacktriangleright d$$



Insertamos este último circuito en el anterior

$$\overbrace{a \blacktriangleright b \blacktriangleright e \blacktriangleright d \blacktriangleright f \blacktriangleright g \blacktriangleright e \blacktriangleright f \blacktriangleright c \blacktriangleright a}^{d \blacktriangleright b \blacktriangleright c \blacktriangleright d}$$

y obtenemos el circuito de Euler del grafo inicial

$$a \blacktriangleright b \blacktriangleright e \blacktriangleright d \blacktriangleright b \blacktriangleright c \blacktriangleright d \blacktriangleright f \blacktriangleright g \blacktriangleright e \blacktriangleright f \blacktriangleright c \blacktriangleright a$$ □

Ejemplo 3.2.52 Consideremos el grafo dado por la siguiente lista de adyacencia:

a	b	c	d	e	f	g	h
c	e	a	a	b	c	b	e
d	g	d	c	c	d	d	g
		e	f	g		e	
		f	g	h		h	

Todos los vértices tienen grado par, por lo que es euleriano. Vamos a utilizar la búsqueda primero en profundidad para construir los circuitos parciales en el algoritmo de Hierholzer. Empezamos por el vértice a y vamos describiendo un circuito eligiendo el primer vértice en la columna del cada vértice que no hayamos incluido previamente. Indicaremos con un superíndice el orden en el que vamos eligiendo y recorriendo los vértices. Empezando en a vamos a c; en la columna de c, descartamos a y vamos al siguiente vértice, que es d; en la columna d, descartamos c y vamos al siguiente vértice que es a, por lo que hemos construido el primer circuito.

a^1	b	c^2	d^3	e	f	g	h
c^2	e	a^1	a^1	b	c	b	e
d^3	g	d^3	c^2	c	d	d	g
		e	f	g		e	
		f	g	h		h	

$$a - c - d - a$$

Borramos las aristas recorridas para ver mejor las que nos quedan por recorrer:

a	b	c	d	e	f	g	h
	e			b	c	b	e
	g			c	d	d	g
		e	f	g		e	
		f	g	h		h	

El primer vértice de este circuito cuya columna tiene vértices sin visitar es c, así que a partir de él construimos el siguiente circuito parcial.

a	b^3	c^1	d^5	e^2	f^6	g^4	h
	e^2			b^3	c^1	b^3	e
	g^4			c^1	d^5	d^5	g
		e^2	f^6	g		e	
		f	g^4	h		h	

$$\overbrace{a - \boxed{c} - d - a}^{c - e - b - g - d - f - c}$$

Insertamos este segundo circuito en el anterior y obtenemos

$$a - c - e - b - g - d - f - c - d - a$$

Borramos las aristas que hemos incluido en el circuito para visualizar mejor las aristas pendientes

a	b	c	d	e	f	g	h
							e
							g
				g		e	
				h		h	

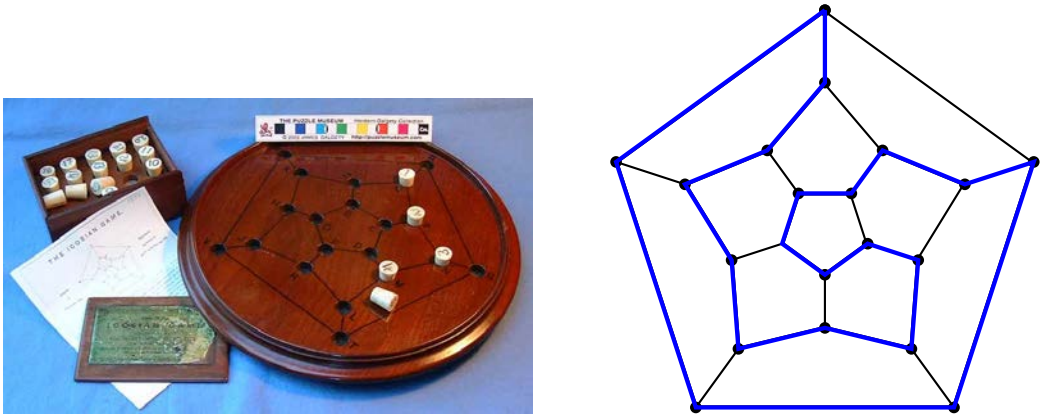


Figura 3.3: Ciclo hamiltoniano en el juego icosiano.[2]

El primer vértice con una arista incidente fuera del circuito es e; a partir de él, construimos el último circuito que insertaremos:

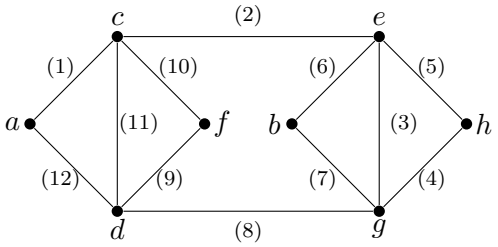
$$\begin{array}{cccccccc} a & b & c & d & e^1 & f & g^2 & h^3 \\ \hline & & & & & & & e^1 \\ & & & & & & & g^2 \\ & & & & g^2 & & e^1 & \\ & & & & h^3 & & h^3 & \end{array}$$

$$a - c - \boxed{e} - \underbrace{b - g - d - f - c - d - a}_{e - g - h - e}$$

Por lo tanto, el resultado es:

$$a - c - e - g - h - e - b - g - d - f - c - d - a$$

Vemos a continuación el dibujo del grafo con las aristas numeradas siguiendo el orden de recorrido en el circuito de Euler.



□

3.2.8. Grafos hamiltonianos

Los grafos hamiltonianos toman su nombre del matemático irlandés William Rowan Hamilton, que en 1857 presentó ante la Asociación Británica un juego llamado *Juego Icosiano*. Este juego contenía un tablero en la que aparecía dibujado

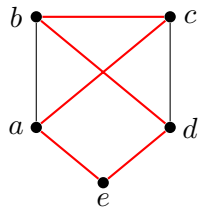
[2] La imagen de la izquierda está tomada de la web de *The Puzzle Museum*, (http://puzzlemuseum.com/month/picm02/200201hamilton.jpg).

un dodecaedro cuyos vértices estaban etiquetados con el nombre de veinte ciudades importantes y el objetivo era encontrar recorridos cerrados a través de las veinte ciudades que pasaran exactamente una vez por cada una de ellas. El juego no tuvo mucho éxito comercial y, aunque su origen estaba en unos estudios sobre estructuras algebraicas que generalizaban a los números complejos, suponía formular por primera vez el problema de encontrar en un grafo lo que en adelante se llamarían **ciclos de Hamilton**.

DEFINICIÓN 3.2.53 *Un **camino de Hamilton** o **hamiltoniano** entre dos vértices distintos u y v, es un camino que pasa por cada vértice del grafo exactamente una vez, es decir, un camino elemental que recorre todos los vértices. Un **ciclo de Hamilton** o **hamiltoniano** es un camino cerrado que recorre todas los vértices del grafo exactamente una vez. Un grafo se dice hamiltoniano si contiene un ciclo de Hamilton.*

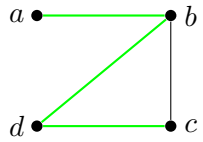
El grafo que aparece a la derecha en la figura 3.3 contiene un ciclo de Hamilton, que aparece resaltado en azul.

EJEMPLO 3.2.54 El siguiente grafo contiene ciclos de Hamilton, por ejemplo, el que aparece resaltado en rojo: $C = a\ c\ b\ d\ e\ a$;



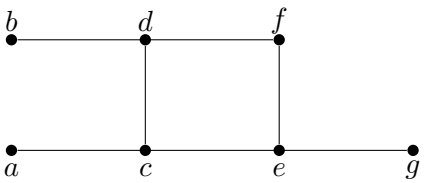
□

EJEMPLO 3.2.55 El siguiente grafo no contiene ciclos de Hamilton, pero sí contiene caminos de Hamilton: el que aparece resaltado en verde.



□

EJEMPLO 3.2.56 El siguiente grafo no contiene ni ciclos ni caminos de Hamilton.



□

El estudio de la existencia de caminos o ciclos de Hamilton es mucho más complejo que el correspondiente para los caminos y circuitos de Euler. No disponemos de resultados que caractericen los grafos que contienen estos caminos y que estén basados en los grados de los vértices. Tampoco hay algoritmos eficientes para determinar, si es posible, este tipo de caminos. Los siguientes resultados establecen condiciones suficientes, pero no necesarias, para afirmar la existencia de ciclos de Hamilton.

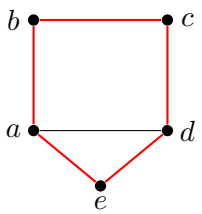
TEOREMA 3.2.57 (DIRAC) *Si $n \geq 3$ es el número de vértices de un grafo simple y todos ellos tienen grado mayor o igual a $\frac{n}{2}$, entonces el grafo contiene un ciclo hamiltoniano.*

TEOREMA 3.2.58 (ORE) *Si $n \geq 3$ es el número de vértices de un grafo simple y para cada par de vértices NO adyacentes u, v, se verifica que $\delta(u) + \delta(v) \geq n$, entonces el grafo contiene un ciclo hamiltoniano.*

El ejemplo 3.2.56 no verifica ni la condición del teorema de Dirac ni la condición del teorema de Ore. Sin embargo, eso no permite concluir que el grafo no contenga ciclos de Hamilton, debemos realizar la búsqueda sistemática de un ciclo y comprobar que no es posible hacerlo.

El ejemplo 3.2.54 no verifica la condición del teorema de Dirac, ya que el grado del vértice e es $2 < \frac{5}{2}$. Sin embargo, sí verifica la condición del teorema de Ore.

EJEMPLO 3.2.59 El siguiente grafo no verifica ni la condición del teorema de Dirac, ni la condición del teorema de Ore, sin embargo, sí contiene un ciclo de Hamilton: $C = a\ b\ c\ d\ e\ a$



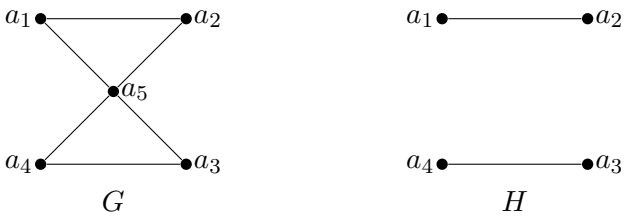
□

Para demostrar que un grafo no es hamiltoniano, podemos usar el siguiente resultado, que establece una condición necesaria, pero no suficiente.

TEOREMA 3.2.60 *Si G un grafo hamiltoniano y H un subgrafo de G obtenido eliminando n vértices (y las aristas incidentes en ellos), entonces el número de componentes conexas de H es menor o igual que n.*

Es decir, si al quitar n vértices obtenemos más de n componentes conexas, podemos afirmar que el grafo no es hamiltoniano.

EJEMPLO 3.2.61 El grafo G no es hamiltoniano, ya que su subgrafo H tiene dos componentes conexas y se ha obtenido eliminado solamente un vértice, el a_5:



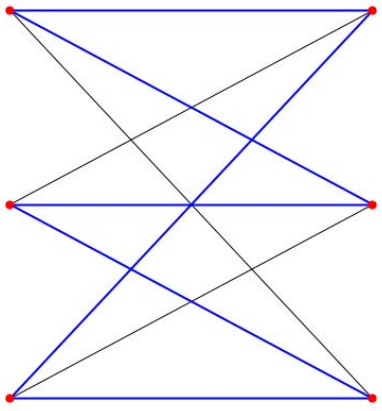
□

EJEMPLO CON MAXIMA 3.2.62 El operador `hamilton_cycle` determina si un grafo es o no Hamiltoniano y en tal caso calcula el ciclo de Hamilton.

```
(%i1) load(graphs)$
(%i2) k33: complete_bipartite_graph(3,3)$
(%i3) hk33: hamilton_cycle(k33);
(%o3)     [0,5,2,4,1,3,0]
```

Utilizando la opción `show_edges` del operador `draw_graph` podemos ver el dibujo de un grafo en el que se resalte un determinado camino, por ejemplo el ciclo de Hamilton de un grafo hamiltoniano.

```
(%i4) draw_graph(k33,show_edges=vertices_to_cycle(hk33));
(%o4)     done
```

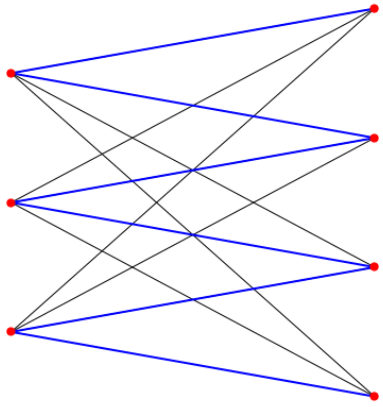


También podemos determinar el camino de Hamilton contenido en un grafo semihamiltoniano. Por ejemplo, sabemos que el grafo $K_{3,4}$ no es hamiltoniano, pero sí es semihamiltoniano.

```
(%i5) k34:complete_bipartite_graph(3,4)$
      hamilton_cycle(k34);
(%o5) []
(%i6) hk34: hamilton_path(k34);
(%o6) [6,2,5,1,4,0,3]
```

En este caso, el operador `vertices_to_path` en la opción `show_edges` también nos permite visualizar el camino de Hamilton.

```
(%i7) draw_graph(k34,show_edges=vertices_to_path(hk34));
(%o7) done
```

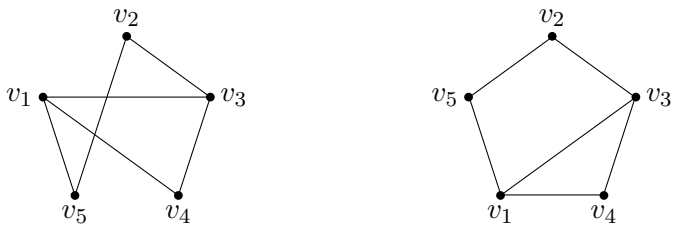


□

3.2.9. Planaridad

DEFINICIÓN 3.2.63 *Se dice que un grafo es* ***plano*** *si puede dibujarse en el plano sin que se corten ningún par de aristas.*

Por ejemplo, las figuras



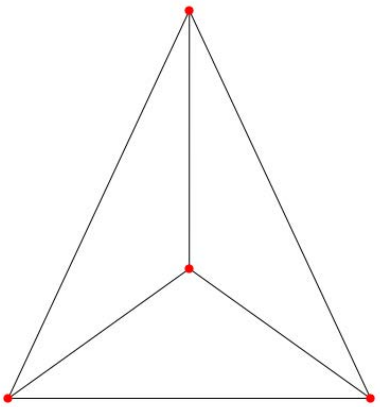
son dos dibujos distintos del mismo grafo; el de la izquierda presenta intersecciones de aristas mientras que el de la derecha no. Por lo tanto, el grafo es plano.

Ejemplo con Maxima 3.2.64 En `Maxima`, podemos analizar la planaridad de un grafo utilizanado el operador `is_planar`.

```
(%i1) load(graphs)$
(%i2) g9: create_graph(7, [[0,1],[0,5],[1,2],[1,4],
        [2,3], [0,3], [2,5],[3,6],[4,5],[4,6],[5,6]])$
      is_planar(g9);
(%o2)    false
```

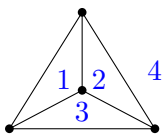
El algoritmo básico que utiliza `Maxima` para generar la representación gráfica de un grafo, no produce siempre una representación plana. Sin embargo, también permite elegir entre varios programas para determinar la posición final de los vértices en la representación gráfica; por ejemplo, `planar_embedding` fuerza la representación plana en la "mayoría" de las situaciones.

```
(%i3) draw_graph(complete_graph(4),redraw=true,
        program=planar_embedding);
(%o3)    done
```



□

Una representación plana de un grafo divide al plano en **regiones**, las partes del plano que quedan delimitadas por las aristas y los vértices, en donde la parte externa, no acotada, también se considera región. Por ejemplo, el grafo completo K_4, determina las cuatro regiones numeradas en la siguiente figura.



En un grafo plano y conexo, el número de vértices, el número de aristas y el número de regiones están relacionados por la fórmula de Euler.

TEOREMA 3.2.65 (FÓRMULA DE EULER) *Sea $G = (V, E)$ un grafo plano conexo con $|V| = v$, $|E| = e$, y sea r el número de regiones de una representación plana de G. Entonces, $v - e + r = 2$*

COROLARIO 3.2.66 *El grafo $K_{3,3}$ es un grafo no plano.*

Demostración: Si $K_{3,3}$ fuese plano, por la fórmula de Euler, su representación plana dividiría el plano en $9 - 6 + 2 = 5$ regiones; vamos a ver que esto es no posible. Por ser bipartito, el grafo $K_{3,3}$ no puede contener ciclos de longitud 3, ya que C_3 no es bipartito. Por lo tanto, los ciclos contenidos $K_{3,3}$ deben tener al menos 4 aristas. En consecuencia, cada región debe estar acotada por un ciclo de al menos 4 aristas. Dado que además cada arista es común a dos regiones, se debe verificar que $4r \leq 2e = 18$, es decir, $r \leq 4.5$. □

Observamos que en esta demostración solo hemos usado que el grafo $K_{3,3}$ no contiene ciclos de longitud tres, y que como mínimo los ciclos que contengan tendrán longitud 4. De ahí hemos deducido que todo grafo plano sin ciclos de lóngitud 3 debe verificar que $4r \leq 2e$ y, en consecuencia, por la fórmula de Euler:

$$2e \geq 4r = 4(e - v + 2) = 4e - 4v + 8 \qquad \Longrightarrow \qquad e \leq 2v - 4$$

Esta condición necesaria de planaridad se puede usar para deducir que un grafo no es plano, tal y como establece el siguiente resultado.

COROLARIO 3.2.67 *Si G es un grafo simple conexo con $v \geq 3$ vértices, e aristas, no contiene ciclos de longitud 3 y verifica que $e > 2v - 4$, entonces G no es plano.*

En general, si un grafo plano contiene ciclos de longitud 3, solo podríamos deducir que $2e \geq 3r$ y en consecuencia,

$$2e \geq 3r = 3(e - v + 2) = 3e - 3v + 6 \qquad \Longrightarrow \qquad e \leq 3v - 6$$

Obtenemos entonces una condición necesaria de planaridad más general. Esta condición, igual que la anterior, es necesaria, pero no suficiente. Por ejemplo, hemos visto que $K_{3,3}$ no es plano y, sin embargo, sí verifica la desigualdad anterior: $e = 9 \leq 3 \cdot v - 6 = 3 \cdot 6 - 6 = 12$. Por esta razón, esta condición se utiliza igualmente para deducir que un grafo no es plano.

COROLARIO 3.2.68 *Si G un grafo conexo con $v \geq 3$ vértices y e aristas y verifica que $e > 3v - 6$, entonces G no es plano.*

COROLARIO 3.2.69 *El grafo K_5 no es plano.*

Demostración: Para el grafo K_5, $3 \cdot v - 6 = 3 \cdot 5 - 6 = 9 < 10 = e$ y por lo tanto, no es plano. □

Finalmente, podemos obtener otra condición necesaria de planaridad a partir del teorema de Euler sobre los grados de los vertices en un grafo simple:

COROLARIO 3.2.70 *Todo grafo plano conexo tiene un vértice de grado menor o igual a 5.*

Demostración: Si todos los vértices tuvieran grado mayor que 5, entonces

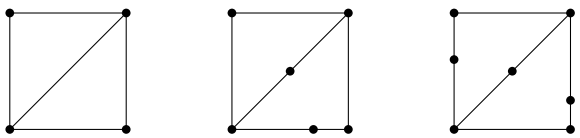
$$e = \frac{1}{2}\sum_{x\in V}\delta(x) \geq \frac{5}{2}\cdot v \geq 3v > 3v - 6,$$

y en consecuencia el grafo no sería plano.

Hemos destacado entre los resultados anteriores el hecho de que los grafos K_5 y $K_{3,3}$ no son planos. Esto se debe a que, estos grafos nos van a permitir caracterizar los grafos planos con el teorema de Kuratowski. Antes de enunciar este teorema, necesitamos introducir el concepto de *homeomorfismo* de grafos.

DEFINICIÓN 3.2.71 *Decimos que el grafo G_1 es **homeomorfo** a G_2 si uno de ellos se puede obtener a partir de el otro insertando vértices en alguna de sus aristas.*

Por ejemplo, los tres grafos de la figura son homeomorfos:

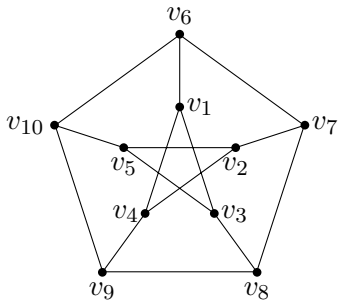


Es obvio que si añadimos vértices en medio de una arista, no cambiamos la condición de planaridad del grafo. El matemático polaco Kazimierz Kuratowski estableció en 1930 el siguiente teorema que caracteriza los grafos planos utilizando el concepto de homeomorfismo de grafos.

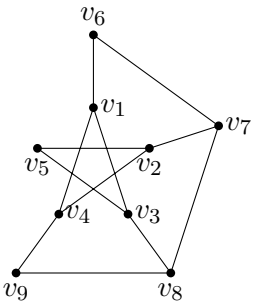
TEOREMA 3.2.72 (KURATOWSKI) *Un grafo es no plano si, y sólo si, contiene un subgrafo que es homeomorfo al grafo K_5 ó al grafo $K_{3,3}$.*

Es decir, podemos deducir que un grafo no es plano si podemos obtener el grafo K_5 o el grafo $K_{3,3}$ eliminando vértices (y las aristas que inciden en él), eliminando aristas o eliminando vértices de grado dos uniendo las aristas que inciden en él en una única arista.

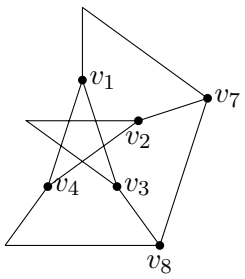
EJEMPLO 3.2.73 El siguiente grafo se conoce como **grafo de Petersen** y no es plano.



Vamos a construir un subgrafo homeomorfo a $K_{3,3}$ para demostrar que efectivamente no es plano. En primer lugar construimos un subgrafo eliminando el vértice v_{10} y todas las aristas que inciden en él.



Este grafo es homeomorfo a $K_{3,3}$, lo que se puede observar fácilmente si eliminamos los vértices v_5, v_6 y v_9, pero manteniendo las aristas $\{v_2, v_3\}$, $\{v_1, v_7\}$ y $\{v_4, v_8\}$:



□

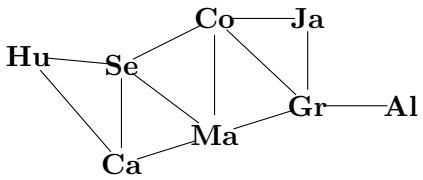
3.2.10. Coloración de Grafos

Colorear los vértices de un grafo consiste en asignar un color a cada vértice del grafo de manera que dos vértices adyacentes no tengan el mismo color.

DEFINICIÓN 3.2.74 *Sea $G = (V, E)$ un grafo simple y C un conjunto de m elementos (colores). Una* ***coloración*** *con m colores de los vértices del grafo G es una función $c\colon V \to C$ tal que si $u, v \in V$ y $\{u, v\} \in E$, entonces $c(u) \neq c(v)$.*

La denominación coloración viene del problema más representativo asociado a este concepto, la coloración de un mapa de tal forma que países o provincias con frontera común no estén coloreadas con el mismo color.

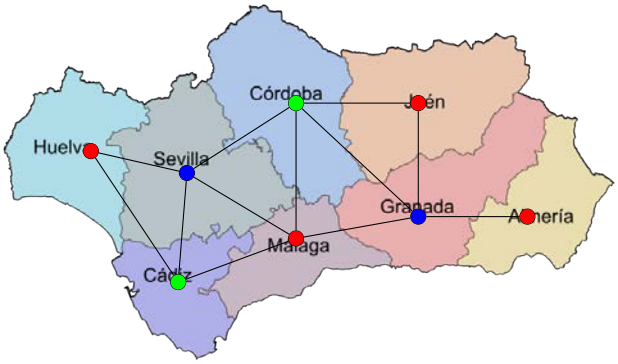
El grafo asociado a este mapa tiene por vértices a las provincias y las aristas unen las provincias con frontera común.



En el mapa de Andalucía hemos utilizado ocho colores diferentes, lo que naturalmente asegura una coloración, pero ¿cuál es el menor número de colores necesario? En 1976 se demostró el denominado **teorema de los 4 colores**, que se había conjeturado en 1852, y que establece que no necesitamos más de 4 colores.

Teorema 3.2.75 (Guthrie/ Appel/ Haken) *Todo grafo plano se puede colorear con cuatro colores o menos.*

Por ejemplo, para colorear el mapa de Andalucía son suficientes tres colores:



El menor número de colores necesario para colorear un grafo se denomina **número cromático** del grafo, y se denota $\chi(G)$. Por lo tanto, para determinar que el

número cromático de un grafo es m, es necesario en primer lugar encontrar una coloración con m colores y, en segundo lugar, demostrar que no es posible colorear con menos colores.

Ejemplo con Maxima 3.2.76 En `Maxima`, disponemos de operadores que nos determinan el número cromático de un grafo y calculan una coloración óptima.

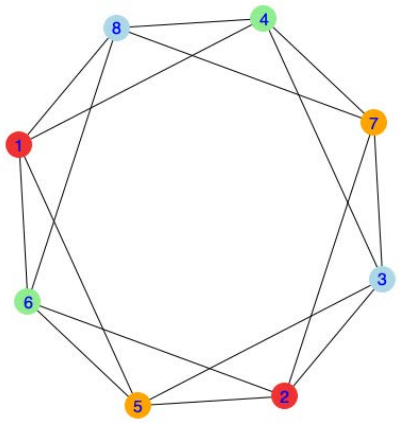
```
(%i1) load(graphs)$
(%i2) gcol:create_graph([1,2,3,4,5,6,7,8],[[3, 4],[4, 8],
          [2,5],[1,8],[5,6],[7,8],[4,7],[2,6],[1,4],[3,7],
          [2,7],[6,8],[2,3],[3,5],[1,6],[1,5]])$
(%i3) chromatic_number(gcol);
(%o4)  4
```

Por otra parte, `vertex_coloring`, también devuelve el número cromático pero incluyendo una coloración con ese número de colores. Los colores están representados por números naturales

```
(%i5) vertex_coloring(gcol);
(%o5)  [4,[[8,2],[7,4],[6,1],[5,4],[4,1],[3,2],[2,3],[1,3]]]
```

Es decir, los vértices 4 y6 están coloreados con el color 1, los vértices 3 y 8 con el color 2, los vértices 1 y 2 con el color 3 y los vértices 5 y7 están con el color 4. Podemos visualizar el grafo con diferentes colores usando la opción `vertex_partition`.

```
(%i6) draw_graph(gcol, vertex_size=4,show_id=true,
            vertex_partition=[[1,2],[3,8],[4,6],[5,7]]);
(%o72)  done
```



□

No existe ningún algoritmo que de forma eficiente determine el número cromático de un grafo y la coloración con el menor número de colores. El siguiente algoritmo

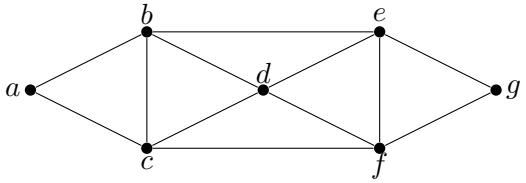
permite obtener fácilmente una coloración de un grafo, pero no garantiza que se use el menor número de colores.

- Empezamos ordenando los vértices según el orden decreciente de sus grados,

$$\delta(v_1) \geq \delta(v_2) \geq \ldots \delta(v_{n-1}) \geq \delta(v_n)$$

- Asignamos el primer color al vértice v_1, es decir: $c(v_1) = 1$ y, siguiendo la secuencia, a los vértices que no sean adyacentes a él y a los que coloreemos con el mismo color.
- Asignamos el segundo color al primer vértice de la secuencia al que no se le haya asignado el primer color, y siguiendo la secuencia a los vértices que no sean adyacentes a él y a los que coloreemos con el mismo color.
- Siguiendo la lista ordenada de vértices, repetimos el proceso hasta colorear todos los vértices.

Ejemplo 3.2.77 Consideremos el grafo



Una ordenación de los vértices según su grado sería

Grados	4	4	4	4	4	2	2
Vértices	b	c	d	e	f	a	g

Asignamos el color 1 al vértice b, después al vértice f. El resto de vértices son adyacentes a b o a f.

Grados	4	4	4	4	4	2	2
Vértices	b	c	d	e	f	a	g
Color	1				1		

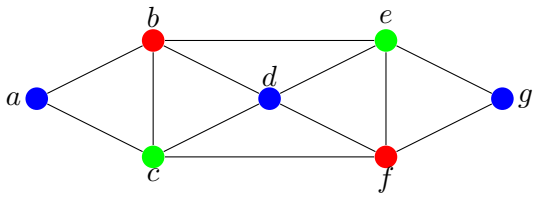
Asignamos el color 2 al vértice c y después al vértice e. El resto de vértices sin colorear son adyacentes a c o a e.

Grados	4	4	4	4	4	2	2
Vértices	b	c	d	e	f	a	g
Color	1	2		2	1		

Finalmente, asignamos el color 3 a los vértices d, a y g.

Grados	4	4	4	4	4	2	2
Vértices	b	c	d	e	f	a	g
Color	1	2	3	2	1	3	3

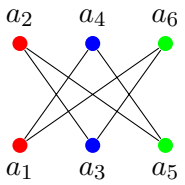
Mostramos el grafo identificando el color 1 con el rojo, el color 2 con el verde y el color 3 con el azul.



Por lo tanto, hemos coloreado el grafo con tres colores, pero todavía no podemos concluir que su número cromático sea 3. Necesitamos probar que no es posible colorearlo con solamente dos colores. Sin embargo esto es bastante inmediato, ya que el grafo contiene ciclos de longitud 3, que evidentemente no se pueden colorear con menos de tres colores. □

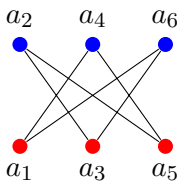
Debemos insistir en que el algoritmo mostrado en el ejemplo anterior no produce necesariamente una coloración óptima, con el menor número de colores posible. Dependiendo de la ordenación inicial de los vértices, el número de colores necesarios puede ser diferente.

Ejemplo 3.2.78 Los vértices del grafo siguiente tienen grado 2, por lo que podemos aplicar el algoritmo de coloración a partir de cualquier ordenación de sus vértices. Si utilizamos la ordenación dada por los subíndices utilizados en las etiquetas, necesitamos tres colores:



a_1	a_2	a_3	a_4	a_5	a_6
2	2	2	2	2	2
R	R	A	A	V	V

Sin embargo, la ordenación $[a_1, a_3, a_5, a_2, a_4, a_6]$ sí permite construir una coloración con dos colores, que es evidentemente su número cromático.



a_1	a_3	a_5	a_2	a_4	a_6
2	2	2	2	2	2
R	R	R	A	A	A

□

El algoritmo, por tanto, nos dará una cota superior del número cromático, pero necesitaremos analizar si es o no posible realizar la coloración con menos colores. Para este trabajo, utilizaremos el número cromático conocido de otros grafos, como los grafos bipartitos, grafos completos o ciclos.

EJEMPLO 3.2.79 El número cromático del grafo K_n es n, ya que no es posible colorear dos vértices con un mismo color, puesto que todos los pares de vértices están conectados al ser completo. □

PROPOSICIÓN 3.2.80 *Si un grafo contiene un subgrafo isomorfo a K_n, entonces su número cromático es mayor o igual que n.*

EJEMPLO 3.2.81 El número cromático de un grafo bipartito es dos. Es más, un grafo es bipartito si y solo si su número cromático es 2. Estas afirmaciones son evidentes, si el grafo es bipartito, asignamos un color a los vértices de una de las partes y el segundo color a los vértices de la otra parte.

De la misma forma, si un grafo se puede colorear con solo dos colores, los conjuntos de vértices coloreados con el mismo color determinan las dos partes del grafo bipartito. □

EJEMPLO 3.2.82 Si n es par, el grafo C_n es bipartito y, en consecuencia, su número cromático es 2. Sin embargo, si n es impar, necesitamos un tercer color, es decir, su número cromático es 3. □

Aplicar y utilizar conceptos y algoritmos de la teoría de grafos requiere modelizar un problema de manera adecuada usando los grafos. Esa parte de la resolución de un problema puede ser compleja y requiere entender que la conexión que definen las aristas pueden corresponder con aspectos muy diversos.

Un problema típico en el que se usa la coloración de grafos es la creación de un calendario de exámenes. En este problema, se busca que en el mismo día no coincidan dos exámenes si un mismo alumno tiene que realizar esos dos exámenes. Para plantear este problema con teoría de grafos, consideramos que las asignaturas son los vértices de un grafo y que las aristas conectan dos asignaturas que comparten estudiantes. De esta forma, una coloración óptima del grafo nos diría el menor número de días o franjas necesarias para programar todos los exámenes de forma que todos los alumnos puedan presentarse a sus asignaturas.

EJEMPLO 3.2.83 En un laboratorio hay una serie de compuestos químicos, a, b, c, d, e, f, g, h que transportar a otro laboratorio. Por cuestiones de seguridad, no se pueden mover juntos dos compuestos que puedan reaccionar si hay un accidente. Las reacciones peligrosas vienen descritas por las adyacencias definidas por la siguiente

tabla.

a	b	c	d	e	f	g	h
b	a	a	b	b	c	d	e
c	d	e	e	c	h	e	f
	e	f	g	d		h	g
				g			
				h			

Una coloración del grafo descrito por esta tabla, los grupos de productos que podemos trasladar en un mismo viaje y la coloración optima nos daría el menor número de viajes necesario. Vamos a construir una primera coloración. El primer color (en este caso primer traslado) se lo asignaríamos a los compuestos e y a.

Grados	5	3	3	3	3	3	2	2
Vértices	e	b	c	d	g	h	a	f
Color	1						1	1

El segundo color se lo podríamos asignar a b, c y g.

Grados	5	3	3	3	3	3	2	2
Vértices	e	b	c	d	g	h	a	f
Color	1	2	2		2		1	1

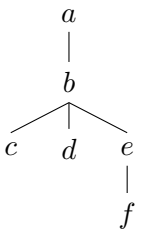
Finalmente, necesitaríamos un tercer color para d y h.

Grados	5	3	3	3	3	3	2	2
Vértices	e	b	c	d	g	h	a	f
Color	1	2	2	3	2	3	1	1

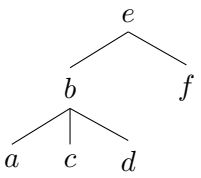
También podemos concluir que no es posible hacer menos viajes para trasladar todos los compuestos, ya que el camino $C = ebde$ es un subgrafo isomorfo a K_3, cuyo número cromático es 3, según hemos visto en el ejemplo 3.2.79. □

3.2.11. Árboles con raíz ordenados

En la construcción de los árboles de búsqueda de las secciones anteriores hemos partido de un vértice, que en principio se puede elegir arbitrariamente, pero que por lo general tendrá un significado destacado en la aplicación. Este nodo destacado recibe el nombre de **raíz**. Habitualmente, los árboles se dibujan tal y como hemos hecho en los ejemplos de las secciones anteriores, orientando hacía abajo los caminos que conectan la raíz con el resto de vértices.



Árbol con raíz a



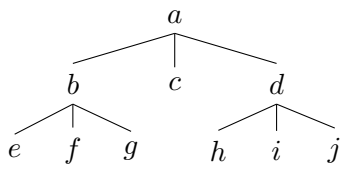
Árbol con raíz e

De esta forma, la disposición de los vértices determina un *orden* entre los vértices, y por ello, los árboles con raíz pueden ser considerados como grafos dirigidos. Además, ese orden establecido por la posición relativa de los vértices introduce los siguientes conceptos.

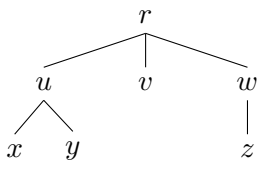
Si T es un árbol con raíz y v un vértice distinto de su raíz: el **padre** de v es el único vértice u tal que hay una arista desde u hasta v en el camino que une v con la raíz; decimos igualmente que v es **hijo** de u. Los vértices con el mismo padre se llaman **hermanos**. Los **antecesores** de un vértice v son todos los vértices del único camino desde la raíz hasta v. Los **descendientes** de un vértice v son aquellos vértices para los que v es un antecesor. Un vértice se dice que es una **hoja** si no tiene hijos. Los **vértices internos** son los vértices que tienen hijos; la raíz se considera vértice interno, a menos que sea el único vértice del grafo, en cuyo caso, se considera hoja. Si v es un vértice de T, el **subárbol** con raíz en x es el subgrafo que contiene al v, a todos sus descendientes y a todas las aristas incidentes en dichos descendientes.

El **nivel** de un vértice en un árbol con raíz es la longitud del único camino desde la raíz hasta dicho vértice; se considera que el nivel de la raíz es cero. La **altura** de un árbol con raíz es el máximo de los niveles de sus vértices. Un árbol con raíz m-ario de altura h se dice que está **equilibrado** si todas sus hojas están en los niveles h o $h-1$.

Un árbol con raíz se llama **árbol m-ario** si todos los vértices internos tienen, a lo sumo, m hijos; en particular, se dirá **binario** si cada vértice interno tiene a los sumo 2 hijos, **ternario** si cada vértice interno tiene a los sumo tres hijos. El árbol se llama **m-ario completo** si cada vértice interno tiene exactamente m hijos. Los árboles m-arios completos se usan habitualmente en problemas de búsqueda, ordenación y codificación.



Árbol ternario **completo**



Árbol ternario **no completo**

TEOREMA 3.2.84 *Un árbol de n vértices tiene $n-1$ aristas.*

TEOREMA 3.2.85 *Un árbol m-ario completo con i vértices internos tiene $n = i \cdot m + 1$ vértices.*

TEOREMA 3.2.86 *Un árbol m-ario de altura h tiene, a lo sumo, m^h hojas.*

Estos resultados permiten abordar problemas en los que necesitemos contar vértices, aristas, hojas,... en estructuras físicas o virtuales dispuestas en forma de árbol.

EJEMPLO 3.2.87 Si sabemos que un árbol ternario es completo y tiene 34 vértices internos, entonces tiene $34 \cdot 3 + 1 = 103$ vértices en total; y por lo tanto, tiene $103 - 1 = 102$ aristas. El número de hojas será el número total de vértices menos los vértices internos, es decir, $103 - 34 = 69$. □

EJEMPLO 3.2.88 Si sabemos que un árbol completo de aridad 5 tiene 817 hojas podemos saber cuántos vértices internos tiene. Si n es el número total de vértices y i es el número de vértices internos, entonces sabemos que $n = i + 817$ y que $n = 5i + 1$; eliminando la variable n, podemos despejar el valor de i:

$$i + 817 = 5i + 1 \qquad \Longrightarrow \qquad 816 = 4i \qquad \Longrightarrow \qquad i = \frac{816}{4} = 204$$ □

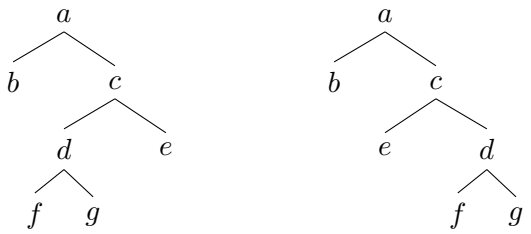
EJEMPLO 3.2.89 En un aula necesitamos conectar 25 ordenadores a un único enchufe de pared. Para ello, disponemos de cables de extensión con con cuatro salidas cada uno. ¿Cuál es el número mínimo de estos cables que necesitamos?

La disposición de los cables, sea cual sea la configuración, tendrá la forma de un árbol con raíz (el aula) que será de aridad 4 y será completo. Dado que tenemos que conectar 25 ordenadores, necesitaremos que el árbol formado tenga 25 hojas, ya que los vértices internos se utilizan para conectar los propios cables. Además, en este caso, el número de cables coincidirá con el número de vértices internos de la configuración. Repetimos el desarrollo que hemos hecho en el ejemplo anterior para calcular este número de vértices internos:

$$i + 25 = 4i + 1 \qquad \Longrightarrow \qquad 24 = 3i \qquad \Longrightarrow \qquad i = \frac{24}{3} = 8$$ □

Árboles con raíz ordenados Un **árbol con raíz ordenado** es un árbol con raíz en el que los hijos de cada vértice interno están ordenados. Este orden se refleja en su representación, disponiéndolos de izquierda a derecha según este orden.

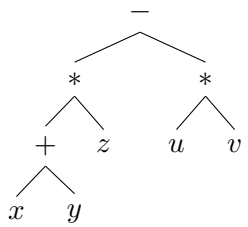
Por ejemplo, los siguientes árboles son iguales (isomorfos) si los consideramos como grafos, pero no son iguales si los consideramos con árboles con raíz ordenados, ya que la hoja e y el subárbol con raíz d están dispuestos en distinto orden.



Los árboles con raíz ordenados se usan para representar expresiones algebraicas, enunciados, expresiones gramáticales,... En estos casos, las hojas del árbol se etiquetan con variables o constantes de algún dominio numérico o semántico, y los vértices interiores se etiquetan con operadores definidos en ese dominio. Es fundamental considerar el orden en un árbol cuando consideramos operaciones que no son conmutativas (como la diferencia o la división entre números), pero también si queremos forzar un orden en la evaluación.

En un árbol ordenado, los hijos de los vértices binarios se denominan **hijo izquierdo**, el primero, e **hijo derecho**, el segundo; además, los correspondiente subárboles se denominan **subárbol izquierdo** y **subárbol derecho** respectivamente.

Ejemplo 3.2.90 La expresión algebraica $\Big(\big((x+y)*z\big)-\big(u*v\big)\Big)$ se representa con el siguiente árbol ordenado:



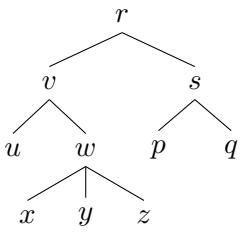
□

Una operación importante cuando trabajamos con árboles ordenados es el recorrido de sus nodos, ya sea para la evaluación en el caso de representar expresiones algebraicas o para la búsqueda de información representada con este tipo de árboles.

Recorrido en orden previo. Este recorrido corresponde a la búsqueda en anchura, que ya hemos aprendido anteriormente, empezando desde la raíz.

- Si T solo consta de la raíz, entonces r es el *recorrido en orden previo* de T.
- En otro caso, si $v_1, v_2, ..., v_k$ son los hijos de r (leídos de izquierda a derecha) en T y $T_1, T_2, ..., T_k$ los subárboles correspondientes, el recorrido en orden previo empieza por visitar la raíz r continúa recorriendo T_1 en orden previo, después T_2 en orden previo y así sucesivamente hasta que T_k.

Ejemplo 3.2.91 El recorrido en orden previo dará la secuencia de vértices en el orden dado por ese recorrido. Vamos a obtener esta secuencia para el siguiente árbol.



Representamos por T_α el subárbol con raíz en α que está pendiente de recorrido.

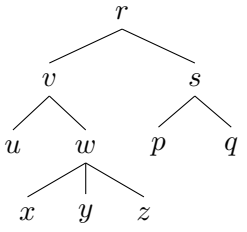
$$r - T_v - T_s$$
$$r - v - u - T_w - T_s$$
$$r - v - u - w - x - y - z - T_s$$
$$r - v - u - w - x - y - z - s - p - q$$

Aunque mostramos tres secuencias previas a la que definitivamente da el recorrido, esta última secuencia debe obtenerse directamente. □

Recorridos en orden posterior. Este recorrido corresponde a la búsqueda en profundidad, pero recorriendo el árbol desde las hojas a la raíz.

- Si T solo consiste en la raíz, entonces r es el *recorrido en orden posterior* de T.
- En otro caso, si $v_1, v_2, ..., v_k$ son los hijos de r (leídos de izquierda a derecha) en T y $T_1, T_2, ..., T_k$ los subárboles correspondientes, entonces el recorrido en orden posterior empieza por recorrer T_1 en orden posterior, después T_2 en orden posterior, así sucesivamente hasta que T_k, y termina visitando la raíz r.

Ejemplo 3.2.92 El recorrido en orden posterior dará la secuencia de vértices en el orden dado por ese recorrido. Vamos a obtener esta secuencia para el siguiente árbol.



Representamos por T_α el subárbol con raíz en α que está pendiente de recorrido.

$$T_v - T_s - r$$
$$u - T_w - v - T_s - r$$
$$u - x - y - z - w - v - T_s - r$$
$$u - x - y - z - w - v - p - q - s - r$$

Aunque mostramos tres secuencias previas a la que definitivamente da el recorrido, esta última secuencia debe obtenerse directamente. □

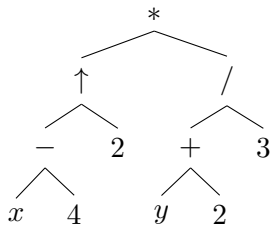
Se pueden realizar otros recorridos en un árbol dependiendo de la aplicación con la que estemos trabajando. Por ejemplo, en los árboles binarios se puede usar el recorrido en *orden simétrico* y en general también podemos utilizar las búsquedas en profundidad para determinar los recorridos, ya sea de la raíz a las hojas o de las hojas a la raíz.

Una de las aplicaciones de los recorridos que acabamos de introducir es obtener la representación de expresiones algebraicas, y expresiones sintácticas en general, sin necesidad de usar paréntesis y situando los operadores ya sea de forma prefija o de forma postfija o sufija.

Ejemplo 3.2.93 La expresión algebraica $(x-4)^2\Big(\dfrac{y+2}{3}\Big)$ se escribe de la siguiente forma usando operadores binarios y paréntesis

$$\big((x-4)\uparrow 2\big) * \big((y+2)/3\big)$$

y se representa por el siguiente árbol con raíz ordenado



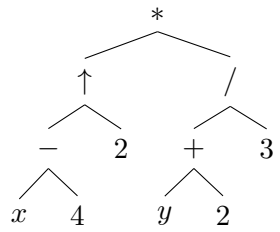
El recorrido en orden previo está dado por la secuencia

$$* \uparrow - x\, 4\, 2\, / + \, y\, 2\, 3$$

que determina la fórmula sin ambigüedades si consideramos que los operadores están escritos de forma prefija. □

Ejemplo 3.2.94 La **Notación polaca** recibe su nombre de la escuela polaca de lógicos, que escribían los operadores lógicos de forma postfija, es decir, primero los argumentos y luego el operador. En términos de recorridos de árbol, esto corresponde a escribir el recorrido del árbol sintáctico en orden posterior.

Volvamos a considerar la expresión algebraica $\big((x-4)\uparrow 2\big) * \big((y+2)/3\big)$ representada por el siguiente árbol con raíz ordenado



El recorrido en orden posterior está dado por la secuencia

$$x\,4 - 2 \uparrow y2 + 3/*$$

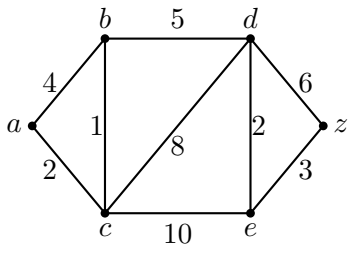
que determina la fórmula sin ambigüedades si consideramos que los operadores están escritos de forma postfija o sufija. □

3.2.12. Grafos Ponderados

DEFINICIÓN 3.2.95 *Un **grafo (simple) ponderado** es una terna $G = (V, E, w)$ tal que $G = (V, E)$ un grafo simple y $w\colon E \to \mathbb{R}^+$ es una aplicación, que se denomina función **peso**.*

Los grafos ponderados permiten representar muchos problemas (redes de transporte en donde el peso representa la distancia, redes de comunicación en donde el peso representa el coste,...) Se pueden considerar grafos ponderados más generales, por ejemplo considerando pesos sobre un grafo dirigido o sobre un multigrafo o también considerando pesos negativos. En este curso, solo vamos a trabajar con grafos simples ponderados.

Cuando representamos gráficamente un grafo ponderado, etiquetamos cada arista con el peso asignado por la aplicación w.



(3.1)

También podemos describir una grafo ponderado utilizando su **matriz de pesos**: a partir de una ordenación de los vértices, el elemento (i, j) de la matriz es el peso de la arista que une el i-ésimo vértices con el j-ésimo vértice, y es un 'guión', $-$,

si los vértices no están conectados. El grafo de arriba se describe por la siguiente matriz de pesos, tomando el orden lexicográfico como el orden entre los vértices:

$$W_G = \begin{pmatrix} - & 4 & 2 & - & - & - \\ 4 & - & 1 & 5 & - & - \\ 2 & 1 & - & 8 & 10 & - \\ - & 5 & 8 & - & 2 & 6 \\ - & - & 10 & 2 & - & 3 \\ - & - & - & 6 & 3 & - \end{pmatrix} \tag{3.2}$$

La **longitud** de un camino en un grafo ponderado es la suma de los pesos de las aristas. Por ejemplo, la longitud del camino $a - b - d - c - e - z$ en el grafo (3.1) es 30.

Un **árbol generador minimal** de un grafo conexo ponderado es un árbol generador tal que la suma de los pesos de sus aristas es mínima, respecto de todos los árboles generadores. Los algoritmos más usados para determinar el árbol generador minimal de un grafo ponderado son el **Algoritmo de Kruskal** y el **Algoritmo de Prim**; en este curso, vamos a ver solamente este último.

Algoritmo de Prim. Sea $G = (V, E, w)$ un grafo conexo ponderado con n vértices. En el algoritmo, vamos a hallar una secuencia de árboles $A_i = (V_i, E_i, w)$, cada uno de ellos subgrafo de G y de tal forma que A_n es el árbol generador minimal.

- Para $i = 1$, tomamos $V_1 = \{v\}$, siendo v un vértice cualquiera, y $E_1 = \varnothing$.
- Supongamos que hemos construido V_{i-1} y E_{i-1}. Sea

  $$w_i = \text{mín}\{w(\{x, y\}); x \in V_{i-1}, y \in V - V_{i-1}\}$$

  y $e_i = \{x_i, y_i\}$ una arista en la que se alcanza ese mínimo, es decir, $x_i \in V_{i-1}$, $y_i \in V - V_{i-1}$, $w(\{x_i, y_i\}) = w_i$.

  Tomamos $V_i = V_{i-1} \cup \{y_i\}$, $E_i = E_{i-1} \cup \{e_i\}$.

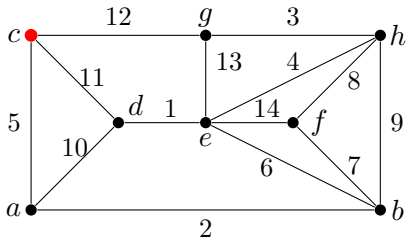
Ejemplo 3.2.96 El estudio de localización de terminales de ordenadores que van a ser instalados en una empresa viene dado por la siguiente tabla, donde los números representan el coste de instalar las conexiones entre los distintos terminales.

	a	b	c	d	e	f	g	h
a	–	2	5	10	–	–	–	–
b	2	–	–	–	6	7	–	9
c	5	–	–	11	–	–	12	–
d	10	–	11	–	1	–	–	–
e	–	6	–	1	–	14	13	4
f	–	7	–	–	14	–	–	8
g	–	–	12	–	13	–	–	3
h	–	9	–	–	4	8	3	–

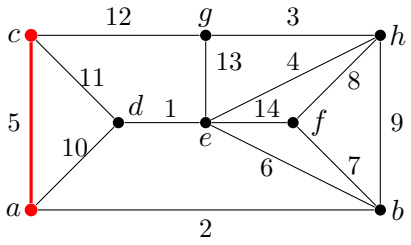
El terminal c corresponde al ordenador principal y el resto de los terminales deben estar conectados a él mediante líneas telefónicas. ¿Cuáles son las conexiones que debemos hacer para que todos los terminales estén conectados a través de c y la inversión realizada sea mínima?

La que necesitamos es obtener el árbol generador minimal, con raíz c, del grafo ponderado determinado por las conexiones entre terminales con su coste.

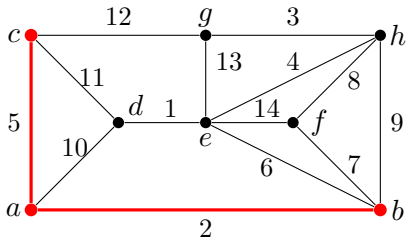
$V_1 = \{c\}$
$E_1 = \varnothing$



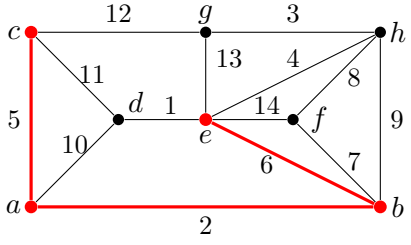
$w_2 = \text{mín}\{5, 11, 12\} = 5 = w(\{c, a\})$
$V_2 = \{c, a\}$
$E_2 = \{\{c, a\}\}$



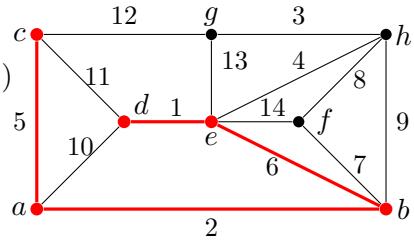
$w_3 = \text{mín}\{2, 10, 11, 12\} = 2 = w(\{a, b\})$
$V_3 = \{c, a, b\}$
$E_3 = E_2 \cup \{\{a, b\}\}$



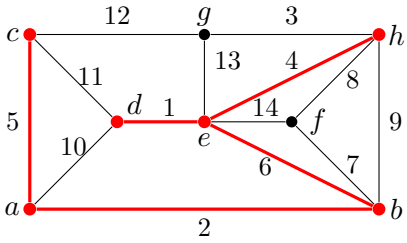
$w_4 = \text{mín}\{6, 7, 9, 10, 11, 12\} = 6 = w(\{b, e\})$
$V_4 = \{c, a, b, e\}$
$E_4 = E_3 \cup \{\{b, e\}\}$



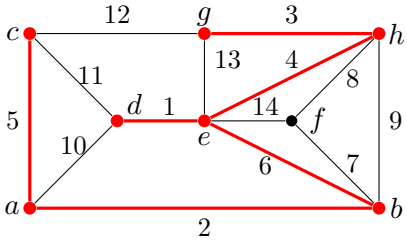
$w_5 = \text{mín}\{1, 4, 7, 9, 10, 11, 12, 13, 14\} = 1 = w(\{e, d\})$
$V_5 = \{c, a, b, e, d\}$
$E_5 = E_4 \cup \{\{e, d\}\}$



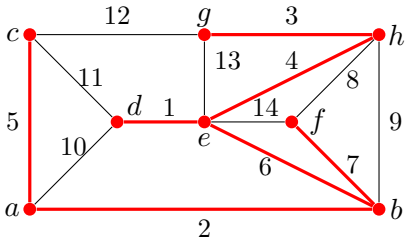
$w_6 = \text{mín}\{4, 7, 9, 12, 13, 14\} = 4 = w(\{e, h\})$
$V_6 = \{c, a, b, e, d, h\}$
$E_6 = E_5 \cup \{\{e, h\}\}$



$w_7 = \text{mín}\{3, 7, 8, 9, 12, 13, 14\} = 3 = w(\{h, g\})$
$V_7 = \{c, a, b, e, d, h, g\}$
$E_7 = E_6 \cup \{\{h, g\}\}$



$w_8 = \text{mín}\{7, 8, 14\} = 7 = w(\{b, f\})$
$V_8 = \{c, a, b, e, d, h, g, f\}$
$E_8 = E_7 \cup \{\{b, f\}\}$



El subgrafo resaltado en rojo en el último dibujo es el árbol generador minimal. □

Ejemplo con Maxima 3.2.97 También disponemos de operadores para trabajar con grafos ponderados. Para definirlos, usamos también el operado `create_graph`, pero en este caso, las aristas se definirán con una lista de dos elementos, siendo el primero de ellos la arista propiamente dicha y el segundo el peso de la misma:

```
(%i1) load(graphs)$
(%i2) grp:create_graph([1,2,3,4,5,6],[
          [[1,2],2],
          [[1,3],3],
```

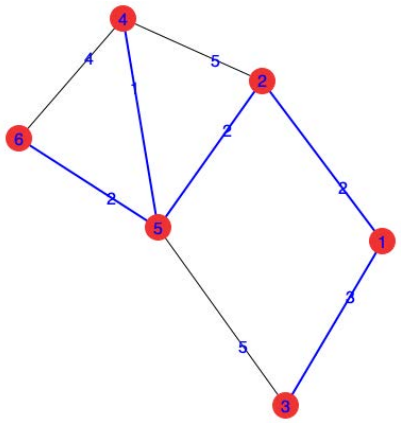
```
[[2,4],5],
[[2,5],2],
[[3,5],5],
[[4,5],1],
[[4,6],4],
[[5,6],2]
]);
```

Podemos calcular árboles generadores minimales usando el operador `minimun_spaning_tree`.

```
(%i3) mstgrp: minimum_spanning_tree(grp);
```

Vamos a utilizar las opciones `show_edges` y `show_weight` para visualizar el grafo con los pesos de cada arista y destacando el subárbol generador minimal.

```
(%i4) draw_graph(grp, vertex_size=4,show_id=true,
                 show_weight=true,
                 show_edges=edges(mstgrp));
(%o88)  done
```



□

Camino de longitud mínima: Algoritmo de Dijkstra. Nos planteamos ahora el problema de encontrar el camino de longitud mínima entre dos vértices de un grafo ponderado, y para ello vamos a utilizar el Algoritmo de Dijkstra.

Dado un grafo ponderado $G = (V, E, w)$ y un vértice $v_0 \in V$, el algoritmo de Dijkstra es un proceso iterativo tal que, en cada iteración, va a determinar el camino más corto de v_0 a un segundo vértice.

En cada paso, partimos del conjunto de vértices S_i, para los cuales hemos encontrado el camino más corto desde v_0 en algún paso anterior al i-ésimo. También

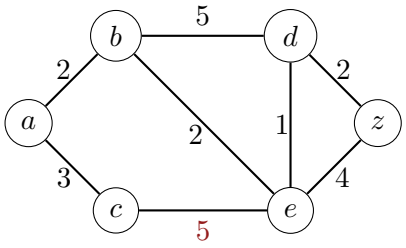
determinamos la función ℓ_i, tal que $\ell_i(v)$ es la longitud del camino más corto desde v_0 hasta v, pasando exclusivamente por vértices de S_i; escribiremos $\ell_i(v) = -$ si $v \in S_i$ y $\ell_i(v) = \infty$ si v no es adyacente a ningún vértice de S_i.

- $S_1 = \{v_0\}$;
  $\ell_1(v_0) = -$, $\ell_1(v) = w(\{v_0, v\})$ si $\{v_0, v\} \in E$, $\ell_1(v) = \infty$ en otro caso.
  Si $\ell_1(v_1) = \text{mín}\{\ell_1(v); v \in V\}$, entonces $S_2 = \{v_0, v_1\}$ y la arista $\{v_0, v_1\}$ es el camino más corto de v_0 a v_1.
- Supongamos que hemos determinado $S_i = S_{i-1} \cup \{v_{i-1}\}$ y ℓ_{i-1}.

$$\ell_i(v) = \begin{cases} \min\{\ell_{i-1}(v), \ell_{i-1}(v_{i-1}) + w(\{v_{i-1}, v\})\} & \text{Si } \{v, v_{i-1}\} \in E \\ \ell_{i-1}(v) & \text{Si } \{v, v_{i-1}\} \notin E \end{cases}$$

  Es decir, después de añadir el vértice v_{i-1} volvemos a calcular las distancias hasta cada vértice para ver si, utilizando el camino más corto hasta v_{i-1}, conseguimos un camino más corto a los otros vértices.
  Tomamos v_i tal que $\ell_i(v_i) = \text{mín}\{\ell_i(v); v \in V\}$; este número es la longitud del camino más corto de v a v_i.
- Continuamos hasta conseguir $S_n = V$ o hasta conseguir el camino más corto hasta el vértice deseado.

Ejemplo 3.2.98 Para el grafo



vamos a hallar el camino más corto desde el vértice a hasta cada uno de los otros vértices. Para seguir el desarrollo del algoritmo, vamos a construir una tabla, en cuya primera columna vamos a escribir los elementos de los conjuntos S_i, destacando el último vértice añadido. El resto de las columnas corresponde a cada vértice del grafo, de tal forma que las filas estarán ocupadas con el correspondiente valor de la función ℓ_i.

	Vértices	a	b	c	d	e	z	
1	$\{a\}$	$-$	$\boxed{2}(a)$	$3(a)$	∞	∞	∞	$a-b$

Los vértices b y c son los únicos adyacentes a a, y por eso son los únicos que aparecen en la primera fila con un número, el peso de la correspondiente arista.

Además, hemos indicado entre paréntesis el vértice anterior en el camino que permite obtener esta longitud; naturalmente, en este primer paso es a. Marcamos el menor valor en la fila, que corresponde a la columna de b, es decir, el camino más corto desde a hasta b es la aristas $a-b$.

	Vértices	a	b	c	d	e	z	
1	$\{a\}$	–	$\boxed{2}(a)$	$3(a)$	∞	∞	∞	$a-b$
2	$\{a\}\cup\{b\}$	–	–	$\boxed{3}(a)$	$7(b)$	$4(b)$	∞	$a-c$

Dado que $a, b \in S_2$, escribimos un guión en las casillas de estos vértices en la segunda fila. Dado que c y z no son adyacentes a b, para estos vértices, copiamos las casillas de la fila superior. Dado que la arista $\{b, d\}$ tiene peso 5, el camino $a-b-d$ tiene peso 7 y por eso escribimos $7(b)$ en la casilla correspondiente a d en la segunda fila. Dado que la arista $\{b, e\}$ tiene peso 2, el camino $a-b-e$ tiene peso 4 y por eso escribimos $4(b)$ en la casilla correspondiente a e en la segunda fila. Por lo tanto, el menor valor en la fila es 3 y nos indica que el camino $a-c$ es el más corto de los que unen a y c.

	Vértices	a	b	c	d	e	z	
1	$\{a\}$	–	$\boxed{2}(a)$	$3(a)$	∞	∞	∞	$a-b$
2	$\{a\}\cup\{b\}$	–	–	$\boxed{3}(a)$	$7(b)$	$4(b)$	∞	$a-c$
3	$\{a,b\}\cup\{c\}$	–	–	–	$7(b)$	$\boxed{4}(b)$	∞	$a-b-e$

Dado que $a, b, c \in S_3$, escribimos un guión en las casillas de estos vértices en la tercera fila. Dado que d y z no son adyacentes a c, copiamos las casillas de la fila superior en las columnas correspondientes a estos vértices. Dado que la arista $\{c, e\}$ tiene peso 5, el camino $a-c-e$ tiene peso 8, que es mayor que el dado por la casilla superior, por eso volvemos a escribir $4(b)$ en la casilla correspondiente a e en la tercera fila.

El valor menor en la tercera fila es $4(b)$, en la columna de e. Esto indica que 4 es la longitud del camino más corto que une a y e; además, este camino viene de b, es decir, $a-b-e$ es el camino más corto desde a hasta e.

	Vértices	a	b	c	d	e	z	
1	$\{a\}$	–	$\boxed{2}(a)$	$3(a)$	∞	∞	∞	$a-b$
2	$\{a\}\cup\{b\}$	–	–	$\boxed{3}(a)$	$7(b)$	$4(b)$	∞	$a-c$
3	$\{a,b\}\cup\{c\}$	–	–	–	$7(b)$	$\boxed{4}(b)$	∞	$a-b-e$
4	$\{a,b,c\}\cup\{e\}$	–	–	–	$\boxed{5}(e)$	–	$8(e)$	$a-b-e-d$

Dado que $a, b, c, e \in S_4$, escribimos un guión en las casillas de estos vértices en la segunda fila. Dado que la arista $\{e, d\}$ tiene peso 1, la longitud del camino $a-b-e-d$

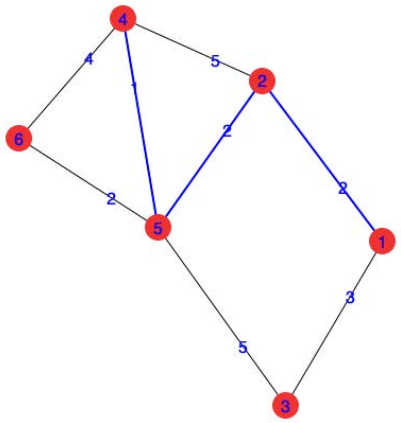
es 5, que es menor que el camino indicado en la fila superior, por eso escribimos $5(e)$ en la casilla correspondiente a d en la cuarta fila. Dado que la arista $\{e, z\}$ tiene peso 4, la longitud del camino $a - b - e - z$ es 8, y por eso escribimos $8(e)$ en la casilla correspondiente a z en la cuarta fila. El menor número en la cuarta fila es 5, en la columna de d; por lo tanto, 5 es la longitud del camino más corto de a hasta d; este camino viene del vértice e y por lo tanto $a - b - e - d$ es el camino más corto desde a hasta d.

	Vértices	a	b	c	d	e	z	
1	$\{a\}$	–	$\boxed{2}(a)$	$3(a)$	∞	∞	∞	$a - b$
2	$\{a\} \cup \{b\}$	–	–	$\boxed{3}(a)$	$7(b)$	$4(b)$	∞	$a - c$
3	$\{a, b\} \cup \{c\}$	–	–	–	$7(b)$	$\boxed{4}(b)$	∞	$a - b - e$
4	$\{a, b, c\} \cup \{e\}$	–	–	–	$\boxed{5}(e)$	–	$8(e)$	$a - b - e - d$
5	$\{a, b, c, e\} \cup \{d\}$	–	–	–	–	–	$\boxed{7}(d)$	$a - b - e - d - z$

En este último paso, solo tenemos que calcular la longitud del camino hasta z sumando 5 al peso de la arista $\{d, z\}$, es decir, $5 + 2 = 7$, que es menor que la indicada en la casilla superior. Por lo tanto, el camino $a - b - e - d - z$ es el más corto desde a hasta z. □

Ejemplo con Maxima 3.2.99 El algoritmo de Dijkstra está implementado en `Maxima` en el operador `shortest_weighted_path`:

```
(%i1) load(graphs)$
(%i2) grp:create_graph([1,2,3,4,5,6],[
          [[1,2],2],
          [[1,3],3],
          [[2,4],5],
          [[2,5],2],
          [[3,5],5],
          [[4,5],1],
          [[4,6],4],
          [[5,6],2]
          ])$
(%i3) dijkgrp: shortest_weighted_path(1,4,grp);
(%o3) [5,[1,2,5,4]]
(%i4) draw_graph(grp,vertex_size=4,show_id=true,
                 show_weight=true,
                 show_edges=vertices_to_path(dijkgrp[2]));
(%o86)  done
```


4
5
2
4
1
6
2
2
2
5
1
3
5
3

□