Árboles

Joaquim Borges
Robert Clarisó
Ramon Masià
Jaume Pujol
Josep Rifà
Joan Vancells
Mercè Villanueva

PID_00174688

UOC
Universitat Oberta de Catalunya

Índice

Introducción

Los árboles ocupan un lugar de primer orden en algorítmica y estructuras de datos; su presencia es ubicua en informática. Por ejemplo, los árboles se pueden utilizar para describir relaciones entre conceptos, como por ejemplo la jerarquía, la inclusión o la descendencia. Algunos ejemplos de árboles que seguramente ya conocéis son: la estructura de directorios en un sistema de ficheros, la organización de los paquetes en un lenguaje de programación, los árboles sintácticos utilizados por el análisis de las oraciones en un lenguaje, los organigramas más sencillos, los árboles genealógicos más sencillos, la taxonomía utilizada para clasificar organismos biológicos (especie, género, etc.)... En este módulo se presenta una introducción a los aspectos más básicos de la teoría.

En primer lugar, se caracterizan los árboles de varias formas, se presentan las propiedades más básicas, se demuestra que todo árbol con un mínimo de dos vértices tiene un mínimo de dos hojas y se caracterizan los grafos conexos como los que admiten árboles generadores.

Seguidamente, se estudia el problema de la obtención del árbol generador de un grafo conexo. En el contexto de los grafos ponderados se presentan dos algoritmos clásicos para obtener árboles generadores minimales, el de Kruskal y el de Prim.

Finalmente, se estudian los árboles con raíz y se obtienen relaciones importantes entre la altura o profundidad de un árbol con raíz y el número de hojas, relaciones que son útiles para obtener evaluaciones de la complejidad de determinados algoritmos, en particular de ordenación.

1. Conceptos básicos

Un *árbol* es un grafo conexo sin ciclos. Los grafos acíclicos también se denominan *bosques* y la razón es que un grafo acíclico es la reunión de sus componentes conexas, que seguirán siendo acíclicas y, en consecuencia, serán árboles. Así, los grafos acíclicos son unión de árboles, es decir, son "bosques". Existen numerosos resultados de caracterización y propiedades de los árboles.

En particular, existe un resultado fundamental que caracteriza los grafos conexos como los grafos que admiten árboles generadores. Uno de los problemas importantes, especialmente en economía, es el de obtener un árbol generador minimal de un grafo ponderado.

Se puede considerar el concepto de recorrido y de camino orientados o dirigidos; las definiciones serían similares al caso no orientado, pero con arcos que se concatenan y con orientaciones concordantes. Se pueden considerar generalizaciones con multiarcos y lazos dirigidos. El orden es el número de vértices y la medida, el número de arcos. En particular, es conveniente introducir los árboles con raíz.

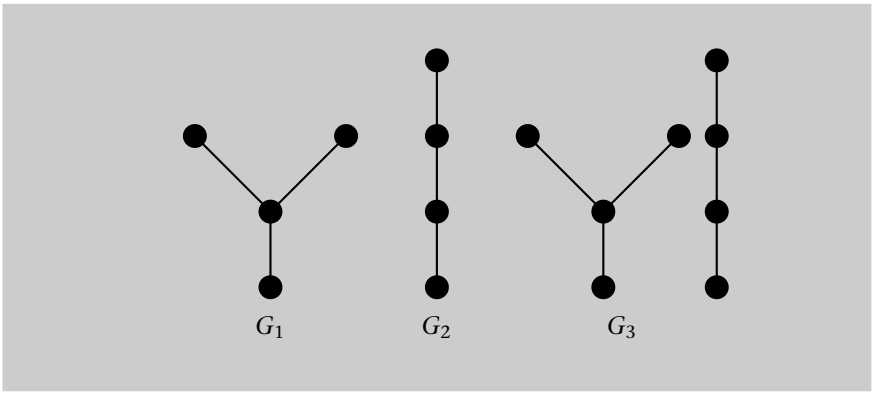
1.1. Caracterización de los árboles

Definición 1

Un **árbol** es un grafo conexo sin ciclos. Si eliminamos la condición de conectividad, obtenemos un **bosque**, es decir, un bosque es un grafo acíclico.

Ejemplo 1

En la figura siguiente podemos ver ejemplos de dos árboles y un bosque. G_1 es un árbol de orden 4 (isomorfo al grafo E_4), G_2 es otro árbol de orden 4 (isomorfo al grafo trayecto T_4). Finalmente, G_3 es un bosque (isomorfo a $E_4 \cup T_4$):



Observad que cada componente conexa de un bosque es un árbol. Por lo tanto, podríamos decir que un bosque es la unión de una colección de árboles.

Teorema 1

Si $T = (V,A)$ es un grafo de orden n y medida m, entonces las propiedades siguientes son equivalentes:

1) T es un árbol.

2) Entre cada pareja de vértices de T existe un único camino.

3) T es conexo y $m = n - 1$.

4) T es acíclico y $m = n - 1$.

Demostración: **1)** $\Leftrightarrow$ **2)** Si T es un árbol, entonces entre cada pareja de vértices hay un camino. Puesto que T no contiene ningún ciclo, este camino tiene que ser único, ya que si hubiera dos caminos diferentes C_1 y C_2 entre u y v entonces el recorrido $C_1 \cup C_2$ sería cerrado y, por lo tanto, contendría un ciclo. Recíprocamente, si entre cada pareja de vértices de T existe un único camino, T es conexo y no contiene ciclos.

1) $\Leftrightarrow$ **3)** T es conexo por la propia definición de árbol. Demostraremos por inducción que $m = n - 1$. Para $n = 1$ el resultado es trivialmente cierto. Supongamos el resultado cierto para todo árbol de orden $k < n$ y vamos a demostrarlo para n. Sea T un árbol de orden n y sea $e = \{u,v\}$ una arista de T. Puesto que ya hemos probado que **1)** $\Leftrightarrow$ **2)**, podemos afirmar que esta arista es el único camino que une los vértices u y v; y, por lo tanto, el grafo $T - e$ está formado exactamente por dos componentes conexas T_u, que contiene u, y T_v, que contiene v. Cada una de estas componentes conexas es un árbol, ya que es un subgrafo de T. Puesto que su orden es menor o igual que n, podemos aplicar la hipótesis de inducción a T_u y a T_v: $m_{T_u} = n_{T_u} - 1$ y $m_{T_v} = n_{T_v} - 1$. Por lo tanto,

$$m = m_{T_u} + m_{T_v} + 1 = n_{T_u} - 1 + n_{T_v} - 1 + 1 = n - 1$$

Recíprocamente, es necesario demostrar que si T es un grafo conexo de orden n y medida $n - 1$, entonces T es acíclico. Ya sabemos que un grafo conexo de orden n debe tener un mínimo de $n - 1$ aristas. Si contuviera un ciclo, eliminando una arista del ciclo, continuaría siendo conexo pero tendría medida $n - 2$, que no es posible.

1) ⇔ 4) La implicación **1) ⇒ 4)** se deduce directamente de la definición de árbol y de la equivalencia anterior. Recíprocamente, debemos demostrar que si T es un grafo acíclico de orden n y medida $n-1$, entonces T es conexo. Sean $T_1,\ldots,T_k$ $(k \geq 1)$ las componentes conexas de T. Puesto que cada T_i no contiene ciclos y es conexo, será un árbol y $m_{T_i} = n_{T_i} - 1$. Por lo tanto,

$$n-1 = m = \sum_{i=1}^{k} m_{T_i} = \sum_{i=1}^{k} (n_{T_i} - 1) = n - k$$

de esta última igualdad se deduce que $k = 1$ y, por lo tanto, T es conexo. ∎

Definición 2

Una **hoja** de un árbol es un vértice de grado 1.

Proposición 2

Todo árbol con un mínimo de dos vértices tiene un mínimo de dos hojas.

Demostración: Sea $T = (V,A)$ un árbol de orden n, y sea F el conjunto de las hojas. Puesto que $|A| = n-1$, por la fórmula de los grados podemos escribir:

$$\begin{aligned} 2(n-1) &= 2|A| = \sum_{v\in V} g(v) = \sum_{v\in F} g(v) + \sum_{v\notin F} g(v) = \\ &= \sum_{v\in F} 1 + \sum_{v\notin F} g(v) \geq |F| + \sum_{v\notin F} 2 = |F| + 2(n-|F|) \end{aligned}$$

Así, de la desigualdad anterior se deriva $|F| \geq 2$. ∎

Ejemplo 2

Demostraremos que un bosque de orden n formado por k árboles tiene medida $n-k$.

En efecto, el bosque $G = (V,A)$ será reunión de las componentes conexas $T_1,\ldots,T_k (k \geq 1)$, que también son árboles. A cada una de ellas se les puede aplicar el resultado $m_{T_i} = n_{T_i} - 1$ y, por lo tanto, podemos escribir

$$|A| = \sum_{i=1}^{k} m_{T_i} = \sum_{i=1}^{k} (n_{T_i} - 1) = (\sum_{i=1}^{k} n_{T_i}) - k = n - k.$$

Ejemplo 3

Calculemos el número de hojas de un árbol que tiene un vértice de grado 3, tres vértices de grado 2 y el resto vértices de grado 1.

Recordemos que si $T = (V,A)$ es un árbol, entonces $m = n-1$, siendo n el orden y m la medida de T.

Si x es el número de hojas, se cumple $n = x + 3 + 1$, y si se aplica la fórmula de los grados se tiene:

$$x + 3 \cdot 2 + 3 = \sum_{v \in V} g(v) = 2(n-1) = 2(x + 3 + 1 - 1) = 2x + 6,$$

así, $x = 3$. La secuencia de grados es, pues, 3,2,2,2,1,1,1.

Ejemplo 4

Sea $T = (V,A)$ un árbol de orden $n = 9$, que tiene tres vértices de grado 3, veamos cuál es la secuencia completa de grados.

Supongamos que $V = \{v_1, \ldots, v_9\}$ y que $x_i = g(v_i)$, $i = 1, \ldots, 9$ son los grados de los vértices del árbol. Puesto que $n \geq 2$, por la proposición 2, existe un mínimo de dos hojas, es decir, un mínimo de dos vértices de grado 1, de manera que la secuencia de grados es $3,3,3,1,1,x_6,x_7,x_8,x_9$ con x_6,x_7,x_8,x_9 a determinar.

Por un lado, por la fórmula de los grados podemos escribir:

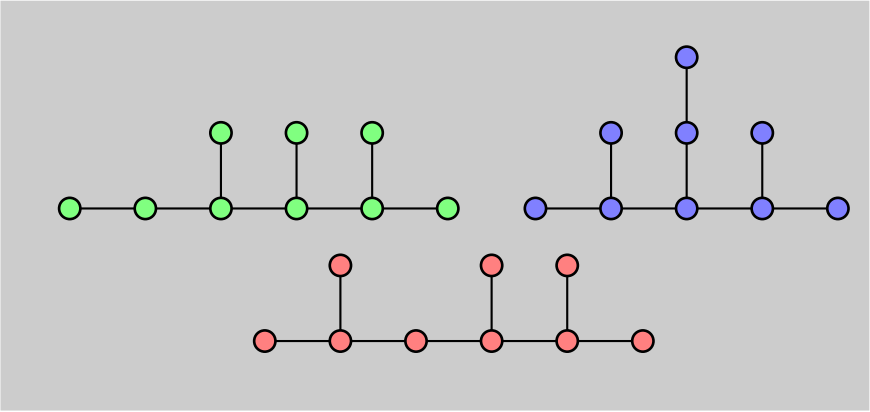
$$2|A| = \sum_{v \in V} g(v) = 3 + 3 + 3 + 1 + 1 + x_6 + x_7 + x_8 + x_9$$

Y, por otro lado, en todo árbol de orden n se cumple $|A| = n - 1 = 9 - 1 = 8$. Si sustituimos en la igualdad anterior, resultará: $x_6 + x_7 + x_8 + x_9 = 5$.

Puesto que un árbol es conexo, no puede haber vértices de grado 0, de donde resulta que $x_6,x_7,x_8,x_9 \geq 1$ y, en consecuencia, el valor de estas incógnitas tiene que ser 1, excepto una, que tiene que ser 2.

Por lo tanto, la secuencia completa es 3, 3, 3, 2, 1, 1, 1, 1, 1.

En el gráfico siguiente podemos ver tres ejemplos de árboles no isomorfos con esta secuencia de grados, lo cual demuestra, en particular, que existen.



Ejercicios

1. Dados nueve puntos en el plano, se trata de unir algunas parejas de manera que el gráfico resulte un árbol. ¿Cuál es el número de trazos que se tendrán que añadir al dibujo?

2. Consideremos un grafo conexo $G = (V,A)$ de orden $|V| = n$. Si $|A| \geq n$, ¿podemos afirmar que el grafo tiene algún ciclo?

3. ¿Cuál es el valor de la suma de los grados de los vértices de un árbol de orden n?

4. ¿Qué árboles son grafos regulares?

5. ¿Puede ser árbol un grafo 3-regular?

6. ¿Existen árboles de orden n, para todo n?

7. Un grafo 3-regular conexo contiene necesariamente algún ciclo. ¿Cierto o falso?

8. Un bosque contiene treinta vértices y veinticinco aristas. ¿Cuántas componentes conexas tiene?

Soluciones

1. El número de trazos es 8, puesto que en un árbol el número de aristas es el número de vértices menos uno.

2. En el caso conexo sí que se puede afirmar; puesto que si fuera acíclico, sería árbol y, en consecuencia sería $|A| = n - 1$.

3. Por la fórmula de los grados, esta suma es $2|A| = 2(n-1) = 2n - 2$.

4. Sólo el grafo trivial y K_2, porque el resto de los árboles tienen vértices de grado 1 (las hojas) y vértices de grado superior.

5. No, puesto que tendría que ser $|A| = |V| - 1$.

6. Sí, por ejemplo los grafos trayecto.

7. Cierto, puesto que de lo contrario sería acíclico y, por lo tanto, árbol; en consecuencia habría vértices de grado 1, contradicción.

8. Recordemos que un bosque es un grafo unión de árboles. Sea, por tanto, $G = T_1 \cup \ldots \cup T_k$ la descomposición de G en reunión de k componentes conexas, que son árboles. Sean n y m el orden y la medida de G; y n_i y m_i el orden y la medida de T_i. De las relaciones $n = \sum_{i=1}^{k} n_i$, $m = \sum_{i=1}^{k} m_i$ y $m_i = n_i - 1$ se deduce

$$m = \sum_{i=1}^{k} m_i = \sum_{i=1}^{k} (n_i - 1) = \sum_{i=1}^{k} n_i - \sum_{i=1}^{k} 1 = n - k$$

Por lo tanto, $k = n - m = 30 - 25 = 5$ componentes conexas.

2. Árboles generadores

2.1. Caracterización de los árboles generadores

Subgrafo generador

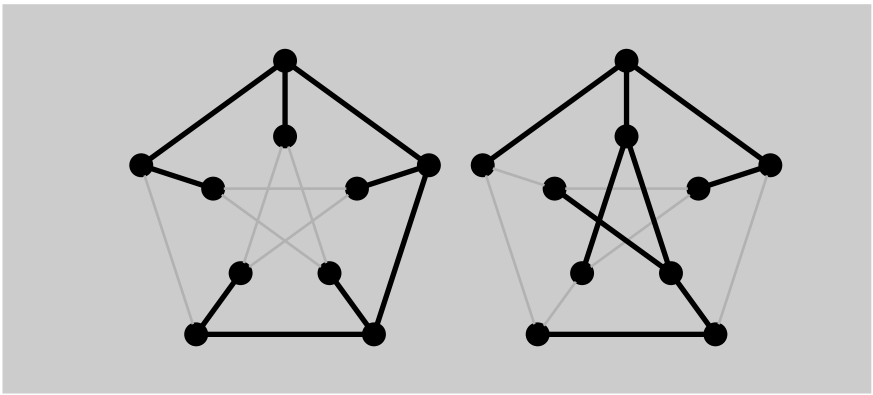
Recordad que un subgrafo es generador si contiene todos los vértices del grafo original.

Definición 3

Un **árbol generador** de un grafo es un subgrafo generador que tiene estructura de árbol. También se denominan **árboles de expansión** (*spanning tree*, en inglés).

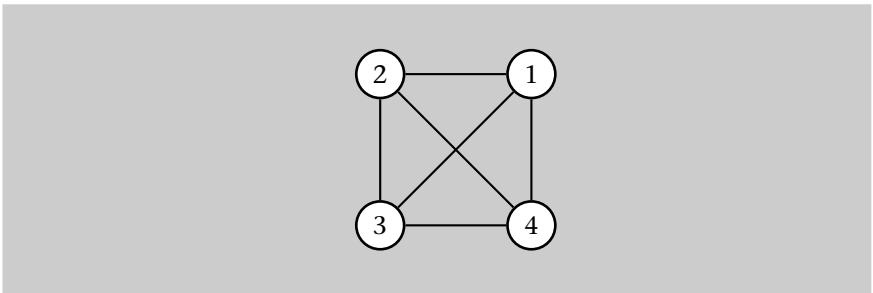
Ejemplo 5

Veamos que un grafo puede contener varios árboles generadores. Estos son dos ejemplos de grafos generadores del grafo de Petersen, indicados con un trazo grueso:



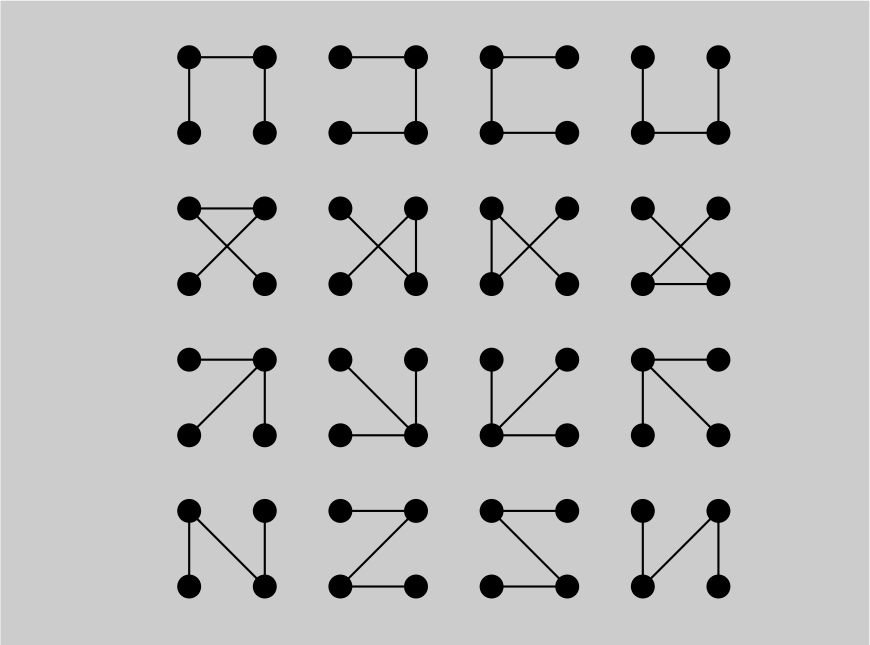
Ejemplo 6

Determinemos todos los árboles generadores, con independencia de isomorfismo, del grafo $G = (V,A)$:



Un árbol generador $T = (V,A')$ es un subgrafo generador de G, es decir, con el mismo conjunto de vértices, que como grafo es un árbol. En particular,

por ser árbol, es $|A'| = |V| - 1 = 3$. Así, los árboles generadores son los siguientes:



El resultado siguiente garantiza la existencia de árboles generadores en un grafo conexo.

Proposición 3

Para un grafo $G = (V,A)$ las propiedades siguientes son equivalentes:

1) G es conexo.

2) G contiene un árbol generador.

Demostración: **2)** $\Rightarrow$ **1)**: Sea $T = (V,A)$ un árbol generador de G, y sean u, v vértices de G, que también lo son de T. Siendo T conexo, hay un camino que conecta los vértices anteriores en T, pero las aristas correspondientes son de G y, por lo tanto, es un camino de G; en consecuencia, G es conexo.

1) $\Rightarrow$ **2)**: Si G es un árbol ya hemos acabado: el árbol generador es el propio grafo. De lo contrario, siendo conexo, tiene que contener algún ciclo C. Eliminamos una arista de este ciclo, creando un nuevo grafo G_1, que seguirá siendo conexo (puesto que la eliminación de una arista de un ciclo no produce desconexión), sigue teniendo los mismos vértices que G (puesto que no hemos eliminado ninguno) y la medida ha disminuido en una unidad. Si G_1 es árbol, será un árbol generador y hemos acabado la demostración. De lo contrario, contiene algún ciclo, y la demostración sigue del mismo modo que antes, eliminando una arista del ciclo. Este proceso es finito, dado que el número de aristas es finito y, por lo tanto, se tiene que llegar a un subgrafo conexo acíclico con todos los vértices del grafo G, es decir, a un árbol generador. ∎

De la definición de árbol generador y del resultado anterior podemos deducir las propiedades siguientes.

Proposición 4

- Si $G = (V,A)$ es conexo de orden n entonces contiene un árbol generador de orden n y medida $n-1$.
- Si $G = (V,A)$ no es conexo entonces cada componente conexa contiene un árbol generador, cuya reunión es un bosque, el **bosque generador** de G.
- Si T es un árbol generador del grafo G entonces, eliminando una arista de T, el árbol deja de ser conexo; si, además, $T \neq G$, añadiendo una arista a T (de las aristas de G que no están en T) entonces T deja de ser acíclico.

2.1.1. Determinación de un árbol generador

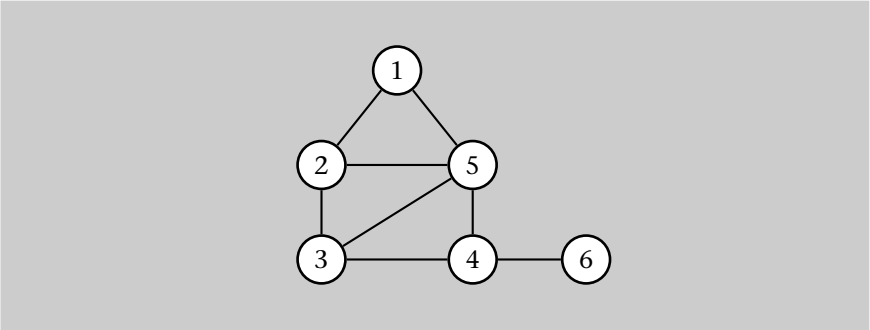
La demostración de la proposición 3 da un método para construir árboles generadores: ir eliminando aristas de los ciclos del grafo. Sin embargo, es más eficiente utilizar los algoritmos de exploración de grafos. Así, el algoritmo siguiente utiliza el *BFS* para obtener un árbol generador del grafo conexo $G = (V,A)$, partiendo del vértice $v \in V$.

```
Entrada : G = (V,A), v ∈ V
Salida : T, árbol generador de G
algoritmo BFS-ArbolGenerador(G,v)
  inicio
    Q ← ∅
    V(T) ← V(G)
    A(T) ← ∅
    para w ∈ V(G)
          estado[w] ← 0
    finpara
    estado[v] ← 1
    añadir(Q,v)
    mientras Q ≠ ∅
                w ← primero(Q)
                para u adyacente a w
                      si estado[u] = 0
                        entonces añadir(Q,u)
                                 estado[u] ← 1
                                 añadir(A(T),{w,u})
                      finsi
                finpara
                eliminar(Q)
    finmientras
    retorno (T)
  fin
```

Simulación del algoritmo

Ejemplo 7

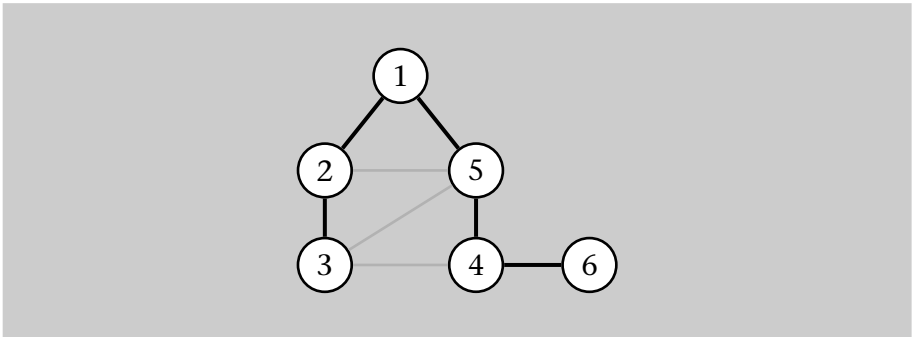
Consideremos el grafo representado en la figura siguiente.



La tabla siguiente registra el funcionamiento del algoritmo para este grafo, con vértice de inicio $v = 1$.

Q	Aristas añadidas	*A (T)*
1	-	$\emptyset$
12	{1,2}	{{1,2}}
125	{1,5}	{{1,2},{1,5}}
25	-	{{1,2},{1,5}}
253	{2,3}	{{1,2},{1,5},{2,3}}
53	-	{{1,2},{1,5},{2,3}}
534	{5,4}	{{1,2},{1,5},{2,3},{5,4}}
34	-	{{1,2},{1,5},{2,3},{5,4}}
4	-	{{1,2},{1,5},{2,3},{5,4}}
46	{4,6}	{{1,2},{1,5},{2,3},{5,4},{4,6}}
6	-	{{1,2},{1,5},{2,3},{5,4},{4,6}}
$\emptyset$	-	{{1,2},{1,5},{2,3},{5,4},{4,6}}

El árbol generador obtenido se puede ver en el gráfico siguiente con las aristas indicadas con un trazo grueso.



Variando el vértice inicial y el orden de elección de las aristas podemos obtener diferentes árboles generadores.

Ejercicios

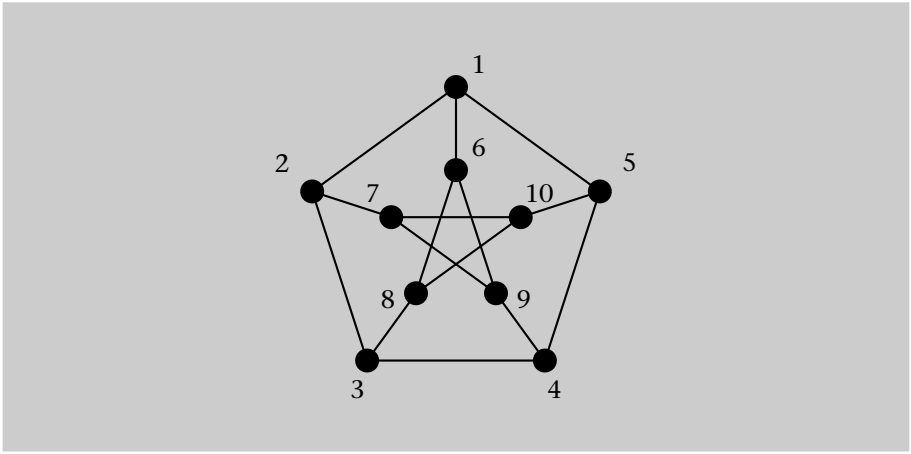
9. ¿Cuántos árboles generadores tiene un árbol?

10. En un árbol generador de un grafo de orden n, ¿cuántas aristas hay?

11. Sea G un grafo conexo y T un árbol generador de G. ¿Cuál es el subgrafo inducido por el conjunto de vértices $V(T)$?

12. Diseñad el algoritmo *DFS-ArbolGenerador*(G,v) que permita obtener un árbol generador de un grafo conexo $G = (V,A)$ a partir del algoritmo *DFS*. Estudiad su complejidad.

13. Utilizando los algoritmos *BFS-ArbolGenerador* y *DFS-ArbolGenerador* encontrad, empezando por el vértice 1, árboles generadores del grafo de Petersen:



Soluciones

9. Sólo uno, él mismo.

10. Hay $n-1$ aristas, puesto que es un árbol.

11. Todo el grafo G.

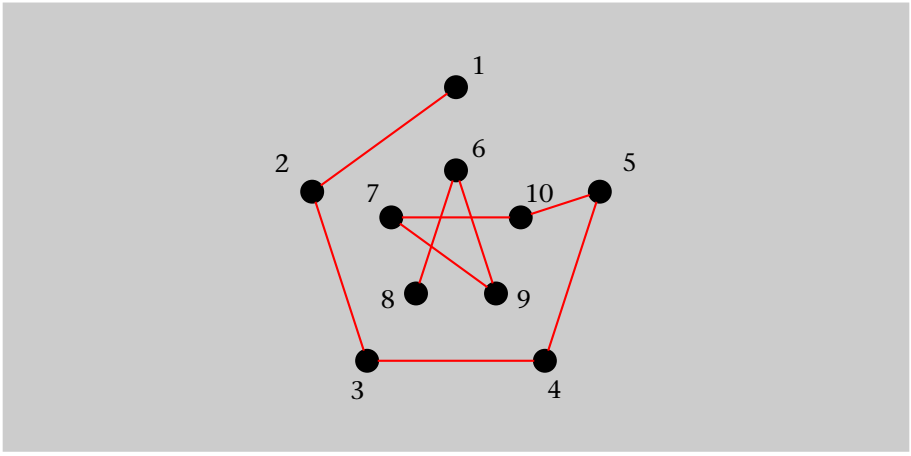
12. Utilizando la versión recursiva del algoritmo *DFS*:

```
Entrada : G = (V,A),v ∈ V
Salida : T,árbol generador de G
algoritmo DFS-ArbolGenerador(G,v)
  inicio
    V(T) ← V(G)
    A(T) ← ∅
    para w ∈ V
          estado[w] ← 0
    finpara
    dfsrec(G,v,T,estado)
    retorno (T)
  fin
```

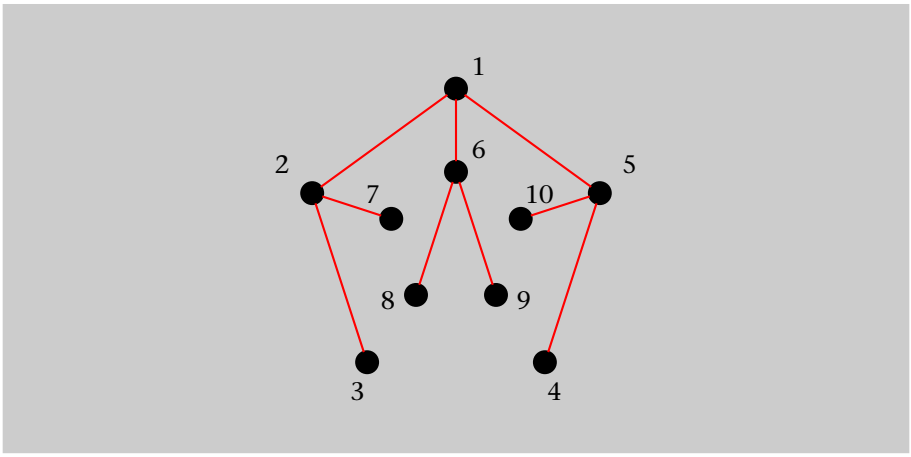
```
función dfsrec(G,v,T,estado)
  inicio
    estado[v] ← 1
    para w adyacente a v
          si estado[w] = 0
             entonces
                        añadir(A(T),{v,w})
                        dfsrec(G,w,T,estado)
          finsi
    finpara
  fin
```

Tanto el *DFS-ArbolGenerador* como el *BFS-ArbolGenerador* utilizan los mismos recursos que el *DFS* y *BFS*, respectivamente. Por lo tanto, su complejidad será $O(n+m)$.

13. Utilizando el algoritmo *DFS-ArbolGenerador* obtendríamos el árbol generador:



Con el algoritmo *BFS-ArbolGenerador* obtendríamos,



2.2. Árboles generadores minimales

Dado un grafo de interconexión entre nodos (ciudades, por ejemplo), cuyas aristas tienen asignadas un peso, que puede corresponder a una cierta medida del coste de la comunicación entre los dos nodos, podría ser necesario establecer un árbol generador minimal, es decir, establecer un subsistema de comunicaciones de tal modo que ningún nodo quede excluido y que globalmente represente un coste mínimo. Este tipo de problemas se pueden resolver con la ayuda de los árboles generadores minimales.

Definición 4

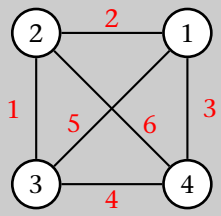
Dado un grafo ponderado (G,w) y un árbol generador T de G definimos el **peso del árbol** T como $w(T) = \sum_{e\in A(T)} w(e)$.

Un **árbol generador minimal** de G es un árbol generador T de G de peso $w(T)$ mínimo.

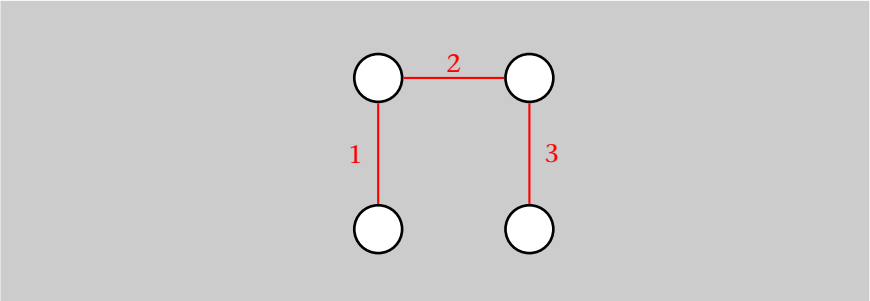
Si G no es conexo, entonces podemos hablar del **bosque generador minimal** de G como aquel que tiene peso mínimo entre todos los bosques generadores de G.

Ejemplo 8

El grafo ponderado siguiente contiene dieciséis árboles generadores diferentes (podéis ver el ejemplo 6).



De todos éstos el árbol generador minimal es el siguiente árbol, con un peso $w(T) = 6$.



Los algoritmos de Kruskal y de Prim son dos algoritmos clásicos de obtención de árboles generadores minimales que corresponden a una tipología bien conocida, la de los algoritmos "*greedy*" o "voraces". Estos algoritmos suelen tomar decisiones que en cada momento son localmente las mejores posibles. Esto no significa, en general, que la solución obtenida sea globalmente la mejor. Curiosamente, esto es así en el caso de los algoritmos mencionados.

2.2.1. Algoritmo de Kruskal

Para construir el árbol generador minimal se empieza con un "árbol vacío"; durante todo el proceso un subgrafo se incrementa con nuevas aristas de la manera siguiente: en cada etapa del proceso se añade una arista que no forme ciclo con las elegidas previamente. Para garantizar la minimalidad, se elige, de entre las posibles, la arista de peso mínimo (no necesariamente adyacente a alguna arista previamente incorporada). El proceso acaba cuando se han incorporado $n-1$ aristas.

Objetivo del algoritmo de Kruskal

El algoritmo de Kruskal busca un árbol generador minimal y elige, en cada paso, la arista de menor peso que no forme ciclo con las ya elegidas.

Formulación del algoritmo de Kruskal

Entrada: un grafo ponderado conexo (G,w) de orden n.
Salida: un árbol generador minimal T de G.
Algoritmo:

inicio
  $k \leftarrow 1$, $T = (V,A')$, $A' = \emptyset$
  mientras $k \leq n-1$
    Elegir la arista $a \in A$ de peso mínimo, no elegida anteriormente y de manera que el subgrafo $T = (V,A' \cup a)$ sea acíclico.
    Añadir a a A'.
    $k \leftarrow k+1$
  finmientras
  retorno(T)
fin.

Implementación del algoritmo de Kruskal

Para implementar el algoritmo de Kruskal se necesita mantener una lista ordenada por el peso de las aristas del grafo y una estructura que permita comprobar, de manera eficiente, que no se forman ciclos.

Intuitivamente, el algoritmo mantiene un bosque. En un principio, existen $|V|$ árboles de un solo vértice. Cuando añadimos una arista se combinan dos árboles en uno. Cuando acaba el algoritmo sólo existe un árbol, el árbol generador minimal.

Estructuras necesarias para la implementación del algoritmo:

- Un grafo ponderado y conexo (G,w) representado mediante una lista de adyacencias.
- Una lista de aristas, E, ordenada según su peso.
- Una estructura denominada **conjuntos-disjuntos** (*disjoint sets*, en inglés) de conjuntos. En principio, cada vértice forma un conjunto que sólo contiene este vértice. Cuando se analiza una arista $\{u,v\}$, se efectúa una **búsqueda** para localizar el conjunto de u y el de v. Si u y v están en el mismo conjunto, la arista no es aceptada porque u y v ya están conectados y, añadiendo la arista $\{u,v\}$, formaría un ciclo. De lo contrario, la arista es aceptada y se ejecuta la **unión** de los dos conjuntos que contienen u y v para formar un nuevo árbol.

  Así las dos operaciones que se deben realizar son:

  – *búsqueda(U,u)*, devuelve el representante del árbol que contiene u.

  – *unión(U,x,y)*, unión de los árboles que tienen como representantes x, y para formar un nuevo árbol.

- Un árbol T que contendrá el árbol generador minimal.

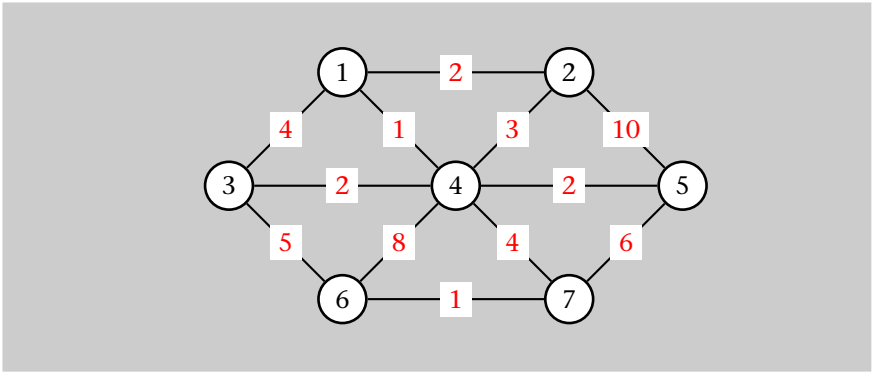
Entrada : (G,w) conexo y ponderado de orden n
Salida : T el árbol generador minimal de G
algoritmo *Kruskal*(G)
  inicio
    U = estructura conjuntos-disjuntos
    E = lista de aristas ordenadas por peso en orden ascendente
    $T \leftarrow (V,A'), A' \leftarrow \emptyset$
    $k \leftarrow 1$
    $i \leftarrow 1$

mientras $(k < n)$
Sea $\{u,v\}$ la arista $E[i]$
$x \leftarrow búsqueda(U,u)$
$y \leftarrow búsqueda(U,v)$
si $(x \neq y)$
entonces $A' \leftarrow A' \cup \{u,v\}$
$k \leftarrow k+1$
$unión(U,x,y)$
finsi
$i \leftarrow i+1$
finmientras
retorno (T)
fin

Simulación del algoritmo de Kruskal

Ejemplo 9

Considerad el grafo definido por el gráfico siguiente.



Podemos utilizar el algoritmo de Kruskal para encontrar el árbol generador minimal. En este caso sólo es necesario hacer una lista de las aristas de menos peso a más peso (en caso de igualdad, se escriben primero las que están formadas por vértices de número menor).

Aristas	Pesos
{1,4}	1
{6,7}	1
{1,2}	2
{3,4}	2
{4,5}	2
{2,4}	3
{1,3}	4
{4,7}	4
{3,6}	5
{5,7}	6
{4,6}	8
{2,5}	10

Tenemos que elegir las 6 primeras aristas (porque hay 7 vértices) que no formen ningún ciclo. Las marcaremos con un asterisco; las aristas descartadas porque forman ciclo las marcaremos en negrita.

Aristas	Pesos
{1,4}∗	1
{6,7}∗	1
{1,2}∗	2
{3,4}∗	2
{4,5}∗	2
{**2,4**}	
{**1,3**}	
{4,7}∗	4
{3,6}	
{5,7}	
{4,6}	
{2,5}	

Por lo tanto, el árbol generador minimal estará formado por las aristas: {1,4}, {6,7}, {1,2}, {3,4}, {4,5}, {4,7} con un peso total 12.

Análisis del algoritmo de Kruskal

En el algoritmo de Kruskal podemos distinguir dos operaciones fundamentales:

1) Ordenación de la lista de aristas según su peso. Si denotamos por m la medida del grafo, entonces podemos ordenar una lista de m aristas con una complejidad $O(m \log m)$, utilizando algoritmos de clasificación como son el *quicksort* o el *heapsort*.

2) El bucle principal del algoritmo se ejecuta $n-1$ veces y, en cada iteración, hacemos dos operaciones de búsqueda y una de unión en la estructura conjuntos-disjuntos. Estas dos operaciones (podéis ver los ejercicios de autoevaluación de este módulo) se pueden hacer utilizando un máximo de $\log m$ pasos. Ja que se ejecuta $n-1$ veces, obtendremos una complejidad $O(n \log m)$.

Todo el algoritmo tendrá una complejidad

$$\max\{O(m \log m), O(n \log m)\}.$$

Puesto que, para un grafo conexo, $m \geq n-1$ podemos concluir que el algoritmo de Kruskal tiene una complejidad $O(m \log m)$.

2.2.2. Algoritmo de Prim

Objetivo del algoritmo de Prim

El algoritmo de Prim busca un árbol generador minimal y elige, en cada paso, la arista de menor peso de las adyacentes a los vértices ya elegidos. De este modo el grafo construido siempre es conexo.

El algoritmo de Prim es similar al algoritmo de Kruskal, con la diferencia de que en cada paso se elige una arista no incorporada previamente, y que sea adyacente a alguna de las ya incorporadas, además, que no forme ciclo con las anteriores y, de entre las aristas disponibles que satisfacen estas condiciones, la que tiene peso mínimo. Si hay más de una, se elige la primera en la ordenación de aristas. De este modo, en todo momento se mantiene un subgrafo

conexo y acíclico. El proceso se termina cuando se han incorporado $n-1$ aristas.

Formulación del algoritmo de Prim

Entrada: un grafo ponderado conexo (G,w) de orden n.
Salida: un árbol generador minimal T de G.
Algoritmo:

inicio
$k \leftarrow 1$, $T = (V,A')$, $A' = \emptyset$
mientras $k \leq n-1$
Elegir la arista $a \in A$ de peso mínimo, no elegida anteriormente,
adyacente a alguna arista de A' y
tal que el subgrafo $T = (V,A' \cup a)$ sea acíclico.
Añadir a a A'.
$k \leftarrow k+1$
finmientras
retorno(T)
fi.

Implementación del algoritmo de Prim

El algoritmo de Prim construye el árbol generador haciendo crecer un solo árbol en pasos sucesivos. Empieza eligiendo cualquier vértice como vértice inicial. En cada paso añadimos la arista de peso mínimo que conecta un vértice del árbol con uno de fuera.

La implementación del algoritmo de Prim es esencialmente idéntica a la del algoritmo de Dijkstra para encontrar la distancia entre dos vértices. Sólo se debe modificar la regla de actualización.

Estructuras necesarias para la implementación del algoritmo:

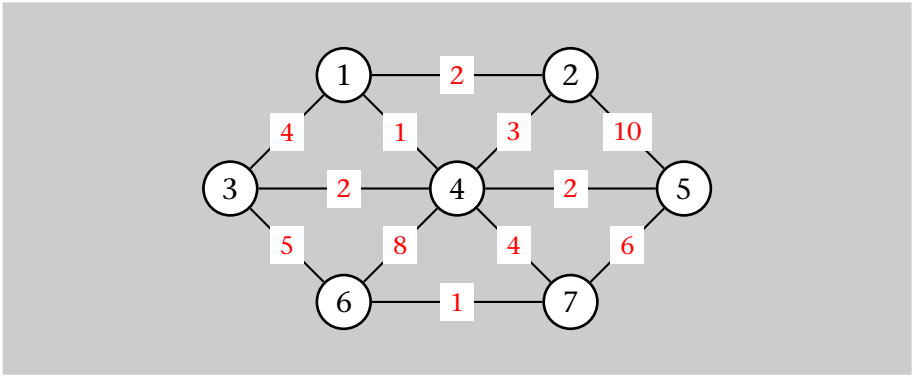
- Un grafo ponderado (G,w) representado mediante una lista de adyacencias.
- Un conjunto U de los vértices que se han visitado, en el orden en el que se ha hecho.
- Una tabla de pesos, $E(\cdot)$, indexada por los vértices de G, que registra el peso de la arista de peso mínimo que conecta un vértice v con un vértice ya visitado.
- Al final, la tabla $E(\cdot)$ registra los pesos de las aristas que forman parte del árbol generador minimal.

En cada paso se fija la etiqueta de uno de los vértices del grafo. Así, tras n pasos habremos calculado el árbol generador minimal.

Entrada : (G,w) de orden n
Salida : T, el árbol generador minimal de G
algoritmo $Prim(G)$
  inicio
    Seleccionamos un vértice inicial $u_0 \in V$
    $U \leftarrow \emptyset$
    para $v \in V \backslash \{u_0\}$
        $E(v) \leftarrow \infty$
        Etiquetamos v con $(E(v),u_0)$
    finpara
    $E(u_0) \leftarrow 0$
    Etiquetamos u_0 con $(0,u_0)$
    $T \leftarrow (V,A'), A' \leftarrow \emptyset$
    para $i \leftarrow 1$ **hasta** n
        u_i vértice que alcanza el $\min\{E(v) \mid v \in V - U\}$
        $U \leftarrow U \cup \{u_i\}$
        $A' \leftarrow A' \cup \{x,u_i\}$ /*donde $(E(u_i),x)$ es la etiqueta de u_i*/
        para $v \in V - U$ adyacente a u_i
            si $w(u_i,v) < E(v)$
              entonces $E(v) \leftarrow w(u_i,v)$
                  Etiquetamos v con $(E(v),u_i)$
            finsi
        finpara
    finpara
    retorno (T)
  fin

Simulación del algoritmo de Prim

Consideremos el grafo definido por el gráfico siguiente.



Si se elige como vértice inicial el vértice 1, la tabla siguiente representa el estado inicial del algoritmo:

Vértices	1	2	3	4	5	6	7
U	0	0	0	0	0	0	0
$(E(\cdot),\cdot)$	(0,1)	$(\infty,1)$	$(\infty,1)$	$(\infty,1)$	$(\infty,1)$	$(\infty,1)$	$(\infty,1)$

En esta tabla, se ha representado el conjunto U y las etiquetas de los vértices. Todos los vértices, excepto el vértice 1, están etiquetados como $(\infty,1)$, que indica que el peso mínimo inicial es ∞ (significa que, en el estado inicial, el vértice no se alcanza).

Los diferentes pasos del algoritmo serán:

$i = 1$

Vértices	1	2	3	4	5	6	7
U	**1**	0	0	0	0	0	0
$(E(\cdot),\cdot)$	(0,1)	**(2,1)**	**(4,1)**	**(1,1)**	$(\infty,1)$	$(\infty,1)$	$(\infty,1)$

$i = 2$

Vértices	1	2	3	4	5	6	7
U	1	0	0	**1**	0	0	0
$(E(\cdot),\cdot)$	(0,1)	(2,1)	**(2,4)**	(1,1)	**(2,4)**	**(8,4)**	**(4,4)**

$i = 3$

Vértices	1	2	3	4	5	6	7
U	1	**1**	0	1	0	0	0
$(E(\cdot),\cdot)$	(0,1)	(2,1)	(2,4)	(1,1)	(2,4)	(8,4)	(4,4)

$i = 4$

Vértices	1	2	3	4	5	6	7
U	1	1	**1**	1	0	0	0
$(E(\cdot),\cdot)$	(0,1)	(2,1)	(2,4)	(1,1)	(2,4)	**(5,3)**	(4,4)

$i = 5$

Vértices	1	2	3	4	5	6	7
U	1	1	1	1	**1**	0	0
$(E(\cdot),\cdot)$	(0,1)	(2,1)	(2,4)	(1,1)	(2,4)	(5,3)	(4,4)

$i = 6$

Vértices	1	2	3	4	5	6	7
U	1	1	1	1	1	0	**1**
$(E(\cdot),\cdot)$	(0,1)	(2,1)	(2,4)	(1,1)	(2,4)	**(1,7)**	(4,4)

$i = 7$

Vértices	1	2	3	4	5	6	7
U	1	1	1	1	1	**1**	1
$(E(\cdot),\cdot)$	(0,1)	(2,1)	(2,4)	(1,1)	(2,4)	(1,7)	(4,4)

De esta última tabla se deduce que el árbol generador minimal estará formado por las aristas {1,2}, {3,4}, {1,4}, {4,5}, {6,7} y {4,7} con un peso mínimo total 12.

Se puede construir la **tabla del algoritmo de Prim** a partir de las filas $(E(\cdot),\circ)$ de cada uno de los pasos (se indica con un asterisco el vértice que se visita):

1	2	3	4	5	6	7
(0,1)	$(\infty,1)$	$(\infty,1)$	$(\infty,1)$	$(\infty,1)$	$(\infty,1)$	$(\infty,1)$
(0,1)*	(2,1)	(4,1)	(1,1)	$(\infty,1)$	$(\infty,1)$	$(\infty,1)$
(0,1)	(2,1)	(2,4)	(1,1)*	(2,4)	(8,4)	(4,4)
(0,1)	(2,1)*	(2,4)	(1,1)	(2,4)	(8,4)	(4,4)
(0,1)	(2,1)	(2,4)*	(1,1)	(2,4)	(5,3)	(4,4)
(0,1)	(2,1)	(2,4)	(1,1)	(2,4)*	(5,3)	(4,4)
(0,1)	(2,1)	(2,4)	(1,1)	(2,4)	(1,7)	(4,4)*
(0,1)	(2,1)	(2,4)	(1,1)	(2,4)	(1,7)*	(4,4)

Análisis del algoritmo de Prim

El algoritmo de Prim tiene la misma estructura que el algoritmo de Dijkstra y, por lo tanto, tendrá la misma complejidad: $O(n^2)$, independientemente del número de aristas del grafo.

Ejercicios

14. Si G es un grafo conexo no ponderado de orden n, ¿cuál es el peso del árbol generador minimal de G?

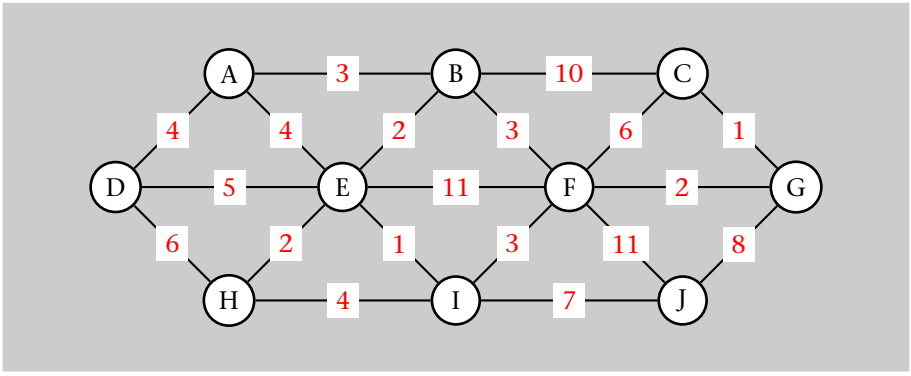
15. Si G es un grafo no ponderado de orden n, ¿cuál es el peso del bosque generador minimal de G?

16. ¿Pueden utilizarse el algoritmo de Kruskal o el de Prim en un grafo conexo no necesariamente ponderado para obtener un árbol generador?

17. Los grafos intermedios que se construyen en el algoritmo de Kruskal, ¿son árboles?

18. Considerad la afirmación: "el algoritmo de Kruskal permite obtener el árbol generador minimal de un grafo conexo ponderado". ¿Es exacta?

19. Encontrad el árbol generador minimal del grafo siguiente utilizando el algoritmo de Kruskal y el algoritmo de Prim. ¿Cuál es el peso mínimo del árbol? ¿Es único el árbol generador minimal? Justificad la respuesta.



20. Si en en el algoritmo de Prim modificamos la condición:

$$w(u_i,v) < E(v),$$

cambiando $<$ por $\leq$, ¿como repercutirá en el funcionamiento del algoritmo? ¿Se obtendrá el mismo árbol generador minimal?

Soluciones

14. El peso es $n-1$.

15. Si G contiene k ($k \geq 1$) componentes conexas, entonces el bosque generador minimal tendrá peso $n - k$.

16. Sí, asignando pesos unidad a todas las aristas.

17. No necesariamente.

18. No, puede haber más de uno.

19. En el algoritmo de Kruskal ordenamos todas las aristas según su peso (en caso de igualdad, consideraremos el orden de la lista de adyacencias):

Aristas	{C,G}	{E,I}	{B,E}	{E,H}	{F,G}	{A,B}	{B,F}	{F,I}	{A,D}
Peso	1	1	2	2	2	3	3	3	4

Aristas	{A,E}	{H,I}	{D,E}	{C,F}	{D,H}	{I,J}	{G,J}	{B,C}	{E,F}	{F,J}
Peso	4	4	5	6	6	7	8	10	11	11

Ahora elegimos las $10 - 1 = 9$ aristas de peso mínimo que no forman ciclo. Se indica con un asterisco las aristas elegidas y en negrita las rechazadas.

Aristas	{C,G}*	{E,I}*	{B,E}*	{E,H}*	{F,G}*	{A,B}*	{B,F}*	{**F,I**}	{A,D}*
Peso	1	1	2	2	2	3	3	3	4

Aristas	{**A,E**}	{**H,I**}	{**D,E**}	{**C,F**}	{**D,H**}	{I,J}*	{G,J}	{B,C}	{E,F}	{F,J}
Peso	4	4	5	6	6	7	8	10	11	11

El árbol generador minimal obtenido tiene un peso total 25.

Ahora repetiremos el problema utilizando el algoritmo de Prim. La tabla siguiente resume la construcción del árbol empezando por el vértice A.

A	B	C	D	E	F	G	H	I	J
(0,A)	(∞,A)	(∞,A)	(∞,A)	(∞,A)	(∞,A)	(∞,A)	(∞,A)	(∞,A)	(∞,A)
(0,A)*	(3,A)	(∞,A)	(4,A)	(4,A)	(∞,A)	(∞,A)	(∞,A)	(∞,A)	(∞,A)
(0,A)	(3,A)*	(10,B)	(4,A)	(2,B)	(3,B)	(∞,A)	(∞,A)	(∞,A)	(∞,A)
(0,A)	(3,A)	(10,B)	(4,A)	(2,B)*	(3,B)	(∞,A)	(2,E)	(1,E)	(∞,A)
(0,A)	(3,A)	(10,B)	(4,A)	(2,B)	(3,B)	(∞,A)	(2,E)	(1,E)*	(7,I)
(0,A)	(3,A)	(10,B)	(4,A)	(2,B)	(3,B)	(∞,A)	(2,E)*	(1,E)	(7,I)
(0,A)	(3,A)	(6,F)	(4,A)	(2,B)	(3,B)*	(2,F)	(2,E)	(1,E)	(7,I)
(0,A)	(3,A)	(1,G)	(4,A)	(2,B)	(3,B)	(2,F)*	(2,E)	(1,E)	(7,I)
(0,A)	(3,A)	(1,G)*	(4,A)	(2,B)	(3,B)	(2,F)	(2,E)	(1,E)	(7,I)
(0,A)	(3,A)	(1,G)	(4,A)*	(2,B)	(3,B)	(2,F)	(2,E)	(1,E)	(7,I)
(0,A)	(3,A)	(1,G)	(4,A)	(2,B)	(3,B)	(2,F)	(2,E)	(1,E)	(7,I)*

Observad que el árbol coincide con el que se ha obtenido con el algoritmo de Kruskal.

Aunque en el algoritmo de Prim siempre hemos elegido la única opción posible, el árbol generador minimal no es único. De hecho, podríamos sustituir la arista {B,F} con peso 3 por la arista {F,I}, con el mismo peso, y obtendríamos un árbol generador minimal diferente. Observad que en el algoritmo de Kruskal sí que hubiéramos podido elegir la arista {F,I} antes que la {B,F} si hubiéramos escogido otro criterio de ordenación de las aristas con el mismo peso.

20. El algoritmo continúa funcionando correctamente y obtenemos un árbol generador minimal aunque puede ser diferente del que se había obtenido en la versión original. De hecho, de entre las aristas del mismo peso, se elegirá la que se examina más tarde.

3. Árboles con raíz

Los grafos dirigidos asimétricos, concepto que definiremos en este apartado, permiten caracterizar los árboles con raíz. Algunos aspectos de estos árboles son muy interesantes para evaluar la complejidad de ciertos algoritmos, especialmente los de ordenación.

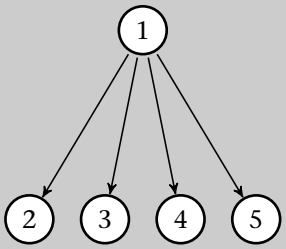
3.1. Caracterización de los árboles con raíz

Definición 5

Dado un arco (u,v) de un digrafo $G = (V,A)$ se dice que u es **padre** del vértice v. Éste es **hijo** del vértice u.

Ejemplo 10

En el digrafo adjunto, el vértice 1 es padre de los vértices 2, 3, 4 y 5; y éstos son hijos del vértice 1.



Definición 6

Un digrafo $G = (V,A)$ es **asimétrico** si no tiene ciclos de longitud 2. Es decir, si $(u,v) \in A$ entonces $(v,u) \notin A$.

Un vértice r de un digrafo $G = (V,A)$ es una **raíz** si existe un $r - v$ camino (dirigido) para todo vértice v del digrafo.

Definición 7

Un digrafo $T = (V,A)$ es un **árbol con raíz** r si:

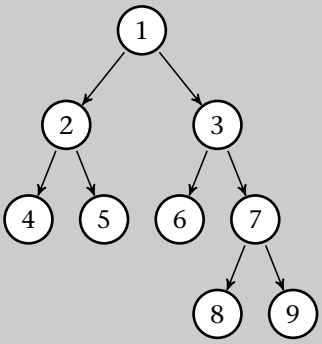
1) T es un digrafo asimétrico.

2) El grafo subyacente es un árbol.

3) Para todo vértice $v \in V$ existe un $r - v$ camino.

Ejemplo 11

El grafo siguiente es un árbol con raíz.

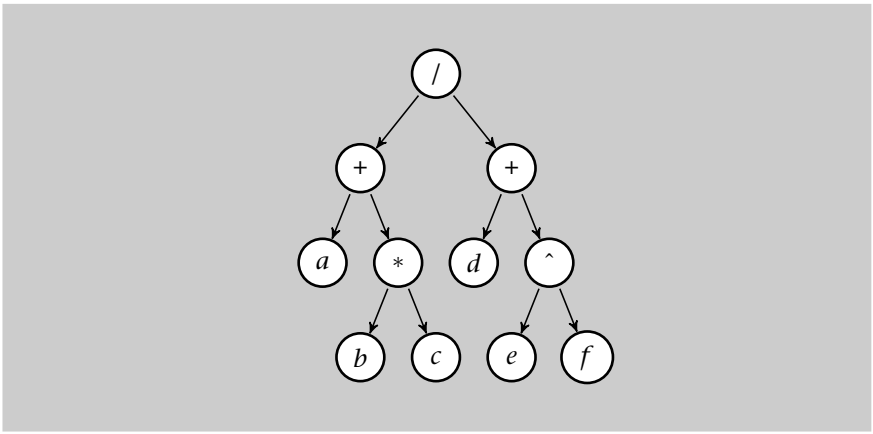


La raíz es el vértice etiquetado 1. Los arcos son (1,2), (1,3), (2,4), (2,5), (3,6), (3,7), (7,8) y (7,9).

Los árboles con raíz siempre se representan en forma "jerárquica" colocando la raíz en lo alto. Esto supone que a menudo sea innecesario indicar la orientación de los arcos que se sobreentiende en "sentido descendente".

Ejemplo 12

Una aplicación importante de los árboles con raíz es la representación de expresiones aritméticas, como por ejemplo $(a + b * c)/(d + e\hat{}f)$. Si una expresión simple como $\alpha \circ \beta$ se guarda como un árbol con raíz $\circ$ e hijos α y β, entonces la expresión anterior, de acuerdo con la precedencia de los operadores $+, *, /, \hat{}$ se representará como:



Esta representación es especialmente útil en el proceso de compilación de programas, puesto que permite evaluar eficientemente cualquier expresión aritmética.

Ved también

En el módulo "Fundamentos de grafos", se define $g^-(v)$ como el número de arcos que tienen el vértice v como destino.

Teorema 5

Sea $T = (V,A)$ un digrafo asimétrico. Entonces las afirmaciones siguientes son equivalentes:

1) T es un árbol con raíz.

2) T tiene una raíz r con $g^-(r) = 0$ y $g^-(v) = 1$ para todo vértice v diferente de la raíz.

3) El grafo subyacente $\overline{T}$ es conexo y existe un vértice r tal que $g^-(r) = 0$ y $g^-(v) = 1$ para todo vértice v diferente de r.

Demostración: **1)** $\Rightarrow$ **2)**: Si $g^-(r) \neq 0$, entonces habría un vértice $v \neq r$ y un arco (v,r). Puesto que también existe un $r - v$ camino, entonces tendríamos un ciclo dirigido en T contra la hipótesis de que T es asimétrico y el grafo subyacente es un árbol. De manera parecida se prueba que $g^-(v) = 1$.

2) $\Rightarrow$ **3)**: Sólo se necesita comprobar que el grafo subyacente es conexo. Si T tiene una raíz r, entonces existe un $r-v$ camino dirigido para todo vértice v de T. Por lo tanto, el grafo subyacente será conexo.

3) $\Rightarrow$ **1)**: Claro está que T tiene que ser asimétrico. Puesto que $\overline{T}$ es conexo, existe un camino (en $\overline{T}$) entre r y v. Si este camino no fuera un $r - v$ camino dirigido, entonces habría un vértice del camino con $g^-(v) > 1$. ∎

Ejercicios

21. ¿Un árbol con raíz puede tener más de una raíz?

22. En un árbol con raíz, ¿el camino que une la raíz con cada vértice es único?

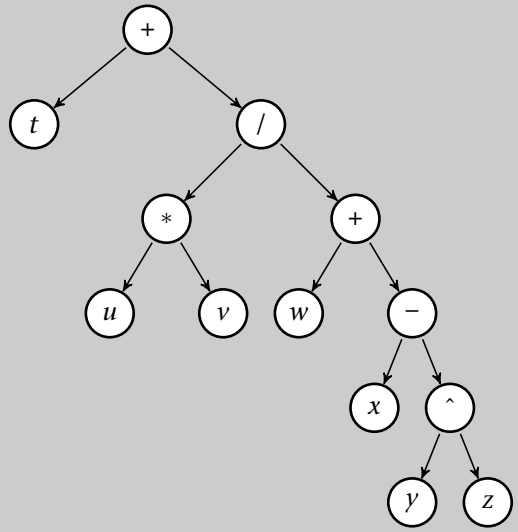
23. Dibujad el árbol con raíz de la expresión aritmética: $t + u * v/(w + x - y\hat{}z)$.

Soluciones

21. No; si hubiera dos raíces r, r', entonces habría un camino orientado $r - r'$ y $g^-(r') = 1$, que es absurdo, ya que contradice la segunda propiedad del teorema 5.

22. Sí. Si hubiera más de uno, entonces el grafo subyacente contendría un ciclo.

23. En este caso, y de acuerdo con la precedencia de los operadores, la raíz es el signo +. El árbol resultante será:



3.2. Árboles *m*-arios

En este subapartado se supone que $T = (V,A)$ es un árbol con raíz r. Demos primero algunas definiciones terminológicas.

Definición 8

Una **hoja** es un vértice que no tiene hijos, es decir, con grado de salida 0; también se denomina **vértice terminal**. Los otros vértices se denominan **internos** (o **no terminales**).

Definición 9

El **nivel** de un vértice v de T es la longitud del único $r - v$ camino.

Definición 10

La **altura** (o **profundidad**) del árbol T es el máximo de los niveles:

$$h(T) = \max\{\text{nivel}(v) \mid v \in V\}$$

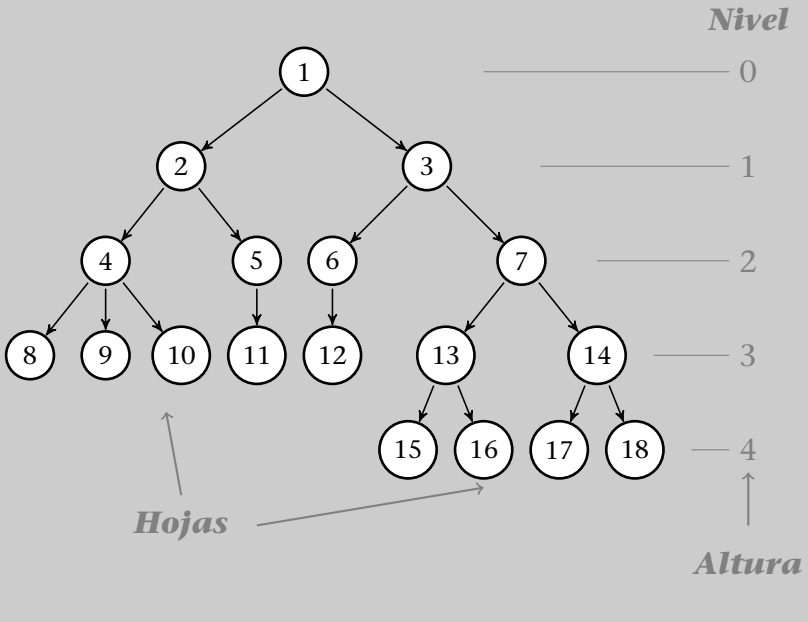
Si el nivel de cada hoja de T es igual a $h(T)$ o a $h(T)-1$ diremos que T es un **árbol equilibrado**.

Ved también

El concepto de árbol equilibrado es muy importante en el almacenamiento eficiente de información en una base de datos o una estructura de datos en memoria. Podéis ver más detalles en la asignatura *Diseño de estructuras de datos*.

Ejemplo 13

En el árbol siguiente los vértices 8, 9, 10, 11, 12, 15, 16, 17, 18 son hojas. El resto son nodos internos.



Cada vértice está en un nivel.

Vértices de nivel 0: 1
Vértices de nivel 1: 2, 3
Vértices de nivel 2: 4, 5, 6, 7
Vértices de nivel 3: 8, 9, 10, 11, 12, 13, 14
Vértices de nivel 4: 15, 16, 17, 18.

Por lo tanto, la altura del árbol (el máximo de los niveles) es 4. Además, es un árbol equilibrado porque todas las hojas están en los dos últimos niveles.

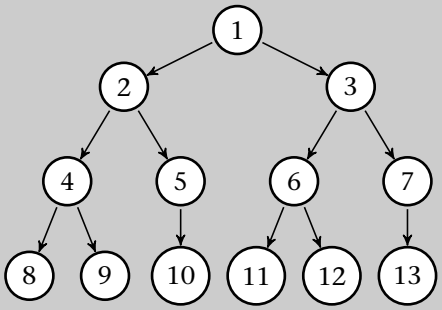
Definición 11

Un árbol con raíz $T = (V,A)$ es m-**ario** ($m \geq 2$) si $0 \leq g^+(v) \leq m$, $\forall v \in V$. En particular, si $m = 2$ diremos que tenemos un **árbol binario**.

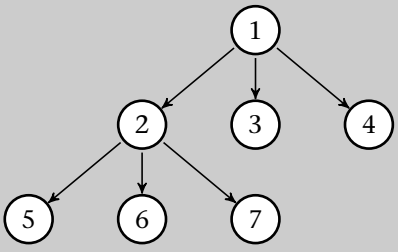
Un árbol m-ario $T = (V,A)$ es **completo** si $\forall v \in V$, $g^+(v) = m$ o $g^+(v) = 0$. Es decir, los vértices internos tienen grado de salida m y las hojas tienen grado de salida 0.

Ejemplo 14

Este árbol con raíz es un árbol binario (cada vértice tiene dos hijos como máximo) no completo (por ejemplo, el vértice etiquetado con un 5 es un vértice interno que no tiene dos hijos):



En cambio, el árbol de la figura siguiente es un árbol 3-ario o ternario y completo.



Observad que los árboles con raíz de una expresión aritmética (podéis ver el ejemplo 12) con operadores binarios son siempre árboles binarios completos.

Existen relaciones importantes que se utilizan en varias situaciones en el análisis de algoritmos.

Proposición 6

Sea $T = (V,A)$ un árbol m-ario,

1) Si T es completo, contiene t hojas e i vértices internos, entonces $t = (m-1)i + 1$.

2) Si T tiene altura h y contiene t hojas, entonces $t \leq m^h$. Es decir, hay un máximo de m^h hojas en un árbol m-ario de altura h.

3) Si $m \geq 2$, de altura h y con t hojas, entonces $h \geq \lceil \log_m t \rceil$ (donde $\lceil x \rceil$ es el mínimo entero mayor o igual que x). Suponiendo que T sea completo y equilibrado entonces $h = \log_m t$.

Demostración: Si denotamos por n el orden de T, entonces el primer resultado es consecuencia de la igualdad $t + i = n = mi + 1$.

El segundo resultado es consecuencia del hecho de que el número de vértices de nivel k ($0 \leq k \leq h$) es menor o igual que m^k.

Finalmente, si T es completo y equilibrado, entonces el nivel $h-1$ tiene m^{h-1} vértices, de los cuales no todos son hojas. Por lo tanto, $m^{h-1} < t$. Por otro lado, ya sabemos que $t \leq m^h$. De las dos desigualdades obtenemos

$$m^{h-1} < t \leq m^h$$

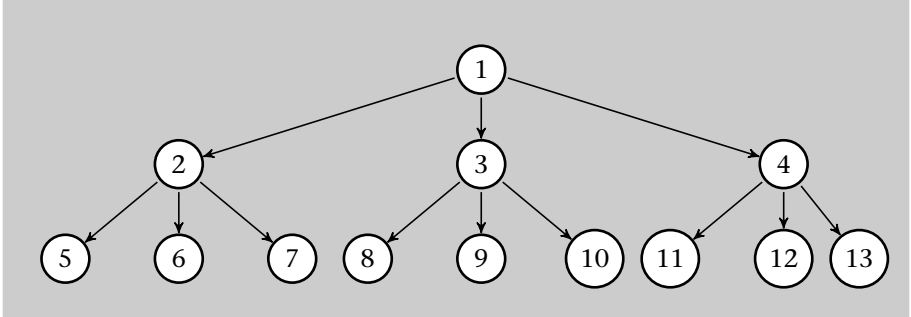
que es equivalente a

$$h - 1 < \log_m t \leq h$$

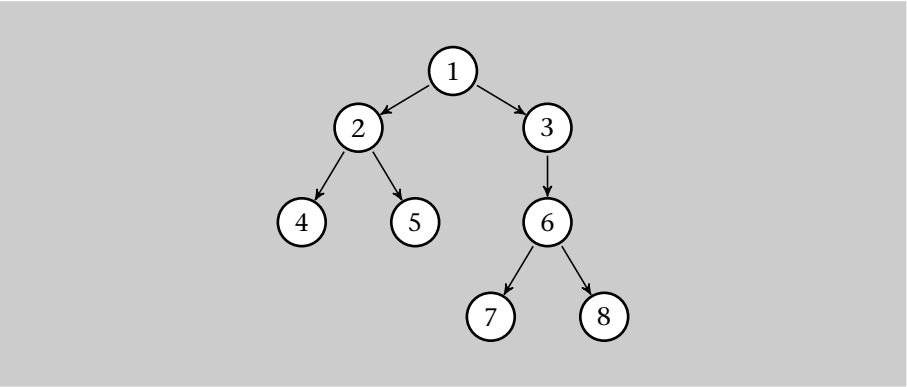
Por la definición de la función $\lceil x \rceil$ obtenemos la igualdad $h = \lceil \log_m t \rceil$. Si no fuera completo o no fuera equilibrado, es evidente que $h \geq \lceil \log_m t \rceil$. ∎

Ejercicios

24. Considerad el siguiente árbol con raíz; indicad cuál es su altura y si es o no equilibrado.



25. Considerad el siguiente árbol con raíz. Indicad cuáles son las hojas, los niveles de los vértices y la altura del árbol:



26. ¿Cuál es la altura mínima de un árbol 4-ario con 127 hojas?

27. En un campeonato de tenis individual participan veintisiete jugadores que compiten en la modalidad de eliminatorias. ¿Cuál es el número de partidos que se deben jugar para determinar el campeón?

Soluciones

24. Es un árbol ternario de altura 2 y equilibrado.

25. Hojas: 4, 5, 7, 8.

Niveles de los vértices:

Vértices de nivel 0: 1
Vértices de nivel 1: 2, 3
Vértices de nivel 2: 4, 5, 6
Vértices de nivel 3: 7, 8

Por lo tanto, la altura será 3.

26. El número de hojas t de un árbol m-ario, de altura h cumple la condición $t \leq m^h$. Si sustituimos, $127 \leq 4^h$, de donde $h \geq \lceil \log_4 127 \rceil = 4$.

27. Los veintisiete jugadores se pueden representar como las hojas de un árbol binario. La raíz del árbol será la final. El número de vértices internos es el número de partidos que deben jugarse. Así, se necesitará organizar un total de $i = (t-1)/(m-1) = 26/1 = 26$ partidos.

3.3. Algoritmos de exploración de los árboles binarios con raíz

Si consideramos una relación de orden entre los hijos de cada vértice interno de un árbol con raíz, entonces nos podemos plantear el problema de explorar de manera exhaustiva todos sus vértices.

En el caso concreto de los árboles binarios, hay tres algoritmos básicos para explorar todos sus vértices: los denominados *recorrido en preorden*, *recorrido en inorden* y *recorrido en postorden*. Los tres recorridos se distinguen básicamente en la forma como se exploran los dos hijos de cada vértice. Antes de estudiarlos se tiene que introducir la notación siguiente: T es el árbol binario de raíz r, T_1 y T_2 son los subárboles de raíces v_1 y v_2 inducidos por los hijos de la raíz r.

- Recorrido en preorden (o en profundidad) de T:

  1) Explorar la raíz r.

  2) Explorar el subárbol T_1 en preorden.

  3) Explorar el subárbol T_2 en preorden.

- Recorrido en inorden de T:

  1) Explorar el subárbol T_1 en inorden.

  2) Explorar la raíz r.

  3) Explorar el subárbol T_2 en inorden.

- Recorrido en postorden de T:

  1) Explorar el subárbol T_1 en postorden.

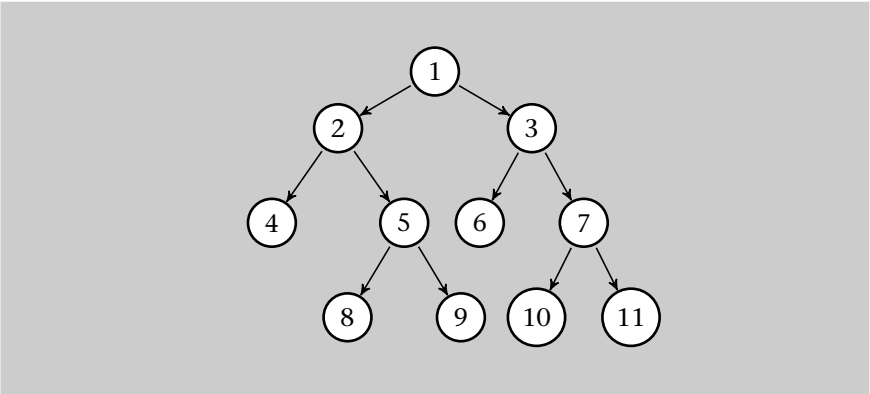
  2) Explorar el subárbol T_2 en postorden.

  3) Explorar la raíz r.

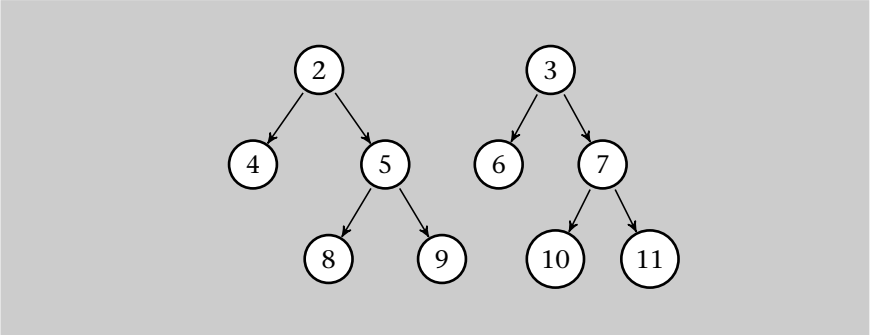
Estos algoritmos visitan cada vértice una sola vez y tiene una complejidad que es una función lineal del orden del árbol, es decir, una complejidad $O(n)$.

Ejemplo 15

El árbol de la figura siguiente es un árbol binario.



La raíz es el vértice etiquetado con un 1. La raíz tiene dos hijos, etiquetados 2 y 3, que son raíces de dos subárboles:



De manera recursiva también podríamos considerar los subárboles de raíz 4, 5, 6,...

Los diferentes recorridos de este árbol son:

Preorden: 1,2,4,5,8,9,3,6,7,10,11;

Inorden: 4,2,8,5,9,1,6,3,10,7,11;

Postorden: 4,8,9,5,2,6,10,11,7,3,1.

Ejercicios

28. La lista siguiente es la lista de adyacencias de un árbol binario. El símbolo "–" representa la ausencia de hijo. Encontrad los recorridos en preorden, inorden y postorden del árbol:

A : B, F
B : –, C
C : D, G
D : –, E
E : –, –
F : –, –
G : –, –

La primera posición de la lista representa el hijo situado a la izquierda de la raíz y la segunda posición, el hijo situado a la derecha.

29. Encontrad los recorridos en preorden, inorden y postorden del árbol de la expresión aritmética $(a + b * c)/(d + e\hat{}f)$.

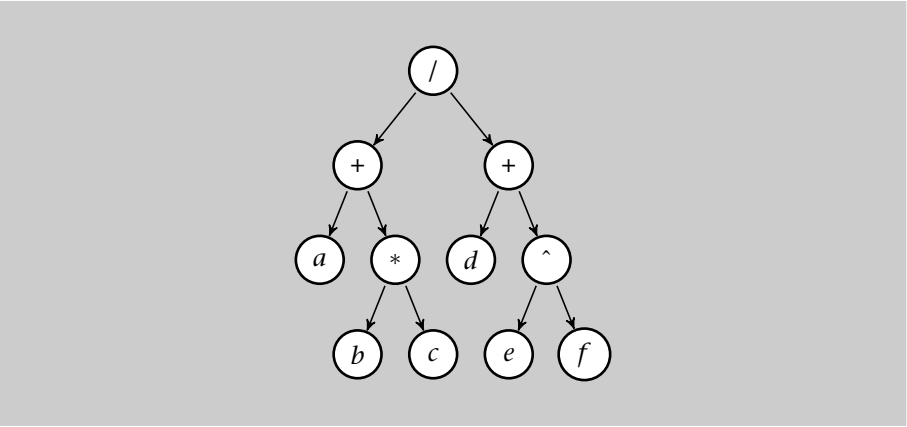
30. Construid un árbol binario T cuyo recorrido en preorden es a,b,d,e,c,f,h,i,g,j,k y el recorrido en postorden es d,e,b,h,i,f,j,k,g,c,a.

Soluciones

28. Los tres recorridos son:

Preorden: A, B, C, D, E, G, F
Inorden: B, D, E, C, G, A, F
Postorden: E, D, G, C, B, F, A

29. El árbol de la expresión es



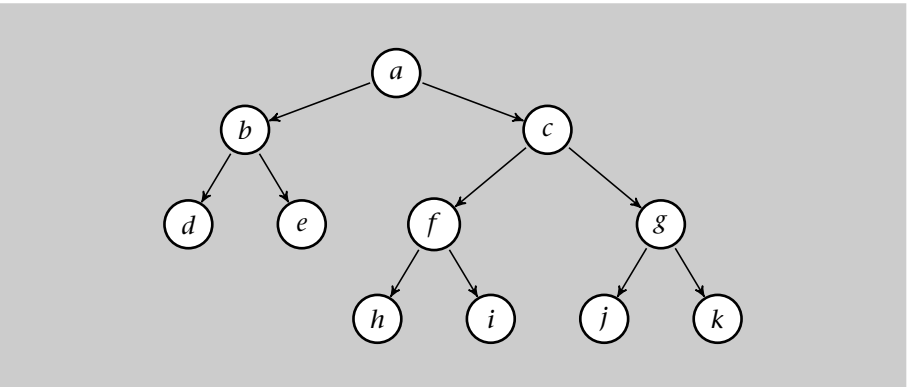
y los tres recorridos son:

Preorden: / + *a* ∗ *b c* + *d* ^ *e f*
Inorden: *a* + *b* ∗ *c* / *d* + *e* ^ *f*
Postorden: *a b c* ∗ + *d e f* ^ + /

Para las expresiones aritméticas el recorrido en preorden da lugar a un tipo de representación de la expresión denominado *notación prefija* (o *notación polaca*, introducida por el lógico Jan Lukasiewicz). El recorrido en inorden da lugar a la *notación infija* (que coincide con la convencional sin paréntesis). Finalmente, el recorrido en postorden da lugar a la *notación postfija* (o también *polaca inversa*).

Las notaciones prefija y postfija son especialmente adecuadas para evaluar expresiones aritméticas. Los compiladores son los encargados de transformar las expresiones convencionales a la notación prefija o postfija, según los casos. También se encargan de evaluar de manera eficiente las expresiones a partir de estas representaciones con la ayuda de una pila.

30. El árbol que se pide es el siguiente.



Ejercicios de autoevaluación

1. Sea $T = (V,A)$ un árbol de orden n con sólo vértices de grado 1 y de grado 3. Expresad, en función del orden n del árbol, el número de hojas y de vértices de grado 3.

2. Probad que un grafo acíclico $G = (V,A)$ con el grado de todos los vértices par es el grafo nulo.

3. ¿Cuántos árboles generadores, con independencia de isomorfismos, tienen los grafos K_2, K_3 y K_4? Intentad encontrar una fórmula que permita calcular el número de árboles generadores del grafo completo de orden n, K_n.

4. Indicad cómo se puede buscar un bosque generador de un grafo no conexo a partir de los algoritmos *BFS* y *DFS*.

5. El nuevo gobierno del archipiélago de Sealand ha decidido unir sus seis islas mediante puentes que permitan conectarlas directamente. El coste de construcción de un puente depende de la distancia entre las islas. Esta tabla recoge las distancias entre islas:

	A	B	C	D	E	F
A	–	5	6	4	3	7
B	–	–	2	4	8	5
C	–	–	–	4	8	8
D	–	–	–	–	2	5
E	–	–	–	–	–	4
F	–	–	–	–	–	–

El gobierno quiere construir un sistema de puentes de manera que el coste total de la obra sea mínimo.

a) Utilizando la teoría de grafos, encontrad los puentes que se deben construir de manera que el coste total de la obra sea mínimo.

b) Supongamos que la capital está en la isla B y que sólo se puede construir un puente entre dos islas si previamente alguna de éstas ya se comunica con la capital (la maquinaria se tiene que transportar a través de los puentes). ¿En qué orden se deberían construir los puentes?

c) Justificad por qué no es aconsejable utilizar el algoritmo de Kruskal para resolver este segundo problema.

6. Para implementar el algoritmo de Kruskal se debe comprobar que, cuando se añade una arista, no forma ciclo. Una manera eficiente de comprobar que no forman ciclo es mediante la estructura denominada conjuntos-disjuntos. Esta estructura se puede representar mediante una tabla U indexada por los vértices del grafo e iniciada a –1. Inicialmente todos los vértices son conjuntos disjuntos. Sobre esta tabla se hacen dos operaciones:

a) **Unión**: se combinan dos conjuntos (dos árboles) si tienen una arista en común.

b) **Búsqueda**: devuelve un representante (la raíz del árbol) del conjunto que contiene un vértice del grafo.

Estas dos operaciones se pueden implementar de la siguiente manera:

```
función unión(U,v1,v2)
  inicio
    si (U[v1] < U[v2])
      entonces U[v2] ← v1
           sino
                si (U[v1] = U[v2])
                  entonces U[v1] ← U[v2] – 1
                finsi
                U[v2] ← v1
    finsi
  fin
```

```
función búsqueda(U,v)
  inicio
    si (U[v] < 0)
      entonces retorno (v)
           sino
                U[v] ← búsqueda(U,U[v])
                retorno (U[v])
    finsi
  fin
```

Utilizando el grafo del ejemplo 9, mostrad el contenido de la tabla U durante la aplicación del algoritmo de Kruskal a este grafo. Notemos que la tabla inicial será:

1	2	3	4	5	6	7
-1	-1	-1	-1	-1	-1	-1

7. Sea (G,w) un grafo ponderado, describid un algoritmo para encontrar un árbol generador maximal. Es decir, un árbol generador T de G tal que su peso $w(T)$ sea máximo. Estudiad la eficiencia del algoritmo propuesto.

8. Los algoritmos de Kruskal y de Prim permiten obtener el árbol generador minimal de un grafo conexo y ponderado. Comparad, según la eficiencia de cada algoritmo, cuál de los dos es más adecuado para encontrar el árbol generador minimal de un grafo.

9. ¿Cuántos vértices tiene un árbol 5-ario completo con doscientos vértices internos?

10. Sea $T = (V,A)$ un árbol binario completo de orden $n = |V| \geq 3$. Probad que el árbol subyacente tiene $\frac{n+1}{2}$ hojas (vértices de grado 1).

11. Tómense 8 monedas idénticas de las cuales se sabe que una es falsa (pesa más que las otras). Si sólo se dispone de una balanza, ¿cuál es el número mínimo de pesadas que se deben hacer para poder asegurar que se descubre la moneda falsa?

12. La secuencia $* a + b * c + d e$ es un recorrido del árbol de una expresión aritmética. Encontrad la expresión original.

Soluciones

1. Denotemos por x_1 el número de hojas y por x_3 el número de vértices de grado 3. Por un lado, tenemos que $n = x_1 + x_3$. Por otro lado, en todo árbol $|A| = n - 1$ y, aplicando la fórmula de los grados, podemos escribir:

$$x_1 + 3x_3 = 2|A| = 2(n-1) = 2(x_1 + x_3 - 1)$$

y obtener así la relación $x_3 = x_1 - 2$, a partir de la cual resulta:

$$n = x_1 + x_3 = x_1 + (x_1 - 2) = 2x_1 - 2,$$

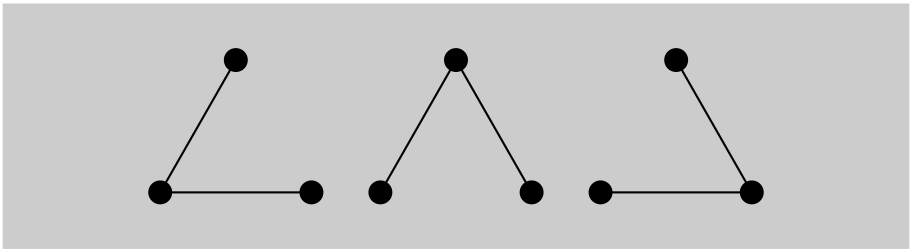
de donde $x_1 = \frac{n+2}{2}$ y finalmente $x_3 = \frac{n-2}{2}$.

2. Sea n el orden del grafo G.

Si el grafo es conexo, entonces, por la aciclicidad, es un árbol y es bien conocido que todo árbol con un mínimo de dos vértices tiene un mínimo de dos hojas o vértices de grado 1, lo cual, en este caso, es imposible por la hipótesis. Por lo tanto, $n < 2$ y, en consecuencia, es $G = N_1$.

Si el grafo no es conexo, entonces el grafo es unión de componentes conexas $G = G_1 \cup \ldots \cup G_k$, y cada una de estas componentes conexas es un grafo conexo acíclico con todos los grados de orden par. Podemos aplicar a cada componente el resultado anterior y de este modo cada componente es isomorfa al grafo nulo N_1. Finalmente, $G = N_1 \cup \ldots \cup N_1 = N_k$.

3. K_2 es el grafo trayecto T_2 y, por lo tanto, tiene un solo árbol generador. K_3 tiene tres árboles generadores, tal y como muestra el siguiente gráfico:



K_4 tiene dieciséis árboles generadores, tal y como se puede ver en el ejemplo 6.

En general, para el grafo K_n existen n^{n-2} árboles generadores diferentes.

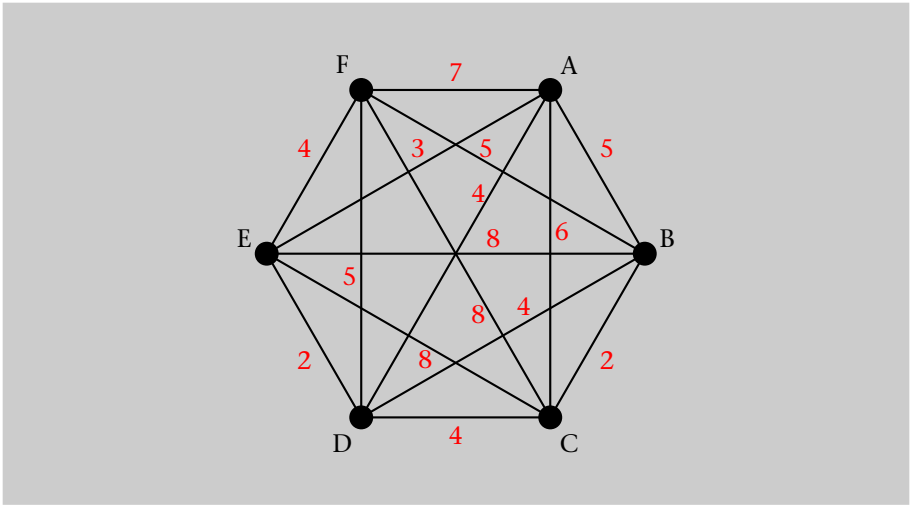
4. Si G no es conexo, entonces es unión de componentes conexas $G = G_1 \cup \ldots \cup G_k$. Para construir un bosque generador será suficiente buscar un árbol generador T_i para cada componente conexa G_i y considerar el bosque $T_1 \cup \ldots \cup T_k$.

El algoritmo siguiente utiliza el algoritmo *BFS-ArbolGenerador* aunque también se podría utilizar el algoritmo *DFS-ArbolGenerador*.

Entrada : $G = (V,A)$ de orden n
Salida : T bosque generador de G
algoritmo *BFS-BosqueGenerador*(G)
  inicio
    $T \leftarrow \emptyset$
    $U \leftarrow V$
    mientras $U \neq \emptyset$
      $v \leftarrow$ cualquier vértice de U
      $R \leftarrow$ *BFS-ArbolGenerador*(G,v)
      $T \leftarrow T \cup R$
      $U \leftarrow U - V(R)$
    finmientras
    retorno (T)
  fin

Observad que la complejidad es la del *BFS*, $O(n+m)$, puesto que exploramos todos los vértices y las aristas de G.

5. El gráfico siguiente muestra la forma del grafo.



a) El problema nos pide buscar el árbol generador minimal de este grafo. Podemos utilizar indistintamente el algoritmo de Kruskal o el de Prim.

Si utilizamos el algoritmo de Kruskal, se necesitará ordenar todas las aristas según su coste:

Aristas	{B,C}	{D,E}	{A,E}	{A,D}	{B,D}	{C,D}	{E,F}	{A,B}	{B,F}
Coste	2	2	3	4	4	4	4	5	5

Aristas	{D,F}	{A,C}	{A,F}	{B,E}	{C,E}	{C,F}
Coste	5	6	7	8	8	8

A continuación, se tienen que elegir cinco aristas de coste mínimo y que no formen ciclo. Estas son: {B,C}, {D,E}, {A,E}, {B,D}, {E,F} con un coste mínimo total 15.

Utilizando el algoritmo de Prim con vértice inicial B:

A	B	C	D	E	F
(∞,B)	(0,B)*	(∞,B)	(∞,B)	(∞,B)	(∞,B)
(5,B)	(0,B)	(2,B)*	(4,B)	(8,B)	(5,B)
(5,B)	(0,B)	(2,B)	(4,B)*	(8,B)	(5,B)
(4,D)	(0,B)	(2,B)	(4,B)	(2,D)*	(5,B)
(3,E)*	(0,B)	(2,B)	(4,B)	(2,D)	(4,E)
(3,E)	(0,B)	(2,B)	(4,B)	(2,D)	(4,E)*

De la tabla se deduce que deben construirse los puentes: {A,E}, {B,C}, {B,D}, {D,E}, {F,E} con un coste total 15.

b) En este caso, la solución válida se obtiene con el algoritmo de Prim, puesto que en cada paso fija un vértice y una arista. Los asteriscos indican los vértices fijados. La solución sería: {B,C}, {B,D}, {D,E}, {A,E}, {F,E}.

c) De acuerdo con el enunciado tenemos que construir el árbol generador minimal de manera que siempre sea conexo. El algoritmo de Kruskal, a diferencia del de Prim, no garantiza que el grafo que se va generando sea conexo.

6. En el ejemplo 9, hemos elegido las aristas marcadas con un asterisco y rechazado las marcadas en negrita:

Aristas	Pesos
{1,4}∗	1
{6,7}∗	1
{1,2}∗	2
{3,4}∗	2
{4,5}∗	2
{**2,4**}	
{**1,3**}	
{4,7}∗	4
{3,6}	
{5,7}	
{4,6}	
{2,5}	

La tabla siguiente muestra las llamadas sucesivas a las funciones *unión* y *búsqueda* y el contenido de la tabla *U*:

Inicialmente

1	2	3	4	5	6	7
-1	-1	-1	-1	-1	-1	-1

búsqueda(*U*,1) → 1; *búsqueda*(*U*,4) → 4;

unión(*U*,1,4)

1	2	3	4	5	6	7
-2	-1	-1	1	-1	-1	-1

búsqueda(*U*,6) → 6; *búsqueda*(*U*,7) → 7;

unión(*U*,6,7)

1	2	3	4	5	6	7
-2	-1	-1	1	-1	-2	6

búsqueda(*U*,1) → 1; *búsqueda*(*U*,2) → 2;

unión(*U*,1,2)

1	2	3	4	5	6	7
-2	1	-1	1	-1	-2	6

búsqueda(*U*,3) → 3; *búsqueda*(*U*,4) → 1;

unión(*U*,3,1)

1	2	3	4	5	6	7
-2	1	1	1	-1	-2	6

búsqueda(*U*,4) → 1; *búsqueda*(*U*,5) → 5;

unión(*U*,1,5)

1	2	3	4	5	6	7
-2	1	1	1	1	-2	6

búsqueda(U,2) → 1; *búsqueda*(U,4) → 1;
búsqueda(U,1) → 1; *búsqueda*(U,3) → 1;
búsqueda(U,4) → 1; *búsqueda*(U,7) → 6;
unión(U,1,6)

1	2	3	4	5	6	7
-3	1	1	1	1	1	6

Esta última tabla describe un árbol con raíz 1 y altura 2. Cualquier nueva aplicación de la función *unión* ya daría un ciclo.

7. Los algoritmos de Kruskal y Prim se pueden utilizar para buscar un árbol generador maximal.

En el algoritmo de Kruskal sólo se debe modificar la ordenación de las aristas:

E = lista de aristas ordenadas por peso en orden descendente

En el algoritmo de Prim sólo se tiene que modificar la comparación entre los pesos:

si $w(u_i,v) > E(v)$ **entonces**

De estas modificaciones se desprende que los algoritmos para buscar un árbol generador maximal tienen la misma eficiencia que para buscar un árbol generador minimal.

8. Sea (G,w) un grafo ponderado conexo de orden n y medida m, el algoritmo de Kruskal determina un árbol generador minimal con una complejidad $O(m \log m)$ y el algoritmo de Prim con una complejidad $O(n^2)$.

Observemos que la eficiencia del algoritmo de Kruskal depende de la medida del grafo, mientras que el algoritmo de Prim tiene una eficiencia que sólo depende del orden del grafo. Para poder comparar los dos algoritmos, habrá que estudiar la relación que hay entre estas dos cantidades, entre $m \log m$ y n^2.

Para un grafo conexo la medida está acotada por

$$n-1 \leq m \leq \frac{n(n-1)}{2}$$

Cuando la medida está próxima a $\frac{n(n-1)}{2}$ (al grafo completo K_n), se dice que el grafo es *denso* (en inglés, *dense graph*). En caso contrario, diremos que el grafo es *poco denso* (*sparse graph*, en inglés).

Por lo tanto, si el grafo es poco denso entonces $m \approx n-1$ y podemos sustituir

$$O(m \log m) = O((n-1)\log(n-1)) = O(n \log n) < O(n^2)$$

En cambio, si el grafo es denso $m \approx \frac{n(n-1)}{2}$ y sustituyendo

$$O(m \log m) = O(\frac{n(n-1)}{2} \log \frac{n(n-1)}{2}) = O(n^2 \log n) > O(n^2)$$

Se puede concluir que para grafos poco densos es más eficiente el algoritmo de Kruskal que el de Prim. En cambio, para grafos densos es mejor el algoritmo de Prim.

La principal ventaja del algoritmo de Prim es que su eficiencia es independiente de la medida del grafo. Como consecuencia, en muchas aplicaciones es preferido al de Kruskal.

9. Si aplicamos el apartado 1 de la proposición 6, podemos calcular el número de hojas, $t = (m-1)i + 1 = (5-1)200 + 1 = 801$. El número total de vértices es $n = t + i = 801 + 200 = 1.001$.

10. Siendo $n = |V|$ el orden del árbol, el número de aristas es $|A| = n - 1$. En un árbol como el indicado se consideran los siguiente conjuntos disjuntos de vértices:

- la raíz, que es el único vértice de grado 2,
- las hojas o vértices de grado 1; se indicará por F el conjunto de las hojas, de cardinal $k = |F|$,
- el resto de los vértices, vértices internos, de grado 3, de los cuales hay m; se indicará por I el conjunto de vértices internos.

De acuerdo con la distribución anterior se puede escribir

$$n = m + k + 1.$$

Usando la fórmula de los grados:

$$2|A| = \sum_{v \in V} g(v) = \sum_{v \in Y} g(v) + \sum_{v \in F} g(v) + 2.$$

Ahora, aplicando la condición $|A| = n - 1$ y teniendo en cuenta los grados de los vértices, se puede escribir:

$$2(n-1) = 3m + k + 2.$$

Sustituyendo en la última igualdad el valor de $m = n - k - 1$, obtenido en la primera relación, obtenemos finalmente $k = \frac{n+1}{2}$.

11. Si se distribuyen las monedas (todas o parte de ellas) entre los platos de la balanza de forma que haya el mismo número en cada plato se pueden considerar tres posibles resultados:

Caso 1: los platos están equilibrados. Significa que las monedas de los dos platos no son falsas.

Caso 2: el plato de la izquierda de la balanza pesa más. Significa que la moneda falsa está a la izquierda.

Caso 3: el plato de la derecha pesa más. Significa que la moneda falsa está a la derecha.

Se puede representar esta situación mediante un árbol ternario que tendrá un mínimo de ocho hojas. Una para cada una de las monedas. La altura del árbol representará el número de pesadas necesarias para descubrir la moneda falsa. Si aplicamos las propiedades de los árboles m-arios, $m = 3$

$$h \geq \lceil \log_3 t \rceil \geq \lceil \log_3 8 \rceil = 2$$

Se necesitarán, pues, un mínimo de dos pesadas para descubrir la moneda falsa. ¿Existe alguna estrategia que permita alcanzar este mínimo?

12. En el árbol binario T de una expresión aritmética, los operandos son siempre las hojas y los operadores los vértices internos. Además, el recorrido en preorden siempre empieza en la raíz del árbol (y de cada subárbol). Por lo tanto, por la forma de la expresión deducimos que se trata del recorrido en preorden del árbol binario de una expresión aritmética con la raíz $*$.

Así, la expresión tendrá la forma $\alpha * \beta$ donde α y β son, respectivamente, los dos subárboles de T. El subárbol α también se tiene que recorrer en preorden y, por lo tanto, tendría que empezar por un operador. Ahora bien, puesto que el símbolo que sigue a $*$ en el recorrido de T es una a, podemos concluir que se trata de una hoja y que $\alpha = a$. Así, tendremos $a * \beta$ con $\beta = + \; b \; * \; c \; + \; d \; e$.

Siguiendo un razonamiento parecido para el subárbol β, podemos deducir que el signo + es la raíz, b es el primer subárbol de β y $* \; c \; + \; d \; e$ el segundo.

Siguiendo este mismo razonamiento de manera recursiva, se llega al resultado $\beta = b + c * (d + e)$.

Finalmente, sustituyendo α y β en la expresión inicial se obtiene la expresión $a * (b + c * (d + e))$.

Bibliografía

Biggs, N. L. (1994). *Matemática Discreta*. (1a. edición, traducción de M. Noy). Barcelona: Ediciones Vicens Vives.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. (2001). *Introduction to Algorithms*. Cambridge: MIT Press.