Arbres

PID_00254247

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

Temps mínim de dedicació recomanat: 4 hores

Universitat Oberta de Catalunya

Índex

Introducció 5

1. Conceptes bàsics 7
1.1. Caracterització dels arbres 7
Exercicis 10
Solucions 11

2. Arbres generadors 12
2.1. Caracterització dels arbres generadors 12
2.1.1. Determinació d'un arbre generador 14
Exercicis 16
Solucions 16
2.2. Arbres generadors minimals 18
2.2.1. Algorisme de Kruskal 19
2.2.2. Algorisme de Prim 22
Exercicis 27
Solucions 27

3. Arbres amb arrel 30
3.1. Caracterització dels arbres amb arrel 30
Exercicis 32
Solucions 33
3.2. Arbres *m*-aris 33
Exercicis 36
Solucions 37
3.3. Algorismes d'exploració dels arbres binaris amb arrel 37
Exercicis 39
Solucions 39

Exercicis d'autoavaluació 41

Solucionari 43

Bibliografia 48

Introducció

Els arbres ocupen un lloc de primer ordre en algorísmica i estructures de dades; la seva presència és ubiqua en informàtica. Per exemple, els arbres es poden utilitzar per a descriure relacions entre conceptes com ara la jerarquia, la inclusió o la descendència. Alguns exemples d'arbres que segurament ja coneixeu són: l'estructura de directoris en un sistema de fitxers; l'organització dels paquets en un llenguatge de programació; els arbres sintàctics utilitzats per a l'anàlisi de les oracions en una llengua; els organigrames més senzills; els arbres genealògics més senzills; la taxonomia utilitzada per a classificar organismes biològics (espècie, gènere, etc.). En aquest mòdul presentarem una introducció als aspectes més bàsics de la teoria.

En primer lloc, es caracteritzen els arbres de diverses maneres, se'n presenten les propietats més bàsiques, es demostra que tot arbre amb un mínim de dos vèrtexs té un mínim de dues fulles i es caracteritzen els grafs connexos com els que admeten arbres generadors.

Seguidament, s'estudia el problema de l'obtenció de l'arbre generador d'un graf connex. En el context dels grafs ponderats es presenten dos algorismes clàssics per a obtenir arbres generadors minimals, el de Kruskal i el de Prim.

Finalment, s'estudien els arbres amb arrel i s'obtenen relacions importants entre l'altura o profunditat d'un arbre amb arrel i el nombre de fulles, relacions que són útils per a obtenir avaluacions de la complexitat de determinats algorismes, en particular dels d'ordenació.

1. Conceptes bàsics

Un *arbre* és un graf connex sense cicles. Els grafs acíclics també s'anomenen *boscos* i la raó és que un graf acíclic és la reunió dels seus components connexos, que seguiran essent acíclics i, en conseqüència, seran arbres. Així, els grafs acíclics són reunió d'arbres, és a dir, són "boscos". Hi ha una sèrie de resultats sobre la caracterització i les propietats dels arbres.

En particular, hi ha un resultat fonamental que caracteritza els grafs connexos com els grafs que admeten arbres generadors. Un dels problemes importants, especialment en economia, és el d'obtenir un arbre generador minimal d'un graf ponderat.

Es poden considerar els conceptes de recorregut i de camí orientats o dirigits; les definicions serien similars al cas no orientat, però amb arcs que es concatenen amb orientacions concordants. Es poden considerar generalitzacions amb multiarcs i llaços dirigits. L'ordre és el nombre de vèrtexs i la mida, el nombre d'arcs. En particular és convenient d'introduir els arbres amb arrel.

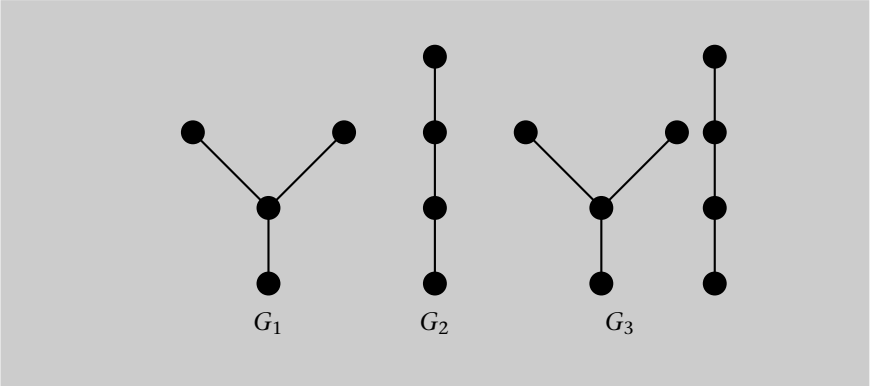
1.1. Caracterització dels arbres

Definició 1

Un **arbre** és un graf connex sense cicles. Si eliminem la condició de connectivitat obtenim un **bosc**, és a dir, un bosc és un graf acíclic.

Exemple 1

En la figura següent podem veure exemples de dos arbres i un bosc. G_1 és un arbre d'ordre 4 (isomorf al graf E_4), G_2 és un altre arbre d'ordre 4 (isomorf al graf trajecte T_4). Finalment, G_3 és un bosc (isomorf a $E_4 \cup T_4$):



Observeu que cada component connex d'un bosc és un arbre. Per tant, podríem dir que un bosc és la unió d'una col·lecció d'arbres.

Teorema 1

Si $T = (V,A)$ és un graf d'ordre n i mida m, aleshores les propietats següents són equivalents:

1) T és un arbre.

2) Entre cada parella de vèrtexs de T hi ha un únic camí.

3) T és connex i $m = n - 1$.

4) T és acíclic i $m = n - 1$.

Demostració: **1)** $\Leftrightarrow$ **2)** Si T és un arbre, entre cada parell de vèrtexs hi ha un camí. Com que T no conté cap cicle, aquest camí ha de ser únic, ja que si hi hagués dos camins diferents C_1 i C_2 entre u i v, aleshores el recorregut $C_1 \cup C_2$ seria tancat i, per tant, contindria un cicle. Recíprocament, si entre cada parell de vèrtexs de T hi ha un únic camí, T és connex i no conté cicles.

1) $\Leftrightarrow$ **3)** T és connex per la pròpia definició d'arbre. Demostrarem per inducció que $m = n - 1$. Per a $n = 1$ el resultat és trivialment cert. Suposem el resultat cert per a tot arbre d'ordre $k < n$ i demostrem-ho per a n. Sigui T un arbre d'ordre n i sigui $e = \{u,v\}$ una aresta de T. Com que ja hem provat que **1)** $\Leftrightarrow$ **2)** podem afirmar que aquesta aresta és l'únic camí que uneix els vèrtexs u i v i, per tant, el graf $T - e$ està format exactament per dos components connexos: T_u, que conté u, i T_v, que conté v. Cadascun d'aquests components connexos és un arbre, pel fet que és un subgraf de T. Com que el seu ordre és més petit o igual que n, podem aplicar la hipòtesi d'inducció a T_u i T_v: $m_{T_u} = n_{T_u} - 1$ i $m_{T_v} = n_{T_v} - 1$. Per tant,

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

Recíprocament, cal demostrar que si T és un graf connex d'ordre n i mida $n-1$ aleshores T és acíclic. Ja sabem que un graf connex d'ordre n ha de tenir un mínim de $n - 1$ arestes. Si contingués un cicle, eliminant una aresta del cicle continuaria essent connex però tindria mida $n - 2$, cosa que no és possible.

1) $\Leftrightarrow$ **4)** La implicació **1)** $\Rightarrow$ **4)** es dedueix directament de la definició d'arbre i de l'equivalència anterior. Recíprocament, cal demostrar que si T és un graf acíclic d'ordre n i mida $n - 1$ aleshores T és connex. Siguin $T_1, \ldots, T_k$ ($k \geq 1$)

els components connexos de T. Com que cada T_i no conté cicles i és connex, serà un arbre i $m_{T_i} = n_{T_i} - 1$. Per tant,

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

D'aquesta darrera igualtat es dedueix que $k = 1$ i, per tant, T és connex. ∎

Definició 2

Una **fulla** d'un arbre és un vèrtex de grau 1.

Proposició 2

Tot arbre amb un mínim de dos vèrtexs té un mínim de dues fulles.

Demostració: Sigui $T = (V,A)$ un arbre d'ordre n, i sigui F el conjunt de les fulles. Com que $|A| = n - 1$, pel lema de les encaixades podem escriure:

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

Així, de la desigualtat anterior es deriva que $|F| \geq 2$. ∎

Exemple 2

Demostrarem que un bosc d'ordre n format per k arbres té mida $n - k$.

En efecte, el bosc $G = (V,A)$ serà reunió dels components connexos $T_1, \ldots, T_k$ ($k \geq 1$), que també són arbres. A cadascun d'ells se li pot aplicar el resultat $m_{T_i} = n_{T_i} - 1$ i, per tant, podem escriure

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

Exemple 3

Calculem el nombre de fulles d'un arbre que té un vèrtex de grau 3, tres vèrtexs de grau 2 i la resta de vèrtexs de grau 1.

Recordem que si $T = (V,A)$ és un arbre, aleshores $m = n - 1$, essent n l'ordre i m la mida de T.

Si x és el nombre de fulles, es compleix $n = x + 3 + 1$, i si s'aplica el lema de les encaixades es té:

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

així, $x = 3$. La seqüència de graus és, doncs, 3,2,2,2,1,1,1.

Exemple 4

Donat un arbre $T = (V,A)$ d'ordre $n = 9$, que té tres vèrtexs de grau 3, vegem quina és la seqüència completa de graus.

Suposem que $V = \{v_1, \ldots, v_9\}$ i que $x_i = g(v_i)$, $i = 1, \ldots, 9$ són els graus dels vèrtexs de l'arbre. Atès que $n \geq 2$, per un resultat anterior, hi ha un mínim de dues fulles, és a dir, un mínim de dos vèrtexs de grau 1, de manera que la seqüència de graus és $3,3,3,1,1,x_6,x_7,x_8,x_9$ amb x_6,x_7,x_8,x_9 a determinar.

D'una banda, pel lema de les encaixades podem escriure

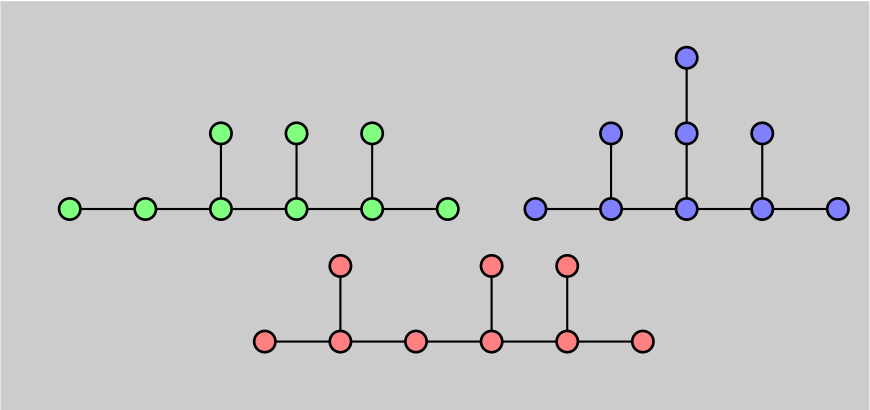
$$2|A| = \sum_{v \in V} g(v) = 3 + 3 + 3 + 1 + 1 + x_6 + x_7 + x_8 + x_9,$$

i, d'altra banda, en tot arbre d'ordre n es compleix $|A| = n - 1 = 9 - 1 = 8$. Si substituïm a la igualtat anterior, resultarà: $x_6 + x_7 + x_8 + x_9 = 5$.

Atès que un arbre és connex, no hi poden haver vèrtexs de grau 0, d'on resulta que $x_6,x_7,x_8,x_9 \geq 1$ i, en conseqüència, el valor d'aquestes incògnites ha de ser 1, excepte una que ha de ser 2.

Per tant, la seqüència completa és 3, 3, 3, 2, 1, 1, 1, 1, 1.

En el gràfic següent podem veure tres exemples d'arbres no isomorfs amb aquesta seqüència de graus, cosa que demostra, en particular, que n'existeixen.



Exercicis

1. Donats nou punts del pla, es tracta d'unir alguns parells de manera que resulti un arbre. Quin és el nombre de traços que s'hauran d'afegir al dibuix?

2. Considerem un graf connex $G = (V,A)$ d'ordre $|V| = n$. Si $|A| \geq n$, podem afirmar que el graf té algun cicle?

3. Quin és el valor de la suma dels graus dels vèrtexs d'un arbre d'ordre n?

4. Quins arbres són grafs regulars?

5. Pot ser arbre un graf 3-regular?

6. Existeixen arbres d'ordre n, per a tot n?

7. Un graf 3-regular connex conté necessàriament algun cicle. Cert o fals?

8. Un bosc conté trenta vèrtexs i vint-i-cinc arestes. Quants components connexos té?

Solucions

1. El nombre de traços és 8, ja que en un arbre el nombre d'arestes és el nombre de vèrtexs menys u.

2. En el cas connex sí que es pot afirmar, ja que si fos acíclic, seria arbre i, en conseqüència, seria $|A| = n - 1$.

3. Pel lema de les encaixades, aquesta suma és $2|A| = 2(n-1) = 2n-2$.

4. Només el graf trivial i K_2, perquè la resta d'arbres tenen vèrtexs de grau 1 (les fulles) i vèrtexs de grau superior.

5. No, ja que hauria de ser $|A| = |V| - 1$.

6. Sí, per exemple els grafs trajecte.

7. Cert, ja que en cas contrari seria acíclic i, per tant, arbre; en conseqüència, hi hauria vèrtexs de grau 1, cosa que ens dóna una contradicció.

8. Recordem que un bosc és un graf unió d'arbres. Sigui, doncs, $G = T_1 \cup \ldots \cup T_k$ la descomposició de G en reunió de k components connexos, que són arbres. Siguin n i m l'ordre i la mida de G; i n_i i m_i, l'ordre i la mida de T_i. De les relacions $n = \sum_{i=1}^{k} n_i$, $m = \sum_{i=1}^{k} m_i$ i $m_i = n_i - 1$ s'obté

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

Per tant, $k = n - m = 30 - 25 = 5$ components connexos.

2. Arbres generadors

2.1. Caracterització dels arbres generadors

Subgraf generador

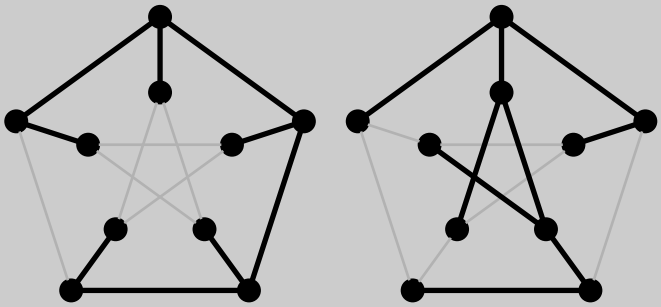
Recordeu que un subgraf és generador si conté tots els vèrtexs del graf original.

Definició 3

Un **arbre generador** d'un graf és un subgraf generador que té estructura d'arbre. També s'anomena **arbre d'expansió** (*spanning tree*, en anglès).

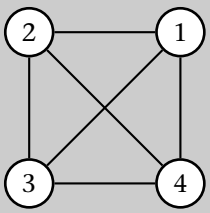
Exemple 5

Vegem que un graf pot contenir diversos arbres generadors. Aquests són dos exemples de grafs generadors del graf de Petersen, indicats amb un traç gruixut:

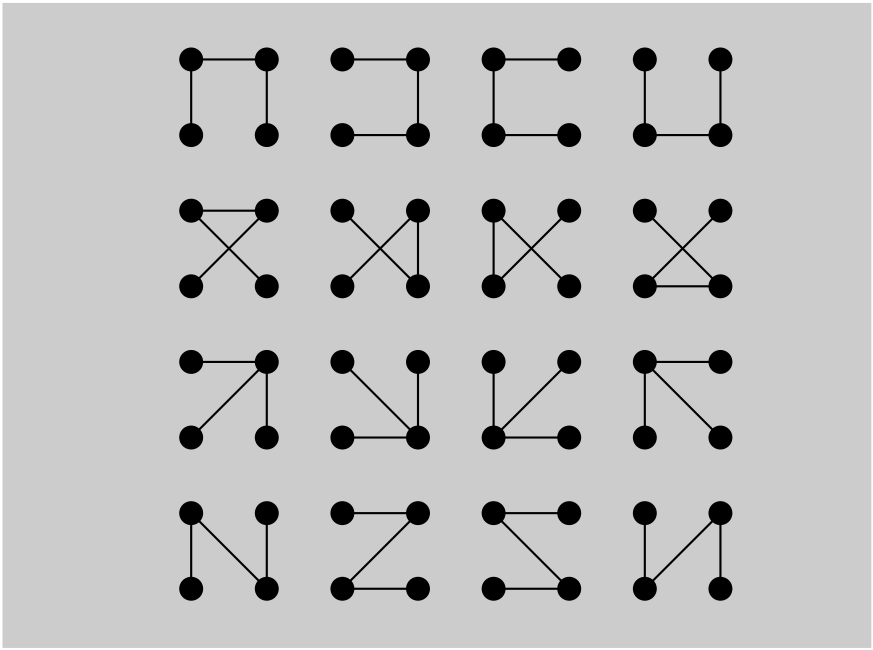


Exemple 6

Determinem tots els arbres generadors, amb independència d'isomorfisme, del graf $G = (V,A)$:



Un arbre generador $T = (V,A')$ és un subgraf generador de G, és a dir, amb el mateix conjunt de vèrtexs, que com a graf és un arbre. En particular, per a ser arbre, és $|A'| = |V| - 1 = 3$. Així, els arbres generadors són els següents:



El resultat següent garanteix l'existència d'arbres generadors en un graf connex.

Proposició 3

Per a un graf $G = (V,A)$ les propietats següents són equivalents:

1) G és connex.

2) G conté un arbre generador.

Demostració: **2)** $\Rightarrow$ **1)**: Sigui $T = (V,A)$ un arbre generador de G, i siguin u, v vèrtexs de G, que també ho són de T. Essent T connex, hi ha un camí que connecta els vèrtexs anteriors en T, però les arestes corresponents són de G i, per tant, és un camí de G; en conseqüència, G és connex.

1) $\Rightarrow$ **2)**: Si G és un arbre, ja hem acabat: l'arbre generador és el graf. En cas contrari, essent connex, ha de contenir algun cicle C. Eliminem una aresta d'aquest cicle, creant un nou graf G_1, que seguirà essent connex (ja que l'eliminació d'una aresta d'un cicle no produeix desconnexió), segueix tenint els mateixos vèrtexs que G (ja que no n'hem eliminat cap) i la mida ha disminuït en una unitat. Si G_1 és arbre, és un arbre generador i hem acabat la demostració. En cas contrari, conté algun cicle, i la demostració segueix de la mateixa manera que abans, amb l'eliminació d'una aresta del cicle. Aquest procés és finit, atès que el nombre d'arestes és finita i, per tant, s'ha d'arribar a un subgraf connex acíclic amb tots els vèrtexs del graf G, és a dir, un arbre generador. ∎

De la definició d'arbre generador i del resultat anterior podem deduir les propietats següents.

Proposició 4

- Si $G = (V,A)$ és connex d'ordre n, aleshores conté un arbre generador d'ordre n i mida $n-1$.
- Si $G = (V,A)$ no és connex, aleshores cada component connex conté un arbre generador, la reunió dels quals és un bosc, el **bosc generador** de G.
- Si T és un arbre generador del graf G, aleshores eliminant una aresta de T l'arbre deixa de ser connex; si, a més, $T \neq G$, afegint una aresta a T (de les arestes de G que no estan a T) T deixa de ser acíclic.

2.1.1. Determinació d'un arbre generador

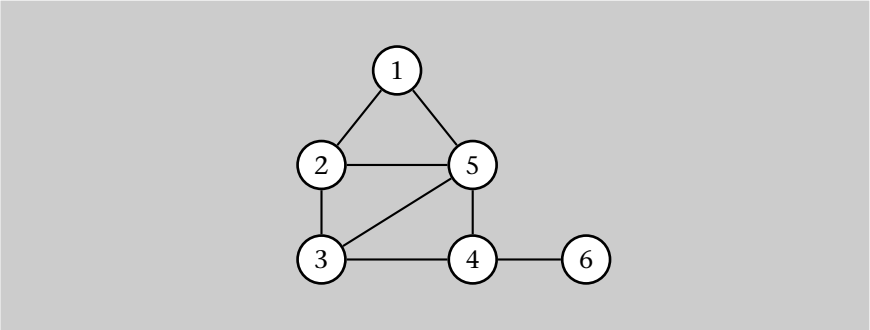
La demostració de la proposició 3 dóna un mètode per a construir arbres generadors: anar eliminant arestes dels cicles del graf. No obstant això, és més eficient utilitzar els algorismes d'exploració de grafs. Així, l'algorisme següent utilitza el *BFS* per a obtenir un arbre generador del graf connex $G = (V,A)$, partint del vèrtex $v \in V$.

```
Entrada : G = (V,A), v ∈ V
Sortida : T, arbre generador de G
algorisme BFS-ArbreGenerador(G,v)
  inici
    Q ← ∅
    V(T) ← V(G)
    A(T) ← ∅
    per w ∈ V(G)
        estat[w] ← 0
    fiper
    estat[v] ← 1
    afegir(Q,v)
    mentre Q ≠ ∅
              w ← primer(Q)
              per u adjacent a w
                  si estat[u] = 0
                     aleshores afegir(Q,u)
                               estat[u] ← 1
                               afegir(A(T),{w,u})
                  fisi
              fiper
              eliminar(Q)
    fimentre
    retorn (T)
  fi
```

Simulació de l'algorisme

Exemple 7

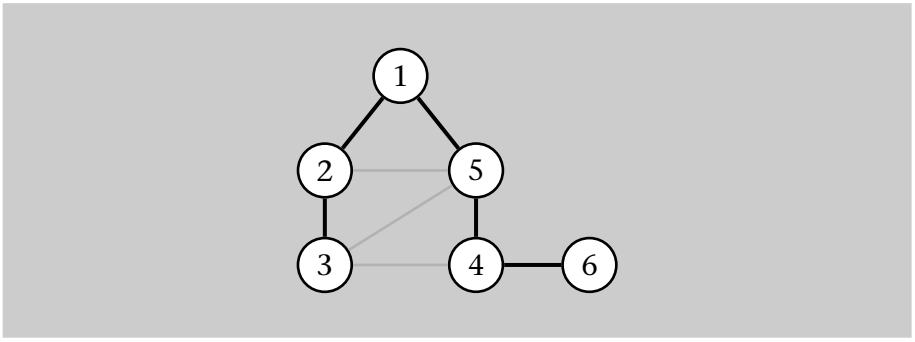
Considerem el graf representat a la figura següent.



Aquesta taula registra el funcionament de l'algorisme per a aquest graf, amb vèrtex d'inici $v = 1$.

Q	Arestes afegides	$A(T)$
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}}

L'arbre generador obtingut es pot veure en el gràfic següent, amb les arestes indicades amb un traç gruixut.



Variant el vèrtex inicial i l'ordre d'elecció de les arestes podem obtenir diferents arbres generadors.

Exercicis

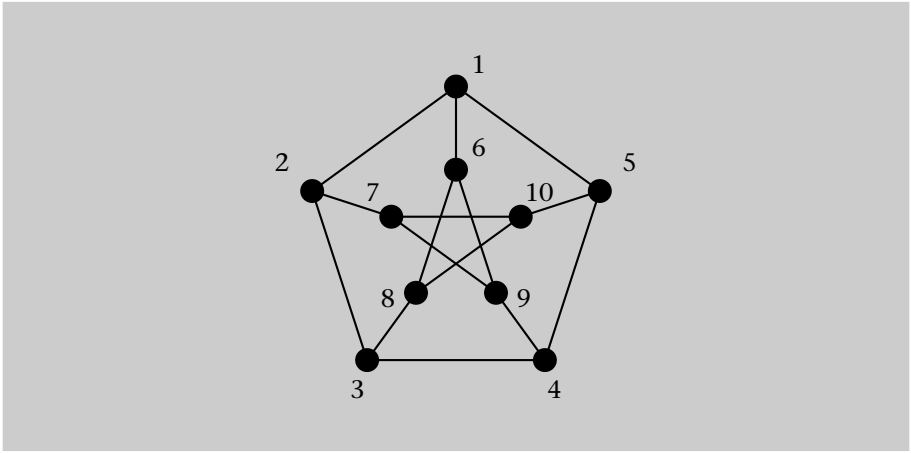
9. Quants arbres generadors té un arbre?

10. En un arbre generador d'un graf d'ordre n, quantes arestes hi ha?

11. Sigui G un graf connex i T un arbre generador de G. Quin és el subgraf induït pel conjunt de vèrtexs $V(T)$?

12. Dissenyeu l'algorisme *DFS-ArbreGenerador*(G,v) que permeti obtenir un arbre generador d'un graf connex $G = (V,A)$. Estudieu-ne la complexitat.

13. Tot utilitzant els algorismes *BFS-ArbreGenerador* i *DFS-ArbreGenerador* trobeu, començant pel vèrtex 1, arbres generadors del graf de Petersen:



Solucions

9. Només un, ell mateix.

10. Hi ha $n-1$ arestes, ja que és un arbre.

11. Tot el graf G.

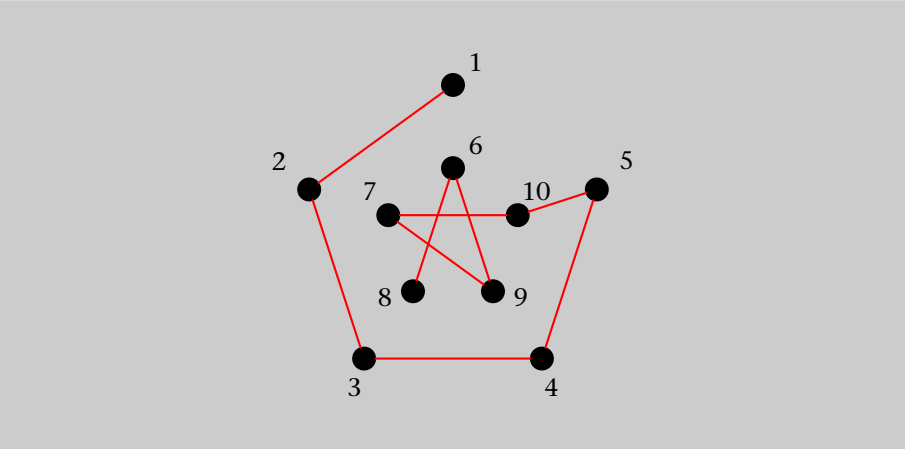
12. Si utilitzem la versió recursiva de l'algorisme *DFS*:

```
Entrada : G = (V,A), v ∈ V
Sortida : T, arbre generador de G
algorisme DFS-ArbreGenerador(G,v)
  inici
    V(T) ← V(G)
    A(T) ← ∅
    per w ∈ V
        estat[w] ← 0
    fiper
    dfsrec(G,v,T,estat)
    retorn (T)
  fi
```

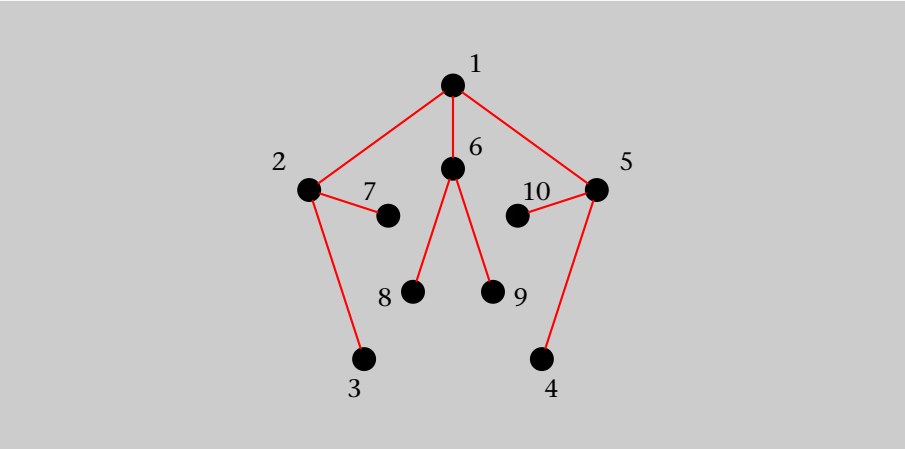
```
funció dfsrec(G,v,T,estat)
  inici
    estat[v] ← 1
    per w adjacent a v
        si estat[w] = 0
          aleshores
                    afegir(A(T),{v,w})
                    dfsrec(G,w,T,estat)
        fisi
    fiper
  fi
```

Tant el *DFS-ArbreGenerador* com el *BFS-ArbreGenerador* utilitzen els mateixos recursos que el *DFS* i *BFS*, respectivament. Per tant, la seva complexitat serà $O(n+m)$.

13. Si fem servir l'algorisme *DFS-ArbreGenerador* obtindríem l'arbre generador següent.



Amb l'algorisme *BFS-ArbreGenerador* obtindríem:



2.2. Arbres generadors minimals

Donat un graf d'interconnexió entre nodes (ciutats, per exemple), les arestes del qual tenen assignades un pes, que pot correspondre a una certa mesura del cost de la comunicació entre els dos nodes, podria ser necessari establir un arbre generador minimal, és a dir, establir un subsistema de comunicacions de tal manera que cap node en quedi exclòs i que globalment representi un cost mínim. Aquest tipus de problemes es poden resoldre amb l'ajut dels arbres generadors minimals.

Definició 4

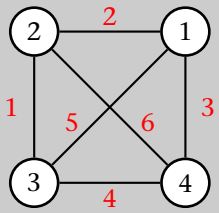
Donat un graf ponderat (G,w) i un arbre generador T de G definim el **pes de l'arbre** T com $w(T) = \sum_{e\in A(T)} w(e)$.

Un **arbre generador minimal** de G és un arbre generador T de G de pes $w(T)$ mínim.

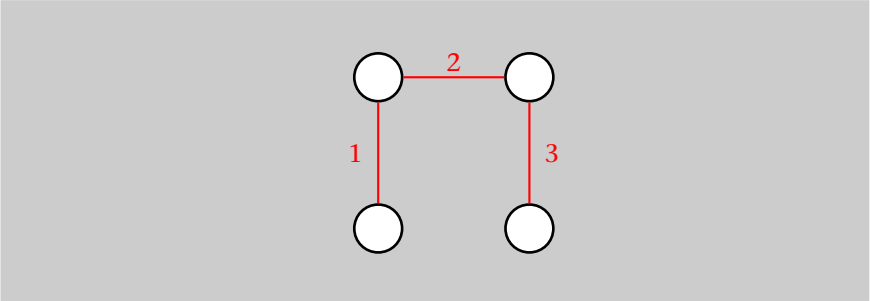
Si G no és connex, aleshores podem parlar del **bosc generador minimal** de G com aquell que té pes mínim entre tots els boscos generadors de G.

Exemple 8

El següent graf ponderat conté setze arbres generadors diferents (vegeu l'exemple 6).



De tots aquests, l'arbre generador minimal és l'arbre següent amb un pes $w(T) = 6$.



Els algorismes de Kruskal i de Prim són dos algorismes clàssics d'obtenció d'arbres generadors minimals que corresponen a una tipologia ben coneguda, la dels algorismes *greedy* o "voraços". Aquests algorismes solen prendre decisions que en cada moment són localment les millors possibles. Això no significa, en general, que la solució obtinguda sigui globalment la millor. Curiosament, això és així en el cas dels algorismes esmentats.

2.2.1. Algorisme de Kruskal

Per a construir l'arbre generador minimal es comença amb un "arbre buit" i durant tot el procés s'incrementa un subgraf amb noves arestes de la manera següent: en cada etapa del procés s'afegeix una aresta que no formi cicle amb les escollides prèviament. Per tal de garantir la minimalitat, es tria, d'entre les possibles, l'aresta de pes mínim (no necessàriament adjacent a alguna aresta prèviament incorporada). El procés acaba quan s'hi han incorporat $n-1$ arestes.

Objectiu de l'algorisme de Kruskal

L'algorisme de Kruskal cerca un arbre generador minimal i tria, en cada pas, l'aresta de menys pes que no formi cicle amb les ja escollides.

Formulació de l'algorisme de Kruskal

Entrada: un graf ponderat connex (G,w) d'ordre n.
Sortida: un arbre generador minimal T de G.
Algorisme:

inici
  $k \leftarrow 1$, $T = (V,A')$, $A' = \emptyset$
  mentre $k \leq n-1$
    Triar l'aresta $a \in A$ de pes mínim, no escollida anteriorment i de manera que el subgraf $T = (V,A' \cup a)$ sigui acíclic.
    Afegir a a A'.
    $k \leftarrow k+1$
  fimentre
  retorn(T)
fi.

Implementació de l'algorisme de Kruskal

Per a implementar l'algorisme de Kruskal cal mantenir una llista ordenada de les arestes del graf en funció del pes i una estructura que permeti comprovar, de manera eficient, que no es formen cicles.

Intuïtivament, l'algorisme manté un bosc. Al principi, hi ha $|V|$ arbres d'un sol vèrtex. Quan afegim una aresta es combinen dos arbres en un. Quan acaba l'algorisme només hi ha un arbre, l'arbre generador minimal.

Estructures necessàries per a la implementació de l'algorisme:

- Un graf ponderat i connex (G,w) representat mitjançant una llista d'adjacències.
- Una llista d'arestes, E, ordenada segons el seu pes.
- Una estructura anomenada **conjunts-disjunts** (*disjoint-sets*, en anglès) de conjunts. Al principi, cada vèrtex forma un conjunt que només conté aquest vèrtex. Quan s'analitza una aresta $\{u,v\}$, es fa una **cerca** per a localitzar el conjunt de u i el de v. Si u i v estan en el mateix conjunt, l'aresta no és acceptada perquè u i v ja estan connectats i afegint l'aresta $\{u,v\}$ formaria un cicle. En cas contrari, l'aresta és acceptada i s'executa la **unió** dels dos conjunts que contenen u i v per a formar un nou arbre.

  Així les dues operacions que cal fer són:

  – *cerca(U,u)*, retorna el representant de l'arbre que conté u.
  – *unió(U,x,y)*, unió dels arbres que tenen com a representants x, y per a formar un nou arbre.
- Un arbre T que contindrà l'arbre generador minimal.

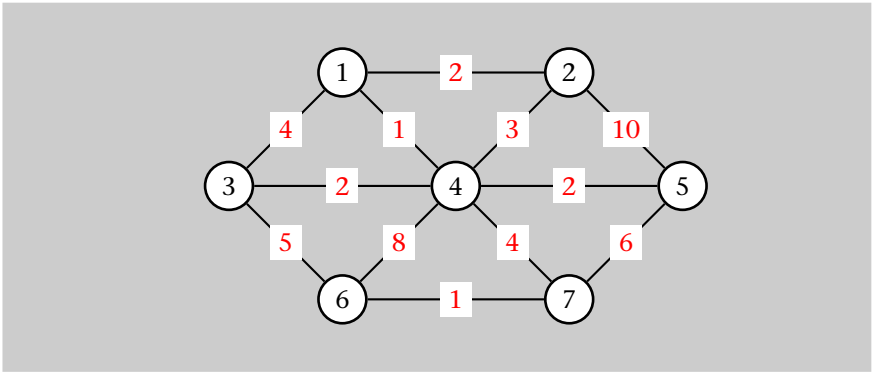
Entrada : (G,w) connex i ponderat d'ordre n
Sortida : T l'arbre generador minimal de G
algorisme *Kruskal*(G)
  inici
    U = estructura conjunts-disjunts
    E = llista d'arestes ordenades per pes en ordre ascendent
    $T \leftarrow (V,A'), A' \leftarrow \emptyset$
    $k \leftarrow 1$
    $i \leftarrow 1$
    mentre $(k < n)$
      Sigui $\{u,v\}$ la aresta $E[i]$
      $x \leftarrow cerca(U,u)$

$y \leftarrow cerca(U,v)$
si $(x \neq y)$
aleshores $A' \leftarrow A' \cup \{u,v\}$
$k \leftarrow k+1$
$unió(U,x,y)$
fisi
$i \leftarrow i+1$
fimentre
retorn (T)
fi

Simulació de l'algorisme de Kruskal

Exemple 9

Considerem el graf definit pel gràfic següent.



Podem utilitzar l'algorisme de Kruskal per a trobar l'arbre generador minimal. En aquest cas només cal fer una llista de les arestes, de menys pes a més pes (en cas d'igualtat, s'escriuen primer les que estan formades per vèrtexs de número més baix).

Arestes	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

Hem de triar les 6 primeres arestes (perquè hi ha 7 vèrtexs) que no formen cap cicle. Les marcarem amb un asterisc; les arestes descartades perquè formen cicle les marcarem en negreta.

Arestes	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}	

Per tant, l'arbre generador minimal estarà format per les arestes: {1,4}, {6,7}, {1,2}, {3,4}, {4,5}, {4,7} amb un pes total 12.

Anàlisi de l'algorisme de Kruskal

En l'algorisme de Kruskal podem distingir dues operacions fonamentals:

1) Ordenació de la llista d'arestes segons el seu pes. Si denotem per m la mida del graf, aleshores podem ordenar una llista de m arestes amb una complexitat $O(m \log m)$, fent servir algorismes de classificació com són el *quicksort* o el *heapsort*.

2) El bucle principal de l'algorisme s'executa $n-1$ vegades i, en cada iteració, fem dues operacions de cerca i una d'unió en l'estructura conjunts-disjunts. Aquestes dues operacions (vegeu els exercicis d'autoavaluació d'aquest mòdul) es poden fer utilitzant un màxim de $\log m$ passos. Com que s'executa $n-1$ vegades, ens donarà una complexitat $O(n \log m)$.

Tot l'algorisme tindrà una complexitat $\max\{O(m \log m), O(n \log m)\}$. Com que, per a un graf connex, $m \geq n-1$ podem concloure que l'algorisme de Kruskal té una complexitat $O(m \log m)$.

2.2.2. Algorisme de Prim

L'algorisme de Prim és similar a l'algorisme de Kruskal, amb la diferència que a cada pas es tria una aresta no incorporada prèviament i que sigui adjacent a alguna de les ja incorporades. A més, no forma cicle amb les anteriors i d'entre les arestes disponibles que satisfacin aquestes condicions és la que té pes mínim. Si n'hi ha més d'una s'agafa la primera en l'ordenació d'arestes. D'aquesta manera, en tot moment es manté un subgraf connex i acíclic. El procés s'acaba quan s'han incorporat $n-1$ arestes.

Objectiu de l'algorisme de Prim

L'algorisme de Prim cerca un arbre generador minimal i tria, en cada pas, l'aresta de menys pes de les adjuntes als vèrtexs ja escollits. D'aquesta manera el graf construït sempre és connex.

Formulació de l'algorisme de Prim

Entrada: un graf ponderat connex (G,w) d'ordre n.

Sortida: un arbre generador minimal T de G.

Algorisme:

inici
$k \leftarrow 1$, $T = (V,A')$, $A' = \emptyset$
mentre $k \leq n-1$
Triar l'aresta $a \in A$ de pes mínim, no escollida anteriorment,
adjacent a alguna aresta de A' i
tal que el subgraf $T = (V,A' \cup a)$ sigui acíclic.
Afegir a a A'.
$k \leftarrow k+1$
fimentre
retorn(T)
fi.

Implementació de l'algorisme de Prim

L'algorisme de Prim construeix l'arbre generador fent créixer un sol arbre en passos successius. Es comença escollint qualsevol vèrtex com a primer vèrtex i a cada pas afegim l'aresta de pes mínim que connecta un vèrtex de l'arbre amb un de fora.

La implementació de l'algorisme de Prim és essencialment idèntica a la de l'algorisme de Dijkstra per a trobar la distància entre dos vèrtexs. Només cal modificar la regla d'actualització.

Estructures necessàries per a la implementació de l'algorisme:

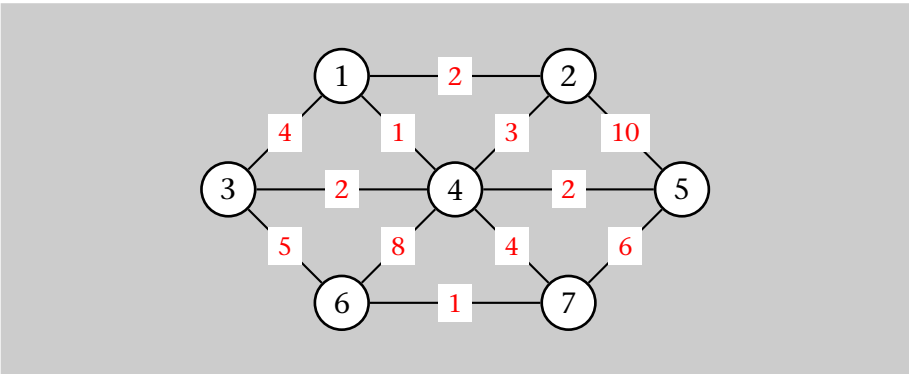
- Un graf ponderat (G,w) representat mitjançant una llista d'adjacències.
- Un conjunt U dels vèrtexs que s'han visitat, en l'ordre en què s'ha fet.
- Una taula de pesos, $E(\cdot)$, indexada pels vèrtexs de G, que registra el pes de l'aresta de pes mínim que connecta un vèrtex v amb un vèrtex ja visitat.
- Al final, la taula $E(\cdot)$ registra els pesos de les arestes que formen part de l'arbre generador minimal.

A cada pas es fixa l'etiqueta d'un dels vèrtexs del graf. Així, després de n passos haurem calculat l'arbre generador minimal.

Entrada : (G,w) d'ordre n.
Sortida : T l'arbre generador minimal de G
algorisme *Prim*(G)
  inici
    Seleccionem un vèrtex inicial $u_0 \in V$
    $U \leftarrow \emptyset$
    per $v \in V \backslash \{u_0\}$
        $E(v) \leftarrow \infty$
        Etiquetem v amb $(E(v),u_0)$
    fiper
    $E(u_0) \leftarrow 0$
    Etiquetem u_0 amb $(0,u_0)$
    $T \leftarrow (V,A'), A' \leftarrow \emptyset$
    per $i \leftarrow 1$ **fins** n
        u_i vèrtex que assoleix el $\min\{E(v) \mid v \in V - U\}$
        $U \leftarrow U \cup \{u_i\}$
        $A' \leftarrow A' \cup \{x,u_i\}$ /*on $(E(u_i),x)$ és l'etiqueta d'u_i*/
        per $v \in V - U$ adjacent a u_i
            si $w(u_i,v) < E(v)$
              aleshores $E(v) \leftarrow w(u_i,v)$
                  Etiquetem v amb $(E(v),u_i)$
            fisi
        fiper
    fiper
    retorn (T)
  fi

Simulació de l'algorisme de Prim

Considerem el graf definit pel gràfic següent.



Si agafem com a vèrtex inicial el vèrtex 1, la taula següent representa l'estat inicial de l'algorisme:

Vèrtexs	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 aquesta taula, hem representat el conjunt U i l'etiquetatge dels vèrtexs. Tots els vèrtexs, excepte el vèrtex 1, estan etiquetats com a $(\infty,1)$, cosa que indica que el pes mínim inicial és ∞ (i, per tant, que, en l'estat inicial, el vèrtex no s'assoleix).

Els diferents passos de l'algorisme seran:

$i = 1$

Vèrtexs	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)**	$(\infty,1)$	$(\infty,1)$	$(\infty,1)$

$i = 2$

Vèrtexs	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)**

$i = 3$

Vèrtexs	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)

$i = 4$

Vèrtexs	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)

$i = 5$

Vèrtexs	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)

$i = 6$

Vèrtexs	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)

$i = 7$

Vèrtexs	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)

D'aquesta darrera taula es dedueix que l'arbre generador minimal estarà format per les arestes {1,2}, {3,4}, {1,4}, {4,5}, {6,7} i {4,7} amb un pes mínim total 12.

Es pot construir la **taula de l'algorisme de Prim** a partir de les files $(E(\cdot),\circ)$ de cadascun dels passos (s'assenyala amb un asterisc el vèrtex que es 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)

Anàlisi de l'algorisme de Prim

L'algorisme de Prim té la mateixa estructura que l'algorisme de Dijkstra i, per tant, tindrà la mateixa complexitat: $O(n^2)$, independentment del nombre d'arestes del graf.

Exercicis

14. Si G és un graf connex no ponderat d'ordre n, quin és el pes de l'arbre generador minimal de G?

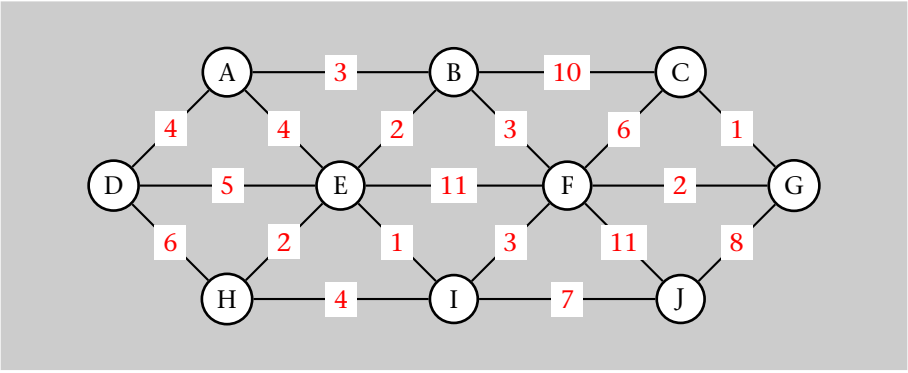
15. Si G és un graf no ponderat d'ordre n, quin és el pes del bosc generador minimal de G?

16. Pot utilitzar-se l'algorisme de Kruskal o el de Prim en un graf connex no necessàriament ponderat per tal d'obtenir un arbre generador?

17. Els grafs intermedis que es construeixen a l'algorisme de Kruskal, són arbres?

18. Considereu l'afirmació: "l'algorisme de Kruskal permet obtenir l'arbre generador minimal d'un graf connex ponderat". Creieu que és exacta?

19. Trobeu l'arbre generador minimal del graf següent fent servir l'algorisme de Kruskal i l'algorisme de Prim. Quin és el pes mínim de l'arbre? És únic, l'arbre generador minimal? Justifiqueu la resposta.



20. Si en l'algorisme de Prim modifiquem la condició:

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

canviant $<$ per $\leq$, com repercutirà en el funcionament de l'algorisme? S'obtindrà el mateix arbre generador minimal?

Solucions

14. El pes és $n-1$.

15. Si G conté k ($k \geq 1$) components, aleshores el bosc generador minimal tindrà pes $n-k$.

16. Sí, assignant pesos unitaris a totes les arestes.

17. No, en general no són necessàriament arbres. Són boscos, ja que l'únic que podem assegurar és que són acíclics.

18. No, n'hi pot haver més d'un.

19. En l'algorisme de Kruskal ordenem totes les arestes segons els seu pes (en cas d'igualtat, considerarem l'ordre de la llista d'adjacències):

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

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

Ara triem les $10-1=9$ arestes de pes mínim que no formen cicle. Assenyalem amb un asterisc les arestes triades i en negreta, les rebutjades.

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

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

L'arbre generador minimal obtingut té un pes total 25.

Ara repetirem el problema fent servir l'algorisme de Prim. La taula següent resumeix la construcció de l'arbre començant pel vèrtex 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)*

Observeu que l'arbre coincideix amb el que hem obtingut amb l'algorisme de Kruskal.

Encara que en l'algorisme de Prim sempre hem triat l'única opció possible, l'arbre generador minimal no és únic. De fet, podríem substituir l'aresta {B,F} amb pes 3 per l'aresta {F,I}, amb el mateix pes, i obtindríem un arbre generador minimal diferent. Observeu que en l'algorisme de Kruskal sí que podíem haver triat l'aresta {F,I} abans que la {B,F} si haguéssim triat un altre criteri d'ordenació de les arestes amb el mateix pes.

20. L'algorisme continua funcionant correctament i obtenim un arbre generador minimal, tot i que pot ser diferent del que s'havia obtingut en la versió original. De fet, d'entre les arestes del mateix pes, s'escollirà la que s'examina més tard.

3. Arbres amb arrel

Els grafs dirigits asimètrics, concepte que definirem en aquest apartat, permeten caracteritzar els arbres amb arrel. Alguns aspectes d'aquests arbres són molt interessants a l'hora d'avaluar la complexitat de certs algorismes, especialment els d'ordenació.

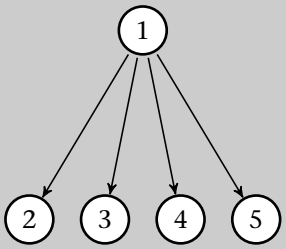
3.1. Caracterització dels arbres amb arrel

Definició 5

Donat un arc (u,v) d'un digraf, $G = (V,A)$ es diu que u és **pare** del vèrtex v i que v és **fill** del vèrtex u.

Exemple 10

En el digraf adjunt, el vèrtex 1 és pare dels vèrtexs 2, 3, 4 i 5; i aquests són fills del vèrtex 1.



Definició 6

Un digraf $G = (V,A)$ és **asimètric** si no té cicles de longitud 2. És a dir, si $(u,v) \in A$, aleshores $(v,u) \notin A$.

Un vèrtex r d'un digraf $G = (V,A)$ és una **arrel** si hi ha un $r - v$ camí (dirigit) per a tot altre vèrtex v del digraf.

Definició 7

Un digraf $T = (V,A)$ és un **arbre amb arrel** r si:

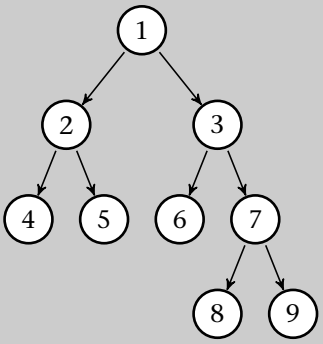
1) T és un digraf asimètric.

2) El graf subjacent és un arbre.

3) Per a tot altre vèrtex $v \in V$ hi ha un $r - v$ camí.

Exemple 11

El graf següent és un arbre amb arrel.

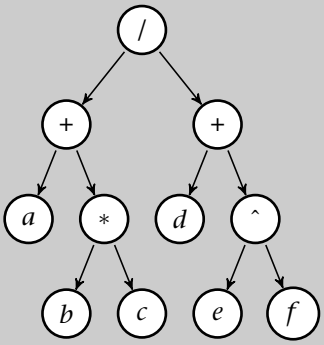


L'arrel és el vèrtex etiquetat 1. Els arcs són (1,2), (1,3), (2,4), (2,5), (3,6), (3,7), (7,8) i (7,9).

Els arbres amb arrel sempre es representen en forma "jeràrquica", posant l'arrel al capdamunt. Això fa que sovint no calgui indicar l'orientació dels arcs, ja que se sobreentén que és en "sentit descendent".

Exemple 12

Una aplicació important dels arbres amb arrel és la representació d'expressions aritmètiques com ara $(a + b * c)/(d + e\hat{}f)$. Si una expressió simple com $\alpha \circ \beta$ es desa com un arbre amb arrel $\circ$ i fills α i β, aleshores l'expressió anterior, d'acord amb la precedència dels operadors $+, *, /, \hat{}$ es representarà com:



Aquesta representació és especialment útil en el procés de compilació de programes, ja que permet avaluar eficientment qualsevol expressió aritmètica.

Vegeu també

En el mòdul "Fonaments de grafs", hem definit $g^-(v)$ com el nombre d'arcs que tenen el vèrtex v com a destí.

Teorema 5

Sigui $T = (V,A)$ un digraf asimètric. Aleshores les afirmacions següents són equivalents:

1) T és un arbre amb arrel.

2) T té una arrel r amb $g^-(r) = 0$ i $g^-(v) = 1$ per a tot vèrtex diferent de l'arrel.

3) El graf subjacent $\overline{T}$ és connex i existeix un vèrtex r de manera que $g^-(r) = 0$ i $g^-(v) = 1$ per a tot vèrtex diferent de r.

Demostració: **1)** $\Rightarrow$ **2)**: Si $g^-(r) \neq 0$, aleshores hi hauria un vèrtex $v \neq r$ i un arc (v,r). Com que també hi ha un $r-v$ camí, aleshores tindrem un cicle dirigit a T, contra la hipòtesi que T és asimètric i el graf subjacent és un arbre. De manera semblant es demostra que $g^-(v) = 1$.

2) $\Rightarrow$ **3)**: Només cal veure que el graf subjacent és connex. Si T té una arrel r, aleshores hi ha un $r-v$ camí dirigit per a tot altre vèrtex v de T. Per tant, el graf subjacent serà connex.

3) $\Rightarrow$ **1)**: És clar que T ha de ser asimètric. Com que $\overline{T}$ és connex, hi ha un camí (a $\overline{T}$) entre r i v. Si aquest camí no fos un $r-v$ camí dirigit, aleshores hi hauria un vèrtex del camí amb $g^-(v) > 1$. ∎

Exercicis

21. Un arbre amb arrel, pot tenir més d'una arrel?

22. En un arbre amb arrel, és únic el camí que uneix l'arrel amb cada vèrtex?

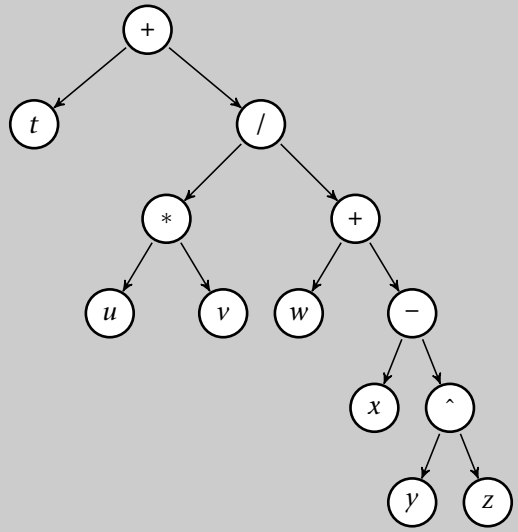
23. Dibuixeu l'arbre amb arrel de l'expressió aritmètica: $t + u * v/(w + x - y\hat{}z)$.

Solucions

21. No; si hi hagués dues arrels r, r', aleshores hi hauria un camí orientat $r - r'$ i $g^-(r') = 1$, fet que és absurd, ja que contradiu la segona propietat del teorema 5.

22. Sí. Si n'hi hagués més d'un, aleshores el graf subjacent contindria un cicle.

23. En aquest cas, i d'acord amb la precedència dels operadors, l'arrel és el signe +. L'arbre resultant serà:



3.2. Arbres m-aris

En aquest subapartat se suposa que $T = (V,A)$ és un arbre amb arrel r. Vegem primer algunes definicions terminològiques.

Definició 8

Una **fulla** és un vèrtex que no té fills, és a dir, amb grau de sortida 0; també s'anomena **vèrtex terminal**. Els altres vèrtexs s'anomenen **interns** (o **no terminals**).

Definició 9

El **nivell** d'un vèrtex v de T és la longitud de l'únic $r-v$ camí.

Definició 10

L'**altura** (o **profunditat**) de l'arbre T és el màxim dels nivells:

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

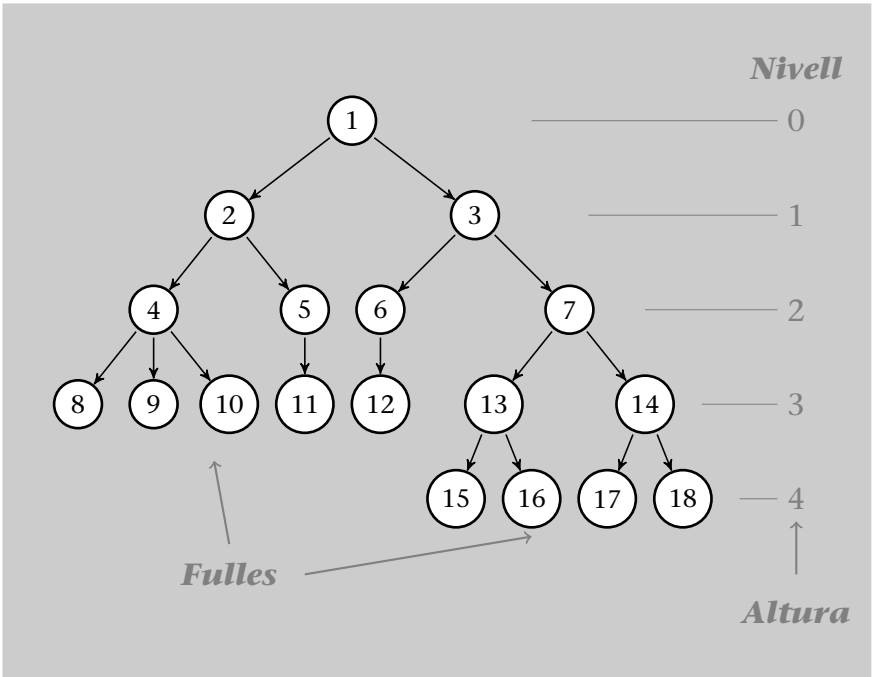
Si el nivell de cada fulla de T és igual a $h(T)$ o a $h(T)-1$, direm que T és un **arbre equilibrat**.

Vegeu també

El concepte d'arbre equilibrat és molt important en l'emmagatzematge eficient d'informació en una base de dades o una estructura de dades en memòria. En podeu veure més detalls a l'assignatura *Disseny d'Estructures de Dades*.

Exemple 13

En l'arbre següent els vèrtexs 8, 9, 10, 11, 12, 15, 16, 17, 18 són fulles. La resta són nodes interns.



Cada vèrtex és en un nivell.

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

Per tant, l'altura de l'arbre (el màxim dels nivells) és 4. A més, és un arbre equilibrat perquè totes les fulles són en els dos darrers nivells.

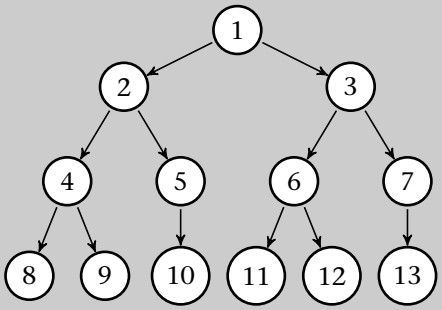
Definició 11

Un arbre amb arrel $T = (V,A)$ és m-**ari** ($m \geq 2$) si $0 \leq g^+(v) \leq m$, $\forall v \in V$. En particular, si $m = 2$ direm que tenim un **arbre binari**.

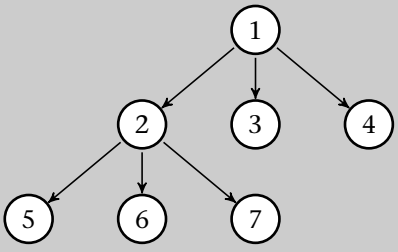
Un arbre m-ari $T = (V,A)$ és **complet** si $\forall v \in V$, $g^+(v) = m$ o $g^+(v) = 0$, és a dir, si els vèrtexs interns tenen grau de sortida m i les fulles tenen grau de sortida 0.

Exemple 14

Aquest arbre amb arrel és un arbre binari (cada vèrtex té dos fills com a màxim) no complet (per exemple, el vèrtex etiquetat amb un 5 és un vèrtex intern que no té dos fills):



En canvi, l'arbre de la figura següent és un arbre 3-ari o ternari i complet.



Observeu que els arbres amb arrel d'una expressió aritmètica (vegeu l'exemple 12) amb operadors binaris són sempre arbres binaris complets.

Hi ha relacions importants que s'utilitzen en diverses situacions en l'anàlisi d'algorismes.

Proposició 6

Sigui $T = (V,A)$ un arbre m-ari,

1) Si T és complet, conté t fulles i i vèrtexs interns, aleshores $t = (m-1)i + 1$.

2) Si T té altura h i conté t fulles, aleshores $t \leq m^h$. És a dir, hi ha un màxim de m^h fulles en un arbre m-ari d'altura h.

3) Si $m \geq 2$, l'altura h i amb t fulles, aleshores $h \geq \lceil \log_m t \rceil$ (on $\lceil x \rceil$ és el mínim nombre enter més gran o igual que x). En el cas que T sigui complet i equilibrat, $h = \log_m t$.

Demostració: Si denotem per n l'ordre de T, aleshores el primer resultat és conseqüència del fet que $t + i = n = mi + 1$.

El segon resultat és conseqüència del fet que el nombre de vèrtexs de nivell k ($0 \leq k \leq h$) és menor o igual a m^k.

Finalment, si T és complet i equilibrat, aleshores el nivell $h-1$ té m^{h-1} vèrtexs, dels quals no tots són fulles. Per tant, $m^{h-1} < t$. D'altra banda, ja sabem que $t \leq m^h$. De les dues desigualtats obtenim

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

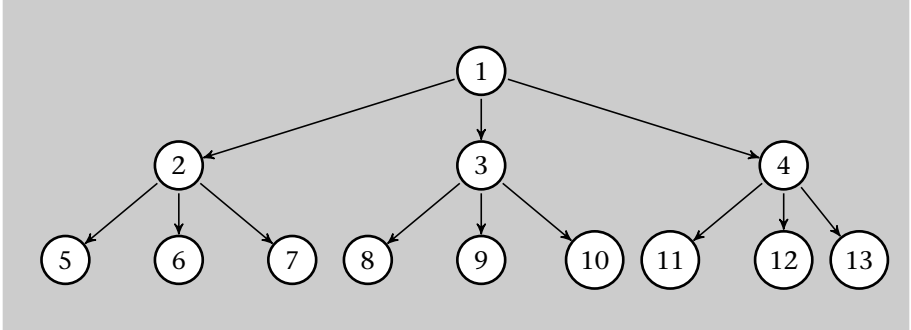
que és equivalent a

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

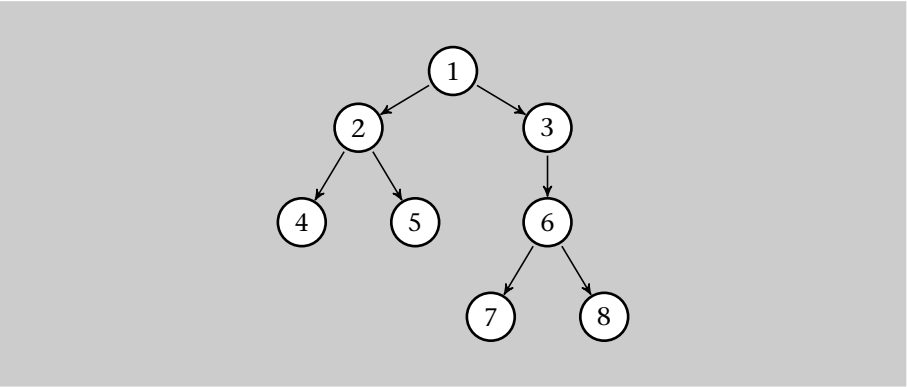
Per la definició de la funció $\lceil x \rceil$ tindrem que $h = \lceil \log_m t \rceil$. Si no fos complet, o bé no fos equilibrat, és evident que $h \geq \lceil \log_m t \rceil$. ∎

Exercicis

24. Considereu l'arbre amb arrel següent; indiqueu quina n'és l'altura i si és equilibrat o no.



25. Considereu l'arbre amb arrel següent. Indiqueu quines són les fulles, els nivells dels vèrtexs i l'altura de l'arbre:



26. Quina és l'altura mínima d'un arbre 4-ari amb 127 fulles?

27. En un campionat de tennis individual participen vint-i-set jugadors que competeixen en la modalitat d'eliminatòries. Quin és el nombre de partits que cal jugar per a determinar el campió?

Solucions

24. És un arbre ternari d'altura 2 i equilibrat.

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

Nivells dels vèrtexs:

Vèrtexs de nivell 0: 1
Vèrtexs de nivell 1: 2, 3
Vèrtexs de nivell 2: 4, 5, 6
Vèrtexs de nivell 3: 7, 8

Per tant, l'altura serà 3.

26. El nombre de fulles t d'un arbre m-ari, d'altura h compleix la condició $t \leq m^h$. Si substituïm, $127 \leq 4^h$, d'on $h \geq \lceil \log_4 127 \rceil = 4$.

27. Els vint-i-set jugadors es poden veure com les fulles d'un arbre binari. L'arrel de l'arbre serà la final. El nombre de vèrtexs interns és el nombre de partits que cal fer. Així, caldrà organitzar un total de $i = (t-1)/(m-1) = 26/1 = 26$ partits.

3.3. Algorismes d'exploració dels arbres binaris amb arrel

Si considerem una relació d'ordre entre els fills de cada vèrtex intern d'un arbre amb arrel, aleshores ens podem plantejar el problema d'explorar de manera exhaustiva tots els seus vèrtexs.

En el cas concret dels arbres binaris, hi ha tres algorismes bàsics per a explorar-ne tots els vèrtexs: els anomenats *recorregut en preordre*, *recorregut en inordre* i *recorregut en postordre*. Els tres recorreguts es distingeixen bàsicament per la manera en què s'exploren els dos fills de cada vèrtex. Abans d'estudiar-los, cal introduir la notació següent: T és l'arbre binari d'arrel r, T_1 i T_2 són els subarbres d'arrels v_1 i v_2 induïts pels fills de l'arrel r.

- Recorregut en preordre (o en profunditat) de T:

  1) Explorar l'arrel r.

  2) Explorar el subarbre T_1 en preordre.

  3) Explorar el subarbre T_2 en preordre.
- Recorregut en inordre de T:

  1) Explorar el subarbre T_1 en inordre.

  2) Explorar l'arrel r.

  3) Explorar el subarbre T_2 en inordre.
- Recorregut en postordre de T:

  1) Explorar el subarbre T_1 en postordre.

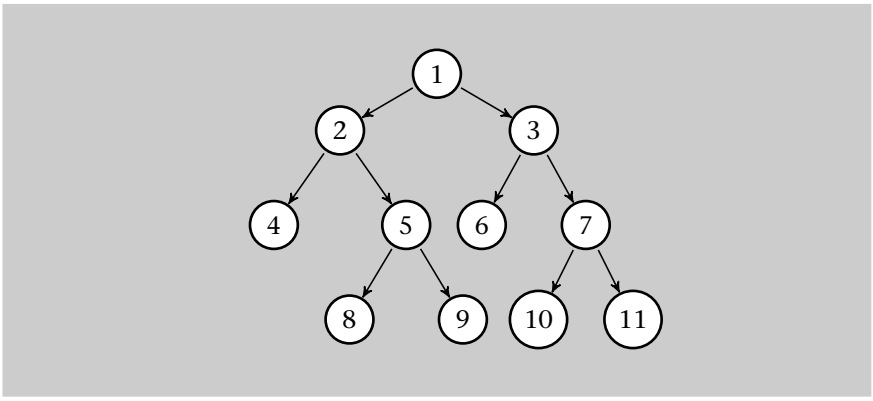
  2) Explorar el subarbre T_2 en postordre.

  3) Explorar l'arrel r.

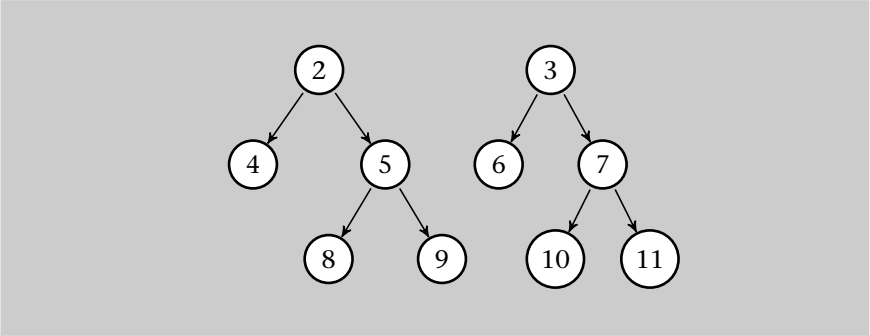
Aquests algorismes visiten cada vèrtex una sola vegada i tenen una complexitat que és funció lineal de l'ordre de l'arbre, és a dir, tenen una complexitat $O(n)$.

Exemple 15

L'arbre de la figura següent és un arbre binari.



L'arrel és el vèrtex etiquetat amb un 1. L'arrel té dos fills, etiquetats 2 i 3, que són arrels de dos subarbres:



De manera recursiva també podríem considerar els subarbres d'arrel 4, 5, 6, ...

Els diferents recorreguts d'aquest arbre són:

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

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

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

Exercicis

28. La llista següent és la llista d'adjacències d'un arbre binari. El símbol "–" representa l'absència de fill. Trobeu els recorreguts en preordre, inordre i postordre d'aquest arbre:

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

La primera posició de la llista representa el fill situat a l'esquerra de l'arrel, i la segona posició, el fill situat a la dreta.

29. Trobeu els recorreguts en preordre, inordre i postordre de l'arbre de l'expressió aritmètica $(a+b*c)/(d+e\hat{}f)$.

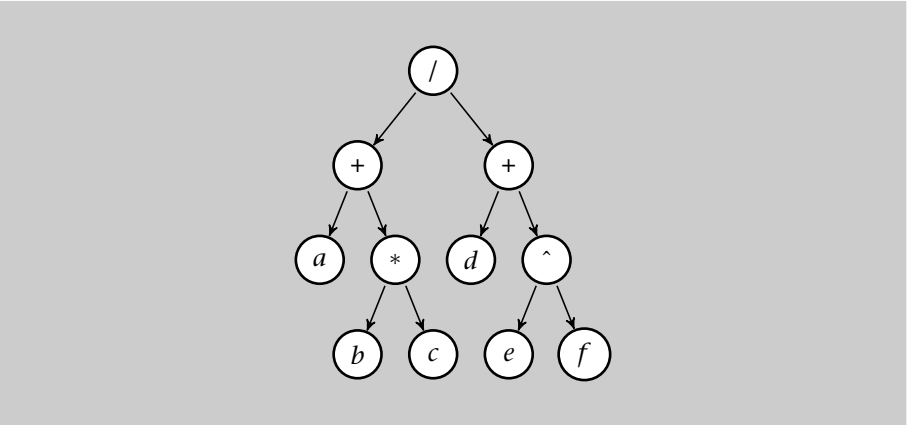
30. Construïu un arbre binari T el recorregut en preordre del qual és a,b,d,e,c,f,h,i,g,j,k i el recorregut en postordre és d,e,b,h,i,f,j,k,g,c,a.

Solucions

28. Els tres recorreguts són:

Preordre: A, B, C, D, E, G, F
Inordre: B, D, E, C, G, A, F
Postordre: E, D, G, C, B, F, A

29. L'arbre d'aquesta expressió és



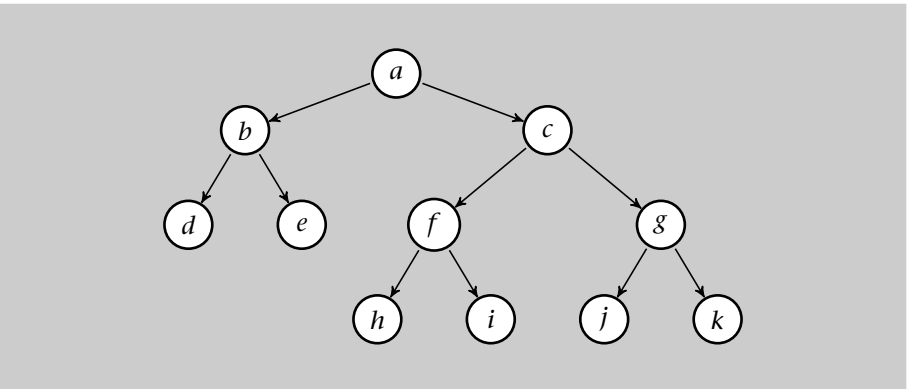
i els tres recorreguts són:

Preordre: / + *a* ∗ *b c* + *d* ^ *e f*
Inordre: *a* + *b* ∗ *c* / *d* + *e* ^ *f*
Postordre: *a b c* ∗ + *d e f* ^ + /

Per a les expressions aritmètiques, el recorregut en preordre dóna lloc a un tipus de representació de l'expressió anomenat *notació prefixa* (o *notació polonesa*, introduïda pel lògic Jan Lukasiewicz). El recorregut en inordre dóna lloc a la *notació infixa* (que coincideix amb la convencional sense parèntesis). Finalment, el recorregut en postordre dóna lloc a la *notació postfixa* (o també *polonesa inversa*).

Les notacions prefixa i postfixa són especialment adequades per a avaluar expressions aritmètiques. Els compiladors són els encarregats de transformar les expressions convencionals a la notació prefixa o postfixa, segons els casos. També s'encarreguen d'avaluar de manera eficient les expressions a partir d'aquestes representacions amb l'ajuda d'una pila.

30. L'arbre que es demana és el següent.



Exercicis d'autoavaluació

1. Sigui $T = (V,A)$ un arbre d'ordre n amb només vèrtexs de grau 1 i de grau 3. Doneu el nombre de fulles i de vèrtexs de grau 3 en funció de l'ordre.

2. Demostreu que un graf acíclic $G = (V,A)$ amb el grau de tots els vèrtexs parell és el graf nul.

3. Quants arbres generadors, amb independència d'isomorfismes, tenen els grafs K_2, K_3 i K_4? Intenteu trobar una fórmula que permeti calcular el nombre d'arbres generadors del graf complet d'ordre n, K_n.

4. Indiqueu com es pot cercar un bosc generador d'un graf no connex a partir dels algorismes *BFS* i *DFS*.

5. El nou govern de l'arxipèlag de Sealand ha decidit unir les seves sis illes mitjançant ponts que permetin connectar-les directament. El cost de construcció d'un pont depèn de la distància entre les illes. Aquesta taula recull les distàncies entre illes:

	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 govern vol construir un sistema de ponts de manera que el cost total de l'obra sigui mínim.

a) Si fem servir la teoria de grafs, trobeu els ponts que s'han de construir de manera que el cost total de l'obra sigui mínim.

b) Suposem que la capital és a l'illa B i que només es pot construir un pont entre dues illes si prèviament alguna d'aquestes ja es comunica amb la capital (la maquinària s'ha de transportar a través dels ponts). En quin ordre s'haurien de construir els ponts?

c) Justifiqueu perquè no és aconsellable utilitzar l'algorisme de Kruskal per a resoldre aquest segon problema.

6. Per a implementar l'algorisme de Kruskal cal comprovar que, quan s'afegeix una aresta, no forma cicle. Una manera eficient de comprovar-ho és mitjançant l'estructura anomenada conjunts-disjunts. Aquesta estructura es pot representar mitjançant una taula U indexada pels vèrtexs del graf i iniciada en el –1. Inicialment, tots els vèrtexs són conjunts disjunts. Sobre aquesta taula es fan dues operacions:

- *Unió*: es combinen dos conjunts (dos arbres) si tenen una aresta en comú.
- *Cerca*: retorna un representant (l'arrel de l'arbre) del conjunt que conté un vèrtex del graf.

Aquestes dues operacions es poden implementar de la manera següent:

```
funció unió(U,v1,v2)
  inici
    si (U[v1] < U[v2])
      aleshores U[v2] ← v1
          si_no
              si (U[v1] = U[v2])
                aleshores U[v1] ← U[v2] – 1
              fisi
              U[v2] ← v1
    fisi
  fi
```

```
funció cerca(U,v)
  inici
    si (U[v] < 0)
      aleshores retorn (v)
          si_no
              U[v] ← cerca(U,U[v])
              retorn (U[v])
    fisi
  fi
```

Si fem servir el graf de l'exemple 9, doneu el contingut de la taula U durant l'aplicació de l'algorisme de Kruskal a aquest graf. Notem que la taula inicial serà:

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

7. Donat un graf ponderat (G,w), descriviu un algorisme per a trobar un arbre generador maximal. És a dir, un arbre generador T de G tal que el seu pes $w(T)$ sigui màxim. Estudieu l'eficiència del l'algorisme proposat.

8. Els algorismes de Kruskal i de Prim permeten obtenir l'arbre generador minimal d'un graf connex i ponderat. Compareu, segons l'eficiència de cada algorisme, quin dels dos és més adequat per a trobar l'arbre generador minimal d'un graf.

9. Donat un arbre 5-ari complet amb dos-cents vèrtexs interns, calculeu quants vèrtexs té.

10. Sigui $T = (V,A)$ un arbre binari complet d'ordre $n = |V| \geq 3$. Demostreu que l'arbre subjacent té $\frac{n+1}{2}$ fulles (vèrtexs de grau 1).

11. Tenim 8 monedes idèntiques, de les quals sabem que una és falsa (pesa més que les altres). Si només disposem d'una balança, quin és el nombre mínim de pesades que cal fer per a poder garantir que descobrim la moneda falsa?

12. La seqüència $* a + b * c + d e$ és un recorregut de l'arbre d'una expressió aritmètica. Trobeu l'expressió original.

Solucionari

1. Indiquem per x_1 el nombre de fulles i per x_3 el nombre de vèrtexs de grau 3. D'una banda, tenim $n = x_1 + x_3$. D'altra banda, pel fet de ser arbre, és $|A| = n-1$ i, finalment, aplicant el lema de les encaixades, podem escriure

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

i obtenir així la relació $x_3 = x_1 - 2$, a partir de la qual resulta:

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

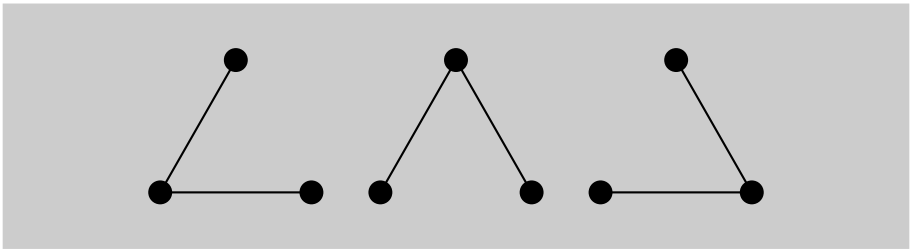
d'on $x_1 = \frac{n+2}{2}$ i finalment $x_3 = \frac{n-2}{2}$.

2. Sigui n l'ordre del graf G.

Si el graf és connex, aleshores, per l'aciclicitat, és un arbre i és ben conegut que tot arbre amb un mínim de dos vèrtexs té un mínim de dues fulles o vèrtexs de grau 1, cosa que en aquest cas és impossible per la hipòtesi. Per tant, $n < 2$ i, en conseqüència, $G = N_1$.

Si el graf no és connex, aleshores el graf és unió de components connexos, o sigui, $G = G_1 \cup \ldots \cup G_k$, i cadascun d'aquests components connexos és un graf connex acíclic amb tots els graus d'ordre parell. Podem aplicar a cada component connex el resultat anterior i, d'aquesta manera, cada component és isomorf al graf nul N_1. Finalment, per tant, $G = N_1 \cup \ldots \cup N_1 = N_k$.

3. K_2 és el graf trajecte T_2 i, per tant, té un sol arbre generador. K_3 té tres arbres generadors tal com mostra el gràfic següent:



K_4 té setze arbres generadors tal com podem veure en l'exemple 6.

En general, per al graf K_n hi ha n^{n-2} arbres generadors diferents.

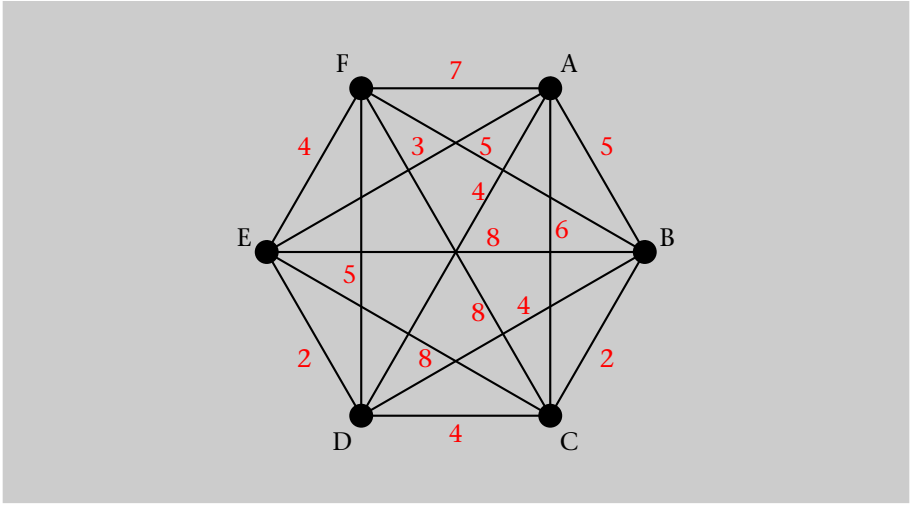
4. Si G no és connex aleshores és unió de components connexos $G = G_1 \cup \ldots \cup G_k$. Per a construir un bosc generador n'hi haurà prou de cercar un arbre generador T_i per a cada component connex G_i i considerar el bosc $T_1 \cup \ldots \cup T_k$.

L'algorisme següent utilitza l'algorisme *BFS-ArbreGenerador*, encara que també es podria utilitzar l'algorisme *DFS-ArbreGenerador*.

Entrada : $G = (V,A)$ d'ordre n.
Sortida : T bosc generador de G
algorisme *BFS-BoscGenerador*(G)
  inici
    $T \leftarrow \emptyset$
    $U \leftarrow V$
    mentre $U \neq \emptyset$
      $v \leftarrow$ vèrtex qualsevol d'U
      $R \leftarrow$ *BFS-ArbreGenerador*(G,v)
      $T \leftarrow T \cup R$
      $U \leftarrow U - V(R)$
    fimentre
    retorn (T)
  fi

Observeu que la complexitat és la del *BFS*, $O(n+m)$, ja que explorem tots els vèrtexs i les arestes de G.

5. El gràfic següent mostra la forma del graf.



a) El problema ens demana cercar l'arbre generador minimal d'aquest graf. Podem utilitzar indistintament l'algorisme de Kruskal o el de Prim.

Si utilitzem l'algorisme de Kruskal caldrà ordenar totes les arestes segons el seu cost:

Arestes	{B,C}	{D,E}	{A,E}	{A,D}	{B,D}	{C,D}	{E,F}	{A,B}	{B,F}
Cost	2	2	3	4	4	4	4	5	5

Arestes	{D,F}	{A,C}	{A,F}	{B,E}	{C,E}	{C,F}
Cost	5	6	7	8	8	8

Tot seguit, cal triar cinc arestes de cost mínim i que no formin cicle. Aquestes són: {B,C}, {D,E}, {A,E}, {B,D} i {E,F}, amb un cost mínim total de 15.

Si utilitzem l'algorisme de Prim amb vèrtex 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 taula es dedueix que cal construir els ponts {A,E}, {B,C}, {B,D}, {D,E} i {F,E}, amb un cost total de 15.

b) En aquest cas, la solució vàlida la dóna l'algorisme de Prim, ja que, a cada pas, fixa un vèrtex i una aresta. Els asteriscos indiquen els vèrtexs fixats. La solució seria: {B,C}, {B,D}, {D,E}, {A,E} i {F,E}.

c) D'acord amb l'enunciat hem de construir l'arbre generador mínim de manera que sempre sigui connex. L'algorisme de Kruskal, a diferència del de Prim, no garanteix que el graf que es va generant sigui connex.

6. En l'exemple 9, hem triat les arestes marcades amb un asterisc i rebutjat les marcades en negreta:

Arestes	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 taula següent mostra les crides successives a les funcions *unió* i *cerca* i el contingut de la taula U:

Inicialment

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

cerca(U,1) → 1; *cerca*(U,4) → 4;

unió(U,1,4)

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

cerca(U,6) → 6; *cerca*(U,7) → 7;

unió(U,6,7)

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

cerca(U,1) → 1; *cerca*(U,2) → 2;

unió(U,1,2)

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

cerca(U,3) → 3; *cerca*(U,4) → 1;

unió(U,1,3)

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

cerca(U,4) → 1; *cerca*(U,5) → 5;

unió(U,1,5)

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

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

unió(U,1,6)

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

Aquesta darrera taula descriu un arbre amb arrel 1 i altura 2. Qualsevol aplicació nova de la funció *unió* ja provocaria un cicle.

7. Els algorismes de Kruskal i de Prim es poden utilitzar per a cercar un arbre generador maximal.

En l'algorisme de Kruskal només cal modificar l'ordenació de les arestes:

E = llista d'arestes ordenades per pes en ordre descendent.

En l'algorisme de Prim només cal modificar la comparació entre el pesos:

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

D'aquestes modificacions es desprén que els algorismes per a cercar un arbre generador maximal tenen la mateixa eficiència que per a cercar un arbre generador minimal.

8. Donat un graf ponderat (G,w) connex d'ordre n i mida m, l'algorisme de Kruskal determina un arbre generador minimal amb complexitat $O(m \log m)$ i l'algorisme de Prim amb complexitat $O(n^2)$.

Observem que l'eficiència de l'algorisme de Kruskal depèn de la mida del graf, mentre que l'algorisme de Prim té una eficiència que només depèn de l'ordre del graf. Si volem comparar els dos algorismes, cal estudiar la relació que hi ha entre aquestes dues quantitats, entre $m \log m$ i n^2.

Per a un graf connex la mida està afitada per

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

Quan la mida és propera a $\frac{n(n-1)}{2}$ (al graf complet K_n) es diu que el graf és *dens* (en anglès, *dense graph*). En cas contrari, direm que el graf és *poc dens* (*sparse graph*, en anglès).

Per tant, si el graf és poc dens, $m \approx n-1$ i podem substituir i obtenir:

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

En canvi, si el graf és dens $m \approx \frac{n(n-1)}{2}$ i substituint obtindrem:

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

Podem concloure que per a grafs poc densos és més eficient l'algorisme de Kruskal que el de Prim. En canvi, per a grafs densos és millor l'algorisme de Prim.

El principal avantatge de l'algorisme de Prim és que la seva eficiència és independent de la mida del graf. Això fa que es prefereixi en moltes aplicacions, en lloc del de Kruskal.

9. Si apliquem l'apartat 1 de la proposició 6, podem calcular el nombre de fulles, $t = (m-1)i + 1 = (5-1)200 + 1 = 801$. El nombre total de vèrtexs és $n = t + i = 801 + 200 = 1.001$.

10. Essent $n = |V|$ l'ordre de l'arbre, el nombre d'arestes és $|A| = n-1$. En un arbre com el que hem indicat considerem els vèrtexs diferenciats següents:

- l'arrel, que és l'únic vèrtex de grau 2;
- les fulles o vèrtexs de grau 1; indicarem per F el conjunt de les fulles, de cardinal $k = |F|$;
- la resta de vèrtexs, vèrtexs interns, de grau 3, dels quals n'hi ha m; indicarem per I el conjunt de vèrtexs interns.

D'acord amb la distribució anterior podem escriure

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

Escrivim ara el lema de les encaixades:

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

Ara apliquem que $|A| = n - 1$ i, tenint en compte els graus dels vèrtexs, podem escriure:

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

Substituïm a l'última igualtat el valor de $m = n - k - 1$, derivat de la primera relació anterior, i finalment obtenim $\frac{n+1}{2}$.

11. Si distribuïm les monedes (totes o una part) entre els plats de la balança, de manera que hi hagi el mateix nombre a cada plat, podem considerar tres possibles resultats:

Cas 1: els plats estan equilibrats. Significa que les monedes dels dos plats no són falses.

Cas 2: el plat de l'esquerra de la balança pesa més. Significa que la moneda falsa és a l'esquerra.

Cas 3: el plat de la dreta pesa més. Significa que la moneda falsa és a la dreta.

Podem representar aquesta situació mitjançant un arbre ternari que tindrà un mínim de vuit fulles, una per a cadascuna de les monedes. L'altura de l'arbre representarà el nombre de pesades necessàries per a descobrir la moneda falsa. Si apliquem les propietats del arbres m-aris, $m = 3$

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

Necessitem, doncs, un mínim de dues pesades per a descobrir la moneda falsa. Sabríeu trobar una estratègia que permetés assolir aquest mínim?

12. En l'arbre binari T d'una expressió aritmètica, els operands són sempre les fulles i els operadors els vèrtexs interns. A més, el recorregut en preordre sempre comença en l'arrel de l'arbre (i de cada subarbre). Per tant, per la forma de l'expressió deduïm que es tracta del recorregut en preordre de l'arbre binari d'una expressió aritmètica amb l'arrel $*$.

Així, l'expressió serà de la forma $\alpha * \beta$ on α i β són, respectivament, els dos subarbres de T. El subarbre α també s'ha de recórrer en preordre i, per tant, hauria de començar per un operador. Ara bé, com que el símbol que segueix a $*$ en el recorregut de T és una a, podem concloure que es tracta d'una fulla i que $\alpha = a$. Així, tindrem $a * \beta$ amb $\beta = + \; b \; * \; c \; + \; d \; e$.

Seguint un raonament semblant sobre el subarbre β podem deduir que el signe + és l'arrel, b és el primer subarbre de β i $* \; c \; + \; d \; e$ el segon. Seguint aquest mateix raonament de manera recursiva arribarem al fet que $\beta = b + c * (d + e)$.

Finalment, substituint α i β en l'expressió inicial obtenim l'expressió $a * (b + c * (d + e))$.

Bibliografia

Biggs, N. L. (1994). *Matemática Discreta*. (1a. edició, traducció 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.