Tema 5. Grafs

Estructures de Dades i Algorismes

FIB

Transparències d' Antoni Lozano
(amb edicions menors d'altres professors)

Q1 2020 – 21

Tema 5. Grafs

1 Propietats
- Introducció
- Grafs
- Grafs dirigits i etiquetats
- Representacions

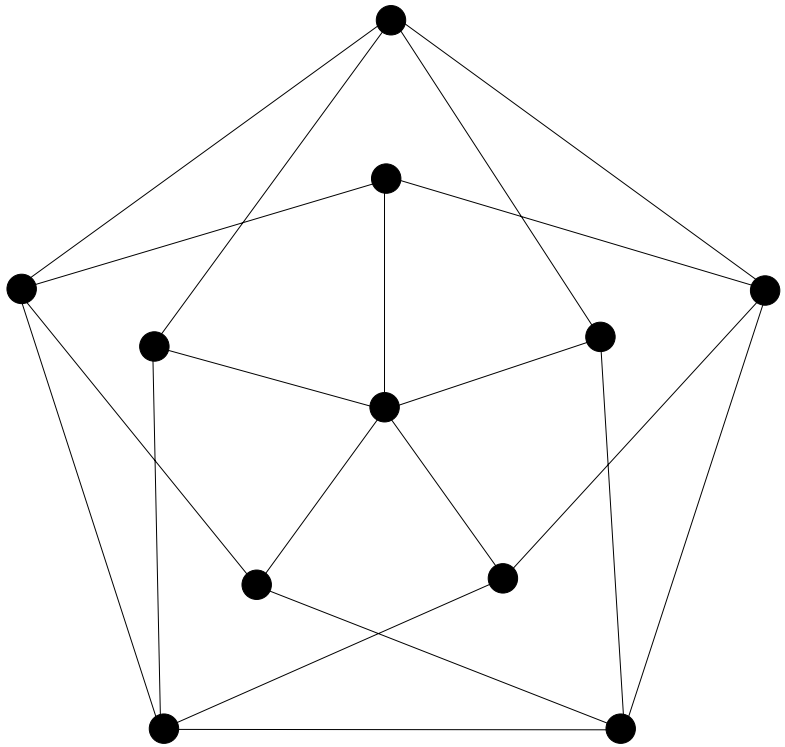
2 Cerca
- Cerca en profunditat
- Cerca en amplada
- Algorisme de Dijkstra
- Ordenació topològica
- Arbres d'Expansió Mínims

Tema 5. Grafs

1 Propietats
- Introducció
- Grafs
- Grafs dirigits i etiquetats
- Representacions

2 Cerca
- Cerca en profunditat
- Cerca en amplada
- Algorisme de Dijkstra
- Ordenació topològica
- Arbres d'Expansió Mínims

Per què els grafs?



Perquè molts problemes es poden expressar de manera clara i acurada mitjançant grafs.

Exemple: acolorir un mapa amb el mínim nombre de colors

Quin és el mínim nombre de colors necessari per acolorir un mapa de manera que dos països veïns no tinguin el mateix color?

Si analitzem un mapa real, trobarem tota mena d'informació irrellevant:

- escala,
- geometria dels països,
- mars, ...

Però tot mapa el podem representar com un graf:

- Un vèrtex correspon a un país o regió.
- Una aresta correspon a una frontera.

De fet, és un graf planar: es pot pintar al pla sense que les arestes es tallin

Exemple: acolorir un mapa amb el mínim nombre de colors

Quin és el mínim nombre de colors necessari per acolorir un mapa de manera que dos països veïns no tinguin el mateix color?

Si analitzem un mapa real, trobarem tota mena d'informació irrellevant:

- escala,
- geometria dels països,
- mars, ...

Però tot mapa el podem representar com un graf:

- Un vèrtex correspon a un país o regió.
- Una aresta correspon a una frontera.

De fet, és un graf planar: es pot pintar al pla sense que les arestes es tallin

Exemple: acolorir un mapa amb el mínim nombre de colors

Quin és el mínim nombre de colors necessari per acolorir un mapa de manera que dos països veïns no tinguin el mateix color?

Si analitzem un mapa real, trobarem tota mena d'informació irrellevant:

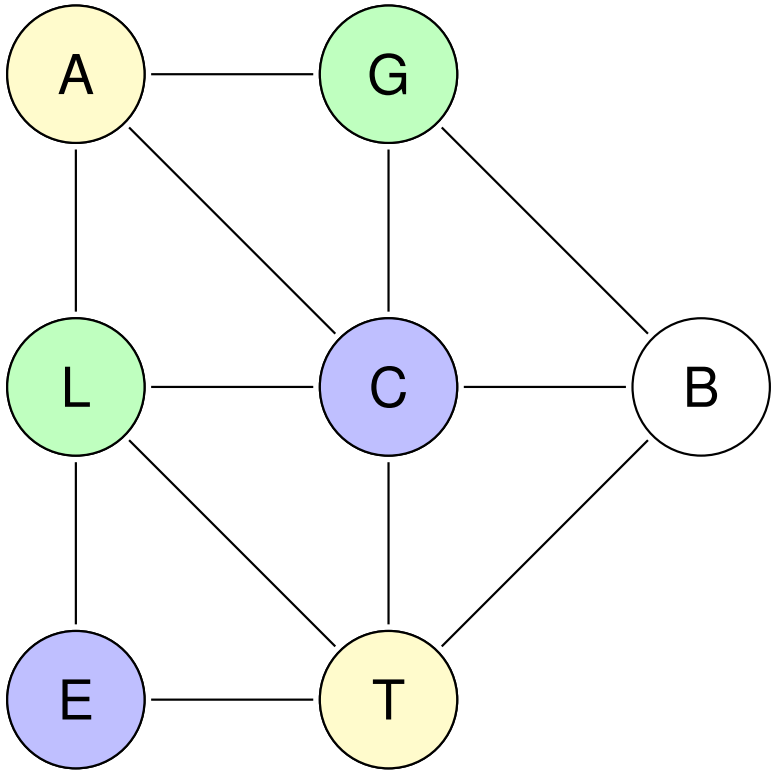
- escala,
- geometria dels països,
- mars, ...

Però tot mapa el podem representar com un graf:

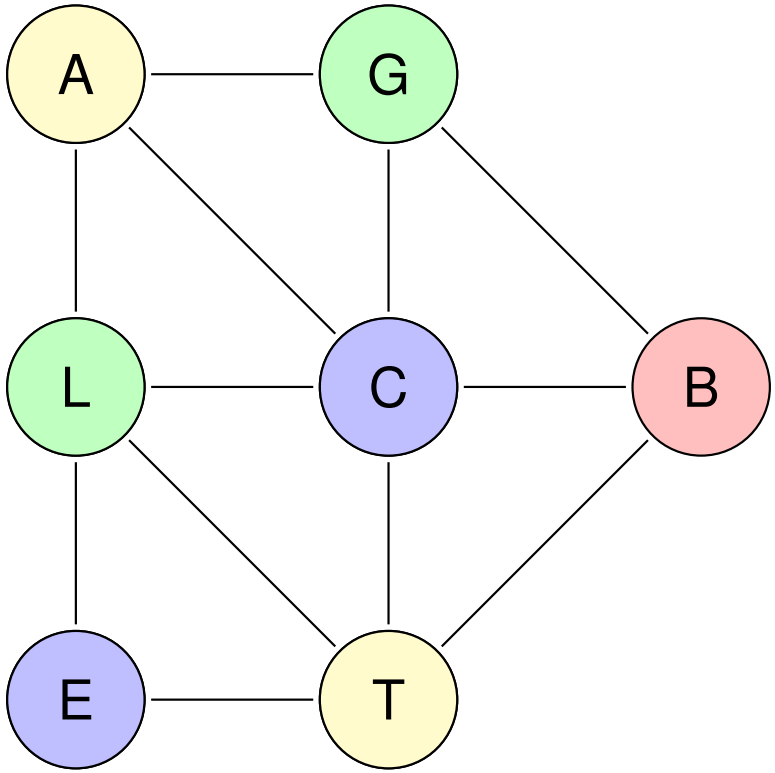
- Un vèrtex correspon a un país o regió.
- Una aresta correspon a una frontera.

De fet, és un graf planar: es pot pintar al pla sense que les arestes es tallin

Introducció



Introducció



Introducció

Fent servir els grafs, podem utilitzar el teorema següent.

Teorema dels quatre colors (Apple/Haken, 1976)

Tot graf planar es pot acolorir amb 4 colors.

Per tant, tot mapa es pot acolorir amb 4 colors.

Introducció

Exemple: el rei Artur i la taula rodona

El rei Artur vol asseure els seus cavallers a la taula rodona però ha d'evitar que alguns d'ells que estan enemistats seguin colze contra colze.

Solució

- Es crea un graf on cada vèrtex és un cavaller i hi ha una aresta entre cada parella que no està enemistada.
- Es busca un cicle que passa per tots els vèrtexs (cicle hamiltonià): aquest indica la manera com han de seure.

Introducció

Exemple: el rei Artur i la taula rodona

El rei Artur vol asseure els seus cavallers a la taula rodona però ha d'evitar que alguns d'ells que estan enemistats seguin colze contra colze.

Solució

- Es crea un graf on cada vèrtex és un cavaller i hi ha una aresta entre cada parella que no està enemistada.
- Es busca un cicle que passa per tots els vèrtexs (cicle hamiltonià): aquest indica la manera com han de seure.

Introducció

Aplicacions dels grafs:

- **Mapes.** Com trobar el millor camí en una xarxa de metro? Quina és la combinació més barata per anar de Barcelona a San Francisco?
- **Hipertextos.** Si una pàgina web és un vèrtex i un enllaç és una aresta, tota la WWW és un graf. Com calcular la importància d'una pàgina respecte d'una cerca d'informació?
- **Programació de tasques.** En els processos industrials, hi ha unes tasques que cal fer abans que d'altres, però es vol completar el procés en el mínim de temps.
- **Xarxes d'ordinadors, circuits, ...**

Definició informal

Un graf està format per un conjunt de vèrtexs connectats per arestes, de forma que entre cada parell de vèrtexs hi ha com a molt una aresta.

Definició formal

Un graf és un parell (V, E), on V és un conjunt finit (de vèrtexs) i E és un conjunt de parells no ordenats de vèrtexs diferents (les arestes).

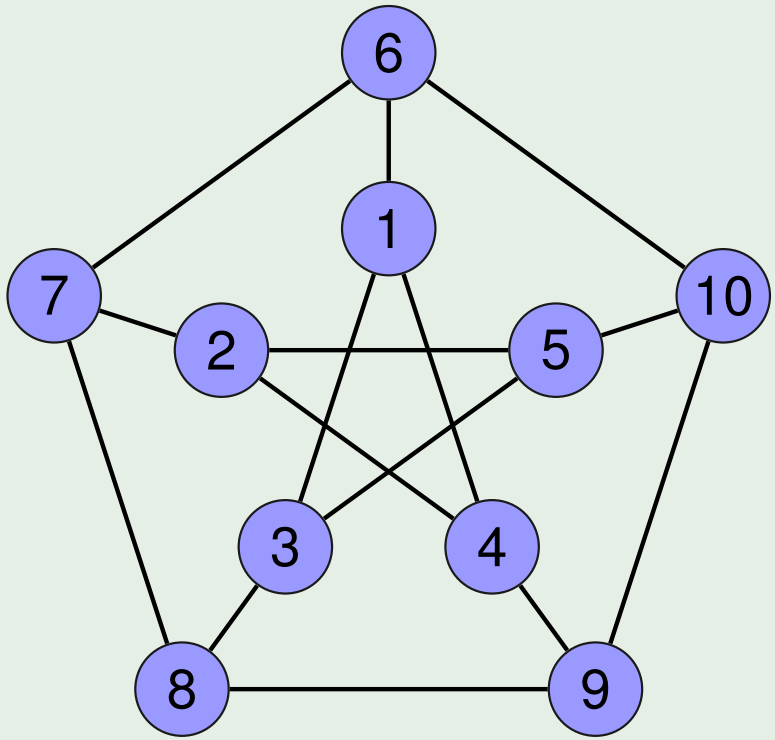
Definició informal

Un graf està format per un conjunt de vèrtexs connectats per arestes, de forma que entre cada parell de vèrtexs hi ha com a molt una aresta.

Definició formal

Un graf és un parell (V, E), on V és un conjunt finit (de vèrtexs) i E és un conjunt de parells no ordenats de vèrtexs diferents (les arestes).

Exemple: graf de Petersen

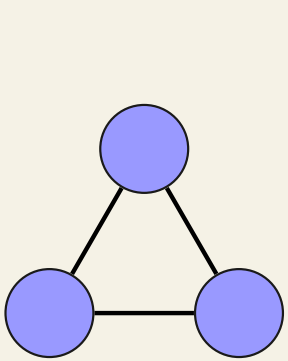


Formalment, és el parell (V, E) on

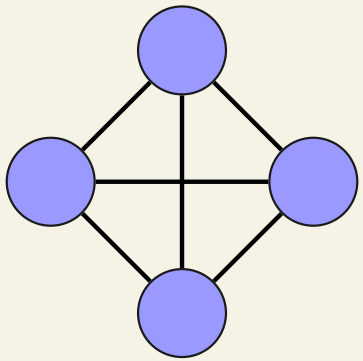
- $V = \{1, \ldots, 10\}$
- $E = \{\{1,3\},\{1,4\},\{1,6\},\{2,4\},\{2,5\},\{2,7\},\{3,5\},$
  $\{3,8\},\{4,9\},\{5,10\},\{6,7\},\{6,10\},\{7,8\},\{8,9\},\{9,10\}\}$

Exemples de grafs

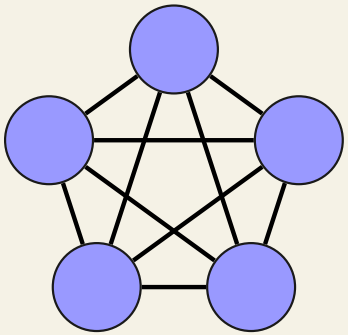
- **Complets:** K_i és el graf complet de i vèrtexs.



K_3

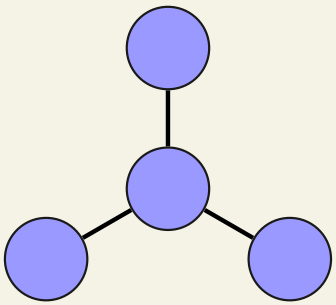


K_4

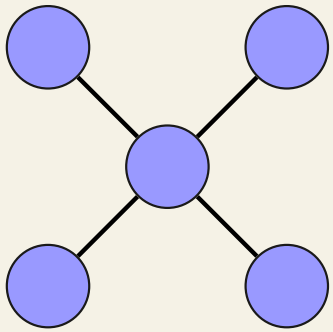


K_5

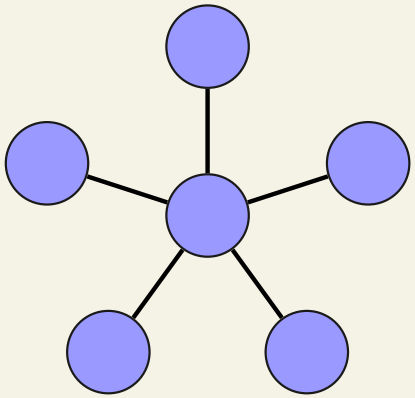
- **Estrelles:** S_i és l'estrella amb $i + 1$ vèrtexs.



S_3



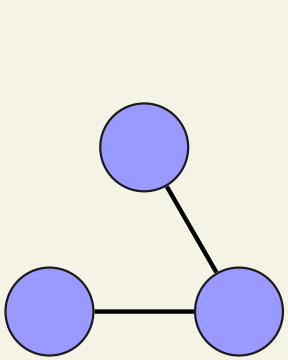
S_4



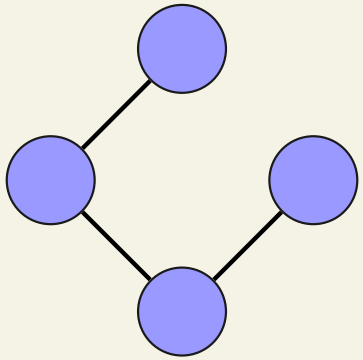
S_5

Exemples de grafs

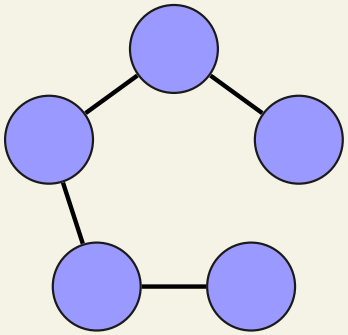
- **Camins:** P_i és el camí de i vèrtexs.



P_3

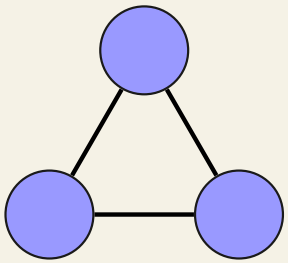


P_4

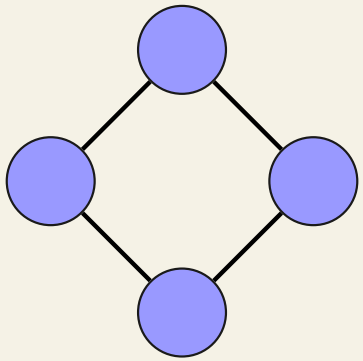


P_5

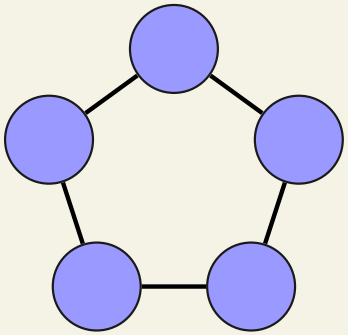
- **Cicles:** C_i és el cicle de i vèrtexs.



C_3



C_4



C_5

Propietat

Tot graf de n vèrtexs té un màxim de $\frac{n(n-1)}{2}$ arestes.

Demostració

Cada vèrtex pot tenir una aresta amb $n-1$ vèrtexs més (però no amb ell mateix). Com que cada aresta està comptada dos cops, s'obtenen

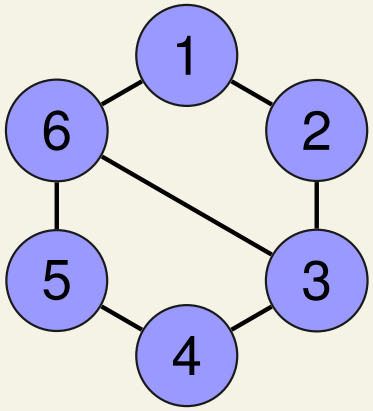
$$\frac{n(n-1)}{2}$$

arestes diferents.

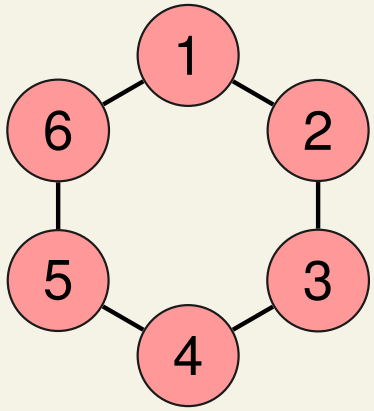
(Són les combinacions de n elements triats de 2 en 2.)

Adjacència i subgrafs

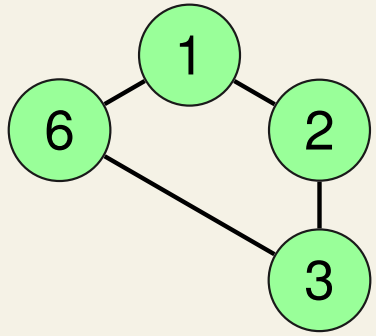
- dos vèrtexs u, v són adjacents si $\{u, v\}$ és una aresta
- una aresta $\{u, v\}$ es diu que és incident en u i v
- grau d'un vèrtex u: nombre d'arestes incidents en u
- un graf $H = (V', E')$ és subgraf del graf $G = (V, E)$ si $V' \subseteq V$ i $E' \subseteq E$.
- un graf H és subgraf induït d'un graf G si
  H és subgraf de G i conté totes les arestes que G té entre vèrtexs de H.



Graf G



Subgraf de G



Subgraf induït de G

Camins i cicles

- Un camí en un graf és una seqüència de vèrtexs en què cada vèrtex (llevat del primer) és adjacent al seu predecessor en el camí.
- Un camí simple és un camí en el qual no hi ha vèrtexs repetits.
- Un cicle és un camí en què no hi ha vèrtexs repetits, excepte l'inicial i el final que són el mateix.
- Un graf és cíclic si conté algun cicle.

Observació

Un graf G té

1. un camí simple de k vèrtexs si i només si P_k és subgraf de G
2. un cicle de k vèrtexs si i només si C_k és subgraf de G

Camins i cicles

- Un camí en un graf és una seqüència de vèrtexs en què cada vèrtex (llevat del primer) és adjacent al seu predecessor en el camí.
- Un camí simple és un camí en el qual no hi ha vèrtexs repetits.
- Un cicle és un camí en què no hi ha vèrtexs repetits, excepte l'inicial i el final que són el mateix.
- Un graf és cíclic si conté algun cicle.

Observació

Un graf G té

1. un camí simple de k vèrtexs si i només si P_k és subgraf de G
2. un cicle de k vèrtexs si i només si C_k és subgraf de G

Connectivitat

- Un graf és connex si existeix un camí entre tot parell de vèrtexs.
- Un component connex d'un graf és un subgraf induït connex maximal (no hi ha cap vèrtex extern adjacent)

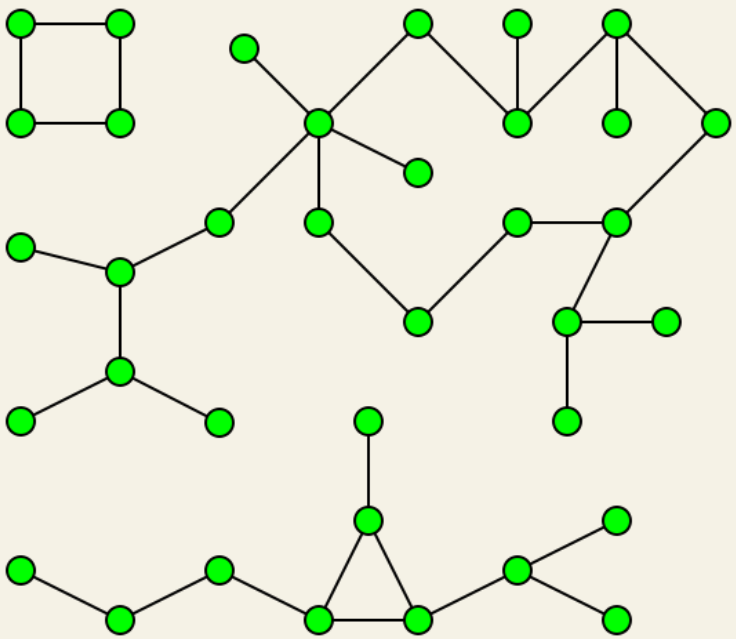


Figura: Graf cíclic amb 3 components connexos

Distància

- La distància entre dos vèrtexs és
  el nombre mínim d'arestes d'un camí que els uneix.
- El diàmetre d'un graf és
  la màxima distància entre qualsevol parell de vèrtexs del graf.

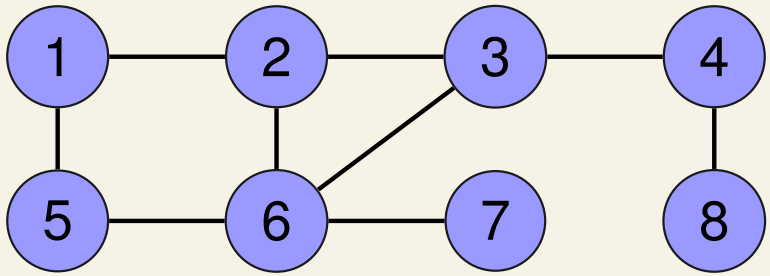
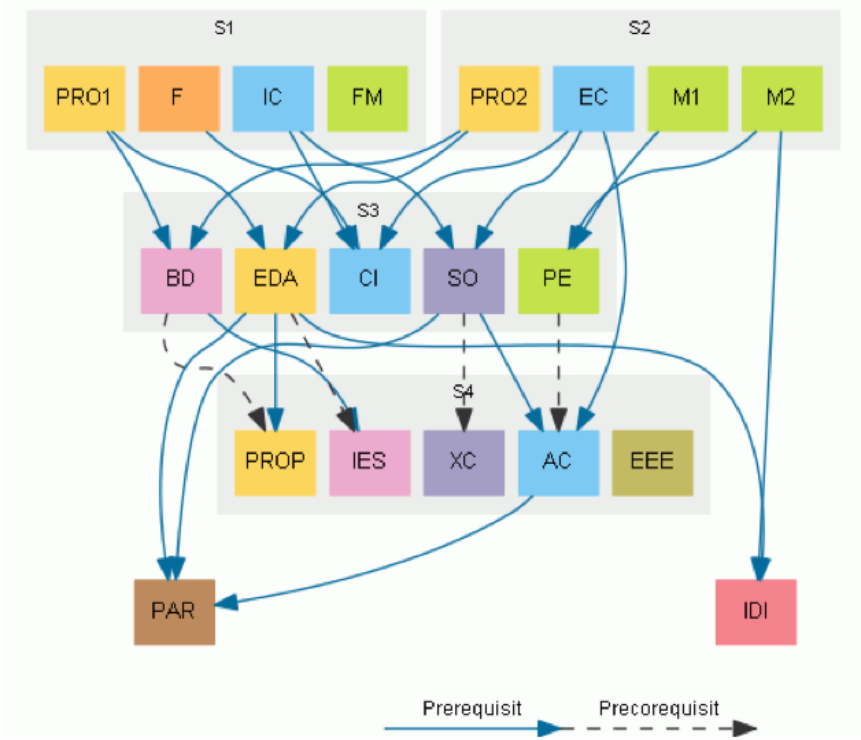


Figura: La distància entre 4 i 5 és 3. El diàmetre del graf és 4.

Grafs dirigits i etiquetats

Definició

- Un graf dirigit o digraf és un parell (V, E), on
  - V és un conjunt finit (de vèrtexs) i
  - E és un conjunt de parells ordenats de vèrtexs (anomenats arcs).



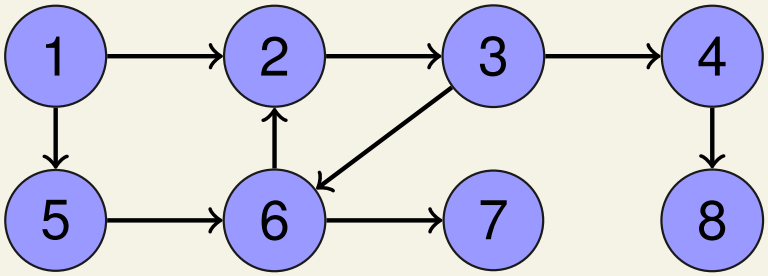
Graf dirigit d'assignatures obligatòries del grau

Grafs dirigits i etiquetats

Els conceptes de grafs es traslladen sense gaires canvis als digrafs.

Digrafs: graus i distàncies

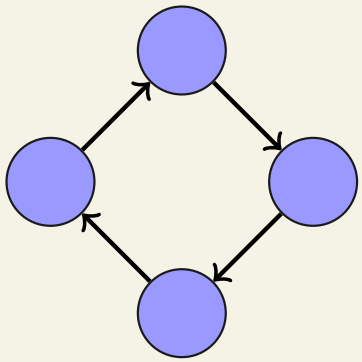
- Es distingeix entre grau d'entrada i grau de sortida.
- En un camí (o camí dirigit), tots els arcs van en la mateixa direcció.
- La distància entre dos vèrtexs es refereix als camins dirigits.



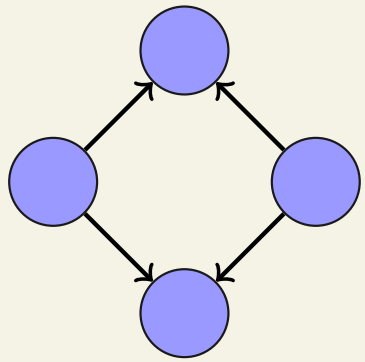
El vèrtex 2 té grau d'entrada 2 i grau de sortida 1.
La distància de 5 a 4 és 4; de 4 a 5, ∞.

Digrafs: connectivitat

- Un digraf és feblement connex (o connex) si
  el graf obtingut substituint els arcs per arestes no dirigides és connex.
- Un digraf és fortament connex si
  existeix un camí dirigit entre qualsevol parell de vèrtexs.
- Si un digraf és fortament connex llavors també és feblement connex
- A la inversa no és cert:



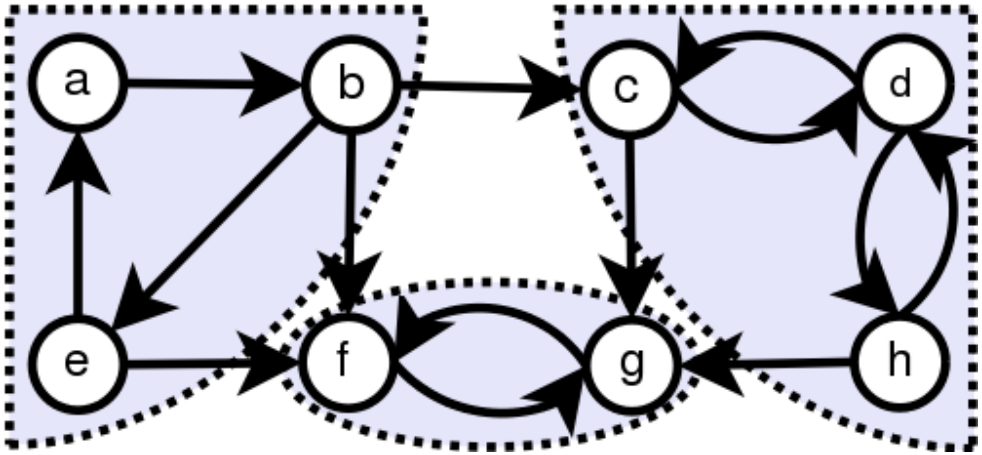
fortament connex



feblement (i no fortament) connex

Digrafs: connectivitat

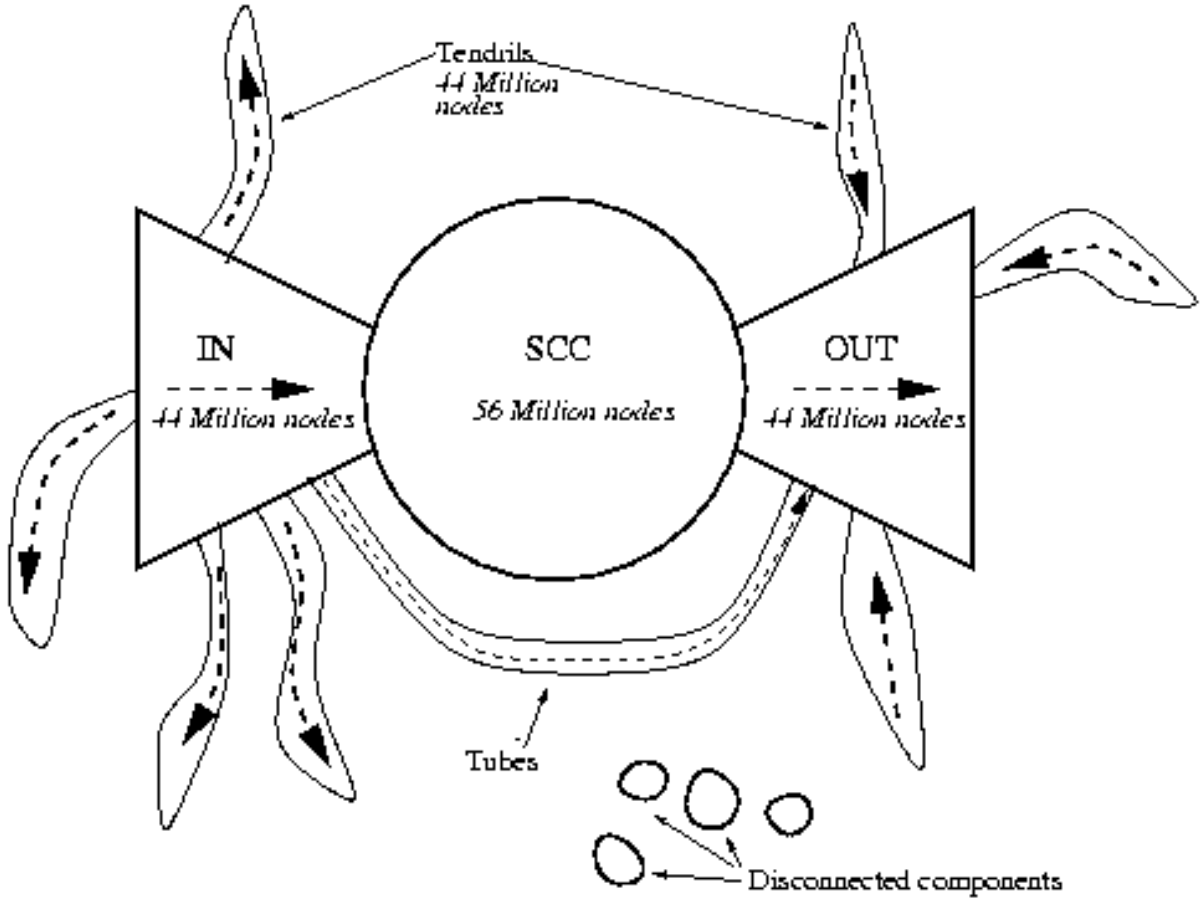
- Els components fortament connexos d'un digraf són els subgrafs induïts fortament connexos maximals.



Digraf amb 3 components connexos

Graf d'internet (any 2000, Broder, Kumar et al.)

Les pàgines web són els vèrtexs; els enllaços, les arestes.

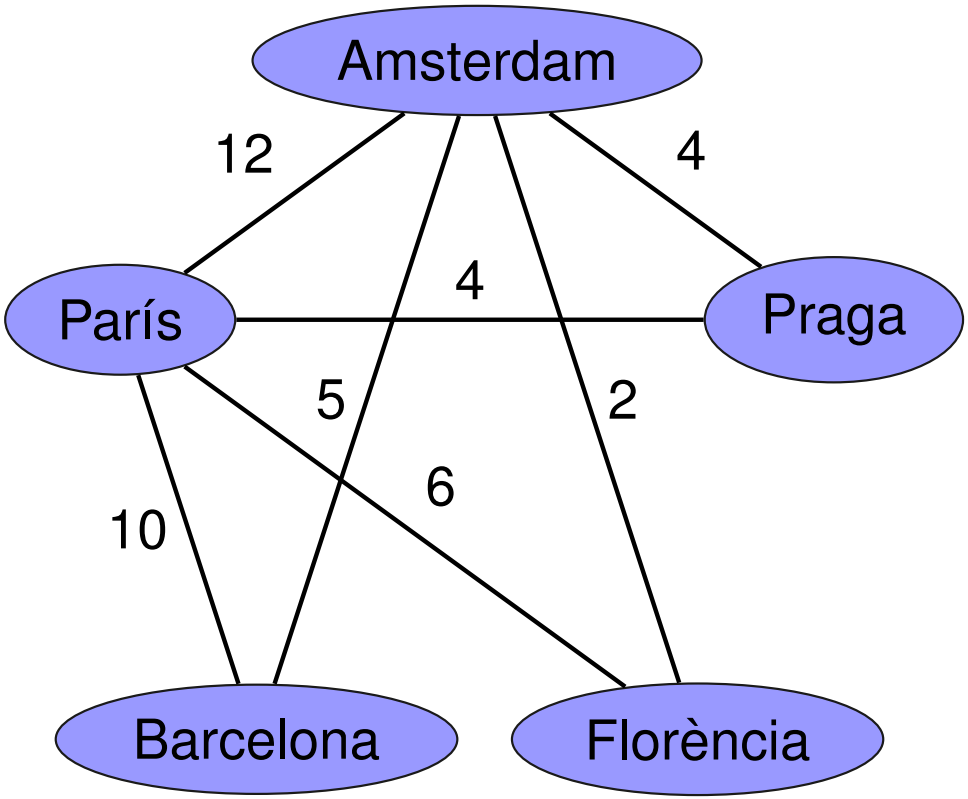


SCC: *(giant) strongly connected component*
diàmetre del SCC: 28

Grafs dirigits i etiquetats

Definició

Anomenarem graf etiquetat (dirigit o no dirigit) a un graf en el qual les arestes tenen etiquetes associades. També se'n diu ponderat o graf amb pesos.



Nombre de vols diaris amb Air France i KLM

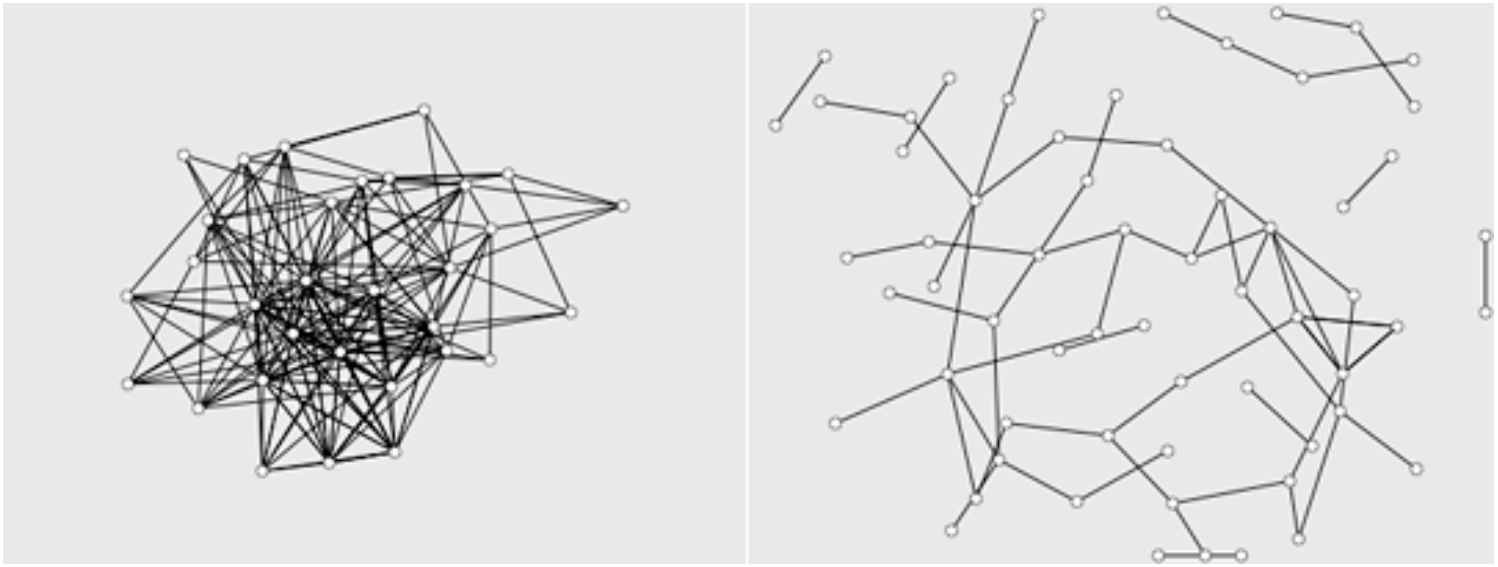
Grafs dirigits i etiquetats

Es fan servir les 4 combinacions:

- Grafs no dirigits no etiquetats
- Grafs no dirigits etiquetats
- Grafs dirigits no etiquetats
- Grafs dirigits etiquetats

Representacions

La representació dels grafs dependrà de la seva densitat d'arestes.



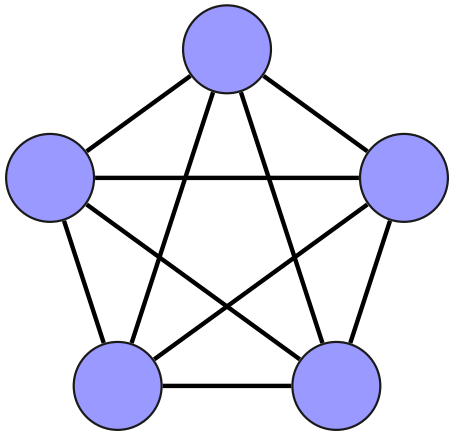
Però com podem definir la densitat d'un graf?

Densitat

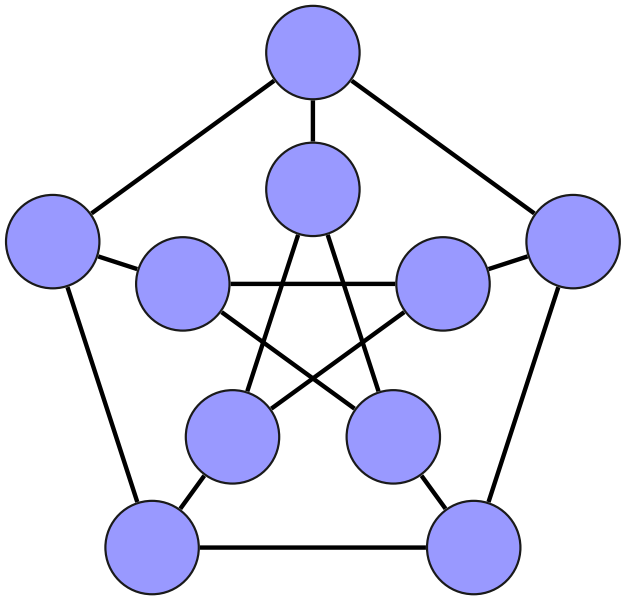
La densitat d'un graf de n vèrtexs i m arestes es defineix com

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

Hem vist que el nombre màxim d'arestes d'un graf de n vèrtexs és $\frac{n(n-1)}{2}$.
Per tant, $0 \leq D \leq 1$.



D=1



D=1/3

Densitat

Un graf de n vèrtexs i m arestes és dens si $m \approx n^2/2$ (si D és proper a 1).
Altrament, se'n diu espars.

El concepte és més formal quan es consideren famílies de grafs.
Per exemple:

- Els grafs complets (K_n) són densos
  perquè $D = 1$ per a tot n.

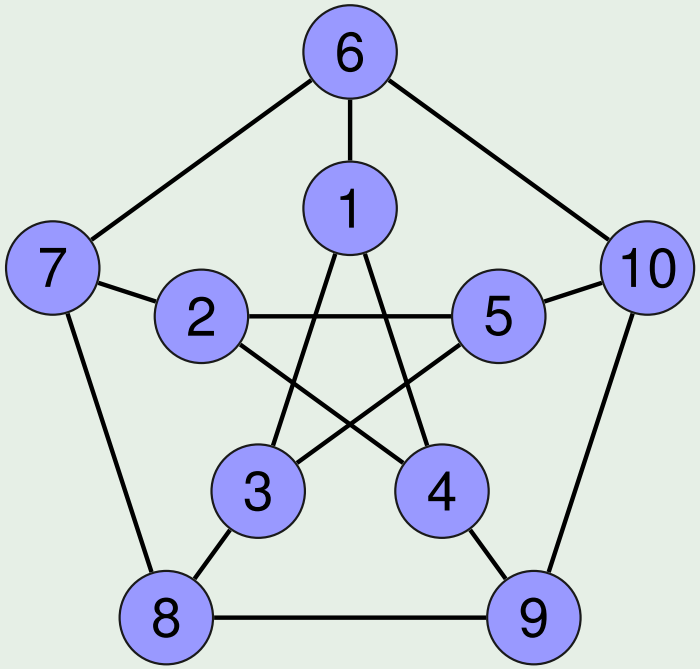
- Els cicles (C_n) són esparsos
  perquè $D = 2/(n-1)$ i la densitat tendeix a 0 quan n tendeix a ∞.

Matriu d'adjacència / grafs no dirigits

La matriu d'adjacència d'un graf no dirigit $G = (V, E)$ és una matriu M de $n \times n$ valors booleans tal que

$$M_{ij} = \begin{cases} 1, & \text{si } \{i,j\} \in E \\ 0, & \text{si } \{i,j\} \notin E \end{cases}$$

Exemple



	1	2	3	4	5	6	7	8	9	10
1	0	0	1	1	0	1	0	0	0	0
2	0	0	0	1	1	0	1	0	0	0
3	1	0	0	0	1	0	0	1	0	0
4	1	1	0	0	0	0	0	0	1	0
5	0	1	1	0	0	0	0	0	0	1
6	1	0	0	0	0	0	1	0	0	1
7	0	1	0	0	0	1	0	1	0	0
8	0	0	1	0	0	0	1	0	1	0
9	0	0	0	1	0	0	0	1	0	1
10	0	0	0	0	1	1	0	0	1	0

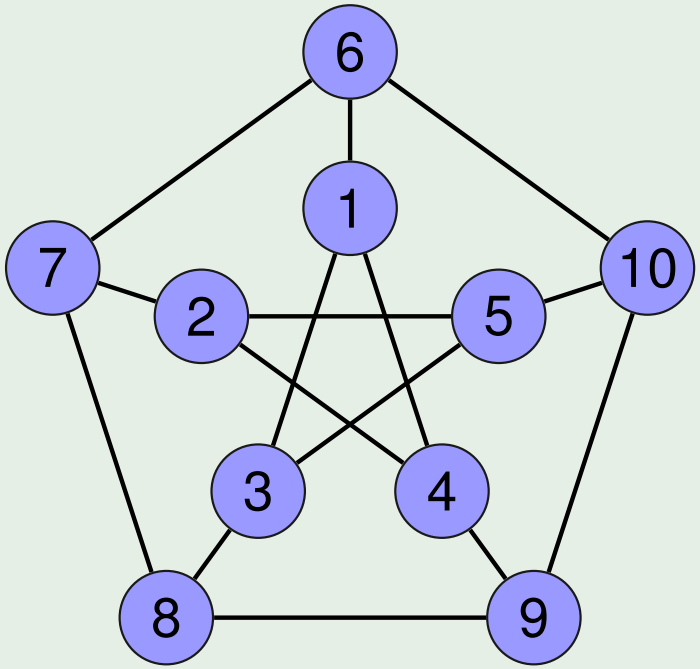
Representacions

Matriu d'adjacència / grafs no dirigits

La matriu d'adjacència d'un graf no dirigit $G = (V, E)$ és una matriu M de $n \times n$ valors booleans tal que

$$M_{ij} = \begin{cases} 1, & \text{si } \{i,j\} \in E \\ 0, & \text{si } \{i,j\} \notin E \end{cases}$$

Exemple



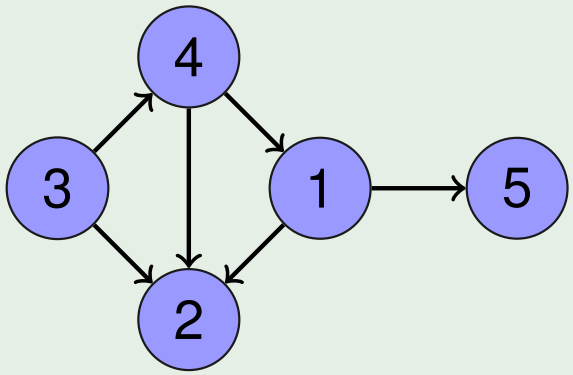
	1	2	3	4	5	6	7	8	9	10
1		0	1	1	0	1	0	0	0	0
2	0		0	1	1	0	1	0	0	0
3	1	0		0	1	0	0	1	0	0
4	1	1	0		0	0	0	0	1	0
5	0	1	1	0		0	0	0	0	1
6	1	0	0	0	0		1	0	0	1
7	0	1	0	0	0	1		1	0	0
8	0	0	1	0	0	0	1		1	0
9	0	0	0	1	0	0	0	1		1
10	0	0	0	0	1	1	0	0	1	

Matriu d'adjacència / grafs dirigits

La matriu d'adjacència d'un graf dirigit $G = (V, E)$ és una matriu M de $n \times n$ valors booleans tal que

$$M_{ij} = \begin{cases} 1, & \text{si } (i,j) \in E \\ 0, & \text{si } (i,j) \notin E \end{cases}$$

Exemple



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Matriu d'adjacència

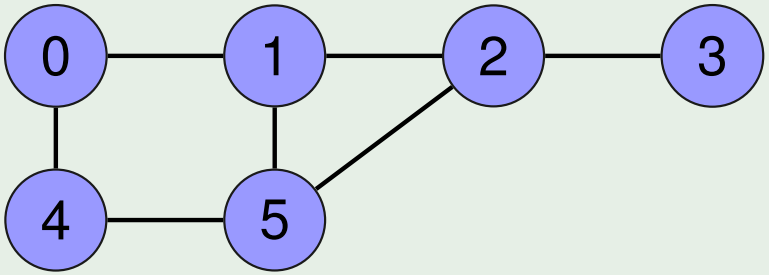
Es pot implementar amb un vector de vectors de booleans.

```
typedef vector< vector<bool> > graph;
```

Si `g` és de tipus `graph` (n vèrtexs):

- El cost en espai és $\Theta(n^2)$.
- La inicialització de `g` és $\Theta(n^2)$.
- El vector `g[i]` conté la informació sobre els veïns de `i`.
- Processar els vèrtexs adjacents a `i` serà $\Theta(n)$.
- Si el graf `g` és no dirigit, llavors `g[i][j] == g[j][i]` (la informació està duplicada).
- Recórrer totes les arestes és $\Theta(n^2)$.
- La matriu d'adjacència és adequada per a grafs densos.

Exemple



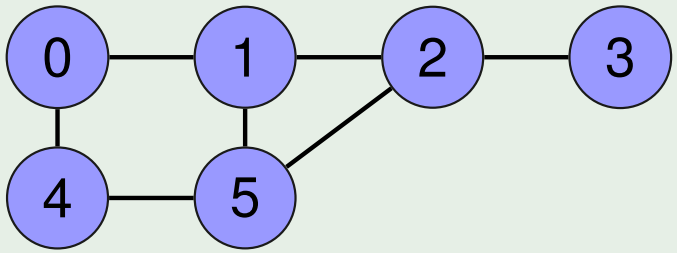
Representacions

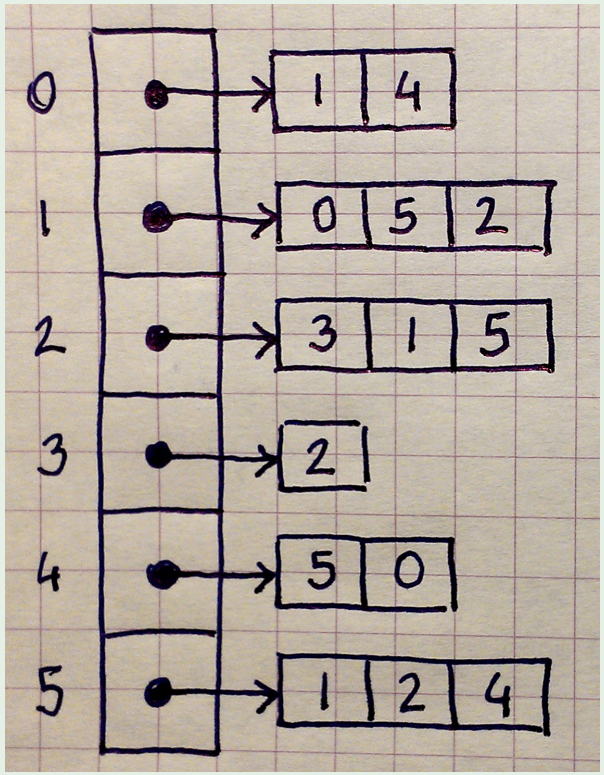
Llistes d'adjacència

Cada vèrtex apunta a la llista dels adjacents (grafs) o successors (digrafs).

```
typedef vector< vector<int> > graph;
```

Exemple





Llistes d'adjacència

Cada vèrtex apunta a la llista dels adjacents (grafs) o successors (digrafs).

```
typedef vector< vector<int> > graph;
```

Si `g` és de tipus `graph` (n vèrtexs i m arestes):

- El cost en espai és $\Theta(n+m)$.
- La inicialització és $\Theta(n)$.
- Els adjacents a `i` estan encadenats sense un ordre especial.
- Processar els vèrtexs adjacents a `i` serà $\Theta(\text{grau}(\texttt{i}))$.
- Si `g` no és dirigit, la informació està duplicada.
- Recórrer totes les arestes és $\Theta(n+m)$.
- Les llistes d'adjacència són adequades per a grafs esparsos.

Cost de les operacions

n: nombre de vèrtexs / m: nombre d'arestes

operacions	matriu d'adjacència	llistes d'adjacència
espai	$\Theta(n^2)$	$\Theta(n+m)$
crear	$\Theta(n^2)$	$\Theta(n)$
afegir vèrtex	$\Theta(n)$	$\Theta(1)$
afegir aresta	$\Theta(1)$	$\mathcal{O}(n)$
esborrar aresta	$\Theta(1)$	$\mathcal{O}(n)$
consultar vèrtex	$\Theta(1)$	$\Theta(1)$
consultar aresta	$\Theta(1)$	$\mathcal{O}(n)$
successors*	$\Theta(n)$	$\mathcal{O}(n)$
predecessors*	$\Theta(n)$	$\Theta(n+m)$
adjacents+	$\Theta(n)$	$\mathcal{O}(n)$

* només en grafs dirigits

\+ només en grafs no dirigits

1 Propietats
- Introducció
- Grafs
- Grafs dirigits i etiquetats
- Representacions

2 Cerca
- Cerca en profunditat
- Cerca en amplada
- Algorisme de Dijkstra
- Ordenació topològica
- Arbres d'Expansió Mínims

Cerca en profunditat

La cerca (o recorregut) en profunditat (en anglès: DFS, de *Depth-First Search*) resol la pregunta:

A quins vèrtexs podem arribar des d'un vèrtex donat?

Un algorisme només pot disposar de la informació de les adjacències: si és possible anar d'un vèrtex a un altre.

La situació és semblant a l'exploració d'un laberint.

Cerca en profunditat

Per explorar un laberint, cal tenir guix i corda:

- El guix evita anar en cercles (saber què hem visitat).
- La corda permet anar enrere i veure passadissos encara no visitats.

Quins són els anàlegs informàtics del guix i la corda?

- el guix és un vector de booleans
- la corda és una pila

Cerca en profunditat

Per explorar un laberint, cal tenir guix i corda:

- El guix evita anar en cercles (saber què hem visitat).
- La corda permet anar enrere i veure passadissos encara no visitats.

Quins són els anàlegs informàtics del guix i la corda?

- el guix és un vector de booleans
- la corda és una pila

Cerca en profunditat

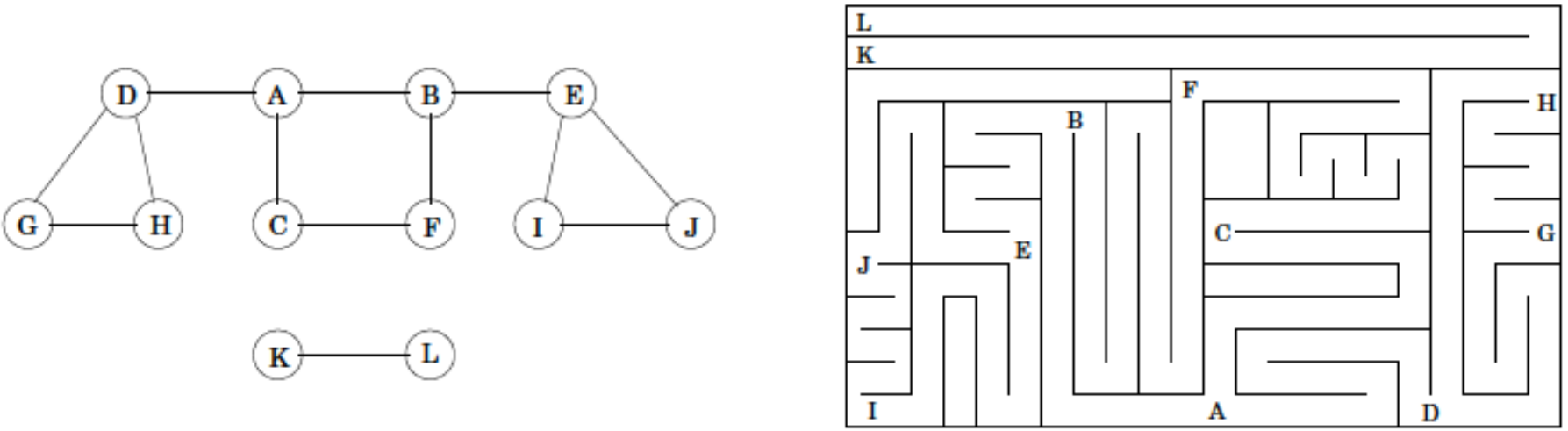


Figura: Per convertir un laberint en un graf: es marquen zones del laberint (vèrtexs) i s'uneixen (arestes) si són veïnes (exemple de *Algorithms*, S. Dasgupta, C.H. Papadimitriou i U.V. Vazirani)

Cerca en profunditat

Cerca en profunditat recursiva

Recorregut en profunditat de tot el graf, encara que no sigui connex.

```
list<int> dfs_rec (const graph& G) {
    int n = G.size();
    list<int> L;
    vector<boolean> vis(n, false);
    for (int u = 0; u < n; ++u) {
        dfs_rec(G, u, vis, L);
    }
    return L;
}
```

Cerca en profunditat

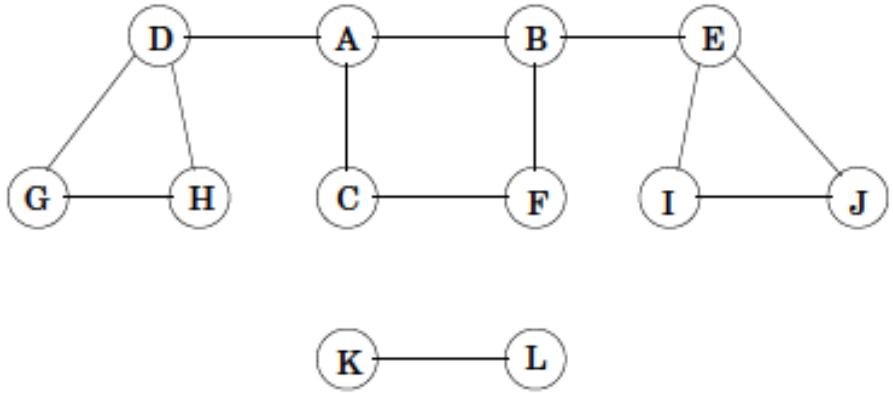
Cerca en profunditat recursiva (des d'un vèrtex)

Visitar tots els vèrtexs accessibles a partir d'un vèrtex donat `u`.

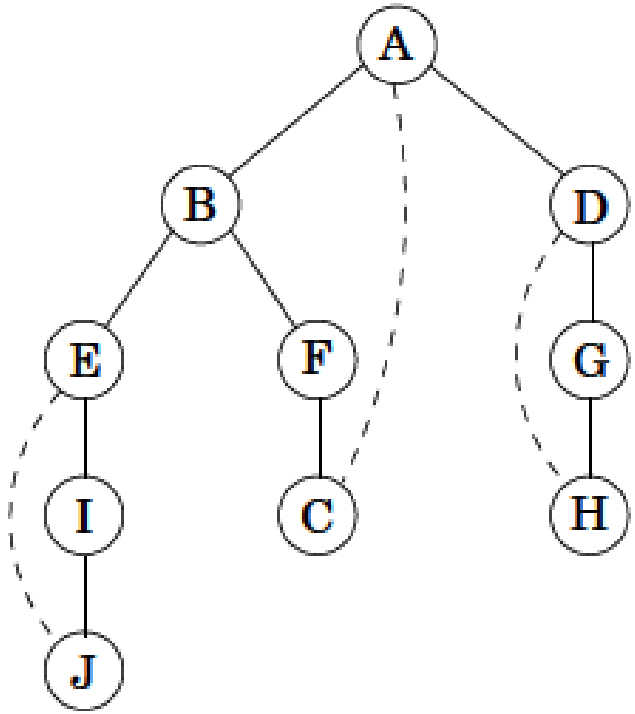
```
void dfs_rec (const graph& G, int u,
              vector<boolean>& vis, list<int>& L) {
    if (not vis[u]) {
        vis[u] = true; L.push_back(u);
        for (int v : G[u]) {
            dfs_rec(G, v, vis, L);
}   }   }
```

Cerca en profunditat

Exemple anterior:
(*Algorithms.* Dasgupta et al.)



Cerca en profunditat en el graf anterior a partir del vèrtex A: (assumint que els vèrtexs adjacents es visiten en ordre alfabètic)



Cerca en profunditat

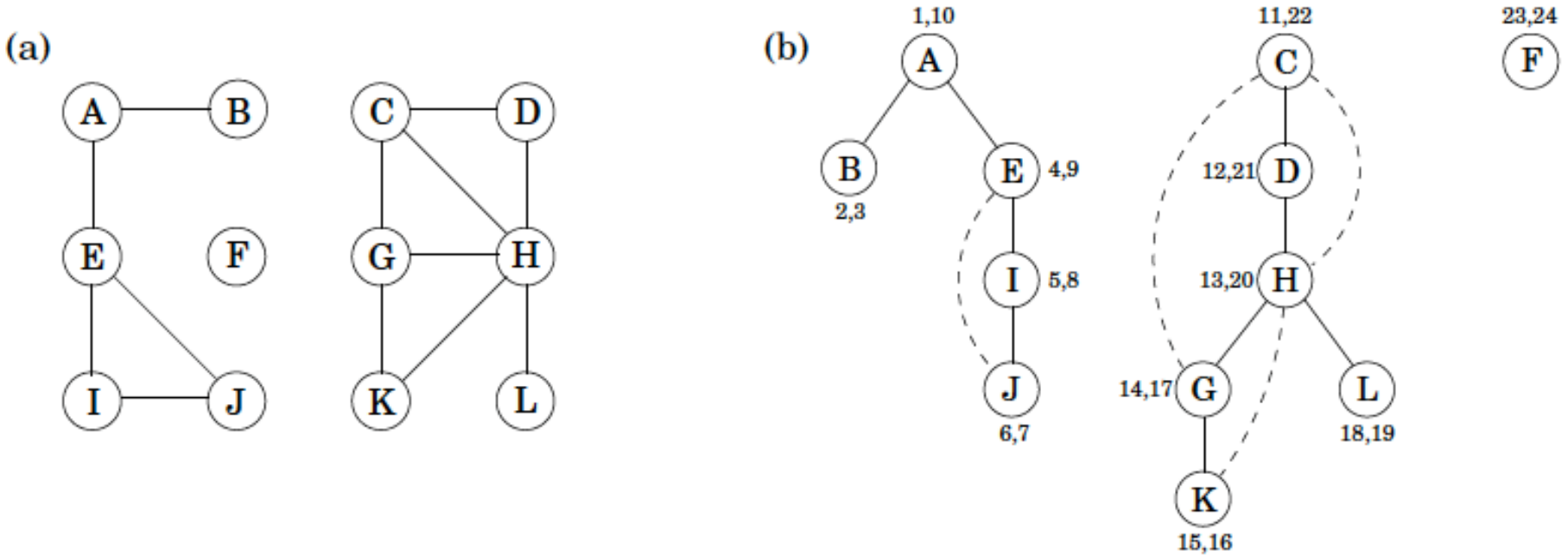


Figura: Cerca en profunditat en graf no dirigit i inconnex (*Algorithms.* Dasgupta et al.)

Cerca en profunditat

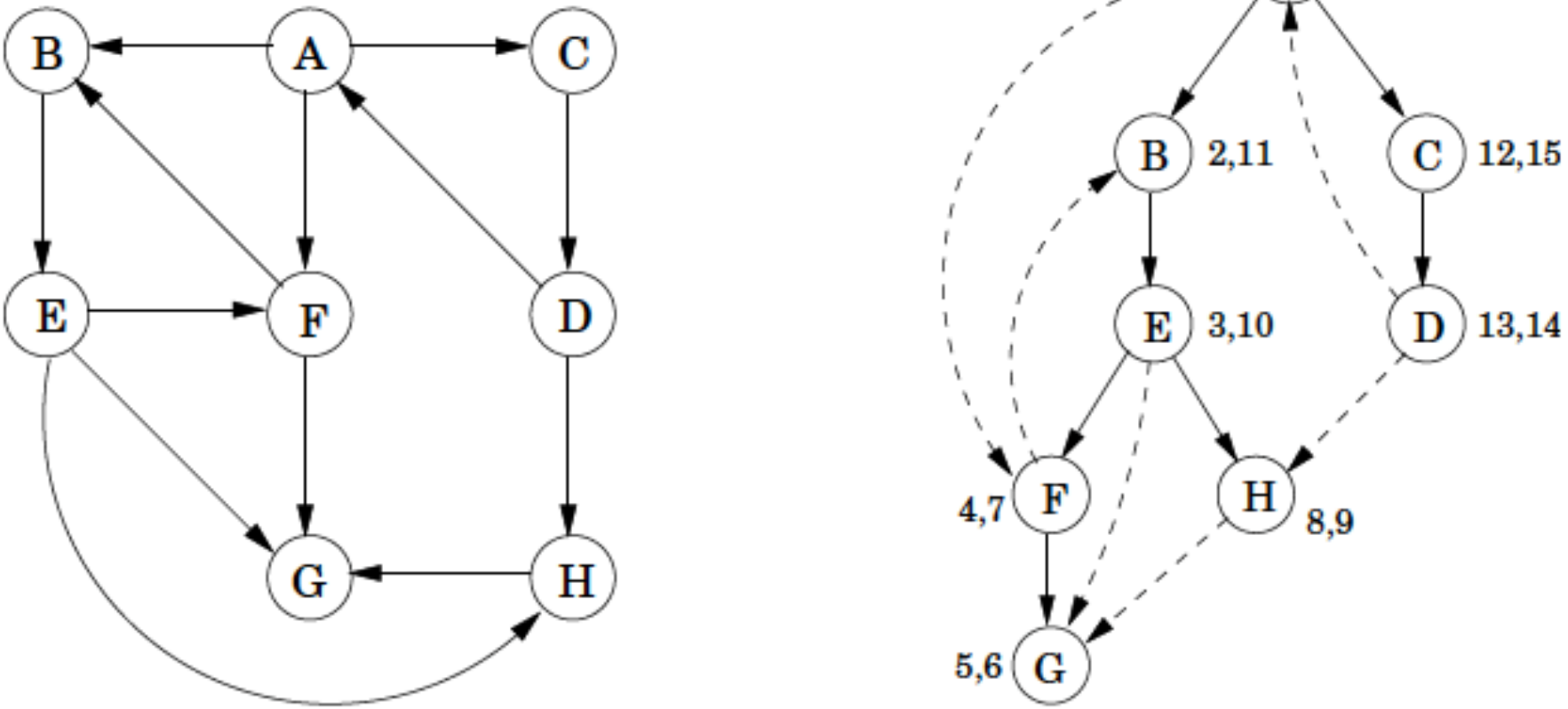


Figura: Cerca en profunditat en graf dirigit (*Algorithms.* Dasgupta et al.)

Cerca en profunditat

Cost de la cerca des d'un vèrtex

```
void dfs_rec (const graph& G, int u,
              vector<boolean>& vis, list<int>& L) {
    if (not vis[u]) {
        vis[u] = true; L.push_back(u);
        for (int w : G[u]) {
            dfs_rec(G, w, vis, L);
}   }   }
```

- Es fa una quantitat fixa de treball (2 primeres línies): $\Theta(1)$
- Es fan crides recursives als veïns. En total,
  - cada vèrtex del component connex es marca un cop
  - cada aresta i, j es visita dos cops: des de `i` i des de `j`

  Per tant, $\mathcal{O}(n+m)$.

Cost total: $\mathcal{O}(n+m)$.

Cerca en profunditat

Cost de la cerca en profunditat

```
list<int> dfs_rec (const graph& G) {
    int n = G.size();
    list<int> L;
    vector<boolean> vis(n, false);
    for (int u = 0; u < n; ++u)
        dfs_rec(G, u, vis, L);
    return L; }
```

Si el graf `G` té

- k components connexos (c.c.),
- $n = \sum_{i=1}^{k} n_i$ vèrtexs (el c.c. i té n_i vèrtexs) i
- $m = \sum_{i=1}^{k} m_i$ arestes (el c.c. i té m_i arestes),

llavors el cost és

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

Cerca en profunditat

Cerca en profunditat iterativa

```
list<int> dfs_ite (const graph& G) {
    int n = G.size();
    list<int> L;
    stack<int> S;
    vector<bool> vis(n, false);

    for (int u = 0; u < n; ++u) {
        S.push(u);
        while (not S.empty()) {
            int v = S.top(); S.pop();
            if (not vis[v]) {
                vis[v] = true; L.push_back(v);
                for (int w : G[v]) {
                    S.push(w);
    }   }   }   }
    return L;
}
```

Cerca en amplada

La cerca (o recorregut) en amplada (en anglès: BFS, de *Breadth-First Search*) avança localment des d'un vèrtex inicial s visitant els vèrtexs a distància $k + 1$ de s després d'haver visitat els vèrtexs a distància k de s.

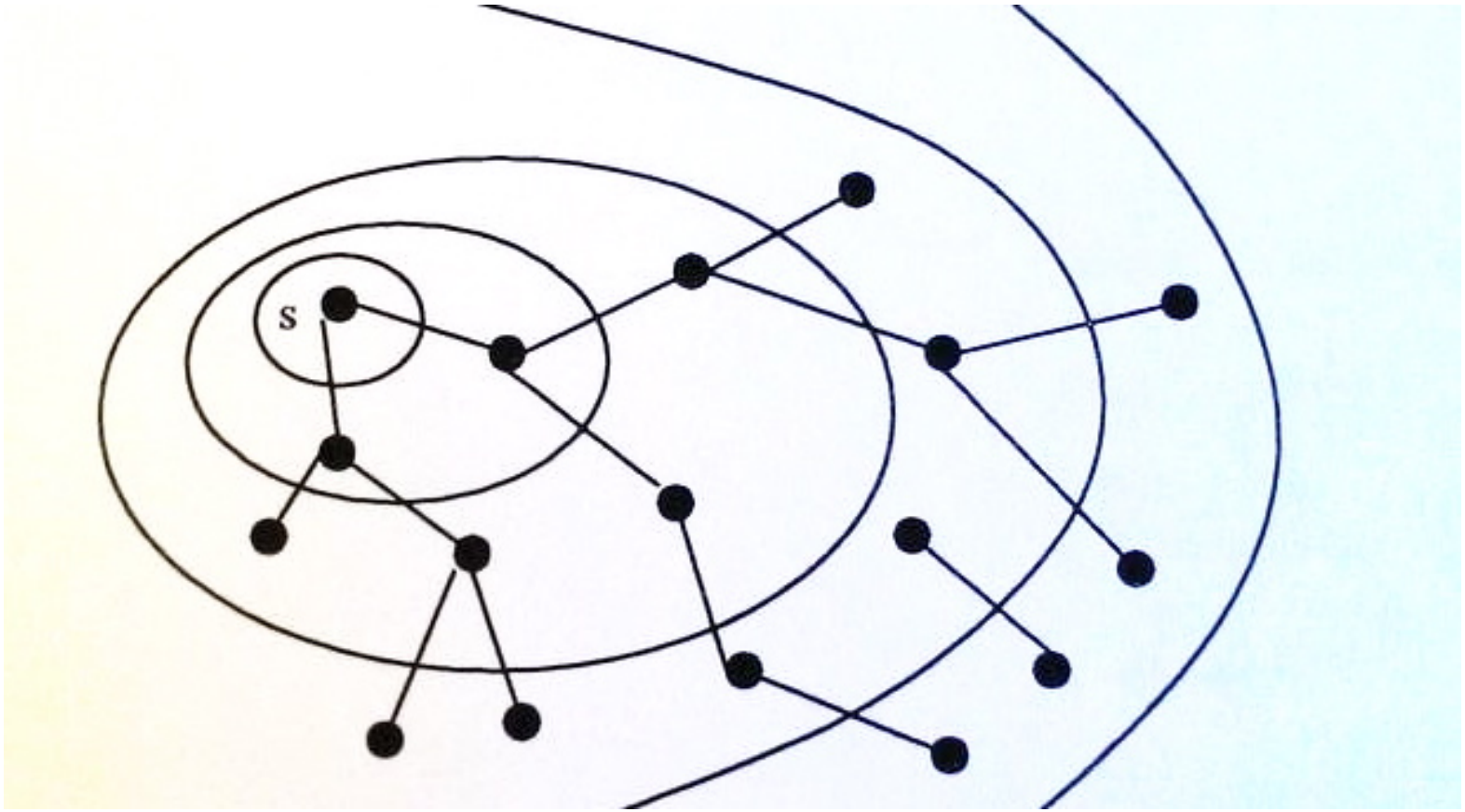


Figura: Cerca en amplada

Cerca en amplada

- L'algorisme de cerca en amplada, a partir d'un vèrtex *s*, calcula
  - un recorregut en amplada a partir de *s*
  - les distàncies mínimes de *s* a tots els vèrtexs
  - els camins mínims de *s* fins tots els vèrtexs
- La cerca en amplada funciona amb grafs
  - dirigits i no dirigits
  - sense pesos (arestes numèriques) en les arestes
- És dels algorismes de grafs més senzills i model d'altres:
  - algorisme de Dijkstra per trobar camins més curts en grafs amb pesos
  - algorisme de Prim per trobar l'arbre d'expansió mínim

Cerca en amplada

- L'algorisme de cerca en amplada, a partir d'un vèrtex *s*, calcula
  - un recorregut en amplada a partir de *s*
  - les distàncies mínimes de *s* a tots els vèrtexs
  - els camins mínims de *s* fins tots els vèrtexs
- La cerca en amplada funciona amb grafs
  - dirigits i no dirigits
  - sense pesos (arestes numèriques) en les arestes
- És dels algorismes de grafs més senzills i model d'altres:
  - algorisme de Dijkstra per trobar camins més curts en grafs amb pesos
  - algorisme de Prim per trobar l'arbre d'expansió mínim

Cerca en amplada

- L'algorisme de cerca en amplada, a partir d'un vèrtex *s*, calcula
  - un recorregut en amplada a partir de *s*
  - les distàncies mínimes de *s* a tots els vèrtexs
  - els camins mínims de *s* fins tots els vèrtexs
- La cerca en amplada funciona amb grafs
  - dirigits i no dirigits
  - sense pesos (arestes numèriques) en les arestes
- És dels algorismes de grafs més senzills i model d'altres:
  - algorisme de Dijkstra per trobar camins més curts en grafs amb pesos
  - algorisme de Prim per trobar l'arbre d'expansió mínim

Cerca en amplada

Entrada: graf $G = (V, E)$ dirigit o no dirigit i vèrtex $s \in V$.
Sortida: per a tots els vèrtexs u accessibles des de s,
dist(u) contindrà la distància de s fins a u
el camí més curt (del revés) de s a u és $(u, prev(u), prev(prev(u)), \cdots, s)$

```
bfs(G, s)
    per a tot u ∈ V
        dist(u) = ∞
        prev(u) = undef
    dist(s) = 0
    Q = [s] (cua que només conté s)
    mentre Q no sigui buida
        u = desencuar(Q)
        per a tota aresta (u, v) ∈ E
            si dist(v) = ∞ llavors
                encuar(Q, v)
                dist(v) = dist(u) + 1
                prev(v) = u
```

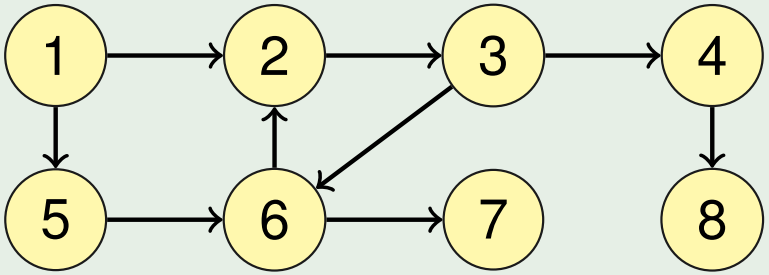
Cerca en amplada

Per a cada $d = 0, 1, 2, \ldots$ hi ha un moment en el qual:

- la cua conté els nodes a distància d
- els vèrtexs a distància $\leq d$ de s tenen la distància correcta
- els vèrtexs a distància $> d$ de s tenen la distància ∞

Cerca en amplada

Exemple 1



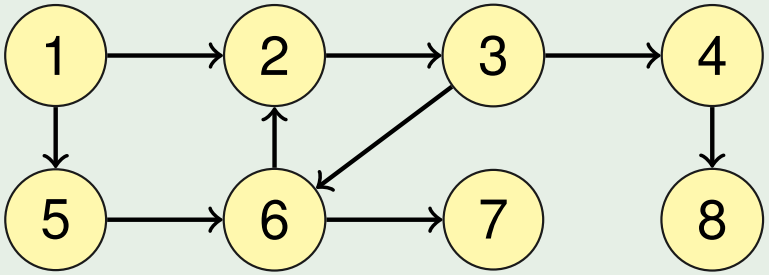
	1	2	3	4	5	6	7	8
dist	0							
prev	-							

Cua: [1]

Proper pas: treure 1 i afegir veïns amb dist ∞: 2, 5

Cerca en amplada

Exemple 1



	1	2	3	4	5	6	7	8
dist	0	1			1			
prev	-	1			1			

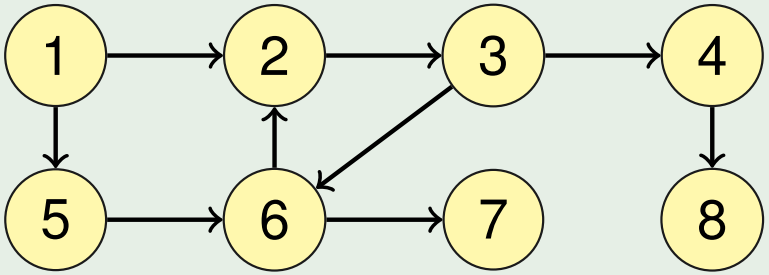
Cua: [2 5]

Proper pas: treure 2 i afegir veïns amb dist ∞: 3

Vèrtexs a distància ≤ 1 tenen *dist* correcte.
Vèrtexs a distància > 1 tenen *dist* infinit.
La cua conté tots els vèrtexs a distància 1

Cerca en amplada

Exemple 1



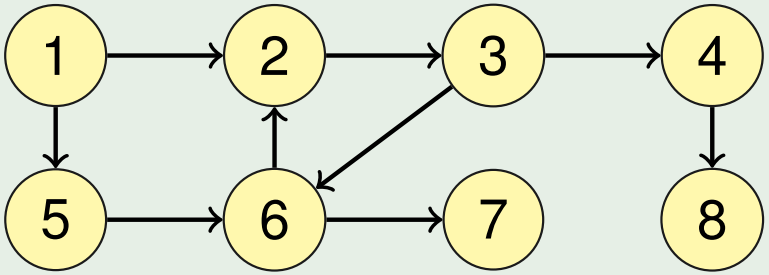
	1	2	3	4	5	6	7	8
dist	0	1	2		1			
prev	-	1	2		1			

Cua: [5 3]

Proper pas: treure 5 i afegir veïns amb dist ∞: 6

Cerca en amplada

Exemple 1



	1	2	3	4	5	6	7	8
dist	0	1	2		1	2		
prev	-	1	2		1	5		

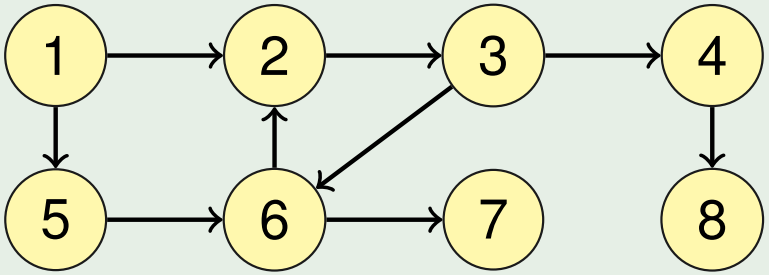
Cua: [3 6]

Proper pas: treure 3 i afegir veïns amb dist ∞: 4

Vèrtexs a distància ≤ 2 tenen *dist* correcte.
Vèrtexs a distància > 2 tenen *dist* infinit.
La cua conté tots els vèrtexs a distància 2

Cerca en amplada

Exemple 1



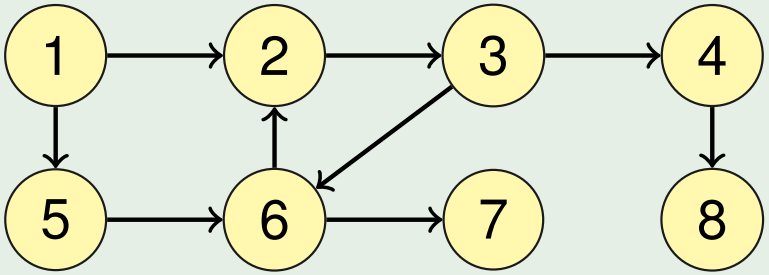
	1	2	3	4	5	6	7	8
dist	0	1	2	3	1	2		
prev	-	1	2	3	1	5		

Cua: [6 4]

Proper pas: treure 6 i afegir veïns amb dist ∞: 7

Cerca en amplada

Exemple 1



	1	2	3	4	5	6	7	8
dist	0	1	2	3	1	2	3	
prev	-	1	2	3	1	5	6	

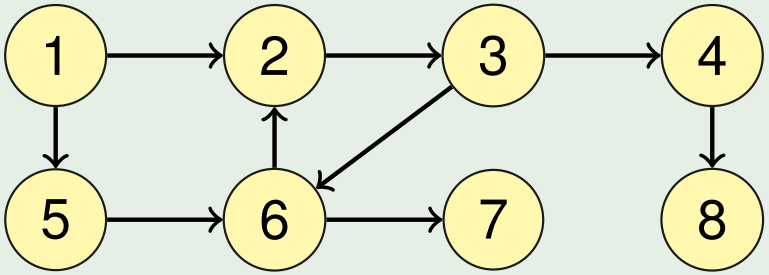
Cua: [4 7]

Proper pas: treure 4 i afegir veïns amb dist ∞: 8

Vèrtexs a distància ≤ 3 tenen *dist* correcte.
Vèrtexs a distància > 3 tenen *dist* infinit.
La cua conté tots els vèrtexs a distància 3

Cerca en amplada

Exemple 1



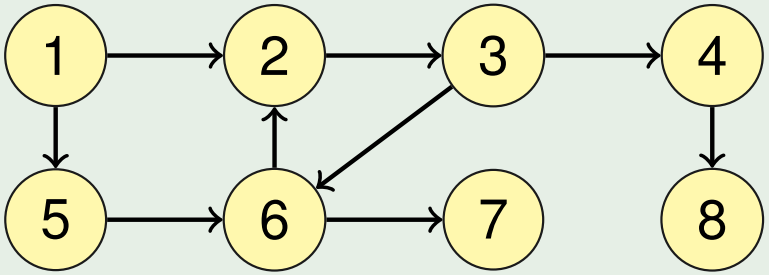
	1	2	3	4	5	6	7	8
dist	0	1	2	3	1	2	3	4
prev	-	1	2	3	1	5	6	4

Cua: [7 8]

Proper pas: treure 7 i afegir veïns amb dist ∞: cap

Cerca en amplada

Exemple 1



	1	2	3	4	5	6	7	8
dist	0	1	2	3	1	2	3	4
prev	-	1	2	3	1	5	6	4

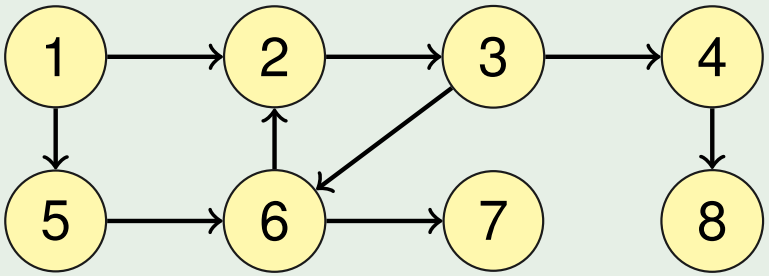
Cua: [8]

Proper pas: treure 8 i afegir veïns amb dist ∞: cap

Vèrtexs a distància ≤ 4 tenen *dist* correcte.
Vèrtexs a distància > 4 tenen *dist* infinit.
La cua conté tots els vèrtexs a distància 4

Cerca en amplada

Exemple 1

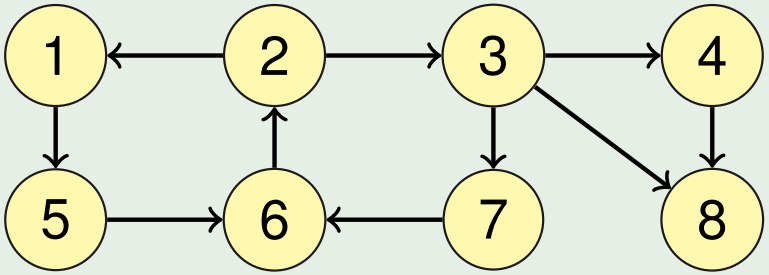


	1	2	3	4	5	6	7	8
dist	0	1	2	3	1	2	3	4
prev	-	1	2	3	1	5	6	4

Cua: [ ]

Cerca en amplada

Exemple 2



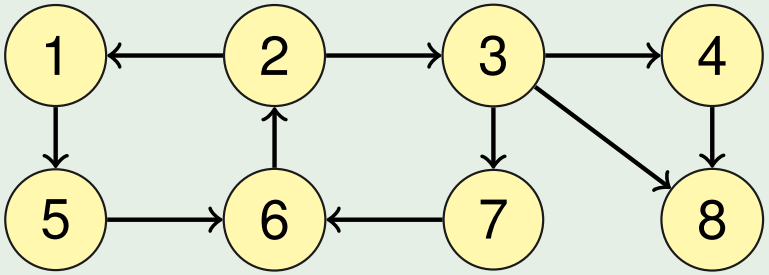
	1	2	3	4	5	6	7	8
dist	0							
prev	-							

Cua: [1]

Proper pas: treure 1 i afegir veïns amb dist ∞: 5

Cerca en amplada

Exemple 2



	1	2	3	4	5	6	7	8
dist	0				1			
prev	-				1			

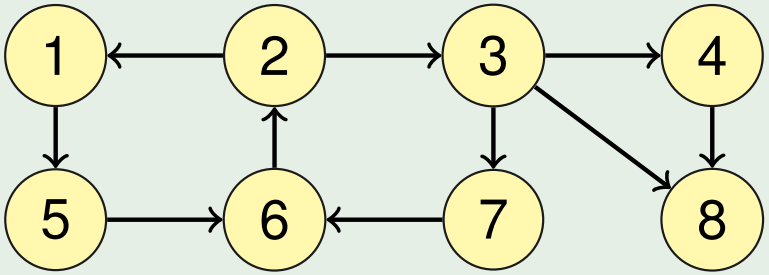
Cua: [5]

Proper pas: treure 5 i afegir veïns amb dist ∞: 6

Vèrtexs a distància ≤ 1 tenen *dist* correcte.
Vèrtexs a distància > 1 tenen *dist* infinit.
La cua conté tots els vèrtexs a distància 1

Cerca en amplada

Exemple 2



	1	2	3	4	5	6	7	8
dist	0				1	2		
prev	-				1	5		

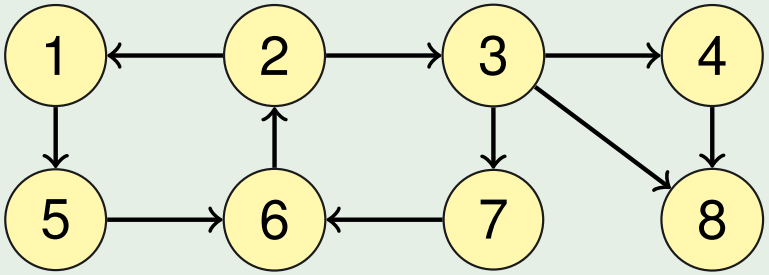
Cua: [6]

Proper pas: treure 6 i afegir veïns amb dist ∞: 2

Vèrtexs a distància ≤ 2 tenen *dist* correcte.
Vèrtexs a distància > 2 tenen *dist* infinit.
La cua conté tots els vèrtexs a distància 2

Cerca en amplada

Exemple 2



	1	2	3	4	5	6	7	8
dist	0	3			1	2		
prev	-	6			1	5		

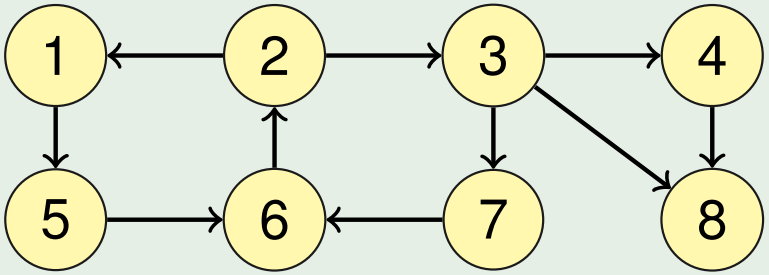
Cua: [2]

Proper pas: treure 2 i afegir veïns amb dist ∞: 3

Vèrtexs a distància ≤ 3 tenen *dist* correcte.
Vèrtexs a distància > 3 tenen *dist* infinit.
La cua conté tots els vèrtexs a distància 3

Cerca en amplada

Exemple 2



	1	2	3	4	5	6	7	8
dist	0	3	4		1	2		
prev	-	6	2		1	5		

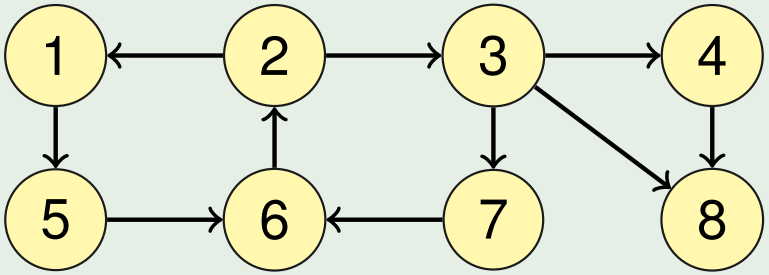
Cua: [3]

Proper pas: treure 3 i afegir veïns amb dist ∞: 4,7,8

Vèrtexs a distància ≤ 4 tenen *dist* correcte.
Vèrtexs a distància > 4 tenen *dist* infinit.
La cua conté tots els vèrtexs a distància 4

Cerca en amplada

Exemple 2



	1	2	3	4	5	6	7	8
dist	0	3	4	5	1	2	5	5
prev	-	6	2	3	1	5	3	3

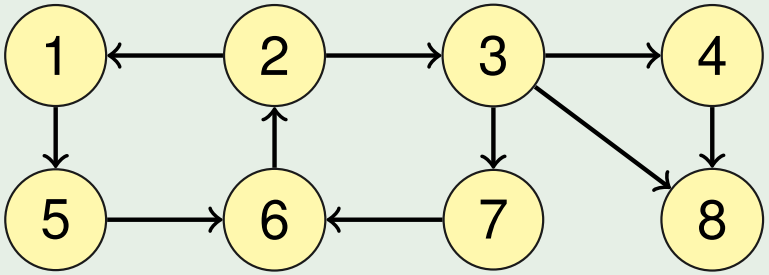
Cua: [4 7 8]

Proper pas: treure 4 i afegir veïns amb dist ∞: cap

Vèrtexs a distància ≤ 5 tenen *dist* correcte.
Vèrtexs a distància > 5 tenen *dist* infinit.
La cua conté tots els vèrtexs a distància 5

Cerca en amplada

Exemple 2



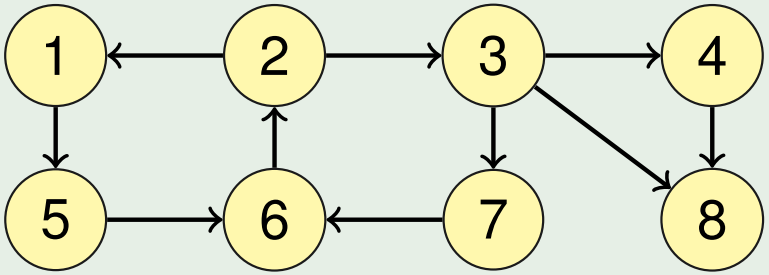
	1	2	3	4	5	6	7	8
dist	0	3	4	5	1	2	5	5
prev	-	6	2	3	1	5	3	3

Cua: [7 8]

Proper pas: treure 7 i afegir veïns amb dist ∞: cap

Cerca en amplada

Exemple 2



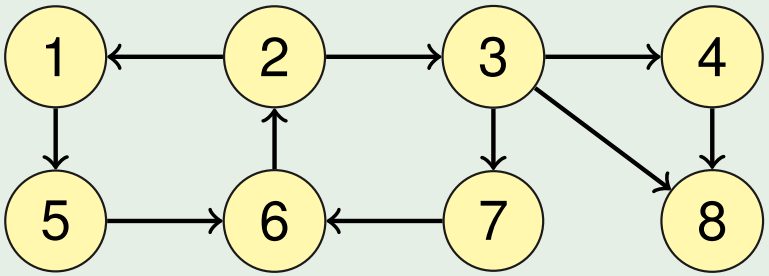
	1	2	3	4	5	6	7	8
dist	0	3	4	5	1	2	5	5
prev	-	6	2	3	1	5	3	3

Cua: [8]

Proper pas: treure 8 i afegir veïns amb dist ∞: cap

Cerca en amplada

Exemple 2



	1	2	3	4	5	6	7	8
dist	0	3	4	5	1	2	5	5
prev	-	6	2	3	1	5	3	3

Cua: [ ]

Cerca en amplada

Cerca en amplada

bfs(G, s)
    per a tot $u \in V$
        $dist(u) = \infty$
    $dist(s) = 0$
    $Q = [s]$ (cua que només conté s)
    mentre Q no sigui buida
        $u = desencuar(Q)$
        per a tota aresta $(u, v) \in E$
            si $dist(v) = \infty$ **llavors**
                $encuar(Q, v)$
                $dist(v) = dist(u) + 1$

Cost amb un graf de n vèrtexs i m arestes:

- Cada vèrtex es posa un cop a la cua: $\Theta(n)$ operacions de cua.
- Cada aresta es visita un cop (dirigits) o dos (no dirigits): $\Theta(m)$

Cost total: $\Theta(n + m)$ (amb llistes d'adjacència).

Cerca en amplada

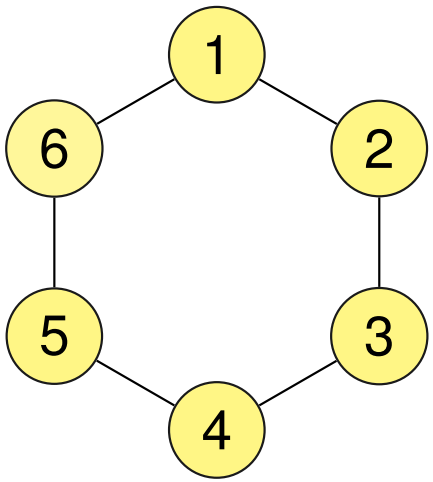
Cerca en amplada (càlcul del recorregut, no de les distàncies)

```
list<int> bfs (const graph& G) {
    int n = G.size();
    list<int> L;
    queue<int> Q;
    vector<bool> enc(n, false);
    for (int u = 0; u < n; ++u) {
        if (not enc[u]) {
            Q.push(u);
            enc[u] = true;
            while (not Q.empty()) {
                int v = Q.front(); Q.pop();
                L.push_back(v);
                for (int w : G[v]) {
                    if (not enc[w]) {
                        Q.push(w); enc[w] = true;
    }   }   }   }
    return L; }
```

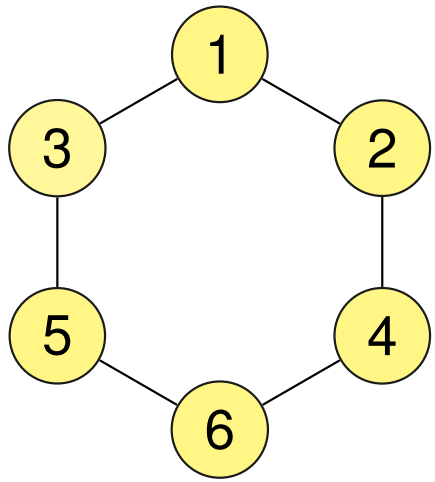
Cerca en amplada

Ara podem comparar les dues cerques: en profunditat (DFS) i en amplada (BFS):

- En DFS, la cerca torna només quan ja no queden vèrtexs per visitar (es pot arribar a fer una gran volta per visitar un veí).
- En BFS, es visiten els vèrtexs per ordre de distància creixent.



DFS

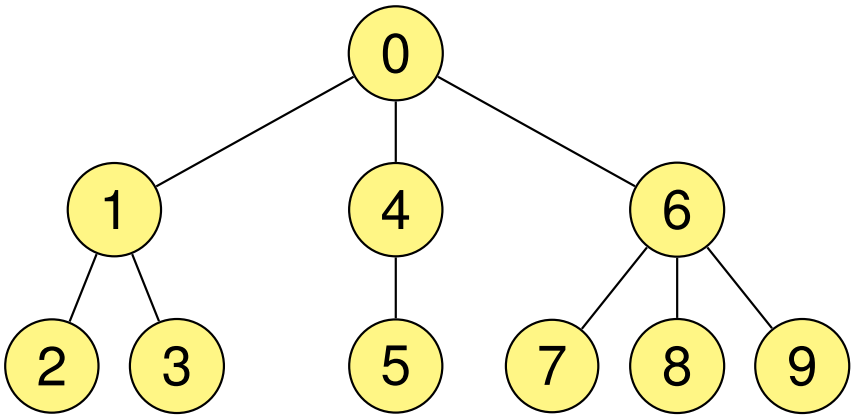


BFS

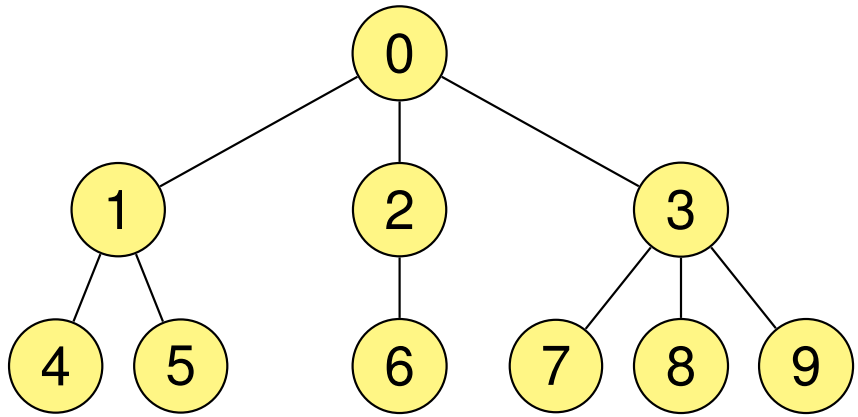
Cerca en amplada

En arbres,

- DFS correspon al recorregut en preordre i és la base del *backtracking*
- BFS els recorre per nivells



DFS



BFS

Cerca en amplada

Els codis de DFS i BFS són molt semblants.

- La diferència fonamental és l'ús
  - d'una pila en DFS
  - d'una cua en BFS

- Una altra diferència important:
  - DFS iteratiu: vis[v] indica si v ha estat visitat (tret de la pila)
  - BFS iteratiu: dist[v]/enc[v] indica si v ha estat encuat (afegit a la cua)

Cerca en amplada

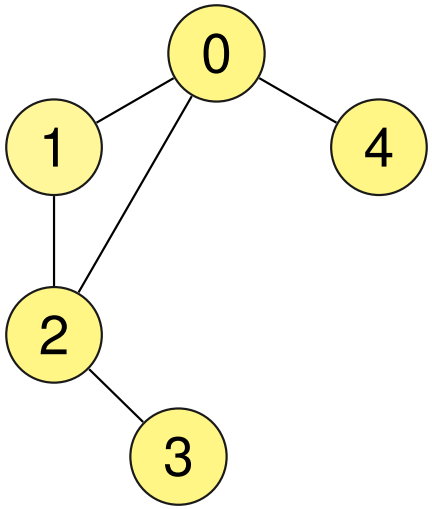
Els codis de DFS i BFS són molt semblants.

- La diferència fonamental és l'ús
  - d'una pila en DFS
  - d'una cua en BFS

- Una altra diferència important:
  - DFS iteratiu: vis[v] indica si v ha estat visitat (tret de la pila)
  - BFS iteratiu: dist[v]/enc[v] indica si v ha estat encuat (afegit a la cua)

Breadth-first search

Pren el següent graf:

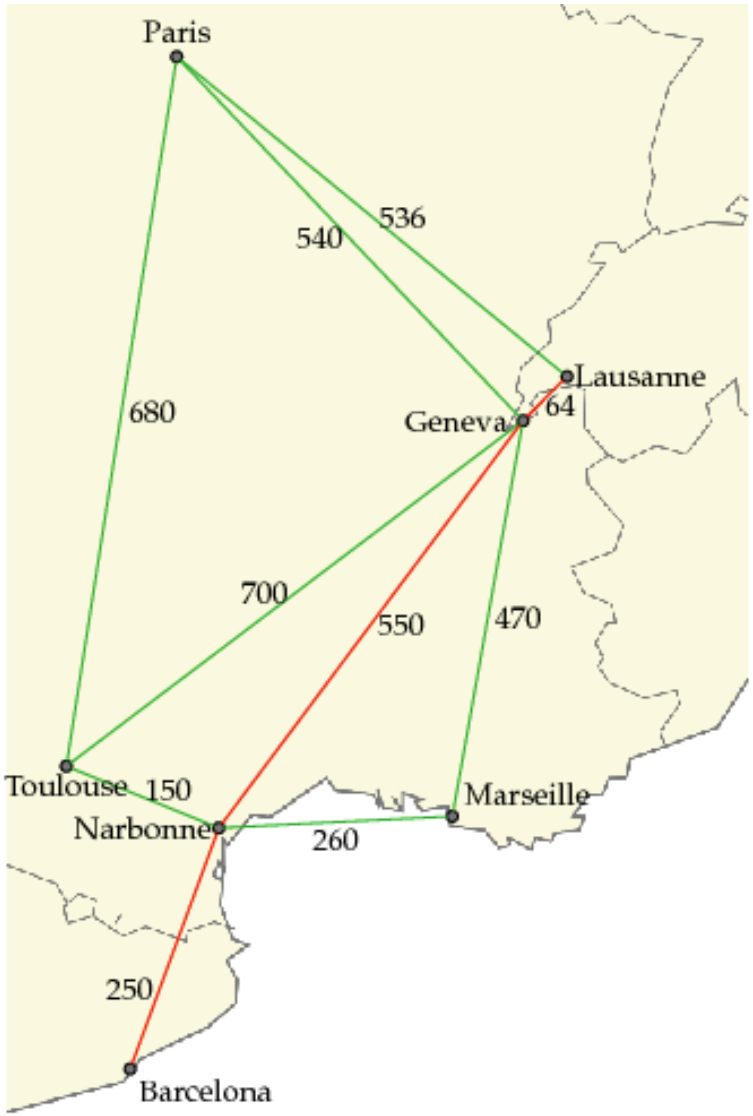


Agafa el codi C++ del BFS i reemplaça la cua per una pila.

En quin ordre es visiten els nodes?

Algorisme de Dijkstra

La cerca en amplada tracta totes les arestes com si tinguessin la mateixa llargada, poc habitual en aplicacions que requereixen camins mínims.

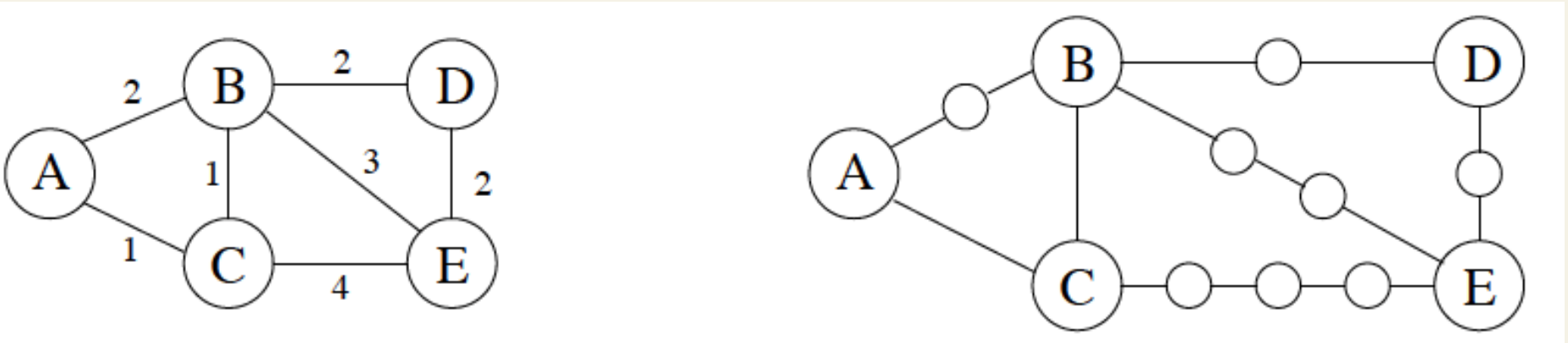


Algorisme de Dijkstra

Per adaptar la cerca en amplada a grafs etiquetats amb “pesos” naturals en les arestes, podem pensar a transformar el graf.

Truc per utilitzar BFS en grafs amb pesos

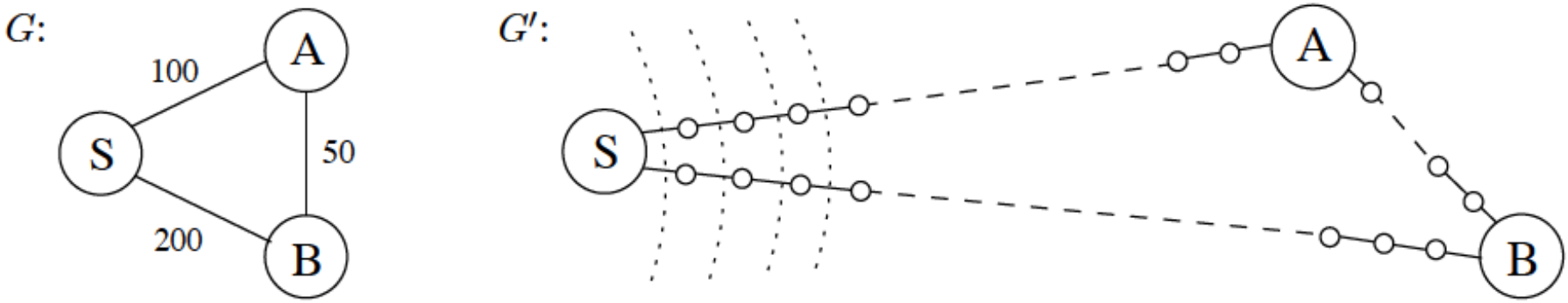
Trencar les arestes del graf d’entrada en arestes de “mida unitària” introduint vèrtexs de farciment.



Per a cada aresta $e = (u, v)$ amb etiqueta $d(e)$, substituir-la per $d(e)$ arestes amb etiqueta 1 afegint $d(e) - 1$ vèrtexs nous entre u i v.

Algorisme de Dijkstra

El truc anterior resultaria massa ineficient amb pesos grans. Però podem imaginar que posem alarmes que avisen de quan arribem d'un vèrtex (dels originals) a un altre.



Per exemple,

- Posem una alarma per a A i $T = 100$ i una altra per a B i $T = 200$.
- Quan arribem a A, el temps de B s'ajusta a $T = 150$.

Algorisme de Dijkstra

Algorisme "de les alarmes"

L'algorisme següent simula BFS sobre G' sense arribar a crear vèrtexs addicionals.

Donat el graf G amb pesos d i un vèrtex inicial S:

1. Posar l'alarma del vèrtex S a 0.
2. **Repetir** fins que no quedin alarmes:
   - Trobar l'alarma següent: vèrtex u i temps T.
   - **Per a cada** veí v de u en G
     si v no té cap alarma **o** és $> T + d(u, v)$, **llavors**
     reajustar l'alarma per a $T' = T + d(u, v)$

Algorisme de Dijkstra

L'algorisme anterior és un exemple d'algorisme voraç (*greedy*).

Esquema voraç

Un algorisme voraç resol un problema per etapes fent, a cada etapa, allò que sembla millor. Habitualment, consisteixen en una estratègia simple que es va repetint.

Exemple: donar canvi

Per donar canvi en monedes d'euro fem servir, primer, la moneda de 2 euros, després la de 1, la de 50 cèntims, la de 20, 10, 5, 2 i 1, per aquest ordre.

Estratègia: mentre no superem la quantitat a pagar, seleccionar la moneda de valor més alt.

Els algorismes voraços són eficients, però no sempre funcionen (per exemple, si afegim una moneda de 12 cèntims i n'hem de tornar 15, la solució que dóna no és bona).

Algorisme de Dijkstra

L'algorisme anterior és un exemple d'algorisme voraç (*greedy*).

Esquema voraç

Un algorisme voraç resol un problema per etapes fent, a cada etapa, allò que sembla millor. Habitualment, consisteixen en una estratègia simple que es va repetint.

Exemple: donar canvi

Per donar canvi en monedes d'euro fem servir, primer, la moneda de 2 euros, després la de 1, la de 50 cèntims, la de 20, 10, 5, 2 i 1, per aquest ordre.

Estratègia: mentre no superem la quantitat a pagar, seleccionar la moneda de valor més alt.

Els algorismes voraços són eficients, però no sempre funcionen (per exemple, si afegim una moneda de 12 cèntims i n'hem de tornar 15, la solució que dóna no és bona).

Algorisme de Dijkstra

L'algorisme de les alarmes calcula distàncies en qualsevol graf amb pesos no negatius i enters en les arestes.

Per implementar el sistema d'alarmes, triem una cua amb prioritat amb les operacions següents:

- **crear-cua**. Construir una cua amb prioritat amb els elements disponibles.
- **afegir**. Inserir un nou element a la cua.
- **treure-min**. Retornar l'element més petit i treure'l de la cua.
- **decrementar**. Decrementar un element.
- **buida**. Retorna un booleà que indica si la cua és buida.

Exercici

Com es pot aconseguir un cost $\Theta(\log n)$ per a **decrementar**?

Algorisme de Dijkstra

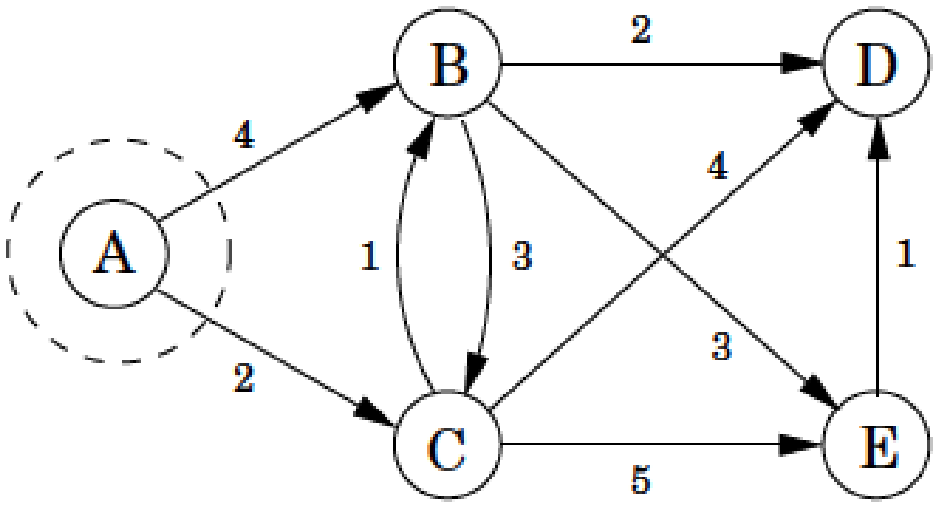
Algorisme de Dijkstra dels camins mínims

Entrada: Graf $G = (V, E)$, dirigit o no dirigit; pesos no negatius en les arestes $\{d(u, v) \mid u, v \in V\}$; vèrtex inicial s;
Sortida: Per a tot vèrtex u accessible des de s, $dist(u) =$ distància de s a u

```
Dijkstra (G,d,s)
    per a tot vèrtex u ∈ V
        dist(u) = ∞
        prev(u) = nil
    dist(s) = 0
    H =crear-cua(V)     (fent servir dist per comparar els elements)
    mentre no buida(H)
        u =esborrar-min(H)
        per a tota aresta (u, v) ∈ E
            si dist(v) > dist(u) + d(u, v)
                dist(v) = dist(u) + d(u, v)
                prev(v) = u
                decrementar(H, v)
```

Algorisme de Dijkstra

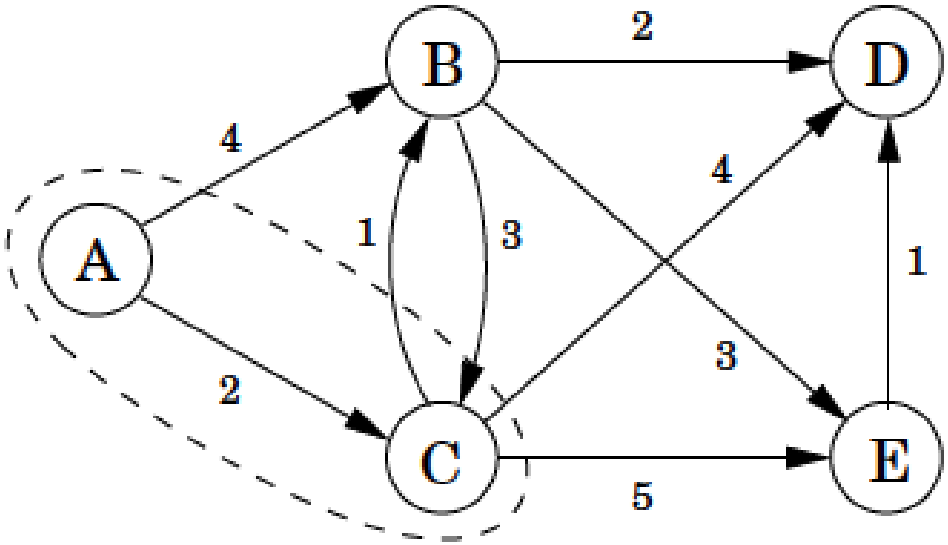
Exemple (*Algorithms*, Dasgupta et al.)



A: 0	D: ∞
B: 4	E: ∞
C: 2	

Algorisme de Dijkstra

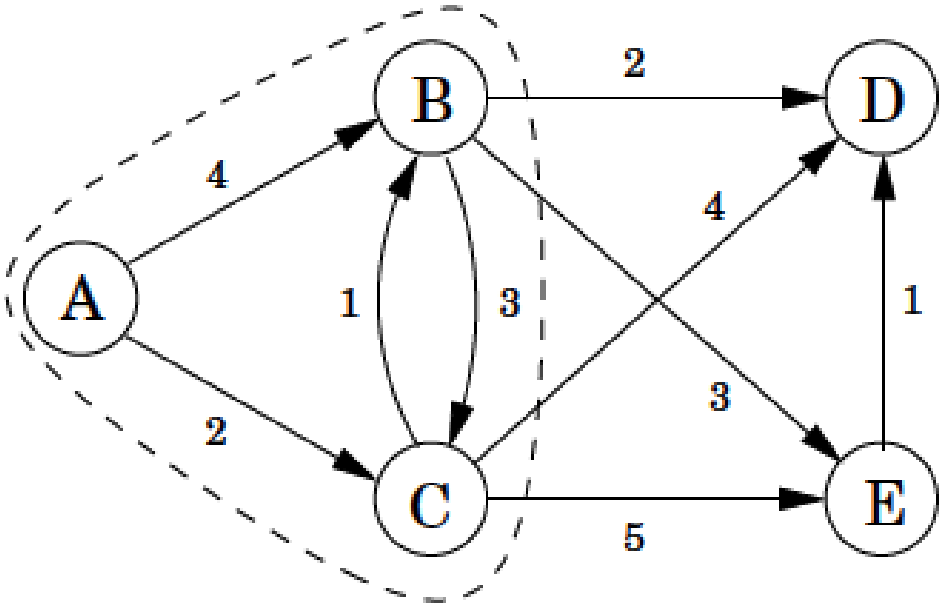
Exemple (*Algorithms*, Dasgupta et al.)



A: 0	D: 6
B: 3	E: 7
C: 2	

Algorisme de Dijkstra

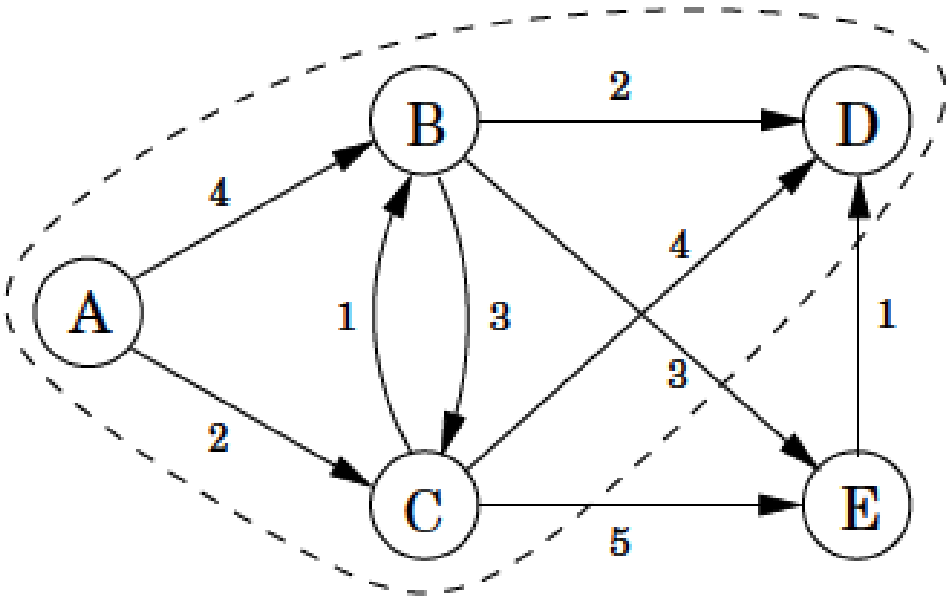
Exemple (*Algorithms*, Dasgupta et al.)



A: 0	D: 5
B: 3	E: 6
C: 2	

Algorisme de Dijkstra

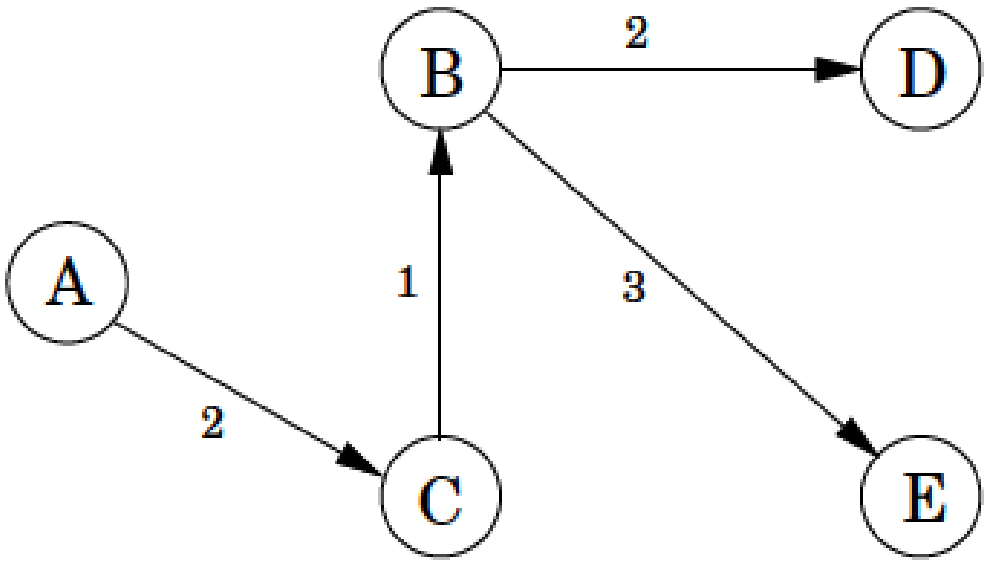
Exemple (*Algorithms*, Dasgupta et al.)



A: 0	D: 5
B: 3	E: 6
C: 2	

Algorisme de Dijkstra

Exemple (*Algorithms*, Dasgupta et al.)



Algorisme de Dijkstra

Cost de l'algorisme de Dijkstra

L'estructura és la mateixa que la de BFS, però cal tenir en compte els costos de les operacions de la cua amb prioritat. Amb un heap binari, el cost per grafs de n vèrtexs i m arestes és: $\Theta((n+m)\log n)$.

Dijkstra (G,d,s)
    per a tot vèrtex $u \in V$
        $dist(u) = \infty$
        $prev(u) = nil$
    $dist(s) = 0$
    $H =$crear-cua(V) (fent servir *dist* per comparar els elements)
    mentre no buida(H)
        $u =$esborrar-min(H)
        per a tota aresta $(u, v) \in E$
            si $dist(v) > dist(u) + d(u, v)$
                $dist(v) = dist(u) + d(u, v)$
                $prev(v) = u$
                decrementar(H, v)

Algorisme de Dijkstra

Quina cua amb prioritat és millor?

Implementació	**esborrar-min**	**afegir/ decrementar**	$n\times$ **esborrar-min** $+(n+m)\times$ **afegir**
vector	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$
heap binari	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}((n+m)\log n)$
heap d-ari*	$\mathcal{O}(\frac{d\log n}{\log d})$	$\mathcal{O}(\frac{\log n}{\log d})$	$\mathcal{O}((nd+m)\frac{\log n}{\log d})$
heap Fibonacci	$\mathcal{O}(\log n)$	$\mathcal{O}(1)^+$	$\mathcal{O}(n\log n+m)$

* Tria òptima per a d: $d = m/n$.

\+ Cost *amortitzat* ($\mathcal{O}(1)$ en mitjana al llarg de tot l'algorisme).

Algorisme de Dijkstra: implementació

Algorisme de Dijkstra

```
typedef pair<double, int> ArcP;          // arc amb pes
typedef vector< vector<ArcP> > GrafP;    // graf amb pesos

void dijkstra(const GrafP& G, int s, vector<double>& d,
                      vector<int>& p) {
    int n = G.size();
    d = vector<double>(n, infinit); d[s] = 0;
    p = vector<int>(n, -1);
    vector<bool> S(n, false);
    priority_queue<ArcP, vector<ArcP>, greater<ArcP> > Q;
    Q.push(ArcP(0, s));
```

Algorisme de Dijkstra: implementació

Algorisme de Dijkstra

```
    while (not Q.empty()) {
        int u = Q.top().second; Q.pop();
        if (not S[u]) {
            S[u] = true;
            for (int i = 0; i < int(G[u].size()); ++i) {
                int v = G[u][i].second;
                int c = G[u][i].first;
                if (d[v] > d[u] + c) {
                    d[v] = d[u] + c;
                    p[v] = u;
                    Q.push(ArcP(d[v], v));
}   }   }   }   }
```

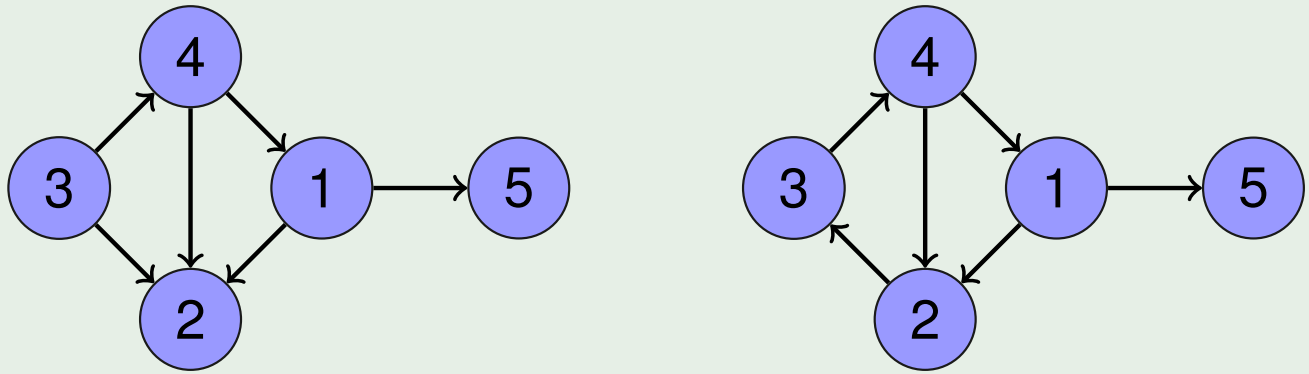
Ordenació topològica

Definició

Un graf dirigit acíclic s'acostuma a anomenar dag.

Els dags expressen precedències o causalitats i són una eina utilitzada en moltes disciplines (informàtica, economia, medicina,...)

Exemple

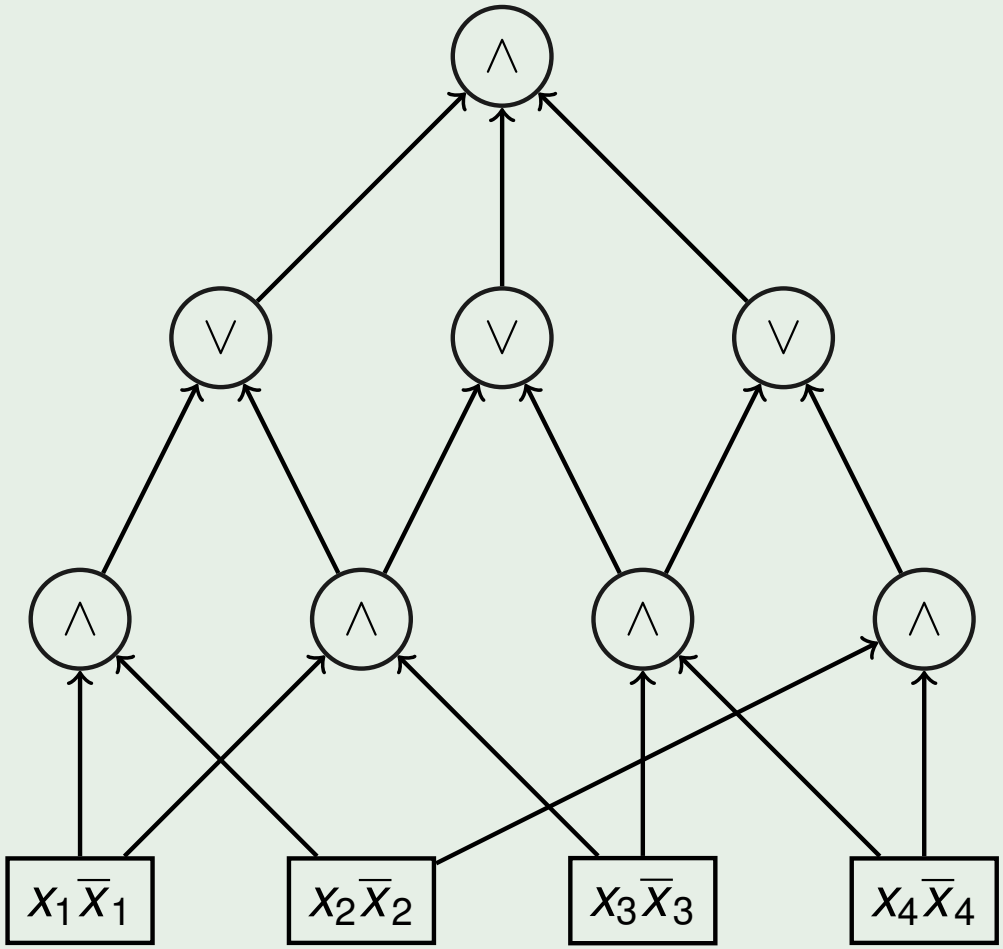


El digraf de l'esquerra és un dag; el de la dreta, no.

Ordenació topològica

Exemple

Un circuit també és un dag.



Ordenació topològica

Definició

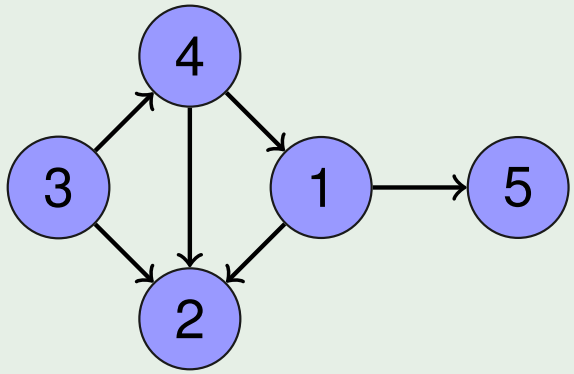
Una ordenació topològica d'un dag $G = (V, E)$ és una seqüència

$$v_1, v_2, v_3, \ldots, v_n$$

tal que $V = \{v_1, \ldots, v_n\}$ i si $(v_i, v_j) \in E$, llavors $i < j$.

Exemple

Un dag pot tenir més d'una ordenació topològica.



Tant 3,4,1,2,5 com 3,4,1,5,2 són ordenacions topològiques.

Ordenació topològica

Definició

Una ordenació topològica d'un dag $G = (V, E)$ és una seqüència

$$v_1, v_2, v_3, \ldots, v_n$$

tal que $V = \{v_1, \ldots, v_n\}$ i si $(v_i, v_j) \in E$, llavors $i < j$.

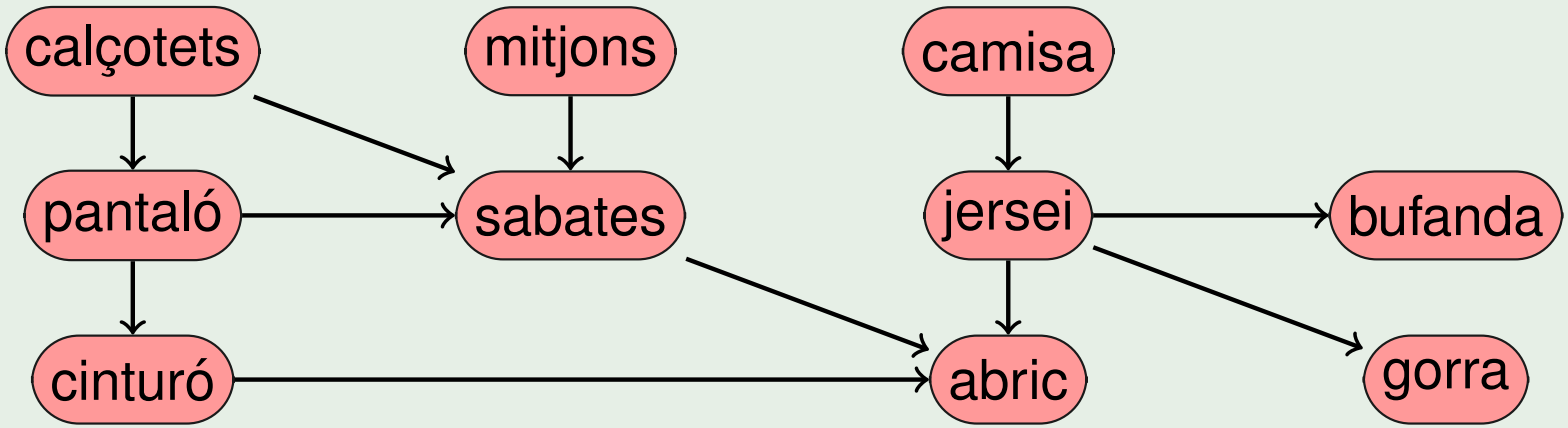
Exercicis

1. Doneu exemples de dags de n vèrtexs que tinguin
   - una única ordenació topològica
   - $(n-1)!$ ordenacions topològiques
2. Digueu per què si un dag de n vèrtexs té $n!$ ordenacions topològiques, ha de consistir en n vèrtexs aïllats.

Ordenació topològica

Exemple

Roba masculina d'hivern



Possibles ordenacions:

- calçotets, mitjons, pantaló, camisa, cinturó, jersei, sabates, abric, bufanda, gorra
- mitjons, camisa, calçotets, jersei, pantaló, cinturó, bufanda, gorra, sabates, abric

Ordenació topològica

Problema de l'ordenació topològica

Donat un dag, trobar una ordenació topològica.

Algorisme (esquema voraç)

Repetir fins que no quedin vèrtexs:

1. Trobar un vèrtex u amb grau d'entrada 0.
2. Escriure u i esborrar-lo del graf.

Cost

Si el graf té n vèrtexs,

- cercar un vèrtex amb grau d'entrada 0 costa $\Theta(n)$
- el pas anterior es repeteix n vegades

Cost total: $\Theta(n^2)$.

Ordenació topològica

Problema de l'ordenació topològica

Donat un dag, trobar una ordenació topològica.

Algorisme (esquema voraç)

Repetir fins que no quedin vèrtexs:

1. Trobar un vèrtex u amb grau d'entrada 0.
2. Escriure u i esborrar-lo del graf.

Cost

Si el graf té n vèrtexs,

- cercar un vèrtex amb grau d'entrada 0 costa $\Theta(n)$
- el pas anterior es repeteix n vegades

Cost total: $\Theta(n^2)$.

Ordenació topològica

Problema de l'ordenació topològica

Donat un dag, trobar una ordenació topològica.

Algorisme (esquema voraç)

Repetir fins que no quedin vèrtexs:

1. Trobar un vèrtex u amb grau d'entrada 0.
2. Escriure u i esborrar-lo del graf.

Cost

Si el graf té n vèrtexs,

- cercar un vèrtex amb grau d'entrada 0 costa $\Theta(n)$
- el pas anterior es repeteix n vegades

Cost total: $\Theta(n^2)$.

Ordenació topològica

El cost es pot millorar si fem més eficient la cerca d'un vèrtex de grau 0. Necessitarem:

- Un vector per emmagatzemar el grau d'entrada de cada vèrtex.
- Una estructura (pila, cua, etc.) que contingui els vèrtexs de grau 0.

Inicialitzem una pila amb els vèrtexs de grau 0. Mentre no sigui buida:

1. Treiem un vèrtex de la pila, l'escrivim i reajustem els graus.
2. Empilem els nous vèrtexs de grau 0.

Ordenació topològica

El cost es pot millorar si fem més eficient la cerca d'un vèrtex de grau 0. Necessitarem:

- Un vector per emmagatzemar el grau d'entrada de cada vèrtex.
- Una estructura (pila, cua, etc.) que contingui els vèrtexs de grau 0.

Inicialitzem una pila amb els vèrtexs de grau 0. Mentre no sigui buida:

1. Treiem un vèrtex de la pila, l'escrivim i reajustem els graus.
2. Empilem els nous vèrtexs de grau 0.

Ordenació topològica

Preparació del vector i de la pila d'un graf de n vèrtexs i m arestes. Cost $\Theta(n+m)$.

```
list<int> ordenacio_topologica (graph& G) {
    int n = G.size();
    vector<int> ge(n, 0);
    for (int u = 0; u < n; ++u) {
        for (int i = 0; i < G[u].size(); ++i) {
            ++ge[G[u][i]];
    }   }

    stack<int> S;
    for (int u = 0; u < n; ++u) {
        if (ge[u] == 0) {
            S.push(u);
    }   }
```

Ordenació topològica

Bucle principal.

```
    list<int> L;
    while (not S.empty()) {
        int u = S.top(); S.pop();
        L.push_back(u);
        for (int i = 0; i < G[u].size(); ++i) {
            int v = G[u][i];
            if (--ge[v] == 0) {
                S.push(v);
    }   }   }
    return L;
}
```

Es visita

- un cop cada vèrtex
- un cop cada aresta

Si el graf es representa amb llistes d'adjacència, el cost és $\Theta(n+m)$.

Arbres d'Expansió Mínims

Sigui $G = (V, E)$ un graf no dirigit connex amb pesos $\omega : E \to \mathbb{R}$.

Definició

Un arbre d'expansió (en anglès, spanning tree) de G és un subgraf $T = (V_A, A)$ de G que

- és un arbre (és a dir, és connex i acíclic)
- conté tots els vèrtexs de G (és a dir, $V_A = V$)

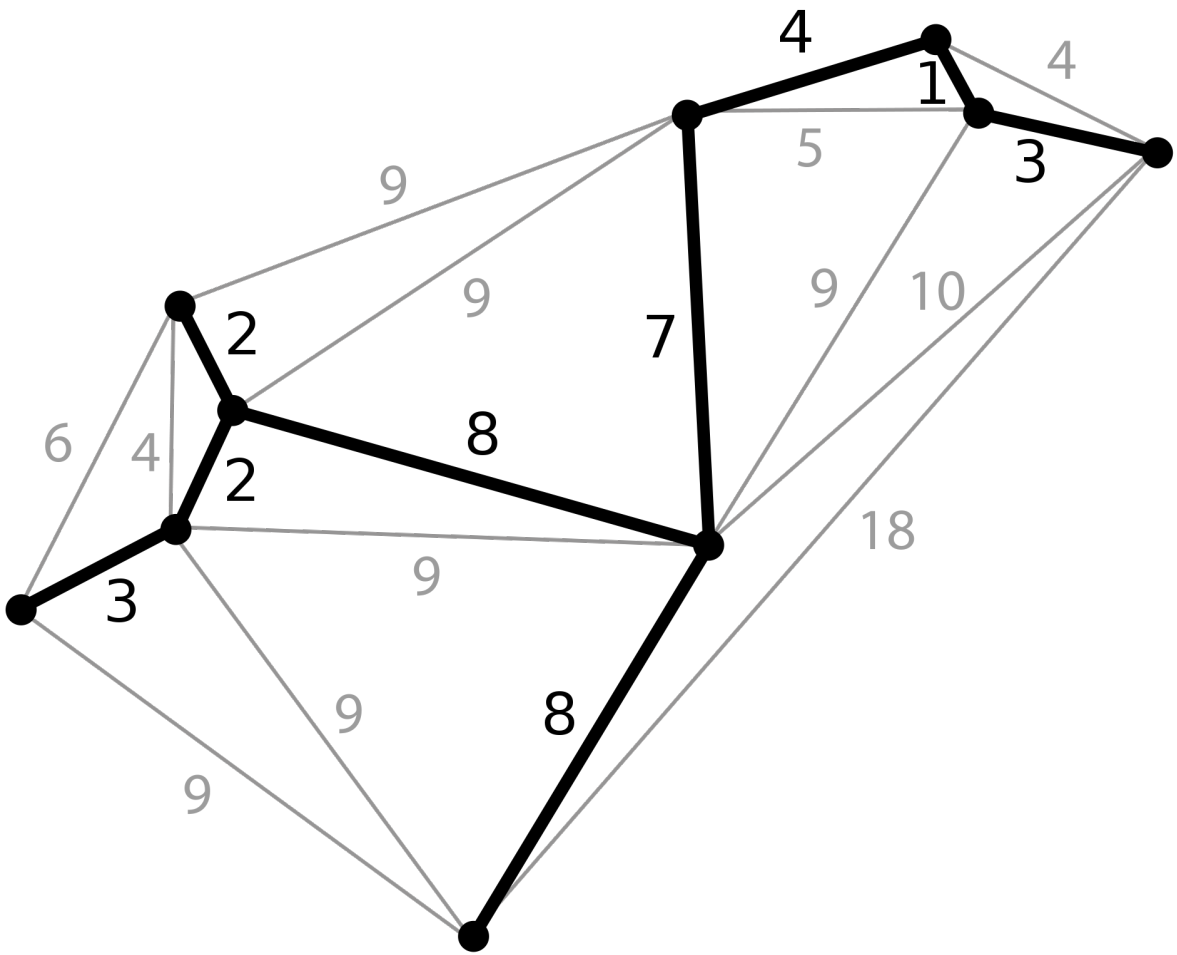
Definició

Un arbre d'expansió mínim (en anglès, minimum spanning tree, MST) de G és un arbre d'expansió $T = (V, A)$ de G tal que el seu pes total

$$\omega(T) = \sum_{e \in A} \omega(e)$$

és mínim entre tots els arbres d'expansió de G

Arbres d'Expansió Mínims



Font: `https://commons.wikimedia.org/wiki/File:Minimum_spanning_tree.svg`

Arbres d'Expansió Mínims

Hi ha molts algorismes diferents per calcular arbres d'expansió mínims.
Tots ells segueixen el següent esquema voraç:

```
A = ∅;
Cand = E;
while (|A| ≠ |V| − 1) {
   escollir e ∈ Cand que no tanca un cicle en T = (V, A);
   A = A ∪ {e};
   Cand = Cand − {e}
}
```

Arbres d'Expansió Mínims

Definició

Un subconjunt d'arestes $A \subseteq E$ és prometedor si
A és un subconjunt de les arestes d'un MST de G

Definició

Un tall en un graf G és una partició del conjunt de vèrtexs V, és a dir, un parell (C, C') tal que $C, C' \neq \emptyset$ i

- $C \cup C' = V$
- $C \cap C' = \emptyset$

Definició

Una aresta e respecta un tall (C, C') si els extrems de e pertanyen tots dos a C o tots dos a C'; altrament, diem que e creua el tall.

Arbres d'Expansió Mínims

Teorema

Sigui A un conjunt prometedor d'arestes que respecta un tall (C, C') de G.
Sigui e una aresta de pes mínim entre les que creuen el tall (C, C').
Aleshores $A \cup \{e\}$ és prometedor.

Aquest teorema dóna una recepta per dissenyar algorismes per a MST:

1. comença amb un conjunt buit d'arestes A
2. defineix quin és el tall de G
3. escull l'aresta e amb pes mínim entre les que creuen el tall
4. afegeix e a A (com que A respecta el tall, això no pot crear un cicle)

Tot algorisme que segueixi aquest esquema és automàticament correcte.

Demostració del teorema

Sigui A' el conjunt d'arestes d'un MST T tal que $A \subseteq A'$
(T existeix perquè suposem que A és prometedor).

Si $e \in A'$ llavors $A \cup \{e\}$ és prometedor i el teorema es compleix.

Suposem ara $e \notin A'$.

Com que A respecta el tall, hi ha una aresta $e' \in A' - A$ que creua el tall
(si no, T no seria connex).

El subgraf $T' = (V, A' - \{e'\} \cup \{e\})$ és un arbre d'expansió i per tant

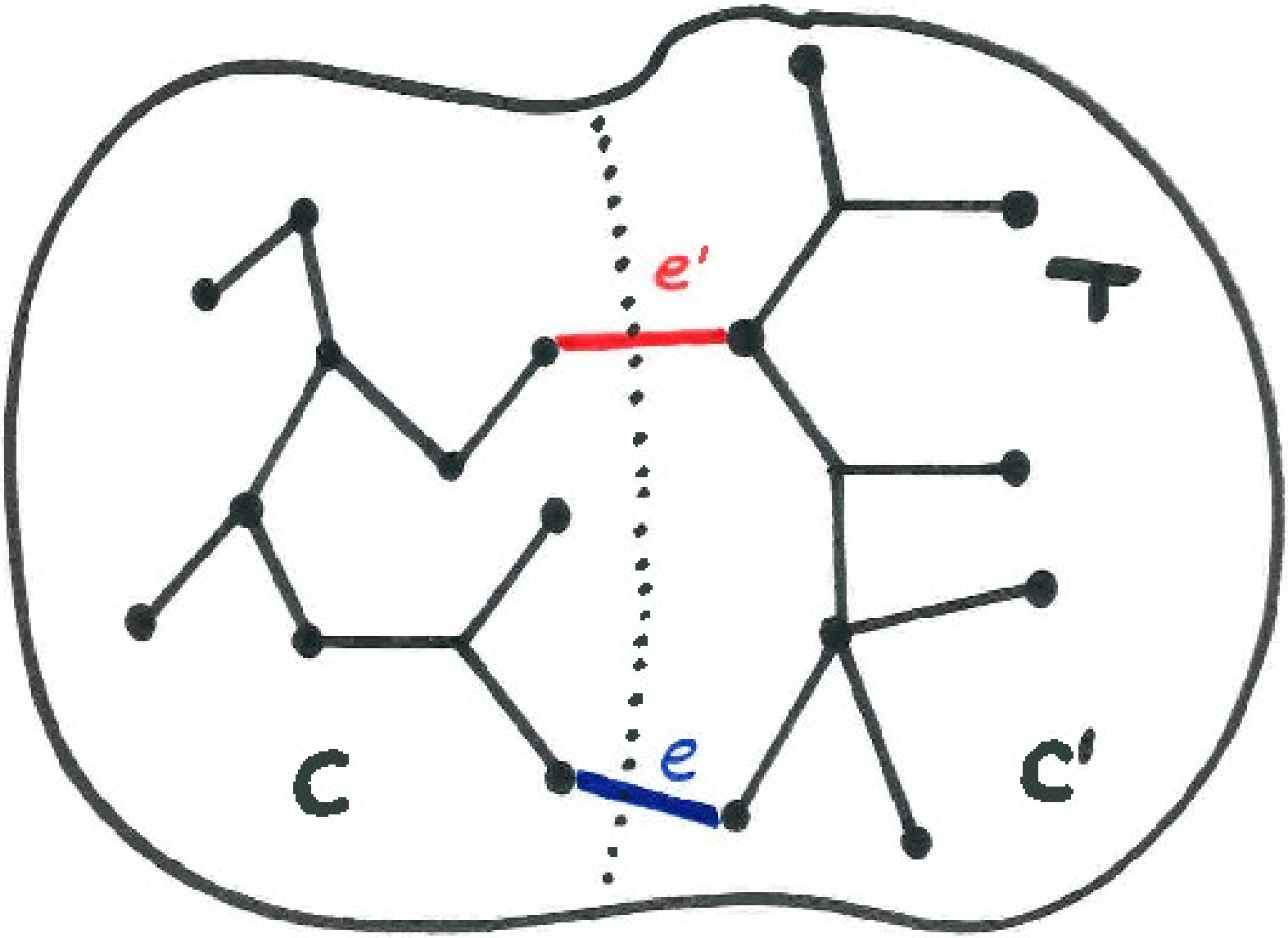
$$\omega(T) \leq \omega(T') = \omega(T) - \omega(e') + \omega(e)$$

d'on $\omega(e') \leq \omega(e)$.

Però per definició de e, tenim $\omega(e') \geq \omega(e)$, i per tant $\omega(T) = \omega(T')$.

Com que T' és un MST, $A \cup \{e\}$ és prometedor.

Arbres d'Expansió Mínims



Algorisme de Prim

A l'algorisme de Prim (també conegut com algorisme de Prim-Jarník), mantenim un subconjunt de vèrtexs visitats.

El conjunt de vèrtexs queda doncs partit entre el visitats i els no visitats.

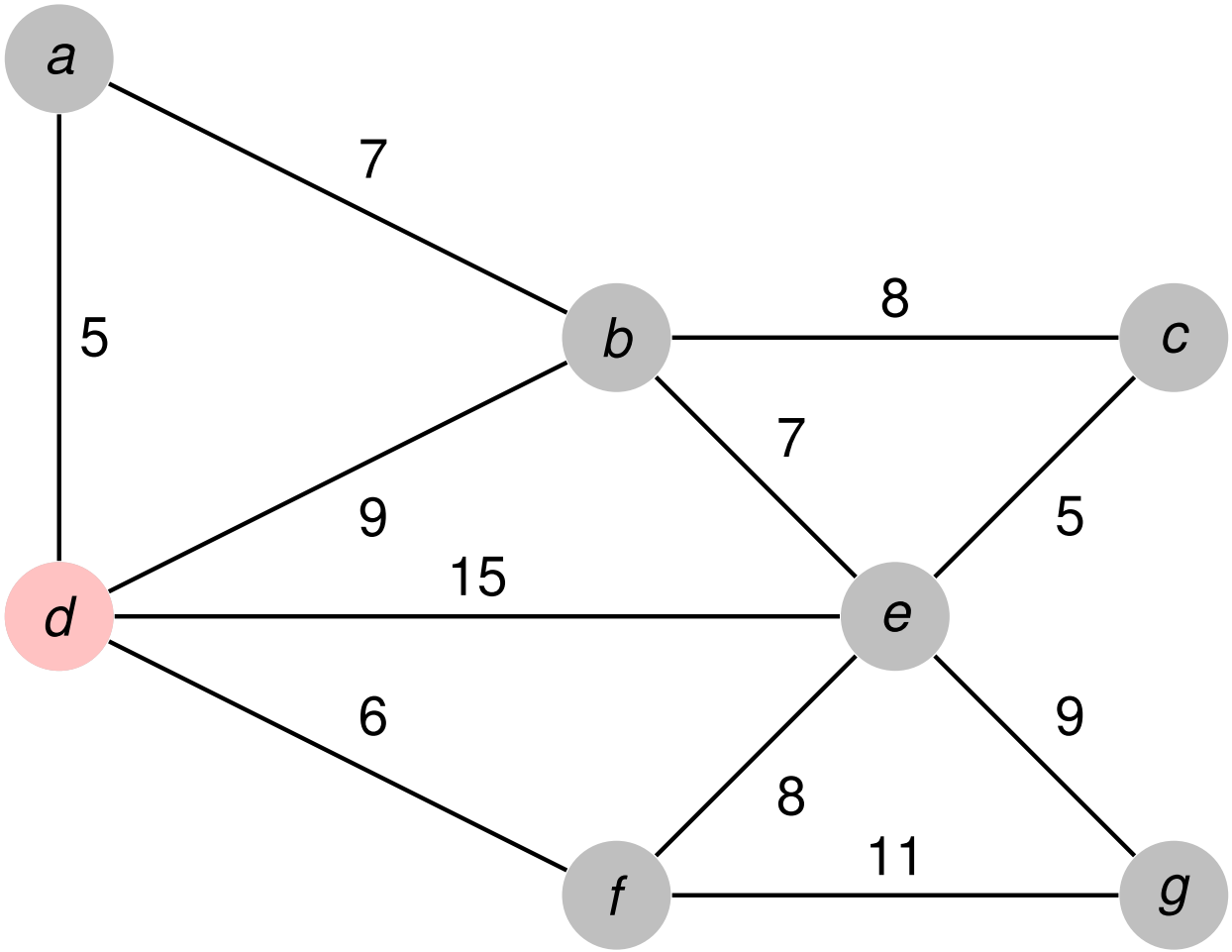
Cada iteració de l'algorisme escull l'aresta de pes mínim entre les que uneixen un vèrtex visitat i un que no.

Pel teorema, l'algorisme és correcte.

Algorisme de Prim

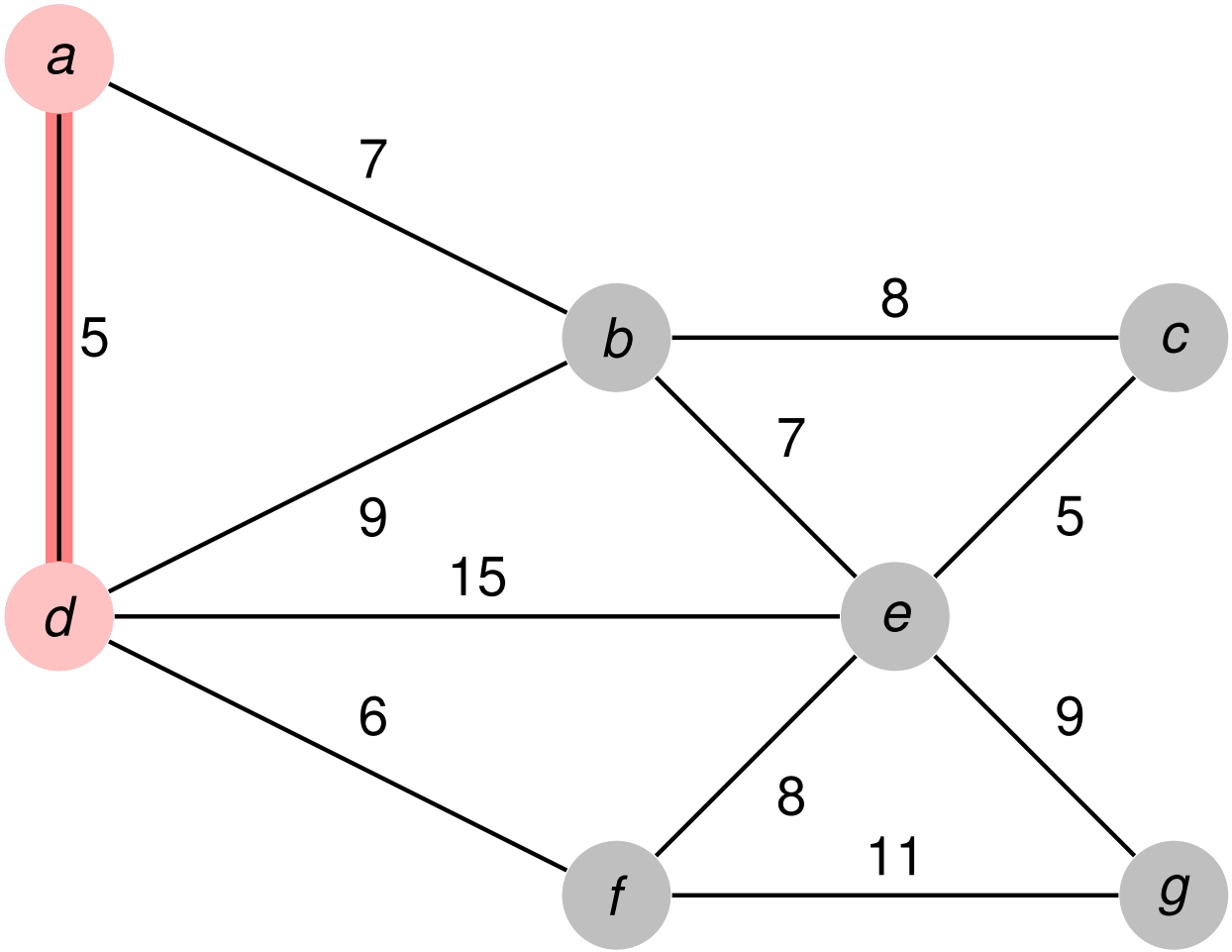
```
typedef pair<int,int> P;

// El graf (connex) esta representat amb llistes d'adjacencia
// Els parells son (cost, vertex)
int mst(const vector<vector<P>>& g) { // Retorna cost d'un MST
  vector<bool> vis(n, false);
  vis[0] = true;
  priority_queue<P, vector<P>, greater<P> > pq;
  for (P x : g[0]) pq.push(x);
  int sz = 1;
  int sum = 0;
  while (sz < n) {
    int c = pq.top().first;
    int x = pq.top().second;
    pq.pop();
    if (not vis[x]) {
      vis[x] = true;
      for (P y : g[x]) pq.push(y);
      sum += c;
      ++sz; } }
  return sum; }
```

Algorisme de Prim



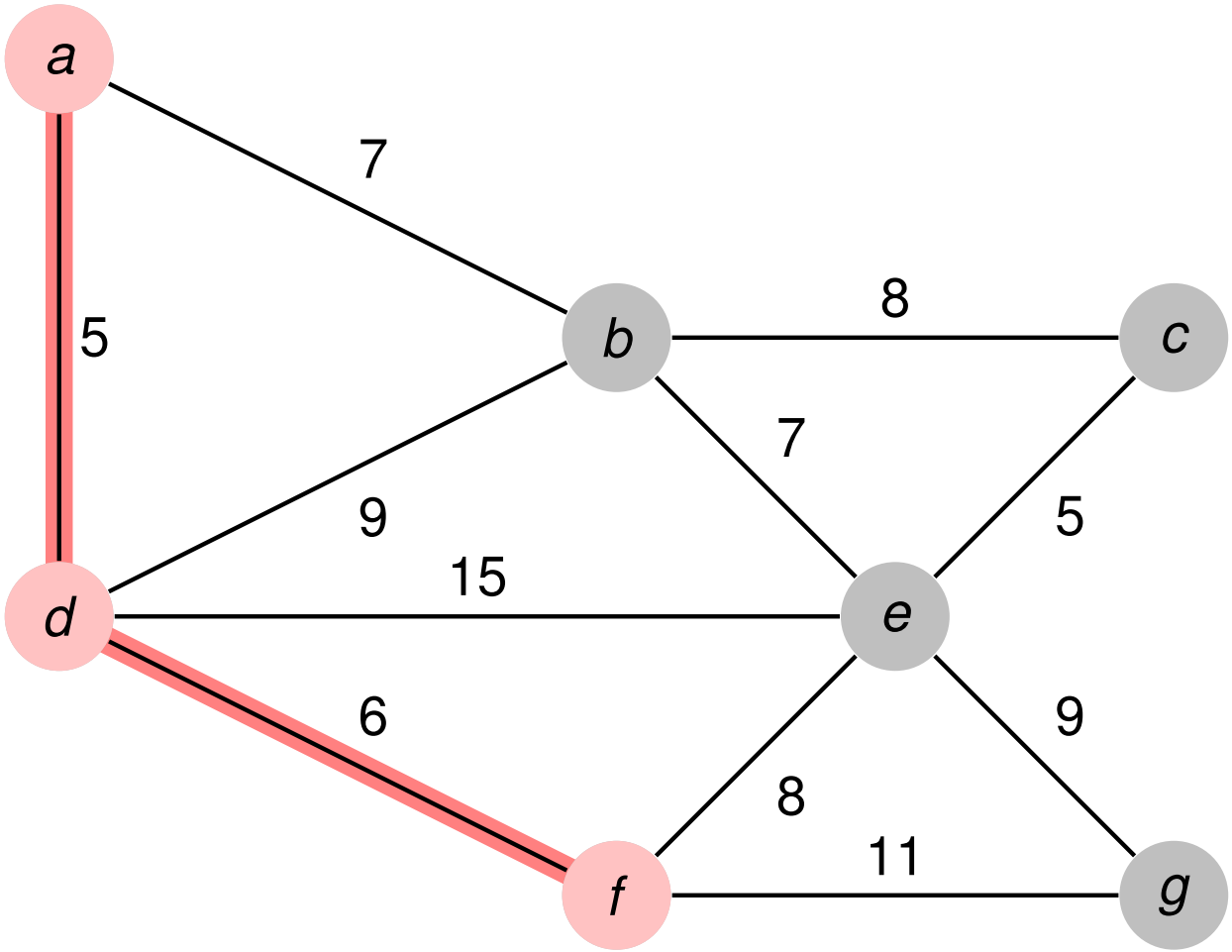
Font: `www.texample.net/tikz/examples/author/kjell-magne-fauske`

Algorisme de Prim



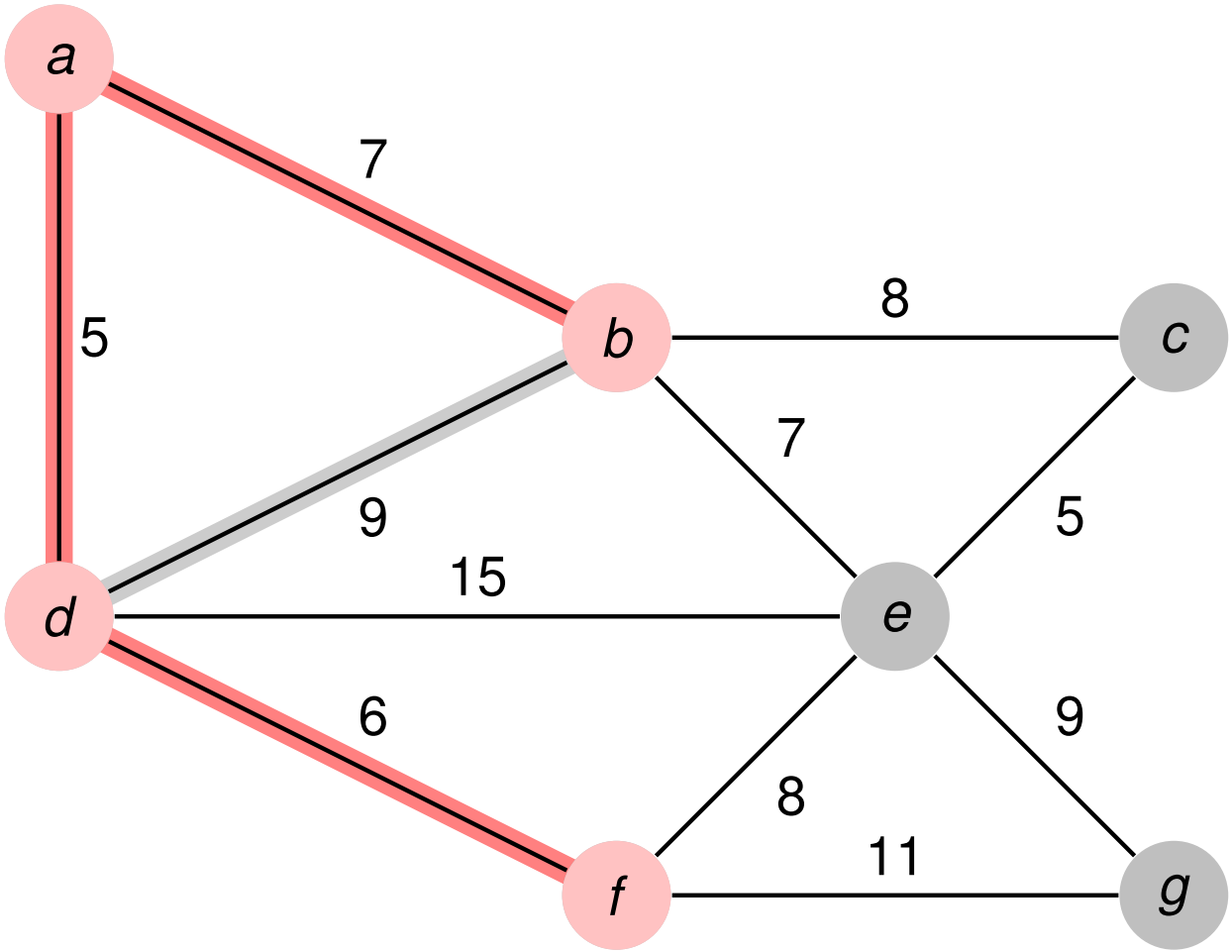
Font: `www.texample.net/tikz/examples/author/kjell-magne-fauske`

Algorisme de Prim



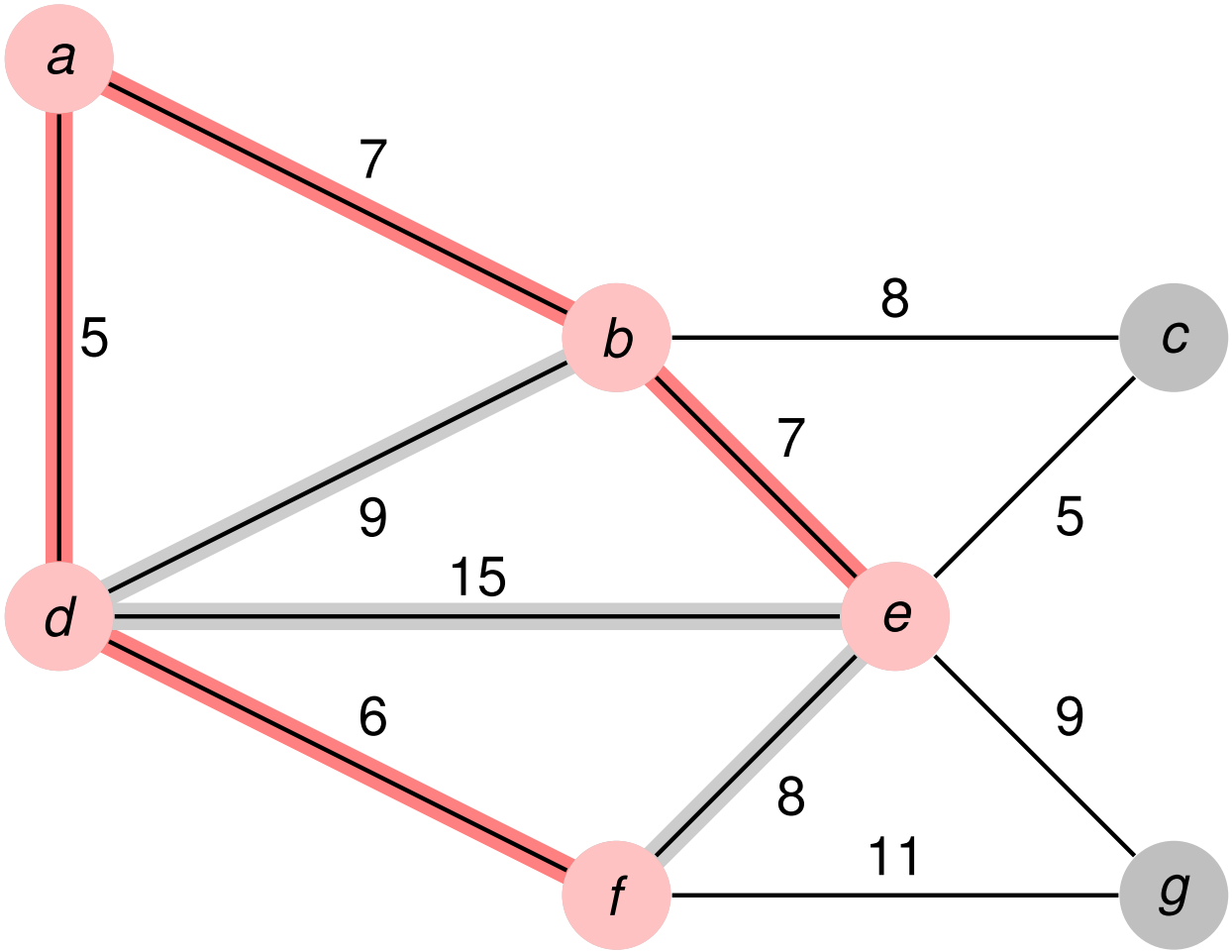
Font: `www.texample.net/tikz/examples/author/kjell-magne-fauske`

Algorisme de Prim



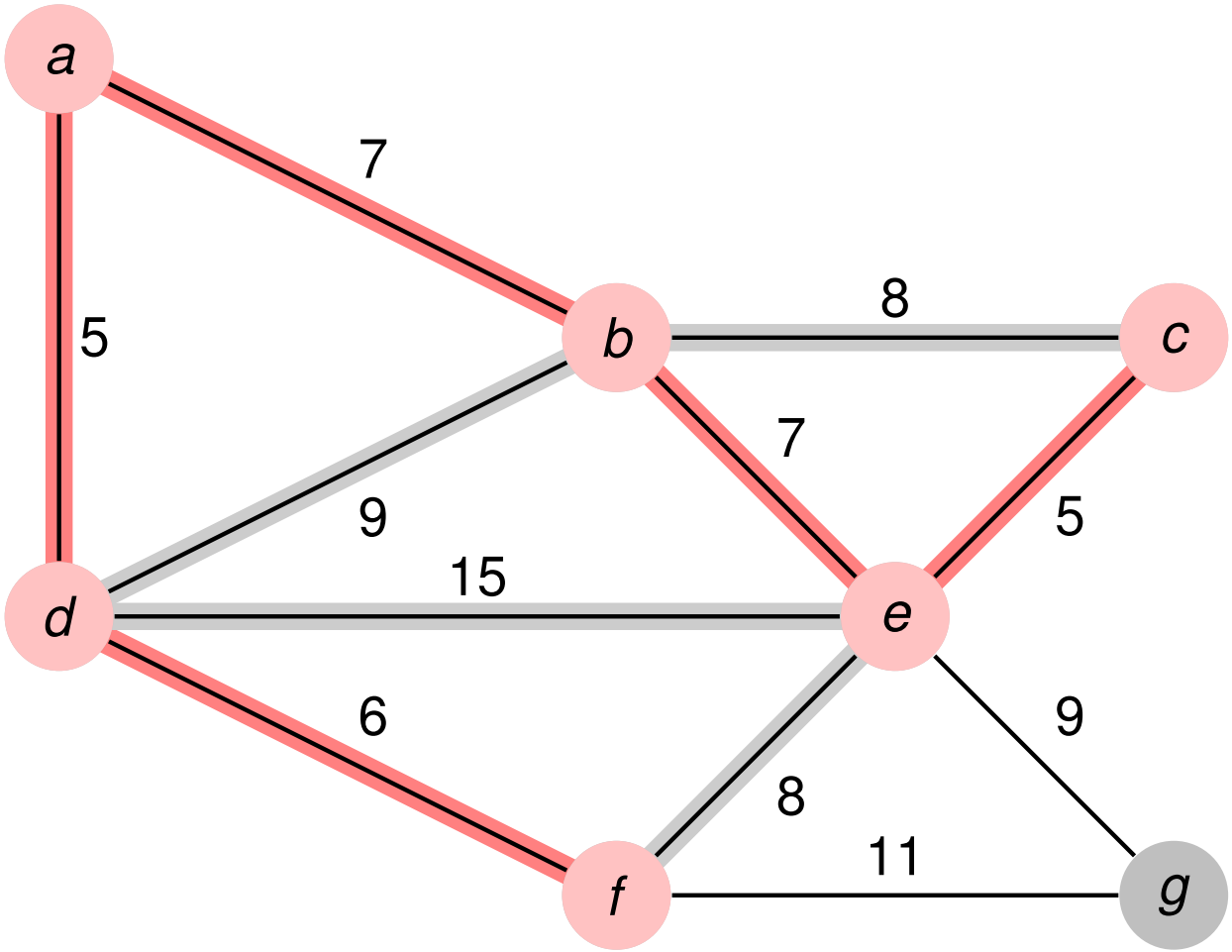
Font: `www.texample.net/tikz/examples/author/kjell-magne-fauske`

Algorisme de Prim



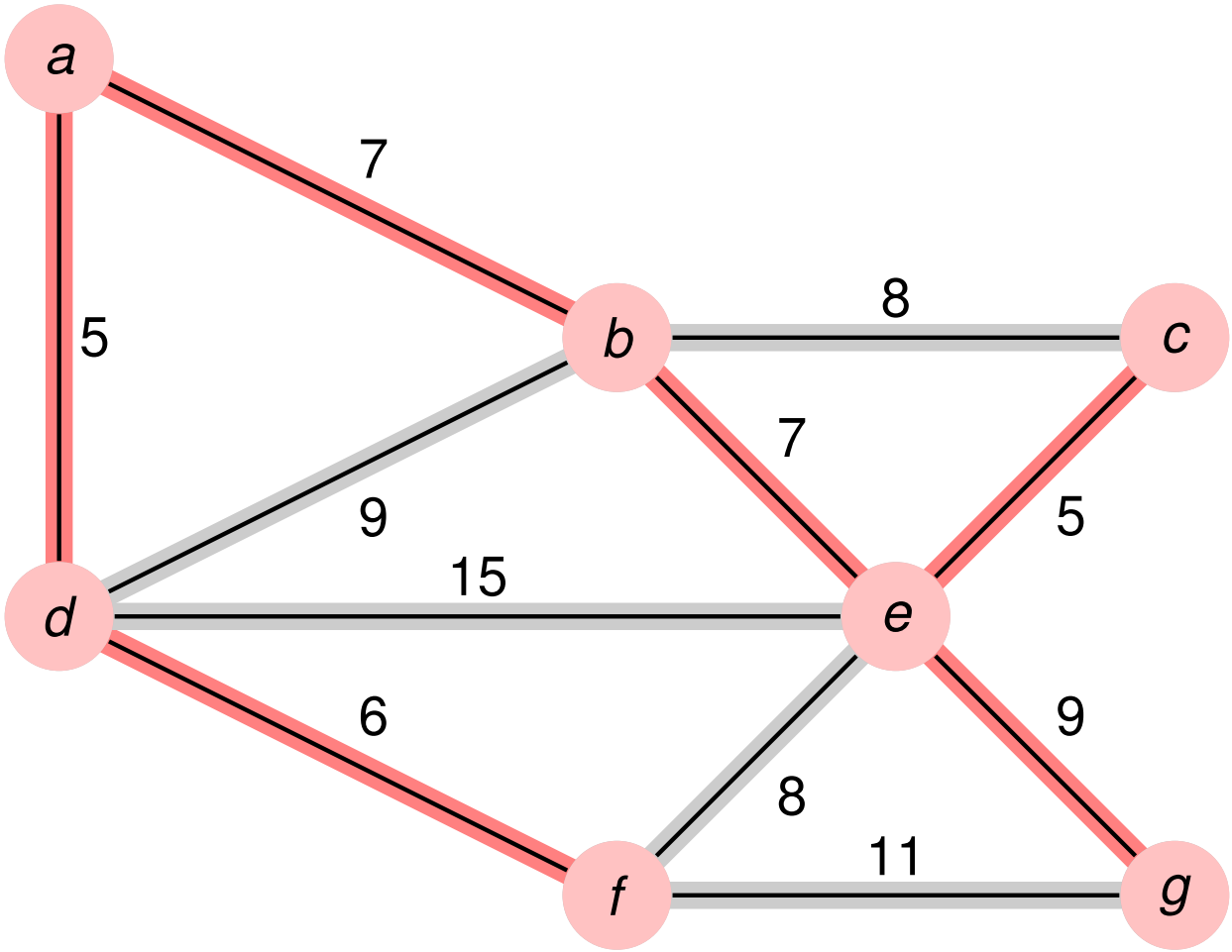
Font: `www.texample.net/tikz/examples/author/kjell-magne-fauske`

Algorisme de Prim



Font: `www.texample.net/tikz/examples/author/kjell-magne-fauske`

Algorisme de Prim



Font: `www.texample.net/tikz/examples/author/kjell-magne-fauske`

Algorisme de Prim

A la implementació anterior, el cost del bucle domina el cost de l'algorisme.

Es fan $O(m)$ voltes.
La selecció de l'aresta a cada volta costa temps $O(\log m)$.
En total seleccionar l'aresta té cost $O(m \log m)$.

Cada vèrtex és marcat exactament una vegada.
Quan es visita el vèrtex x, afegir nous candidats costa $O(\deg(x) \log m)$.

En total:

$$O(m \log m) + \sum_{x \in V} O(\deg(x) \log m) = O(m \log m) = O(m \log n)$$

De fet es pot veure que en el cas pitjor efectivament el cost és $\Theta(m \log n)$.