

Tema 5. Grafs

Estructures de Dades i Algorismes

FIB

Antoni Lozano
(amb material d'Enric Rodríguez)

Q2 2017–2018
Versió de 24 d'abril de 2018

1 Propietats

- Introducció
- Grafs
- Grafs dirigits i etiquetats
- Representacions

2 Algorismes elementals

- Cerca en profunditat
- Cerca en amplada
- Ordenació topològica

3 Distàncies mínimes

- Algorisme de Dijkstra
- Arbres d'expansió mínims

1 Propietats

- Introducció
- Grafs
- Grafs dirigits i etiquetats
- Representacions

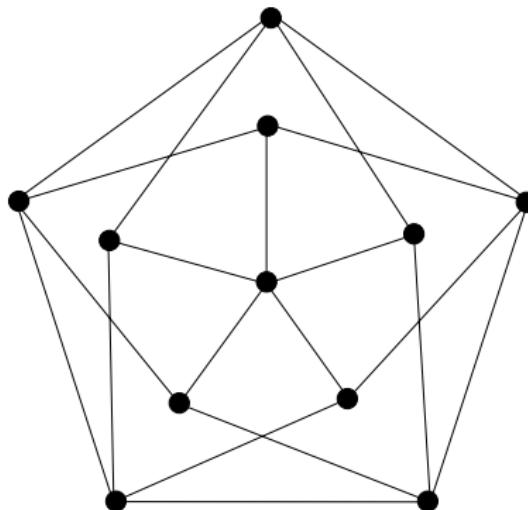
2 Algorismes elementals

- Cerca en profunditat
- Cerca en amplada
- Ordenació topològica

3 Distàncies mínimes

- Algorisme de Dijkstra
- Arbres d'expansió mínims

Per què els grafs?



Perquè hi ha molts problemes que es poden expressar de manera **clara i acurada** mitjançant els 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 els països veïns tinguin colors diferents?

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

- fronteres irregulars
- mars
- punts de confluència de més de dos països

Però tot mapa el podem representar com un graf planar:

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

Exemple: acolorir un mapa amb el mínim nombre de colors

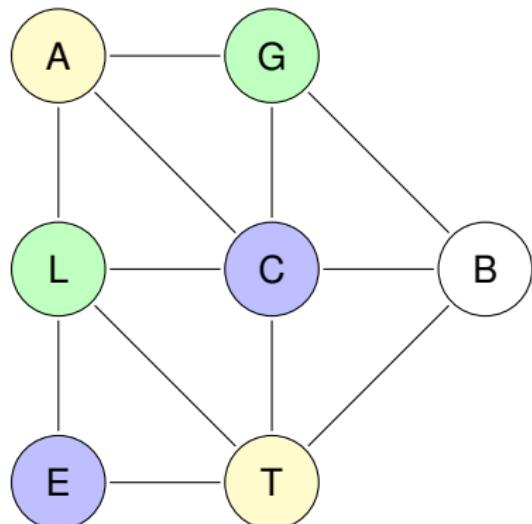
Quin és el mínim nombre de colors necessari per acolorir un mapa de manera que els països veïns tinguin colors diferents?

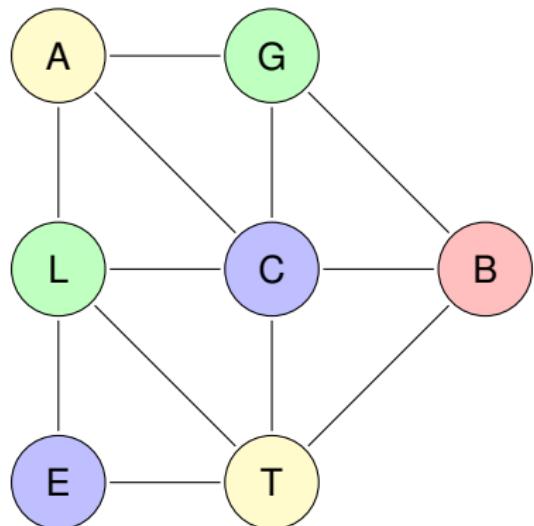
Si analitzem un mapa real, trobarem tota mena d'informació irrellevant:

- fronteres irregulars
- mars
- punts de confluència de més de dos països

Però tot mapa el podem representar com un graf planar:

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





Fent servir els grafs, podem utilitzar el teorema següent.

Teorema dels quatre colors (Appel/Haken, 1976)

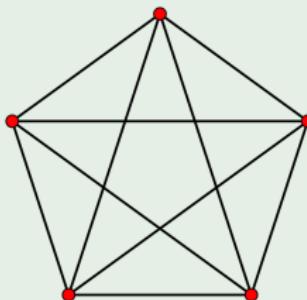
Tot graf planar es pot acolorir amb 4 colors.

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

Exemple: connectar cinc objectes en el pla

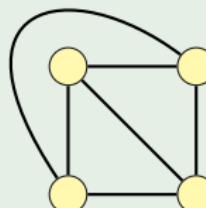
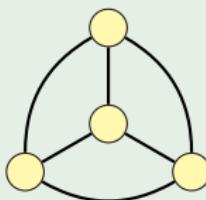
No es poden connectar 5 objectes sobre el pla sense creuar connexions.

En teoria de grafs, és el mateix que dir que el graf K_5

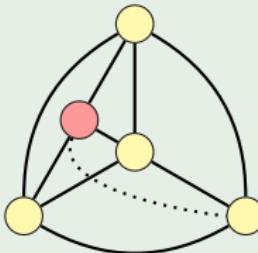
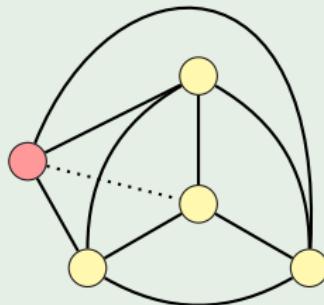


no es pot dibuixar en el pla sense creuar arestes (no és *planar*).

Es pot argumentar que K_5 ha de contenir K_4 (el graf complet de 4 vèrtexs) que, dibuixat, sempre defineix 4 àrees en el pla (1 externa i 3 internes):



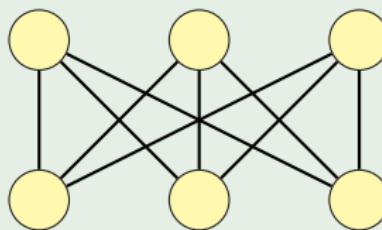
Tant si el cinquè vèrtex es dibuixa en l'àrea externa com en una d'interna, hi haurà un vèrtex amb el qual no pot connectar-se sense creuar alguna aresta.



Exemple: connectar tres objectes amb d'altres tres

No es poden connectar 3 cases a 3 serveis (aigua, llum i gas) sobre un mateix pla sense creuar connexions.

En teoria de grafs, és el mateix que dir que el graf $K_{3,3}$ no és planar:



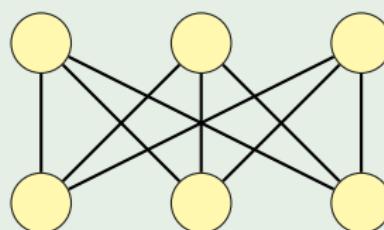
Exercici

Argumenteu de manera semblant a l'exemple de K_5 per deduir que $K_{3,3}$ no és planar.

Exemple: connectar tres objectes amb d'altres tres

No es poden connectar 3 cases a 3 serveis (aigua, llum i gas) sobre un mateix pla sense creuar connexions.

En teoria de grafs, és el mateix que dir que el graf $K_{3,3}$ no és planar:



Exercici

Argumenteu de manera semblant a l'exemple de K_5 per deduir que $K_{3,3}$ no és planar.

Teorema (Kuratowski, 1922)

Un graf és planar si i només si no conté un subgraf que sigui una subdivisió de K_5 o $K_{3,3}$.

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 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.
- S'introduceix el graf en un algorisme que troba un cicle que passa per tots els vèrtexs (cicle hamiltonià): aquest serà l'ordre en què 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 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.
- S'introduceix el graf en un algorisme que troba un cicle que passa per tots els vèrtexs (cicle hamiltonià): aquest serà l'ordre en què han de seure.

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 París?
- **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, mapes de carreteres, processos industrials...**

Definició

Un graf és un **parell** (V, E), on

- V és un conjunt finit (**vèrtexs**)
- E és un conjunt de parells no ordenats de vèrtexs (**arestes**)

Observacions

Donat un graf $G = (V, E)$:

- no existeix més d'una aresta entre dos vèrtexs
(perquè E és un conjunt, no un multiconjunt)
- no conté bucles, és a dir, arestes (u, u) per a cap $u \in V$
(perquè una aresta és un parell no ordenat \equiv conjunt de 2 elements)

Definició

Un graf és un **parell** (V, E) , on

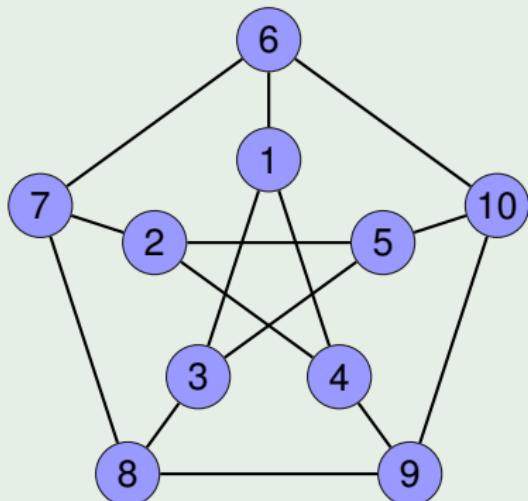
- V és un conjunt finit (**vèrtexs**)
- E és un conjunt de parells no ordenats de vèrtexs (**arestes**)

Observacions

Donat un graf $G = (V, E)$:

- no existeix més d'una aresta entre dos vèrtexs
(perquè E és un conjunt, no un multiconjunt)
- no conté bucles, és a dir, arestes (u, u) per a cap $u \in V$
(perquè una aresta és un parell no ordenat \equiv conjunt de 2 elements)

Exemple: graf de Petersen

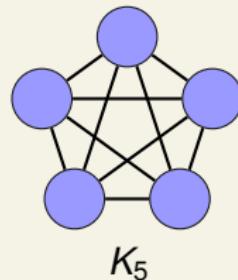
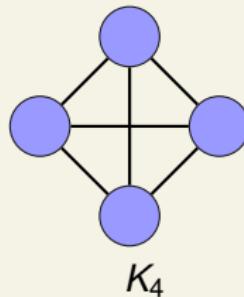
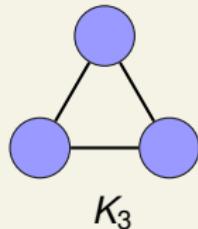


Formalment, és $GP(5, 2) = (V, E)$ on

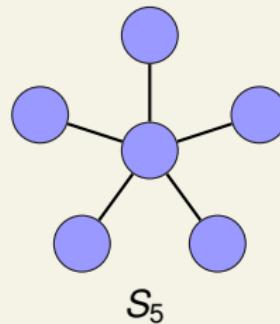
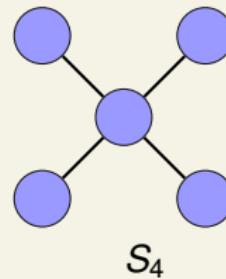
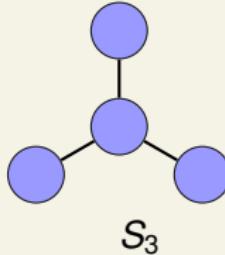
- $V = \{1, \dots, 10\}$
- $E = \{\{1, 3\}, \{1, 4\}, \{1, 6\}, \{2, 4\}, \{2, 5\}, \{2, 7\}, \{3, 5\}, \{3, 8\}, \{4, 9\}, \{5, 10\}, \{6, 7\}, \{6, 10\}, \{7, 8\}, \{8, 9\}, \{9, 10\}\}$

Tipus de grafs

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

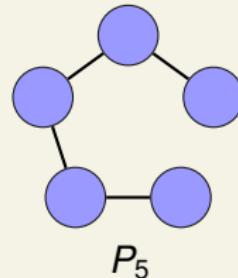
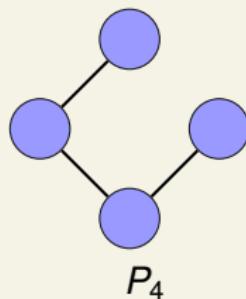
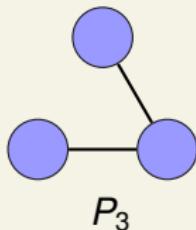


- **Estrelles:** S_i és l'estrella amb $i + 1$ vèrtexs.

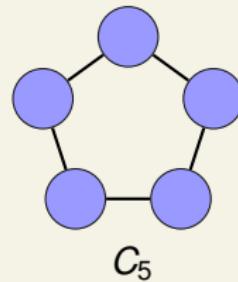
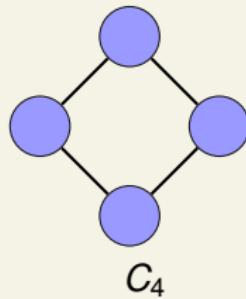
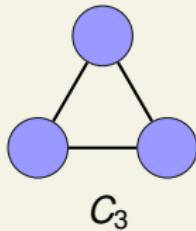


Tipus de grafs

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



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



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.)

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

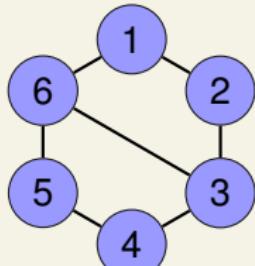
$$\frac{n(n-1)}{2}$$

arestes diferents.

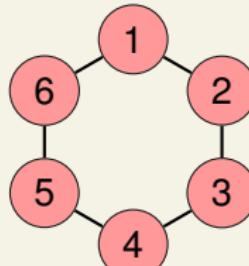
(Són les combinacions de n elements triats de 2 en 2.)

Adjacència i subgrafs

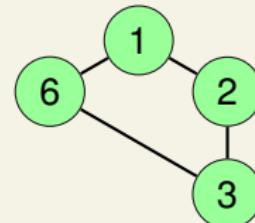
- dos vèrtexs u, v són **adjacents** si $\{u, v\}$ és una aresta
- una aresta $\{u, v\}$ es diu que és **incident** en u i v
- **grau** d'un vèrtex u : nombre d'arestes incidents en u
- un graf $H = (V', E')$ és **subgraf** del graf $G = (V, E)$ si $V' \subseteq V$ i $E' \subseteq E$.
- un graf H és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



Graf G



Subgraf de G



Subgraf induït de G

Exercici

Per què en qualsevol reunió sempre hi ha dues persones que tenen el mateix nombre de coneguts?

Solució

Volem veure que en tot graf no dirigit hi ha dos vèrtexs amb el mateix grau.

Donat un graf G de n vèrtexs, els graus possibles són $0, 1, \dots, n - 1$. Però els graus 0 i $n - 1$ no es poden donar alhora en G .

Per tant:

- el nombre de graus possibles en G és només de $n - 1$
- el nombre total de vèrtexs és n

Pel principi de les caselles, hi ha dos vèrtexs amb el mateix grau.

Exercici

Per què en qualsevol reunió sempre hi ha dues persones que tenen el mateix nombre de coneguts?

Solució

Volem veure que en tot graf no dirigit hi ha dos vèrtexs amb el mateix grau.

Donat un graf G de n vèrtexs, els graus possibles són $0, 1, \dots, n - 1$. Però els graus 0 i $n - 1$ no es poden donar alhora en G .

Per tant:

- el nombre de graus possibles en G és només de $n - 1$
- el nombre total de vèrtexs és n

Pel principi de les caselles, hi ha dos vèrtexs amb el mateix grau.

Camins i cicles

- Un **camí** en un graf és una seqüència de vèrtexs en la qual cada vèrtex (excepte el primer) és adjacent al seu predecessor en el camí
- Un **camí simple** és un camí sense arestes ni vèrtexs repetits
- Un **cicle** és un camí que és simple excepte en els vèrtexs inicial i 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 la qual cada vèrtex (excepte el primer) és adjacent al seu predecessor en el camí
- Un **camí simple** és un camí sense arestes ni vèrtexs repetits
- Un **cicle** és un camí que és simple excepte en els vèrtexs inicial i 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 que no té cap vèrtex adjacent a cap vèrtex extern

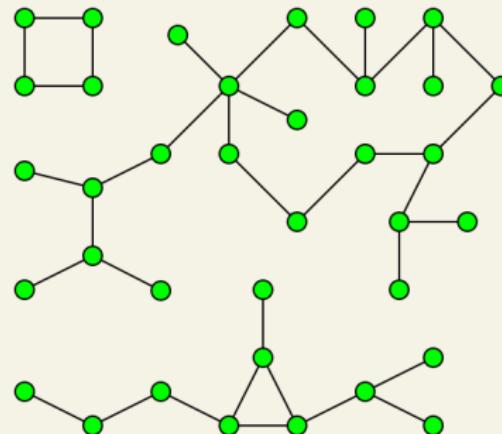


Figura: Graf cíclic amb 3 components connexos

Exercici: Dibuixeu el graf

Dibuixeu el graf $G = (V, E)$ donat per:

- $V = \{1, 2, 3, 4, 5, 6\}$
- $E = \{\{1, 2\}, \{1, 4\}, \{3, 2\}, \{4, 5\}, \{5, 1\}, \{5, 2\}\}$

És connex? Si no ho és, quants components connexos té?

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

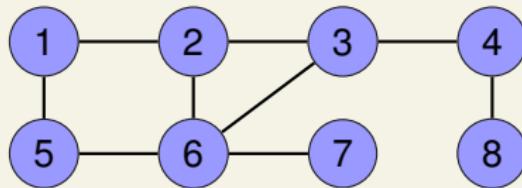
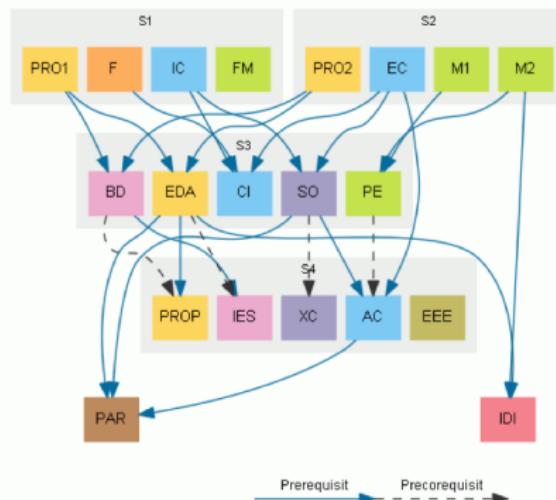


Figura: La distància entre 4 i 5 és 3. El diàmetre del graf és 4.

Grafs dirigits i etiquetats

Definició

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



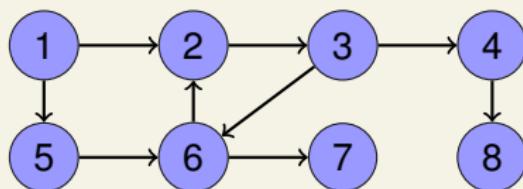
Graf dirigit d'assignatures obligatòries del grau (amb dos tipus d'arcs)

Grafs dirigits i etiquetats

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

Digrafs: graus i distàncies

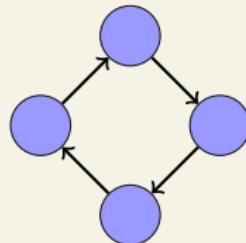
- Es distingeix entre **grau d'entrada** i **grau de sortida**
- En un **camí** (o **camí dirigit**), tots els arcs van en la mateixa direcció
- La **distància** entre dos vèrtexs es refereix als camins dirigits



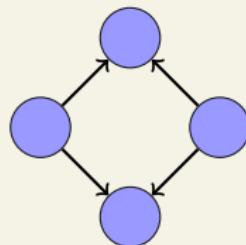
El vèrtex 2 té grau d'entrada 2 i grau de sortida 1.
La distància de 5 a 4 és 4. La de 4 a 5, ∞ .

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



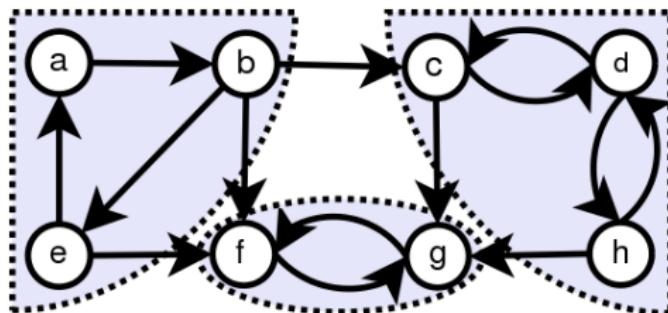
fortament connex



feblement connex

Digrafs: connectivitat

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



Digraf amb 3 components connexos

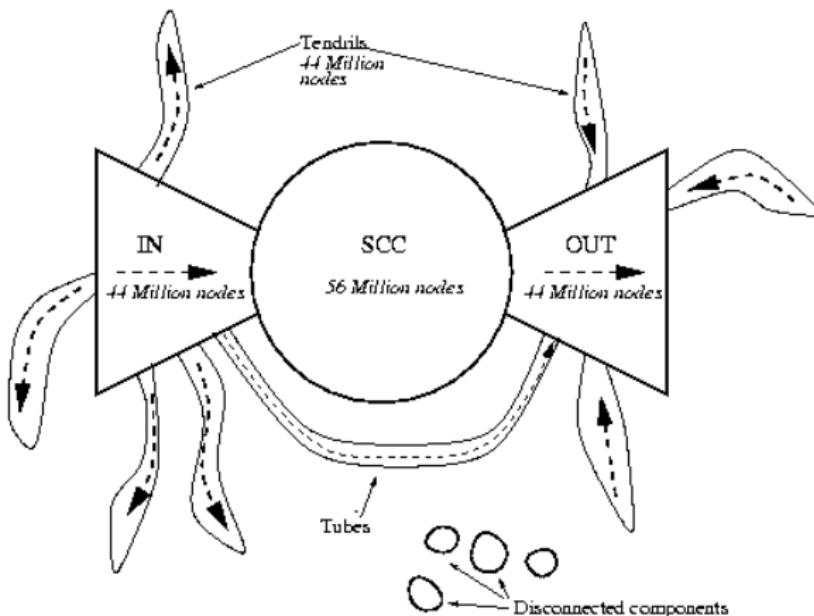
Exercici: Dibuixeu el graf II

Dibuixeu el graf $G = (V, E)$ donat per:

- $V = \{1, 2, 3, 4, 5\}$
- $E = \{(1, 2), (1, 4), (3, 2), (4, 5), (5, 1), (5, 2)\}$

És feblement connex? És fortament connex? Quants components fortament connexos té?

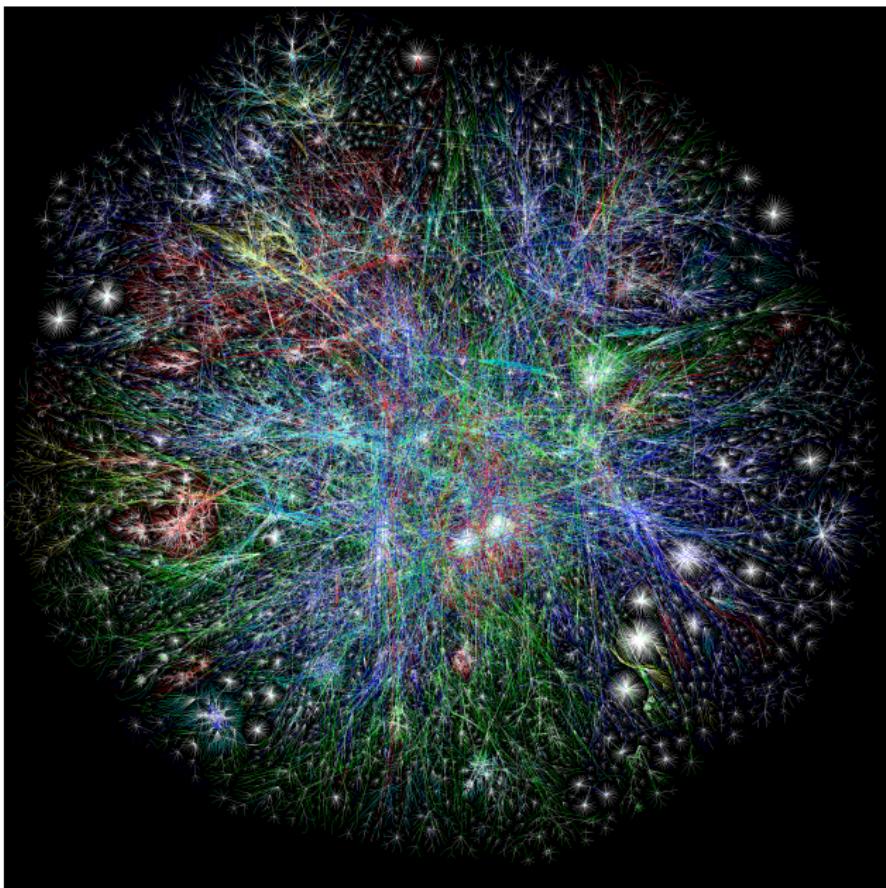
Les pàgines web són els vèrtexs. Els enllaços, les arestes.



SCC: (*giant*) *strongly connected component*

diàmetre del SCC: 28

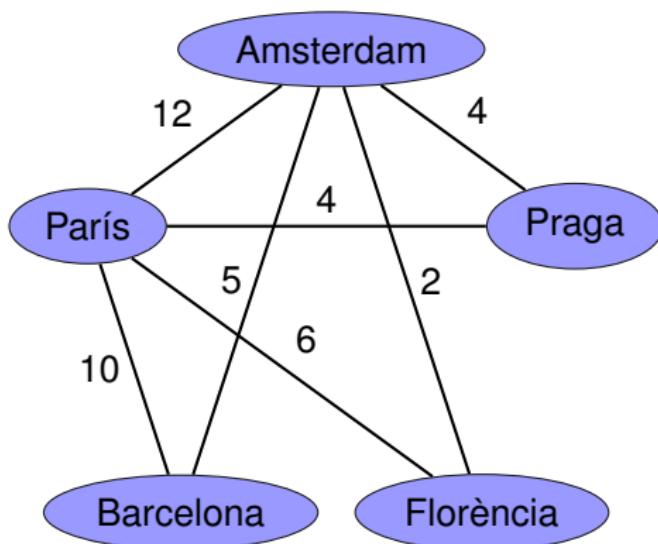
Graf d'internet



Grafs dirigits i etiquetats

Definició

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



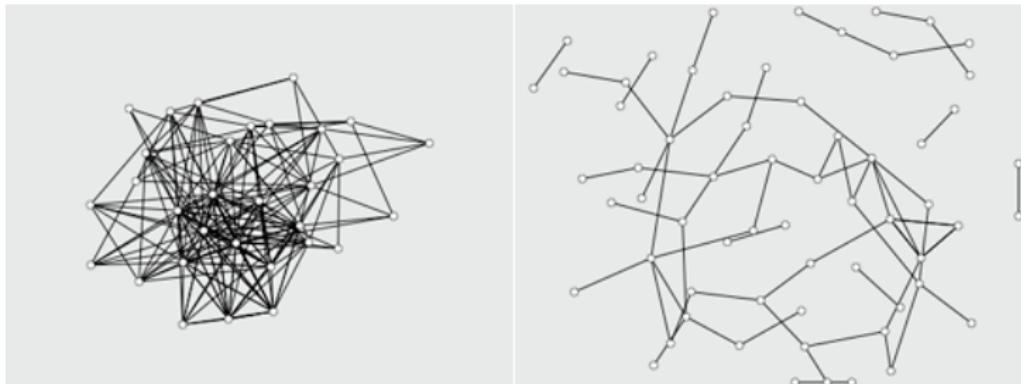
Nombre de vols diaris amb Air France i KLM

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.



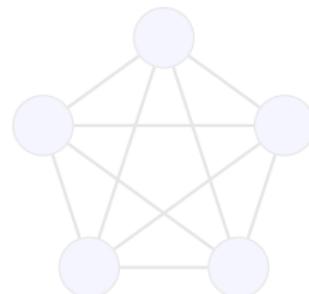
Però com podem definir la densitat d'un graf?

Densitat

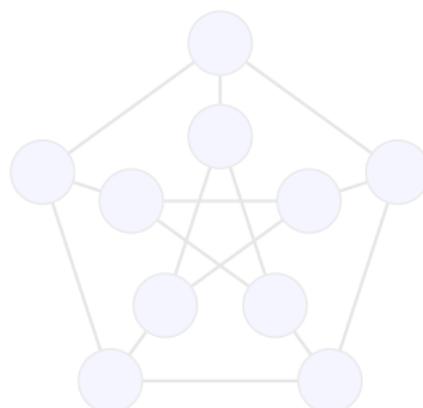
La **densitat** d'un graf G de n vèrtexs i m arestes es defineix com

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

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



$$D(K_5) = 1$$



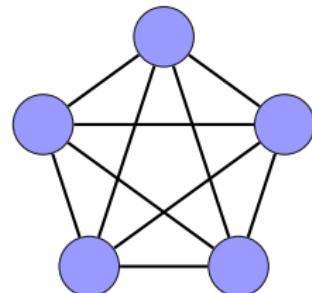
$$D(GP(5,2)) = 1/3$$

Densitat

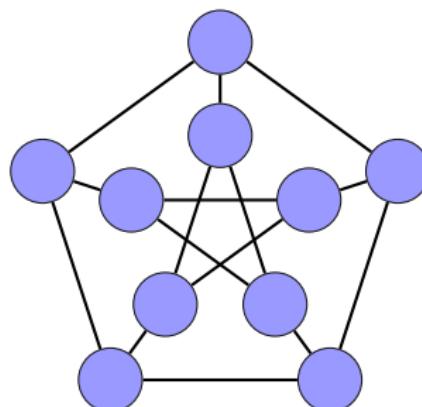
La **densitat** d'un graf G de n vèrtexs i m arestes es defineix com

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

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



$$D(K_5) = 1$$



$$D(GP(5,2)) = 1/3$$

Densitat

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

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

Per exemple:

- Els **grafs complets** són densos perquè $D(K_n) = 1$ per a tot n
- Els **cicles** són esparsos perquè $D(C_n) = 2/(n - 1)$ i $\lim_{n \rightarrow \infty} D(C_n) = 0$

Densitat

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

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

Per exemple:

- Els **grafs complets** són densos perquè $D(K_n) = 1$ per a tot n
- Els **cicles** són esparsos perquè $D(C_n) = 2/(n - 1)$ i $\lim_{n \rightarrow \infty} D(C_n) = 0$

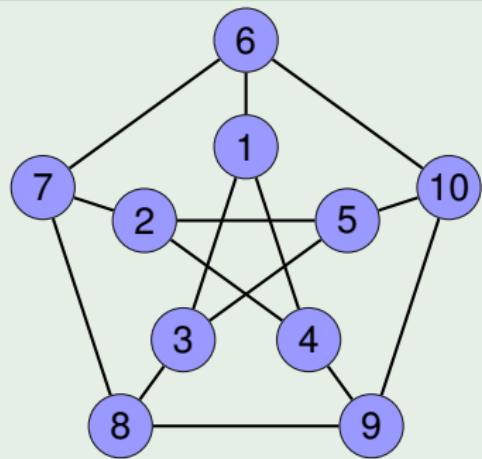
Representacions

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 booleanos tal que

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

Exemple



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	1	0	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

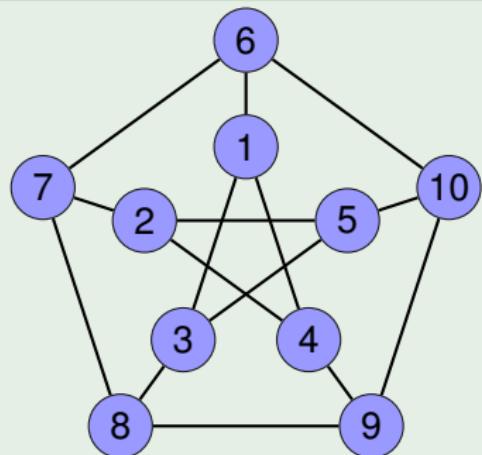
Representacions

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 booleanos tal que

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

Exemple



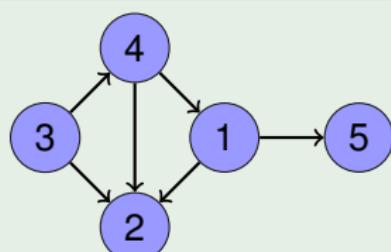
1	0	1	1	0	1	0	0	0	0
2	0	0	1	1	0	1	0	0	0
3	1	0	1	0	0	0	1	0	0
4	1	1	0	0	0	0	0	1	0
5	0	1	1	0	0	0	0	0	1
6	1	0	0	0	0	1	0	0	1
7	0	1	0	0	0	1	1	0	0
8	0	0	1	0	0	0	1	1	0
9	0	0	0	1	0	0	0	1	1
10	0	0	0	0	1	1	0	0	1

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



$$\begin{matrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 1 & 1 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Representacions

Matriu d'adjacència: estructura de dades

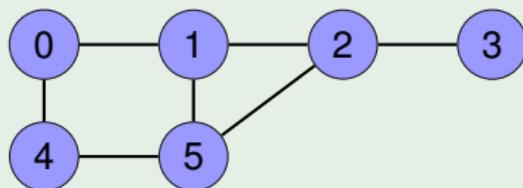
Les matrius d'adjacència es poden implementar amb un vector de vectors.

```
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 no és 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



0	0 1 0 0 1 0
1	1 0 1 0 0 1
2	0 1 0 1 0 1
3	0 0 1 0 0 0
4	1 0 0 0 0 1
5	0 1 1 0 1 0

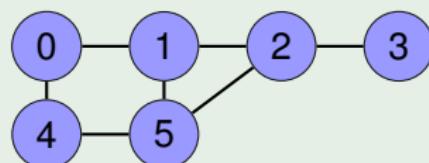
Representacions

Llistes d'adjacència

Cada vèrtex apunta a la llista dels adjacents.

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

Exemple



0	1 4
1	0 5 2
2	3 1 5
3	2
4	5 0
5	1 2 4

Llistes d'adjacència

Cada vèrtex apunta a la llista dels adjacents.

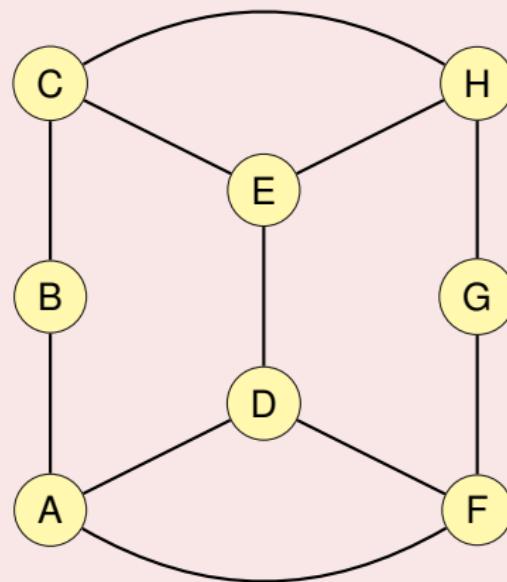
```
typedef vector< vector<int> > graph;
```

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 $O(m)$
- 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**

Exercici: Representeu el graf

Mostreu com es representa el graf següent utilitzant una matriu d'adjacència i llistes d'adjacència. Compteu el grau de cada vèrtex i calculeu el diàmetre i la densitat del graf.



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	$\Theta(1)$	$O(n)$
esborrar aresta	$\Theta(1)$	$O(n)$
consultar vèrtex	$\Theta(1)$	$\Theta(1)$
consultar aresta	$\Theta(1)$	$O(n)$
és v aïllat?	$\Theta(n)$	$\Theta(1)$
successors*	$\Theta(n)$	$O(n)$
predecessors*	$\Theta(n)$	$O(m)$
adjacents ⁺	$\Theta(n)$	$O(n)$

* només en grafs dirigits

+ només en grafs no dirigits

1 Propietats

- Introducció
- Grafs
- Grafs dirigits i etiquetats
- Representacions

2 Algorismes elementals

- Cerca en profunditat
- Cerca en amplada
- Ordenació topològica

3 Distàncies mínimes

- Algorisme de Dijkstra
- Arbres d'expansió mínims

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

Quines parts del graf són accessibles des d'un vèrtex donat?

Un algorisme només pot comprovar 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**

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

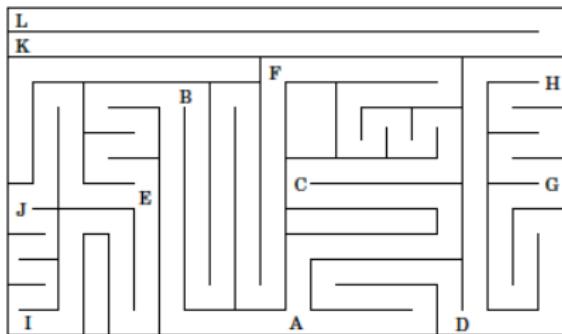
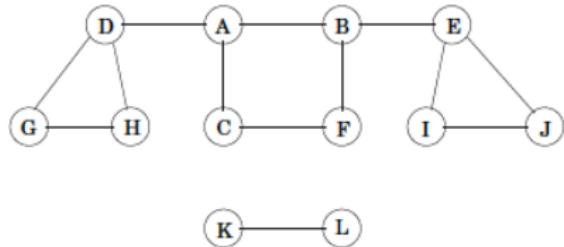
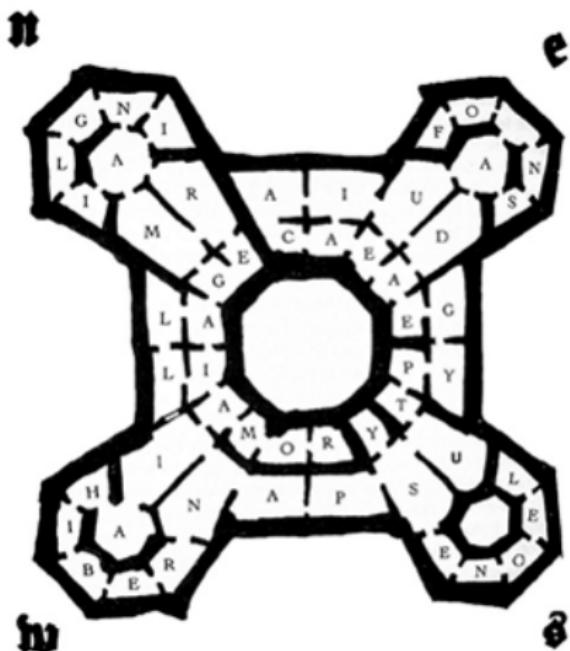
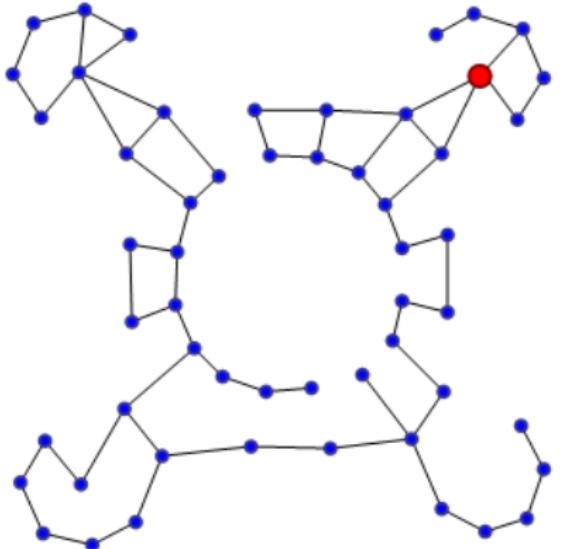


Figura: Per convertir un laberint en un graf: es marquen zones del laberint (vèrtexs) i s'uneixen (amb arestes) si són veïnes (*exemple de Algorithms, S. Dasgupta, C.H. Papadimitriou i U.V. Vazirani*)

Cerca en profunditat



a)



b)

Figura: Laberint de la biblioteca a *El nom de la rosa*, U. Eco

Cerca en profunditat

Cerca en profunditat recursiva (des d'un vèrtex)

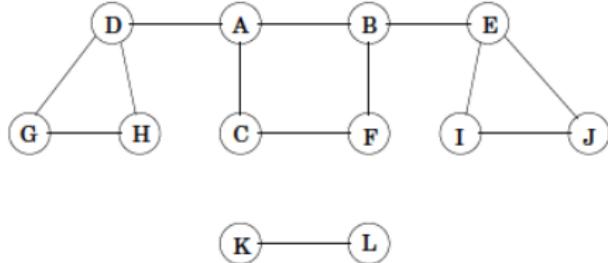
Visitar tots els vèrtexs accessibles a partir d'un vèrtex donat u . La pila és implícita en la recursió.

```
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);
    }
}
```

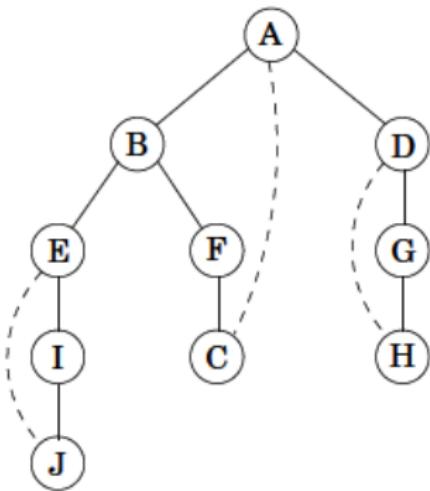
Cerca en profunditat

Exemple anterior:

(*Algorithms. Dasgupta et al.*)



Cerca en profunditat a partir del vèrtex A:
(els vèrtexs es visiten en ordre alfabètic)



Cerca en profunditat

Cerca en profunditat recursiva

Cerca 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;
}
```

Qüestió

Com es pot adaptar l'algorisme `dfs_rec` per comprovar si un graf és connex?

Cerca en profunditat

Cerca en profunditat recursiva

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

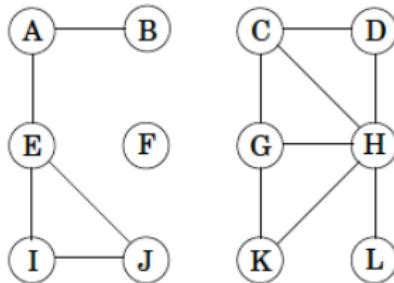
```
list<int> dfs_rec (const graph& G) {
    int n = G.size();
    list<int> L;
    vector<boolean> vis(n, false);
    for (int u = 0; u < n; ++u) {
        dfs_rec(G, u, vis, L);
    }
    return L;
}
```

Qüestió

Com es pot adaptar l'algorisme `dfs_rec` per comprovar si un graf és connex?

Cerca en profunditat

(a)



(b)

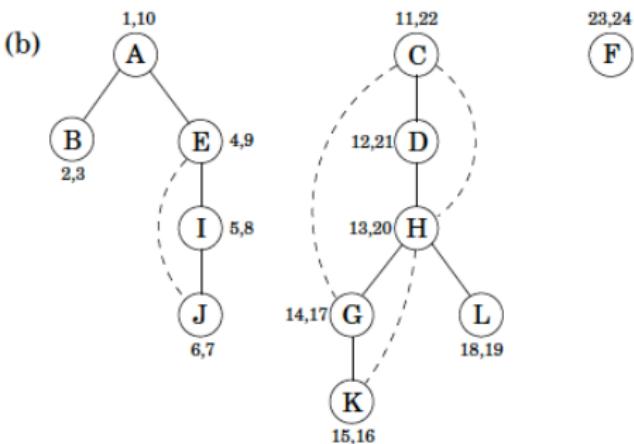


Figura: Cerca en profunditat en graf no dirigit i inconnex (*Algorithms. Dasgupta et al.*)

Cerca en profunditat

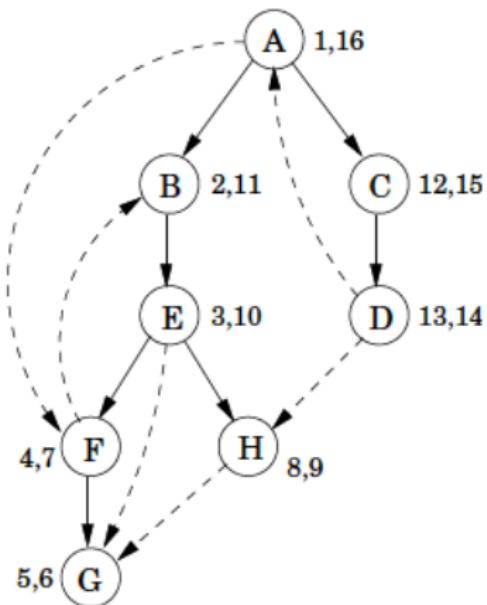
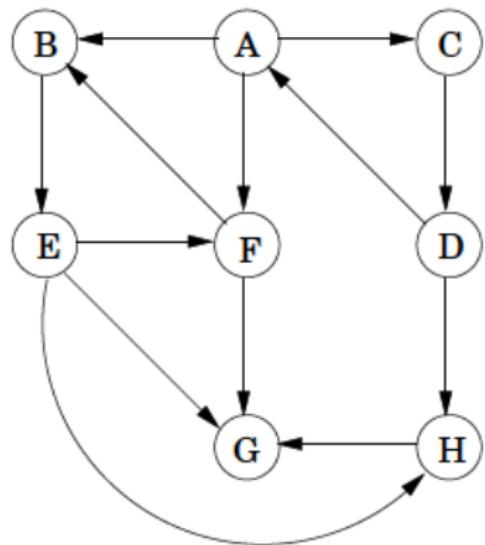


Figura: Cerca en profunditat en graf dirigit (*Algorithms. Dasgupta et al.*)

Cerca en profunditat

```
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);
    }
}
```

Cost de la cerca des d'un vèrtex

- 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, $O(n + m)$

Cost total: $O(n + m)$

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; }
```

Cost de la cerca en profunditat

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

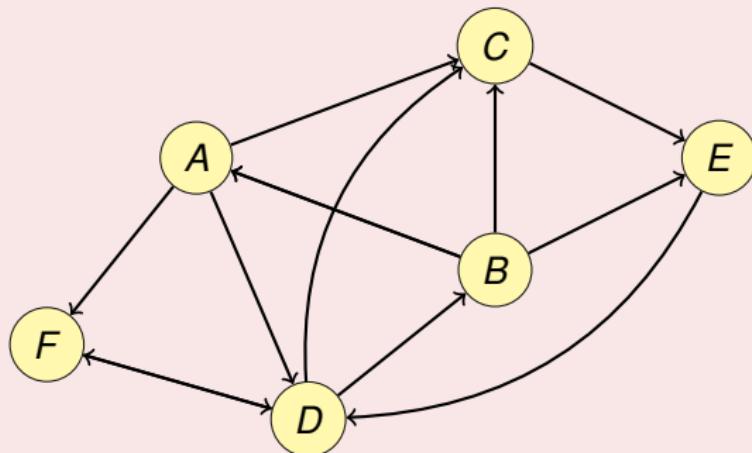
Cerca en profunditat iterativa

```
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;
}
```

Exercici: Recorregut en profunditat

Llisteu totes les possibles seqüències de visita dels vèrtexs d'aquest graf dirigit, tot aplicant un recorregut en profunditat que comenci en el vèrtex *E*.



Cerca en amplada

La **cerca 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 .

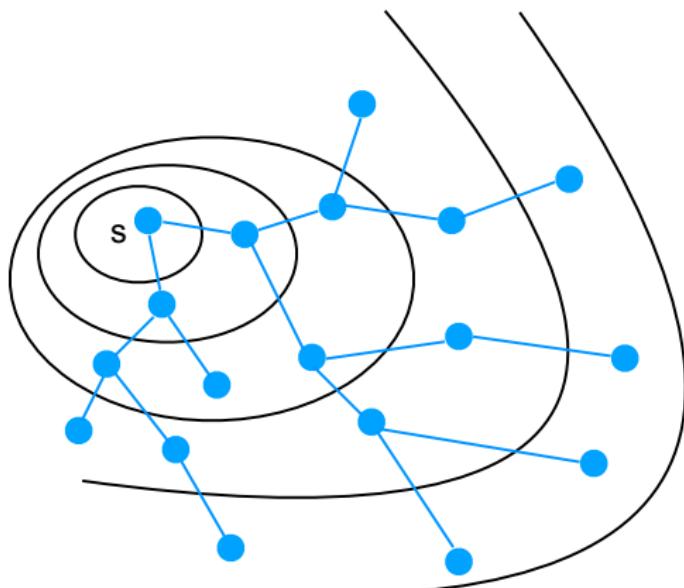


Figura: Cerca en amplada

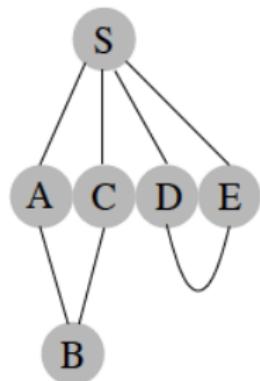
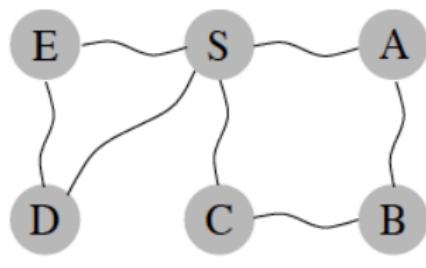


Figura: Model físic d'un graf. Quan es penja el graf des de S, les arestes dels camins mínims es tensen. L'aresta $\{D, E\}$ no juga cap paper en els camins mínims. *Algorithms, Dasgupta et al.*

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** (etiquetes 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

- 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** (etiquetes 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

- 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** (etiquetes 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

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 , $\text{dist}(u)$ contindrà la distància des de s fins a u .

$\text{bfs}(G, s)$

per a tot $u \in V$

$\text{dist}(u) = \infty$

$\text{dist}(s) = 0$

$Q = [s]$ (cua que només conté s)

mentre Q no sigui buida

$u = \text{desencuar}(Q)$

per a tota aresta $(u, v) \in E$

si $\text{dist}(v) = \infty$ **llavors**

$\text{encuar}(Q, v)$

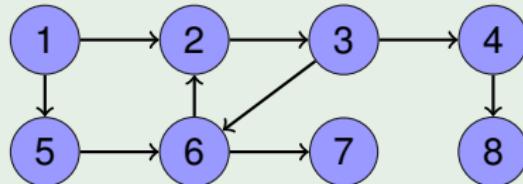
$\text{dist}(v) = \text{dist}(u) + 1$

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

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

Cerca en amplada

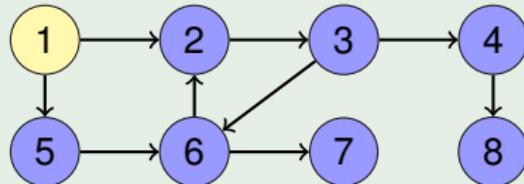
Exemple 1



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
		[1]
1	0	[2 5]
2	1	[5 3]
5	1	[3 6]
3	2	[6 4]
6	2	[4 7]
4	3	[7 8]
7	3	[8]
8	4	[]

Cerca en amplada

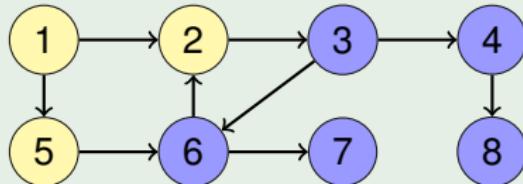
Exemple 1



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
		[1]
1	0	[2 5]
2	1	[5 3]
5	1	[3 6]
3	2	[6 4]
6	2	[4 7]
4	3	[7 8]
7	3	[8]
8	4	[]

Cerca en amplada

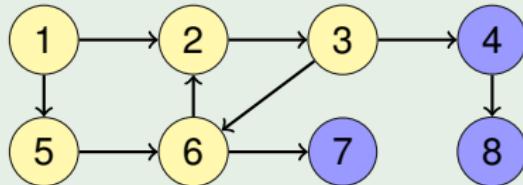
Exemple 1



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
		[1]
1	0	[2 5]
2	1	[5 3]
5	1	[3 6]
3	2	[6 4]
6	2	[4 7]
4	3	[7 8]
7	3	[8]
8	4	[]

Cerca en amplada

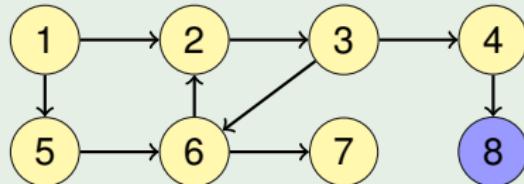
Exemple 1



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
		[1]
1	0	[2 5]
2	1	[5 3]
5	1	[3 6]
3	2	[6 4]
6	2	[4 7]
4	3	[7 8]
7	3	[8]
8	4	[]

Cerca en amplada

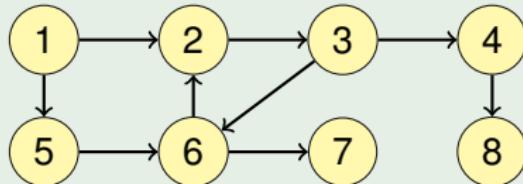
Exemple 1



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
		[1]
1	0	[2 5]
2	1	[5 3]
5	1	[3 6]
3	2	[6 4]
6	2	[4 7]
4	3	[7 8]
7	3	[8]
8	4	[]

Cerca en amplada

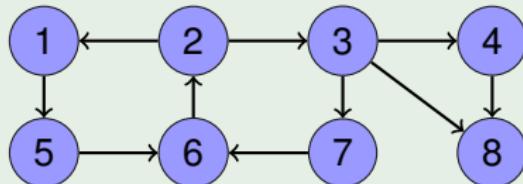
Exemple 1



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
		[1]
1	0	[2 5]
2	1	[5 3]
5	1	[3 6]
3	2	[6 4]
6	2	[4 7]
4	3	[7 8]
7	3	[8]
8	4	[]

Cerca en amplada

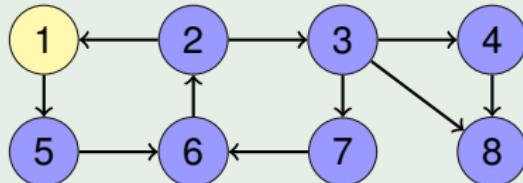
Exemple 2



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
1	0	[1]
5	1	[5]
6	2	[6]
2	3	[2]
3	4	[3]
4	5	[4 7 8]
7	5	[7 8]
8	5	[8]
		[]

Cerca en amplada

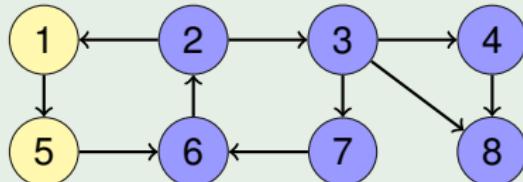
Exemple 2



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
1	0	[1]
5	1	[5]
6	2	[6]
2	3	[2]
3	4	[3]
4	5	[4 7 8]
7	5	[7 8]
8	5	[8]
		[]

Cerca en amplada

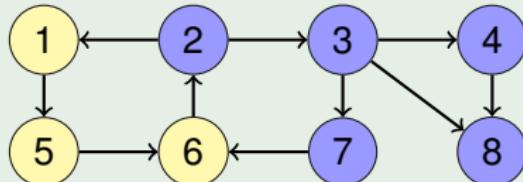
Exemple 2



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
1	0	[1]
5	1	[5]
6	2	[6]
2	3	[2]
3	4	[3]
4	5	[4 7 8]
7	5	[7 8]
8	5	[8]
		[]

Cerca en amplada

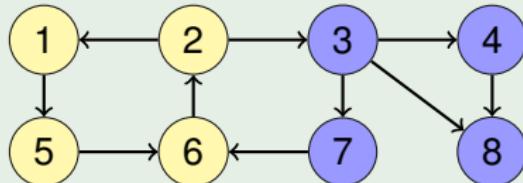
Exemple 2



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
1	0	[1]
5	1	[5]
6	2	[6]
2	3	[2]
3	4	[3]
		[4 7 8]
4	5	[7 8]
7	5	[8]
8	5	[]

Cerca en amplada

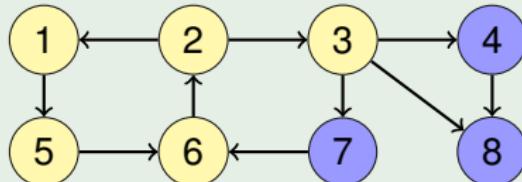
Exemple 2



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
1	0	[1]
5	1	[5]
6	2	[6]
2	3	[2]
3	4	[3]
4	5	[4 7 8]
7	5	[7 8]
8	5	[8]
		[]

Cerca en amplada

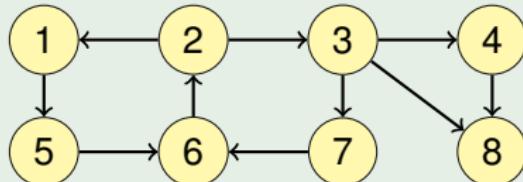
Exemple 2



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
1	0	[1]
5	1	[5]
6	2	[6]
2	3	[2]
3	4	[3]
4	5	[4 7 8]
7	5	[7 8]
8	5	[8]
		[]

Cerca en amplada

Exemple 2



ordre de visita	distància des de 1	contingut de la cua després de processar el vèrtex
1	0	[1]
5	1	[5]
6	2	[6]
2	3	[2]
3	4	[3]
4	5	[4 7 8]
7	5	[7 8]
8	5	[8]
		[]

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:

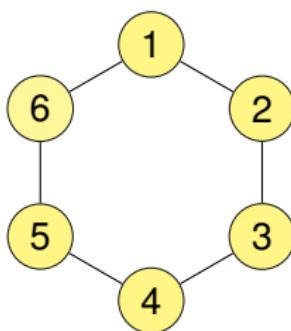
- Cada vèrtex es posa un cop a la cua: $\Theta(n)$ operacions de cua.
- Cada aresta es visita un cop (dirigits) o dos (no dirigits): $\Theta(m)$

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

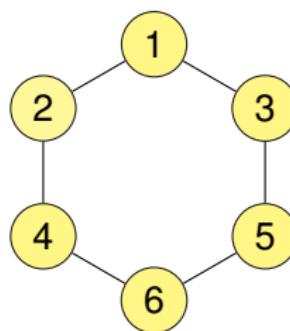
Cerca en amplada

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

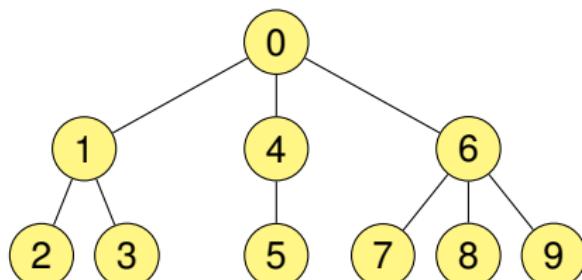


BFS

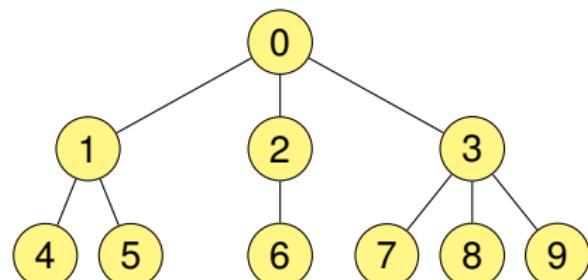
Cerca en amplada

En arbres,

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



DFS



BFS

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
- Com en DFS, en BFS només **s'explora el component connex del vèrtex inicial**. Per explorar la resta del graf, podem recomençar la cerca des dels altres vèrtexs amb l'ajut d'un bucle

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
- Com en DFS, en BFS només **s'explora el component connex del vèrtex inicial**. Per explorar la resta del graf, podem recomençar la cerca des dels altres vèrtexs amb l'ajut d'un bucle

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 (w : G[v])
                    if (not enc[w])
                        Q.push(w); enc[w] = true;
            }
        }
    }
    return L;
}
```

Cost de la cerca en amplada (com en DFS)

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)
- $m = \sum_{i=1}^k m_i$ arestes (el c.c. i té m_i arestes)

llavors el cost és

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

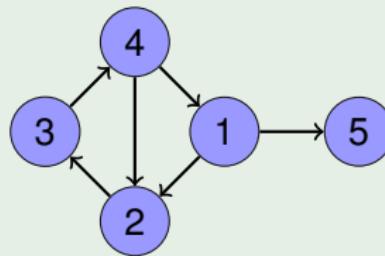
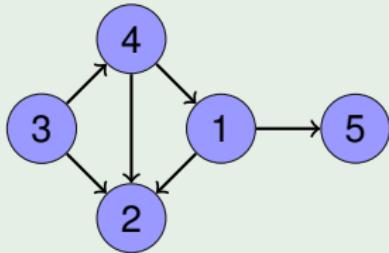
Ordenació topològica

Definició

Un **dag** és un **graf dirigit acíclic**.

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

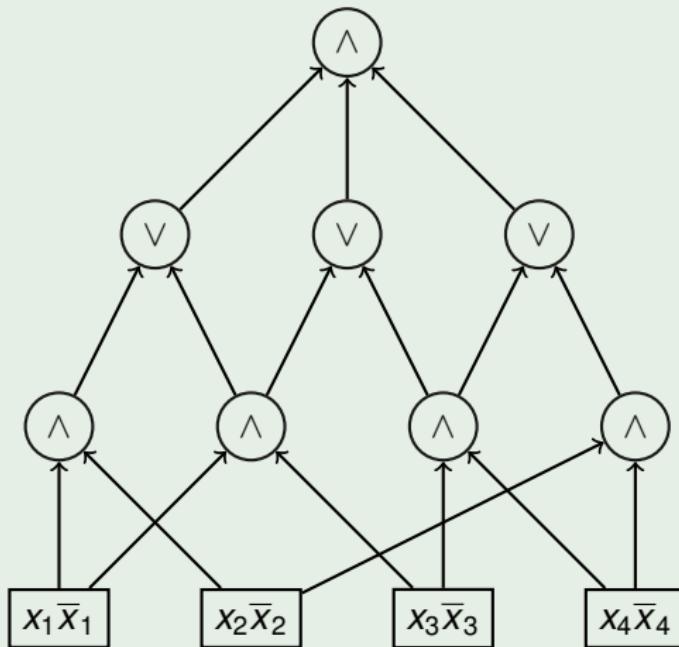
Exemple



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

Exemple

Un circuit també és un dag.



Ordenació topològica

Definició

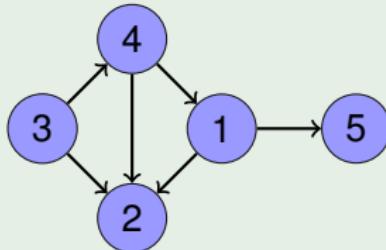
Una **ordenació topològica** d'un dag $G = (V, E)$ és una seqüència

$$v_1, v_2, v_3, \dots, v_n$$

tal que $V = \{v_1, \dots, v_n\}$ 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

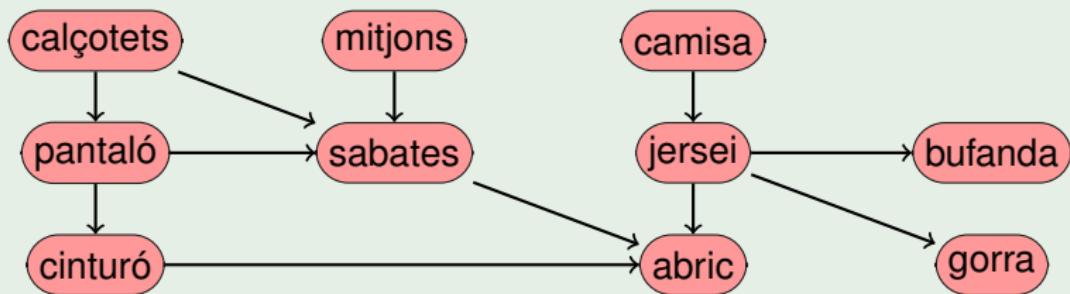
$$v_1, v_2, v_3, \dots, v_n$$

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

Qüestions

- 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.
- ③ Quantes ordenacions topològiques són possibles en el dag $(\{1, \dots, n\}, \{(1, 2)\})$?

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 vorac)

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 vorac)

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 vorac)

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: $\Theta(n^2)$.

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 o cua) 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

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 o cua) 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

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 v : G[u]) {
            ++ge[v];
        }
    }

    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 v : G[u])
        if (--ge[v] == 0)
            S.push(v);
}
return L;
```

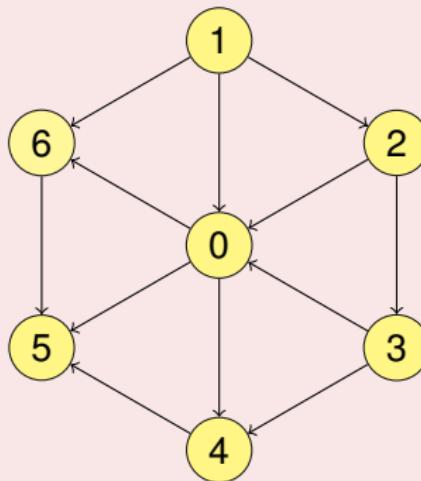
Es visita

- un cop cada vèrtex
- un cop cada aresta

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

Exercici

Donat el graf següent, calculeu l'evolució del vector **ge**, la pila **S** i la llista **L** al llarg de l'execució a partir del vèrtex 1.



1 Propietats

- Introducció
- Grafs
- Grafs dirigits i etiquetats
- Representacions

2 Algorismes elementals

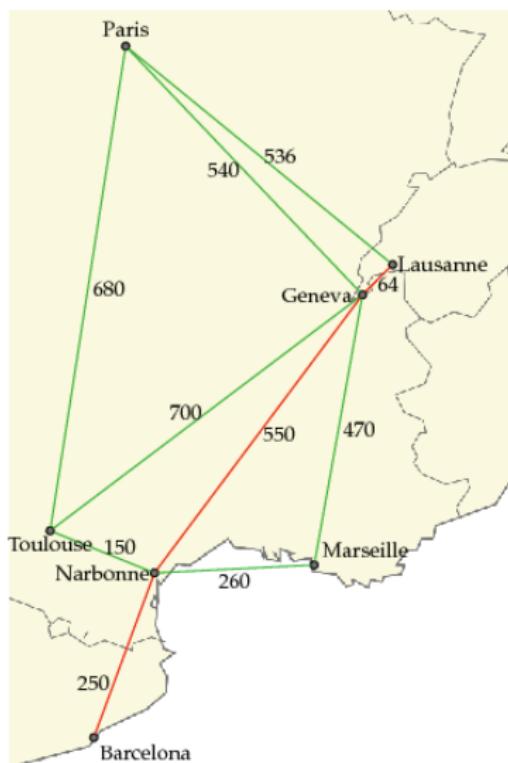
- Cerca en profunditat
- Cerca en amplada
- Ordenació topològica

3 Distàncies mínimes

- Algorisme de Dijkstra
- Arbres d'expansió mínims

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.

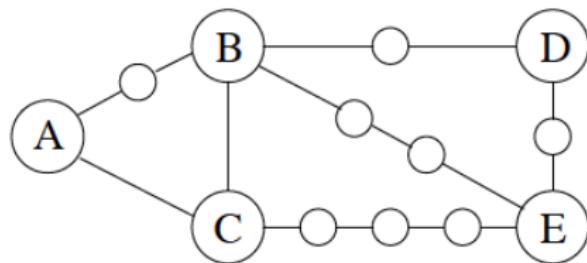
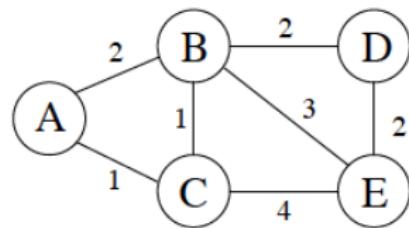


Algorisme de Dijkstra

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

Truc per utilitzar BFS en grafs amb distàncies

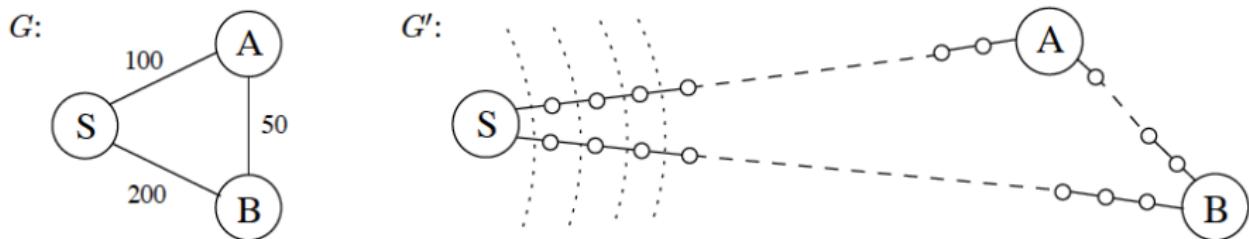
Trencar les arestes del graf d'entrada en arestes de “mida unitària” introduint vèrtexs de farciment.



Cada aresta $e = (u, v)$ amb etiqueta $d(e)$ se substitueix per $d(e)$ arestes amb etiqueta 1 afegint-hi $d(e) - 1$ vèrtexs nous entre u i v .

Algorisme de Dijkstra

El truc anterior resultaria massa ineficient amb distàncies grans. Però podem imaginar que posem **alarms** 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 les alarmes”

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

Donat el graf G amb distàncies 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 dona no és òptima).

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 algoritmes 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 dona no és òptima).

L'algorisme de les alarmes calcula distàncies en qualsevol graf que tingui pesos positius i enters en les arestes.

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

- **crear-cua**: construir cua amb prioritat amb els elements i claus disponibles
- **afegir**: inserir un nou element a la cua
- **treure-min**: retornar l'element amb clau més baixa i treure'l de la cua
- **decrementar-clau**: actualitzar el decrement en la clau d'un element
- **buida**: retorna un booleà que indica si la cua és buida

Algorisme de Dijkstra

Algorisme de Dijkstra dels camins mínims

Entrada: Graf $G = (V, E)$, dirigit o no dirigit; distàncies positives 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 $u \in V$

$dist(u) = \infty$

$prev(u) = nil$

$dist(s) = 0$

$H = \text{crear-cua}(V)$ (fent servir $dist$ per les claus)

mentre no buida(H)

$u = \text{esborrar-min}(H)$

per a tota aresta $(u, v) \in E$

si $dist(v) > dist(u) + d(u, v)$

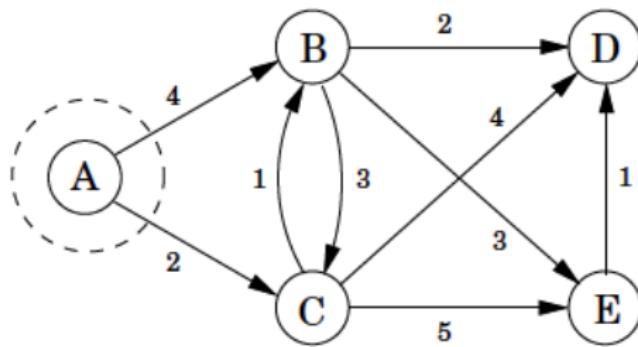
$dist(v) = dist(u) + d(u, v)$

$prev(v) = u$

decrementar-clau(H, v)

Algorisme de Dijkstra

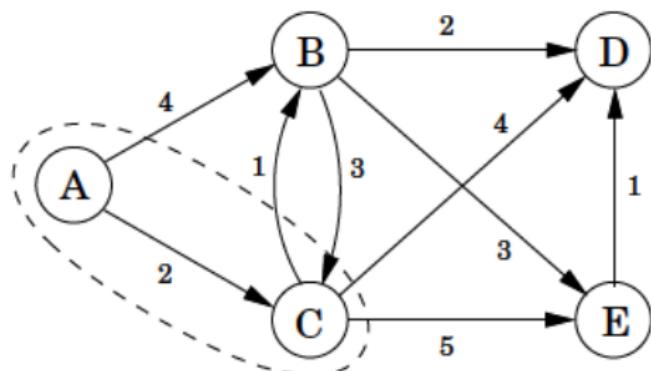
Exemple (*Algorithms*, Dasgupta et al.)



A: 0	D: ∞
B: 4	E: ∞
C: 2	

Algorisme de Dijkstra

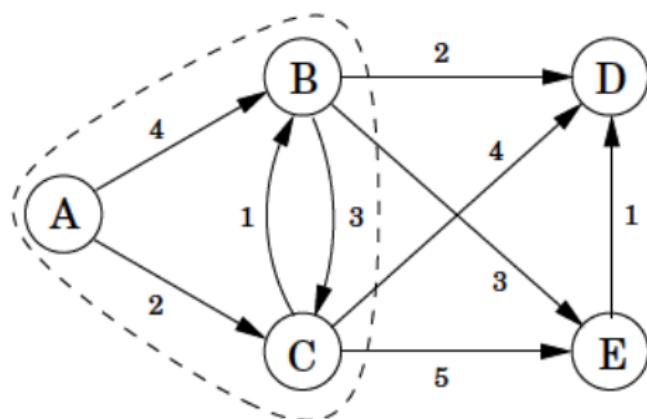
Exemple (*Algorithms*, Dasgupta et al.)



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

Algorisme de Dijkstra

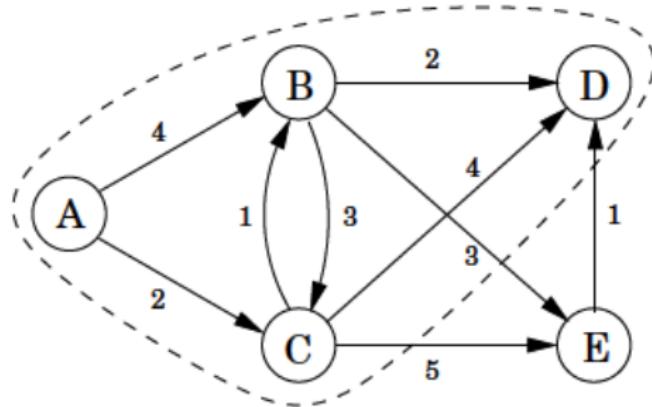
Exemple (*Algorithms*, Dasgupta et al.)



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

Algorisme de Dijkstra

Exemple (*Algorithms*, Dasgupta et al.)

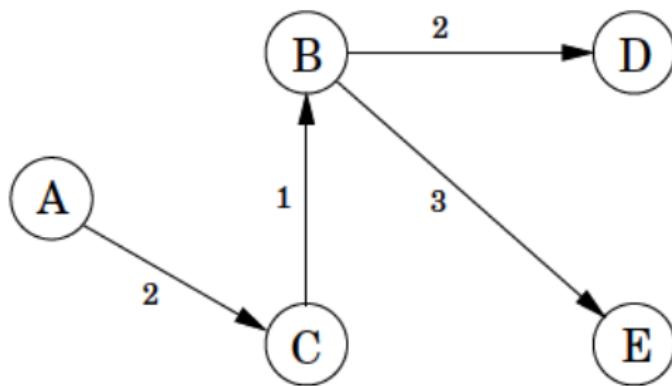


A: 0	D: 5
B: 3	E: 6
C: 2	

Algorisme de Dijkstra

Exemple (*Algorithms*, Dasgupta et al.)

Vector prev:



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, el cost per a grafs de n vèrtexs i m arestes és: $\Theta((n + m) \log n)$.

Dijkstra (G, d, s)

per a tot vèrtex $u \in V$

$dist(u) = \infty$

$prev(u) = nil$

$dist(s) = 0$

$H = \text{crear-cua}(V)$ **(fent servir** $dist$ **per les claus)**

mentre no buida(H)

$u = \text{esborrar-min}(H)$

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-clau(H, v)

Algorisme de Dijkstra

Quina cua amb prioritat és millor?

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

* Tria òptima per a d : $d = m/n$.

+ Cost amortitzat ($O(1)$ en mitjana al llarg de tot l'algorisme).

Algorisme de Dijkstra: implementació

Algorisme de Dijkstra

En lloc de decrementar les claus, s'acumulen en la cua amb prioritat però un vector **S** evitarà que un vèrtex es tracti dos cops.

```
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

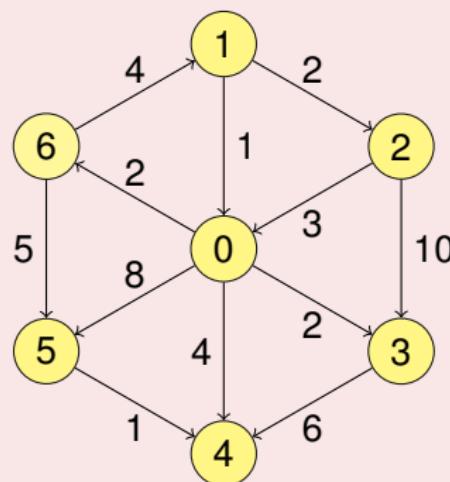
Algorisme de Dijkstra

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

Algorisme de Dijkstra

Exercici

Apliqueu l'algorisme de Dijkstra per trobar els camins mínims en el graf següent a partir del vèrtex 1.



Arbres d'expansió mínims

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

Definition

Un **arbre d'expansió** (*spanning tree*, en anglès) de G és un subgraf $T = (V, A)$ de G , on $A \subseteq E$, que és connex i acíclic.

Observem que T és un arbre i conté tots els vèrtexs de G

Definició

Un **arbre d'expansió mínim** (MST, de l'anglès *minimum spanning tree*) de G és un arbre d'expansió $T = (V, A)$ de G el pes total del qual

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

és mínim entre tots els arbres d'expansió de G .

Arbres d'expansió mínims

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

Definition

Un **arbre d'expansió** (*spanning tree*, en anglès) de G és un subgraf $T = (V, A)$ de G , on $A \subseteq E$, que és connex i acíclic.

Observem que T és un arbre i conté tots els vèrtexs de G

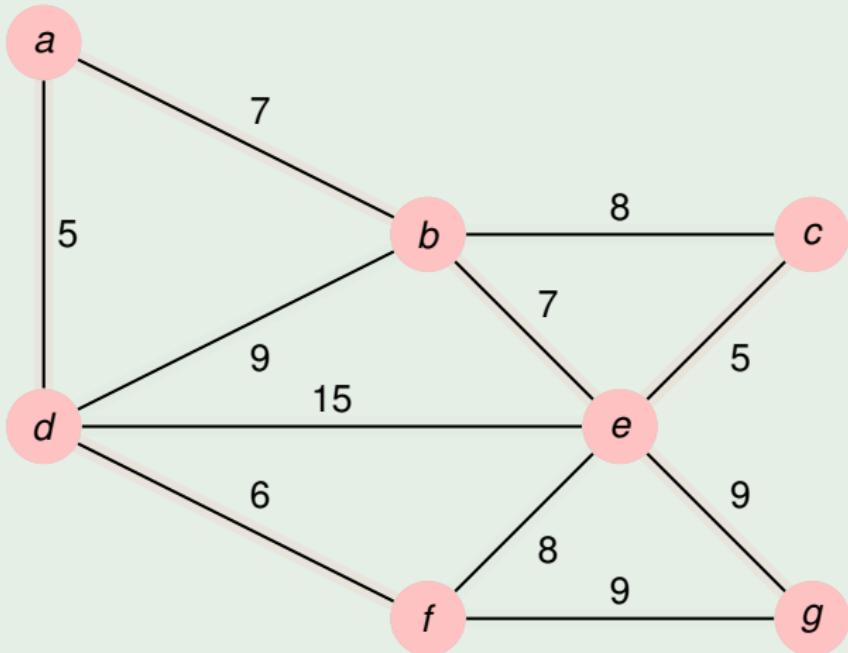
Definició

Un **arbre d'expansió mínim** (MST, de l'anglès *minimum spanning tree*) de G és un arbre d'expansió $T = (V, A)$ de G el pes total del qual

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

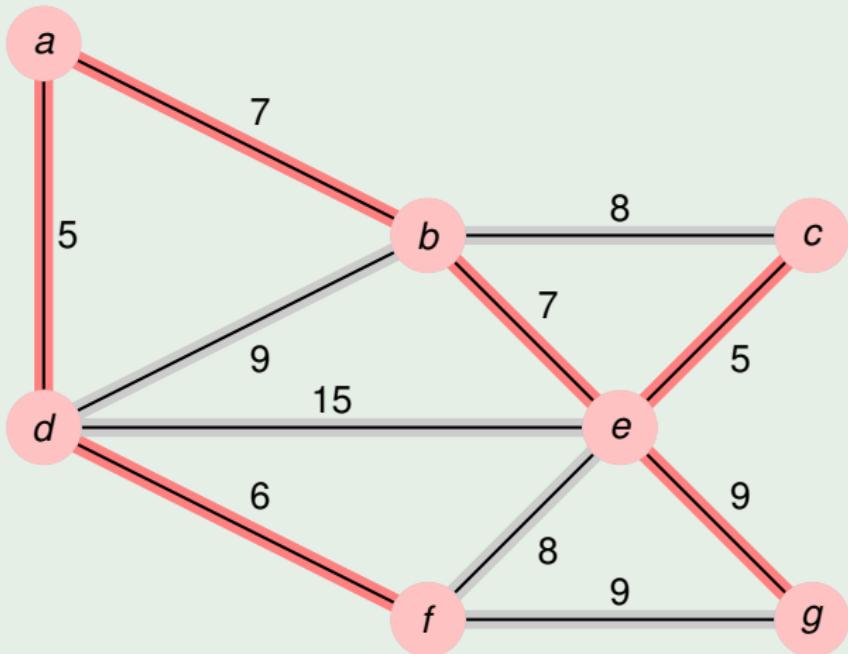
és mínim entre tots els arbres d'expansió de G .

Dos arbres d'expansió mínims



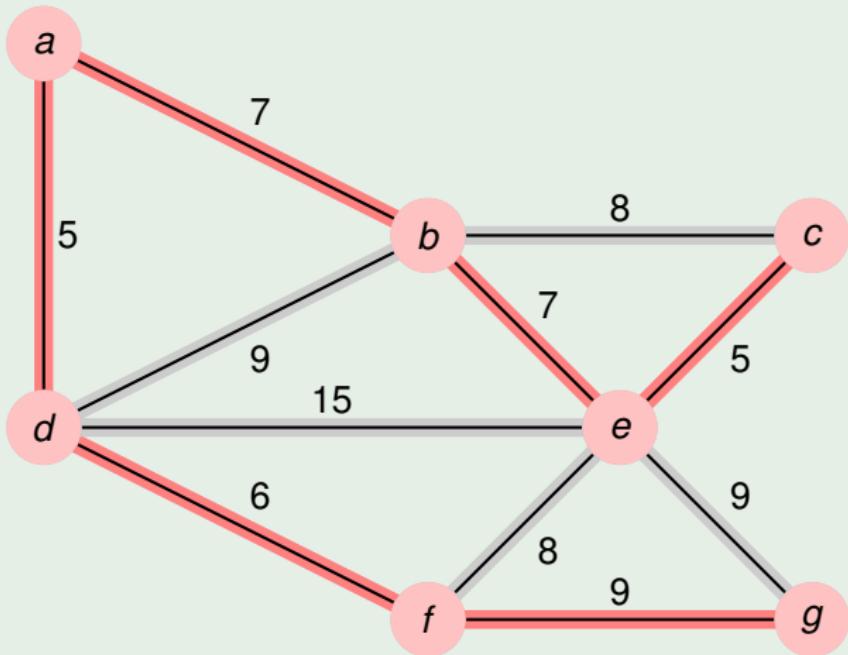
Font: www.texexample.net/tikz/examples/author/kjell-magne-fauske

Dos arbres d'expansió mínims



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

Dos arbres d'expansió mínims



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

Arbres d'expansió mínims

Hi ha molts algorismes diferents per calcular MST. Tots segueixen un **esquema voraç**, és a dir, repeteixen una acció que sembla òptima a cada moment.

```
A = ∅;  
Cand = E;  
while (|A| ≠ |V| - 1) {  
    triar e ∈ Cand tal que  $T = (V, A \cup \{e\})$  no tingui cicles  
    A = A ∪ {e}  
    Cand = Cand - {e}  
}  
}
```

Els conceptes següents ens portaran a una estratègia voraç per triar una aresta del conjunt de candidates.

Arbres d'expansió mínims

Hi ha molts algorismes diferents per calcular MST. Tots segueixen un **esquema voraç**, és a dir, repeteixen una acció que sembla òptima a cada moment.

```
A = ∅;  
Cand = E;  
while (|A| ≠ |V| - 1) {  
    triar e ∈ Cand tal que  $T = (V, A \cup \{e\})$  no tingui cicles  
    A = A ∪ {e}  
    Cand = Cand - {e}  
}  
}
```

Els conceptes següents ens portaran a una estratègia voraç per triar una aresta del conjunt de candidates.

Arbres d'expansió mínims

Definition

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

Definition

Un **tall** d'un graf $G = (V, E)$ és una partició de V , és a dir, un parell (C, C') tal que

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

Definition

Una aresta e **respecta** un tall (C, C') si ambdós extrems de e pertanyen a C o ambdós pertanyen a C' ; altrament, diem que e **travessa** el tall.

Un conjunt d'arestes A **respecta** un tall si totes les arestes de A el respecten.

Arbres d'expansió mínims

Definition

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

Definition

Un **tall** d'un graf $G = (V, E)$ és una partició de V , és a dir, un parell (C, C') tal que

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

Definition

Una aresta e **respecta** un tall (C, C') si ambdós extrems de e pertanyen a C o ambdós pertanyen a C' ; altrament, diem que e **travessa** el tall.

Un conjunt d'arestes A **respecta** un tall si totes les arestes de A el respecten.

Arbres d'expansió mínims

Definition

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

Definition

Un **tall** d'un graf $G = (V, E)$ és una partició de V , és a dir, un parell (C, C') tal que

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

Definition

Una aresta e **respecta** un tall (C, C') si ambdós extrems de e pertanyen a C o ambdós pertanyen a C' ; altrament, diem que e **travessa** el tall.

Un conjunt d'arestes A **respecta** un tall si totes les arestes de A el respecten.

Arbres d'expansió mínims

Teorema

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

Aquest teorema dona un mètode per dissenyar algorismes per a un MST:

- ① començar amb un conjunt d'arestes buit A
- ② definir quin és el tall inicial de G
- ③ mentre el tall no sigui trivial
 - triar l'aresta e amb pes mínim entre les que travessen el tall
 - afegir e a A i passar a C l'extrem de e que pertanyia a C'
(com que e travessava el tall, no es pot crear un cicle)

Arbres d'expansió mínims

Teorema

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

Aquest teorema dona un mètode per dissenyar algorismes per a un MST:

- ① començar amb un conjunt d'arestes buit A
- ② definir quin és el tall inicial de G
- ③ mentre el tall no sigui trivial
 - triar l'aresta e amb pes mínim entre les que travessen el tall
 - afegir e a A i passar a C l'extrem de e que pertanyia a C'
(com que e travessava el tall, no es pot crear un cicle)

Arbres d'expansió mínims

Teorema

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

Demostració

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

Considerem dos casos:

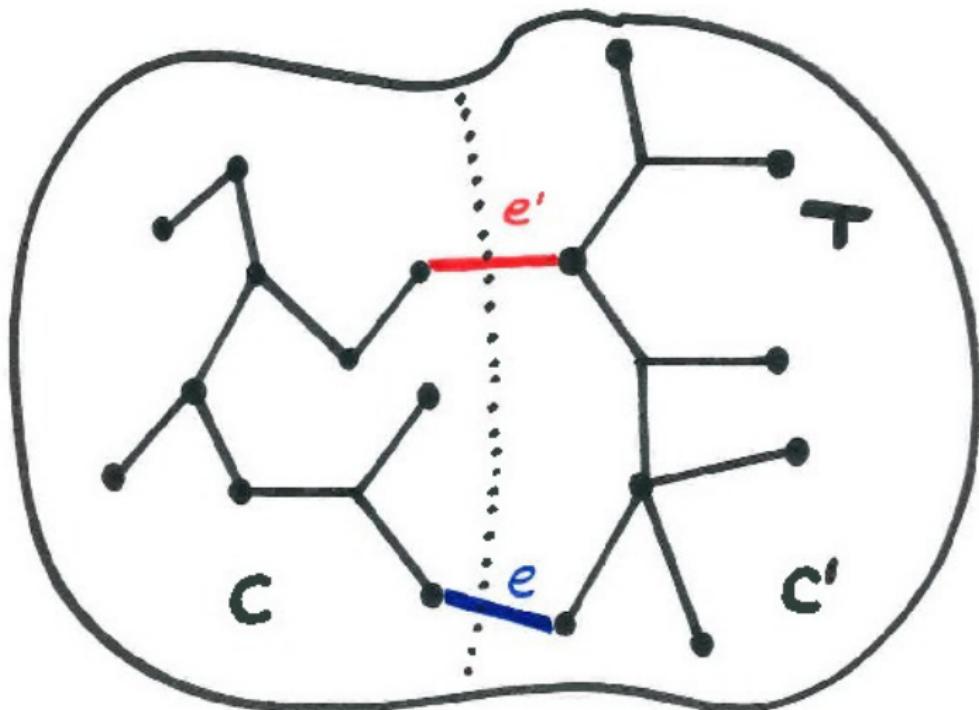
- 1 $e \in A'$

Llavors, $A \cup \{e\}$ és prometedor.

- 2 $e \notin A'$

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

Arbres d'expansió mínims



Arbres d'expansió mínims

Teorema

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

Demostració

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

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

Llavors, $\omega(e') \leq \omega(e)$.

Però, com que e tenia pes mínim, es compleix que $\omega(e) \leq \omega(e')$.

Per tant, $\omega(T) = \omega(T')$ i T' és un MST.

Aleshores, $A \cup \{e\}$ és prometedor.

Arbres d'expansió mínims

Teorema

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

Demostració

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

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

Llavors, $\omega(e') \leq \omega(e)$.

Però, com que e tenia pes mínim, es compleix que $\omega(e) \leq \omega(e')$.

Per tant, $\omega(T) = \omega(T')$ i T' és un MST.

Aleshores, $A \cup \{e\}$ és prometedor.

Arbres d'expansió mínims

Teorema

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

Demostració

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

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

Llavors, $\omega(e') \leq \omega(e)$.

Però, com que e tenia pes mínim, es compleix que $\omega(e) \leq \omega(e')$.

Per tant, $\omega(T) = \omega(T')$ i T' és un MST.

Aleshores, $A \cup \{e\}$ és prometedor.

Arbres d'expansió mínims

Teorema

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

Demostració

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

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

Llavors, $\omega(e') \leq \omega(e)$.

Però, com que e tenia pes mínim, es compleix que $\omega(e) \leq \omega(e')$.

Per tant, $\omega(T) = \omega(T')$ i T' és un MST.

Aleshores, $A \cup \{e\}$ és prometedor.

En l'**algorisme de Prim** (conegit també com **algorisme Prim-Jarník**), mantenim un subconjunt de vèrtexs visitats.

- El conjunt de vèrtexs es divideix entre visitats i no visitats
- Cada iteració de l'algorisme tria l'aresta de pes mínim entre les que uneixen vèrtexs visitats amb no visitats
- Pel teorema, l'algorisme és correcte

En l'**algorisme de Prim** (conegit també com **algorisme Prim-Jarník**), mantenim un subconjunt de vèrtexs visitats.

- El conjunt de vèrtexs es divideix entre visitats i no visitats
- Cada iteració de l'algorisme tria l'aresta de pes mínim entre les que uneixen vèrtexs visitats amb no visitats
- Pel teorema, l'algorisme és correcte

En l'**algorisme de Prim** (conegit també com **algorisme Prim-Jarník**), mantenim un subconjunt de vèrtexs visitats.

- El conjunt de vèrtexs es divideix entre visitats i no visitats
- Cada iteració de l'algorisme tria l'aresta de pes mínim entre les que uneixen vèrtexs visitats amb no visitats
- Pel teorema, l'algorisme és correcte

En l'**algorisme de Prim** (conegit també com **algorisme Prim-Jarník**), mantenim un subconjunt de vèrtexs visitats.

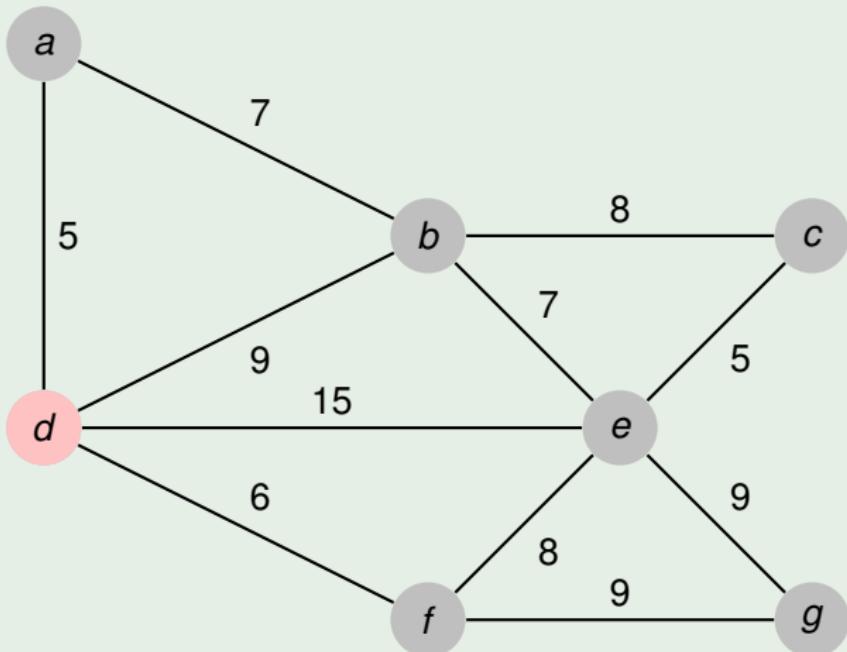
- El conjunt de vèrtexs es divideix entre visitats i no visitats
- Cada iteració de l'algorisme tria l'aresta de pes mínim entre les que uneixen vèrtexs visitats amb no visitats
- Pel teorema, l'algorisme és correcte

Algorisme de Prim

El graf es representa amb llistes d'adjacència. Els parells són (cost, vertex).

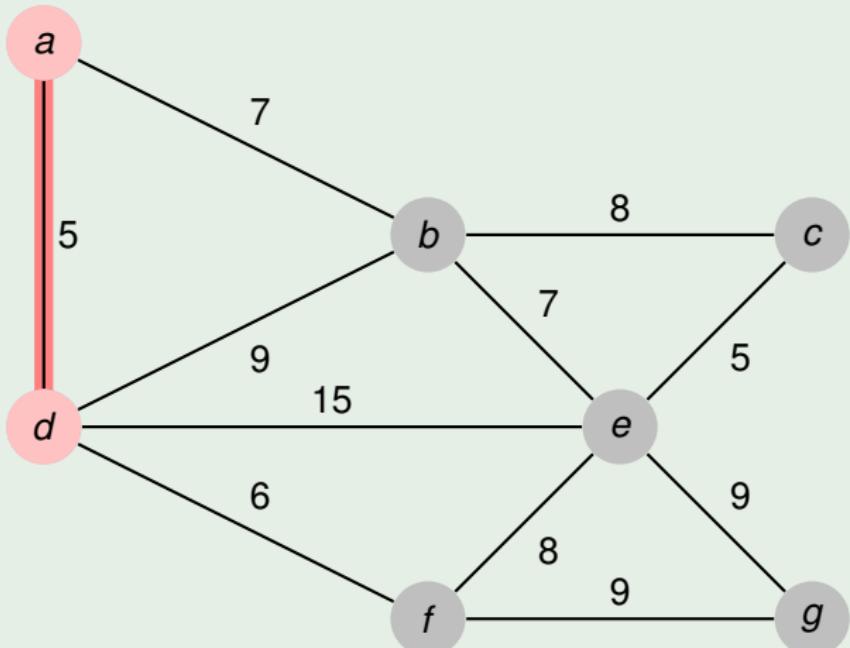
```
typedef pair<int, int> P;
int mst(const vector<vector<P>>& g) { // Ret. el cost d'un MST
    vector<bool> vis(n, false);
    vis[0] = true;
    priority_queue<P, vector<P>, greater<P> > pq;
    for (P x : g[0]) pq.push(x);
    int sz = 1;
    int sum = 0;
    while (sz < n) {
        int c = pq.top().first;
        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; }
```

Exemple



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

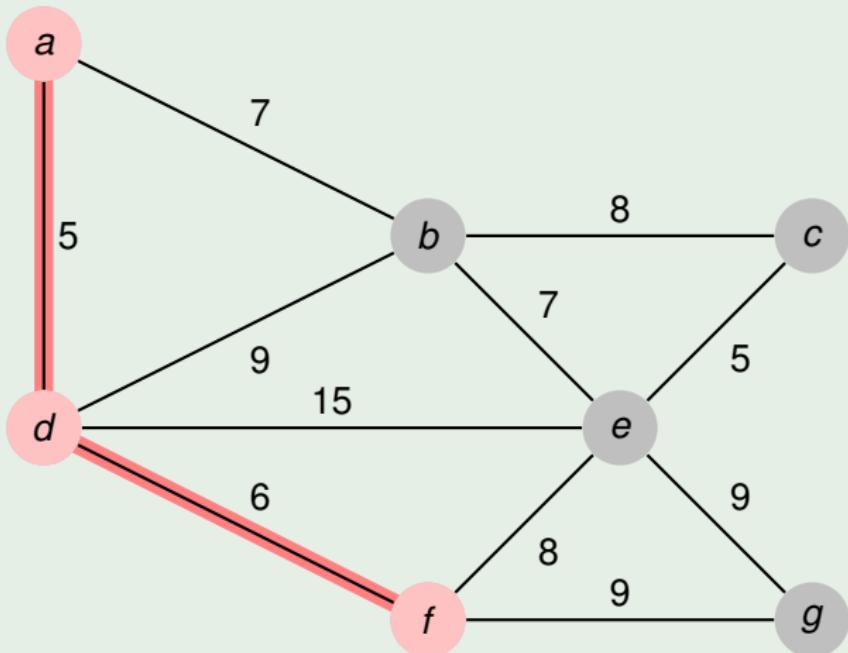
Exemple



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

Algorisme de Prim

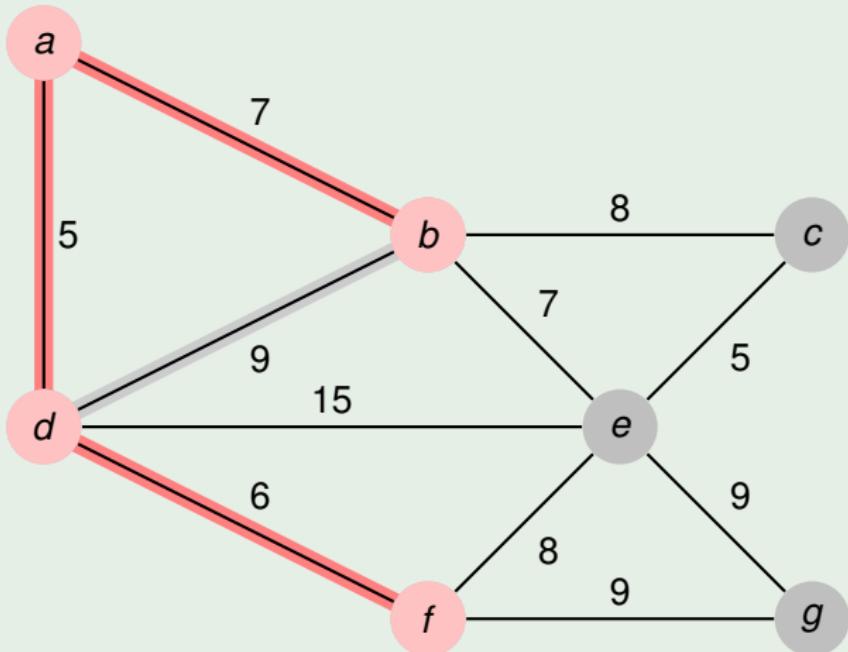
Exemple



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

Algorisme de Prim

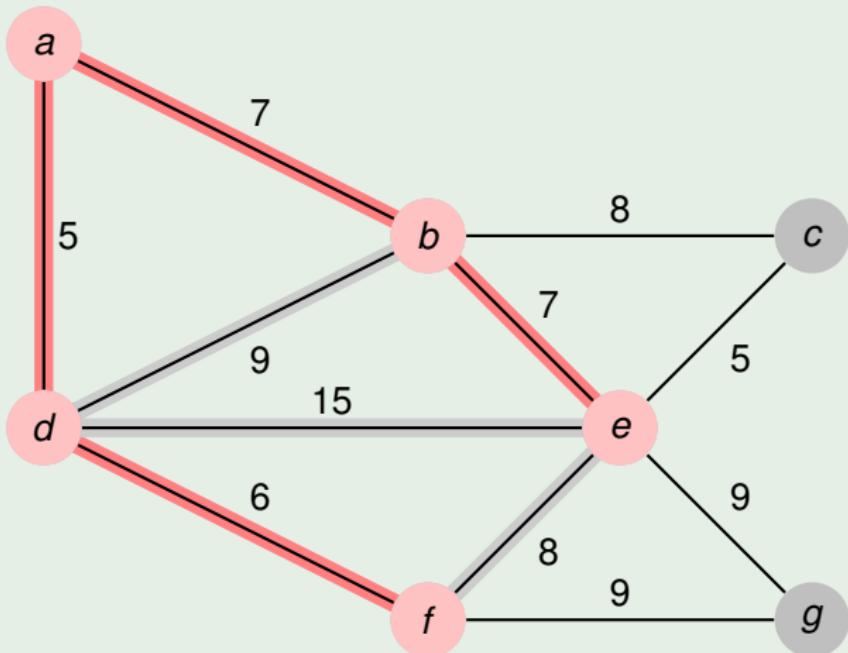
Exemple



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

Algorisme de Prim

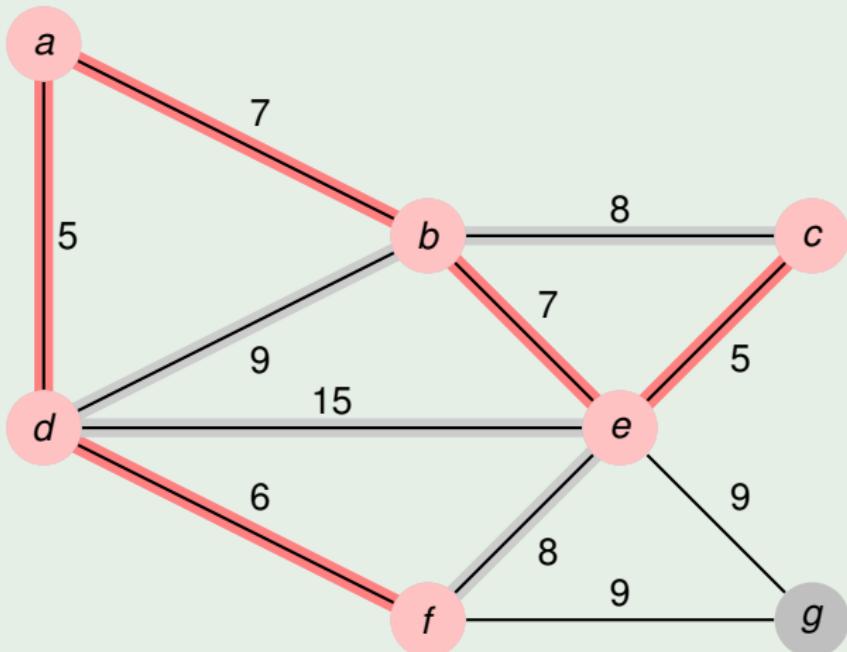
Exemple



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

Algorisme de Prim

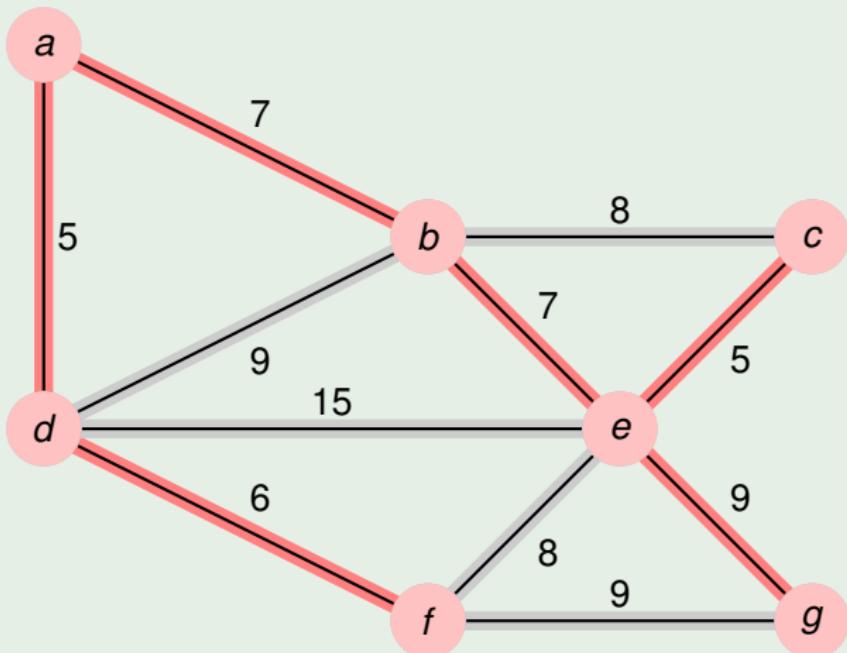
Exemple



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

Algorisme de Prim

Exemple



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

Anàlisi del cost

El cost del bucle, amb $O(m)$ iteracions, domina el cost de l'algorisme.

Comptem per separat la selecció d'aresta i la visita de nous candidats dins de cada iteració:

- **Selecció d'aresta.** La selecció d'aresta en cada iteració és $O(\log m)$.
- **Visita de nous candidats.** Cada vèrtex es marca exactament un cop.
Quan es visita un vèrtex x , afegir nous candidats costa $O(\deg(x) \log m)$.

El cost total és, doncs,

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

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