UNIVERSITAT DE BARCELONA

Grafs I

Algorísmica Avançada | Enginyeria Informàtica

Santi Seguí | 2019-2020

Algunes preguntes interessants!

- **Camí (Path):** Hi ha cap camí entre els nodes *s* i *t*?
- **Camí més curt:** Quin és el camí més curt entre els nodes *s* i *t*?

Quins d'aquests problemes són fàcils? difícil? intractable?

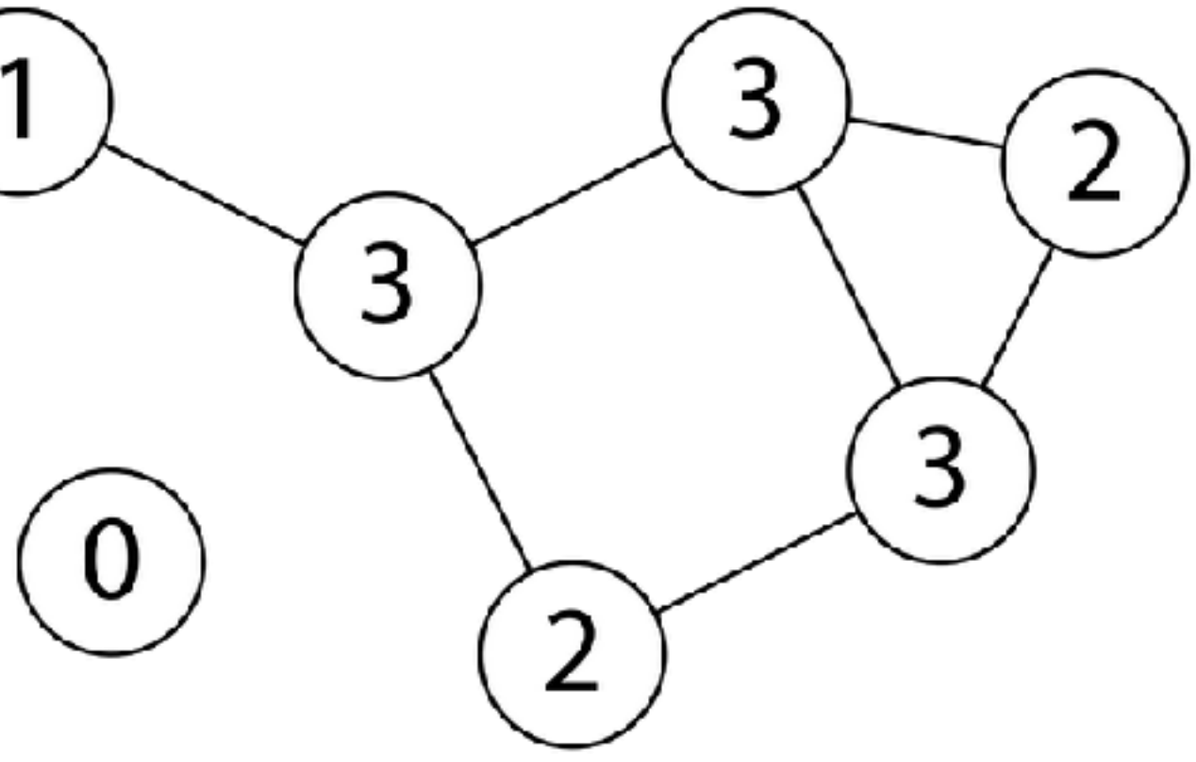
- **Cicle:** Hi ha cap cicle en el graf?
- **Camí d'Euler:** Hi ha cap cicle que utilitzi cap aresta exactament un cop?
- **Camí Hamiltonià:** Hi ha cap cicle que utilitzi cada node exactament un cop?

- **Connectivitat**. Hi ha cap manera de connectar tots els nodes
- **MST**. Quina és la millor manera de connectar tots els nodes d'un graf?
- **Bioconnectivitat**. Hi ha cap node que la seva eliminació desconnecti el graf.

- **Planaritat**. Podeu dibuixar el gràfic en el pla sense vores de creuament
- **Isomorfisme gràfic.** Les dues llistes adjacència representen el mateix gràfic?

Propietats dels grafs

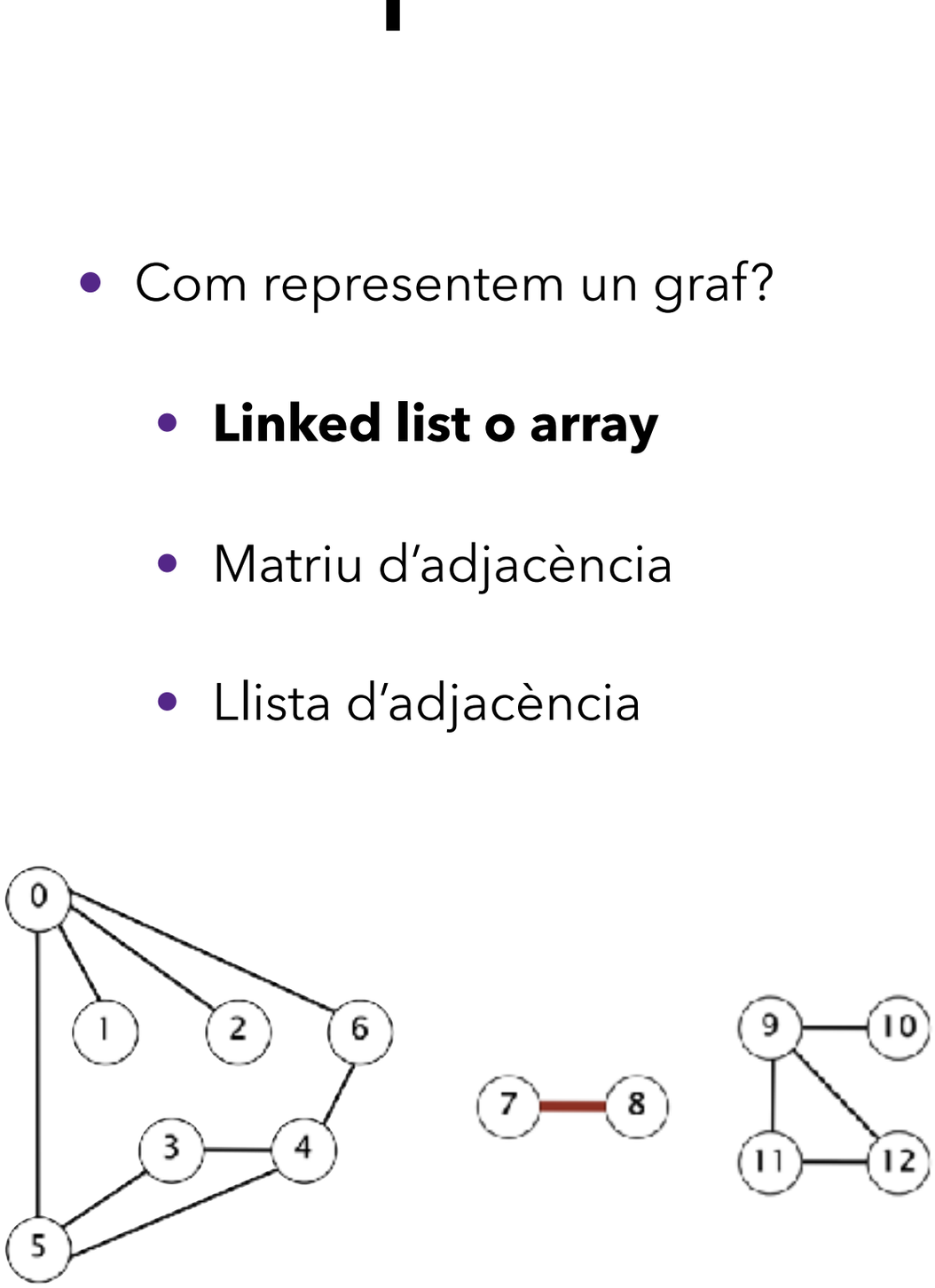
- El **grau** d'un vèrtex és el nombre d'arestes que hi incideixen

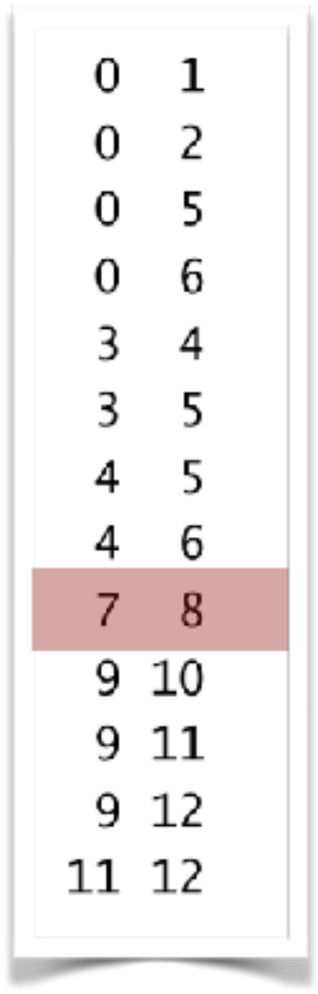


Un graf amb vèrtexs etiquetats segons el seu grau. El *vèrtex aïllat* s'etiqueta amb 0, ja que no és adjacent a cap altre vèrtex.

Representació dels grafs

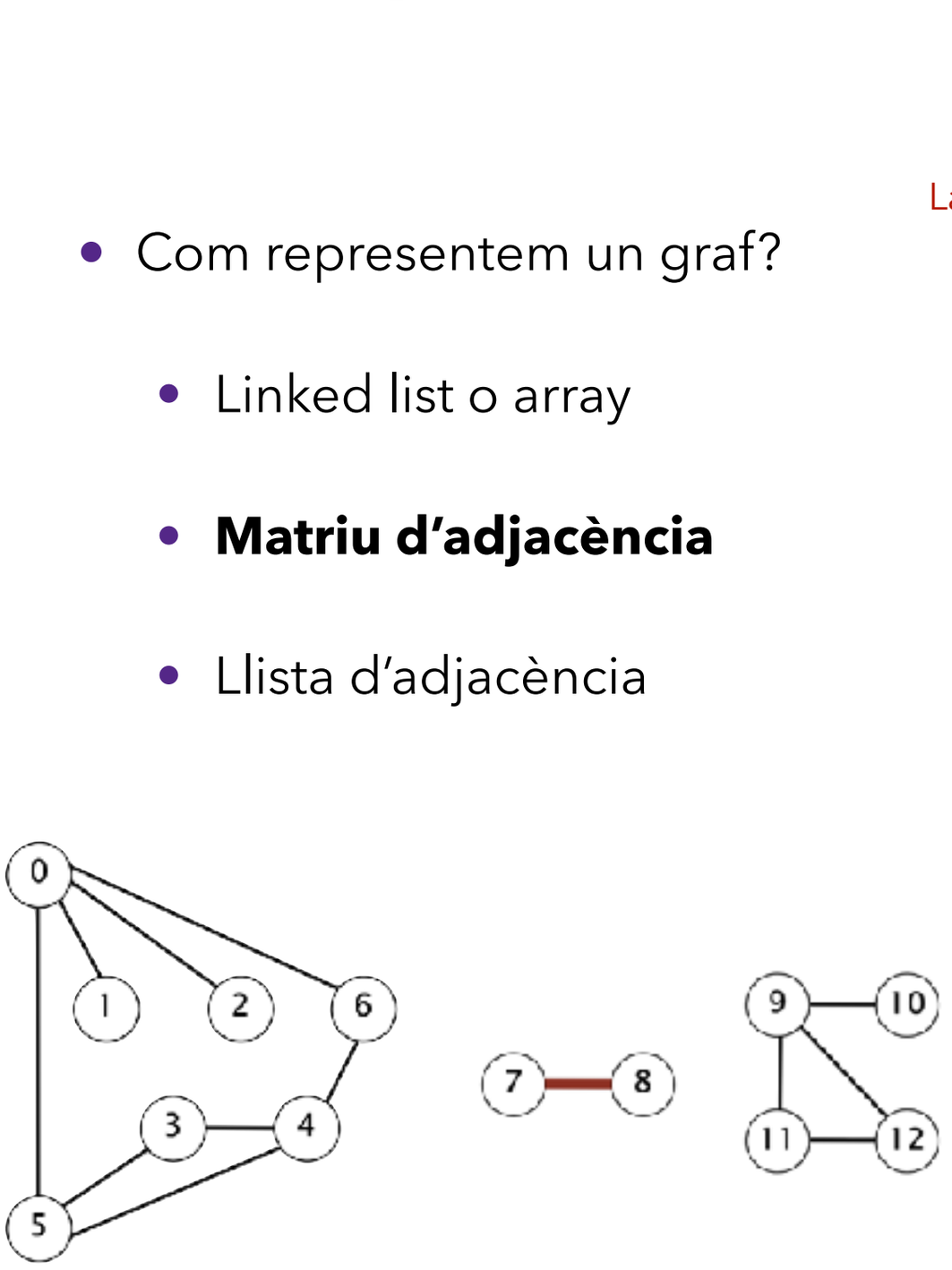
- Com representem un graf?
  - **Linked list o array**
  - Matriu d'adjacència
  - Llista d'adjacència





Representació dels grafs

- Com representem un graf?
  - Linked list o array
  - **Matriu d'adjacència**
  - Llista d'adjacència

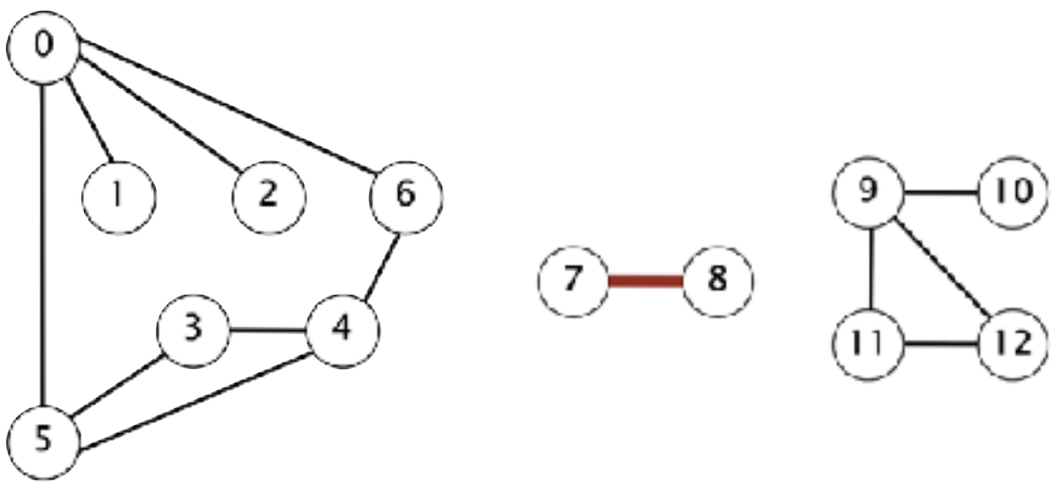


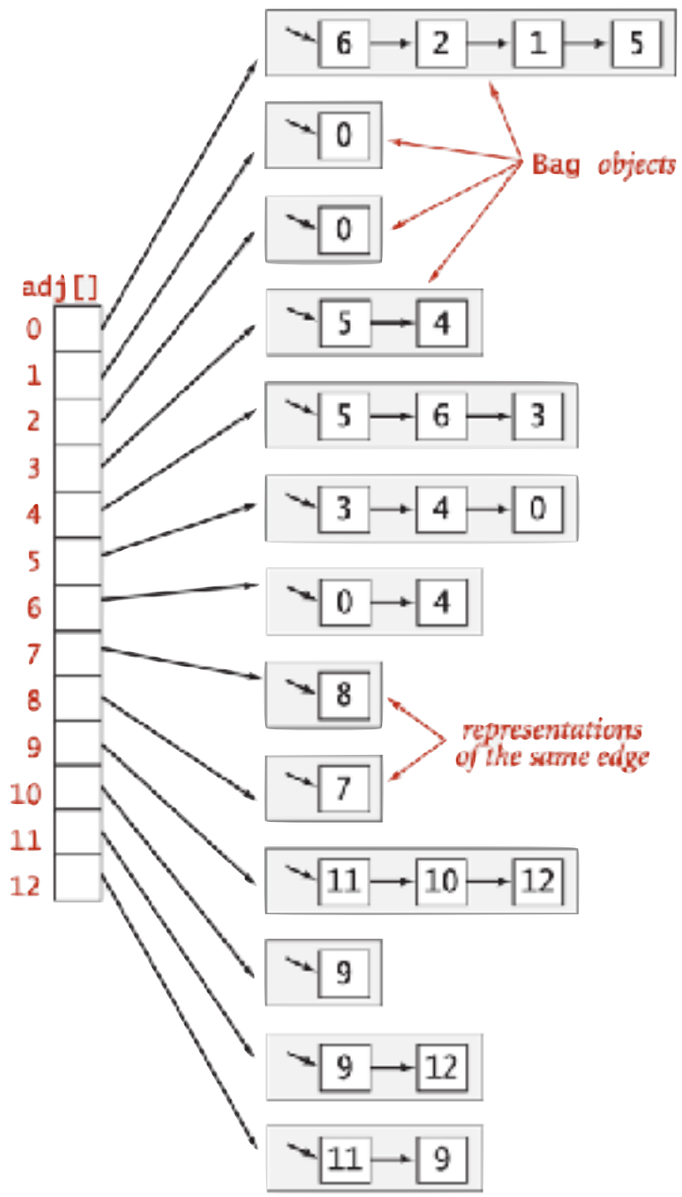
La mateixa dada

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Representació dels grafs

- Com representem un graf?
  - Linked list o array
  - Matriu d'adjacència
  - **Llista d'adjacència**

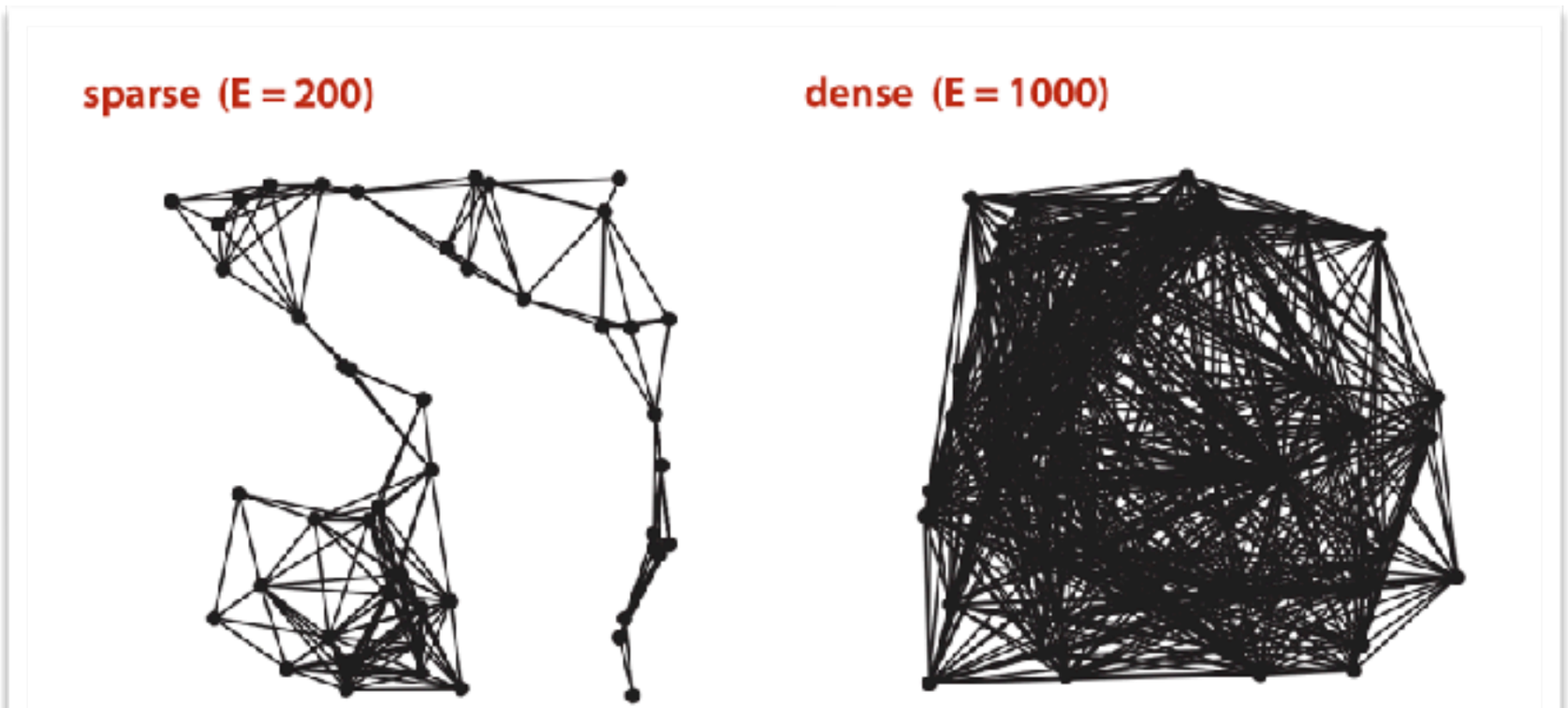




Representació dels grafs

- **A la pràctica**: Quin utilitzar?
  - Els grafs reals acostumen a ser ***sparse***

Alt nombre de nodes,
grau mitja del nodes petit



Two graphs (V = 50)

Representació dels grafs

- **A la pràctica**: Quin utilitzar?
  - Els algorítmies es basen en iterar sobre els nodes adjacents del *v*
  - Els grafs reals acostumen a ser ***sparse***

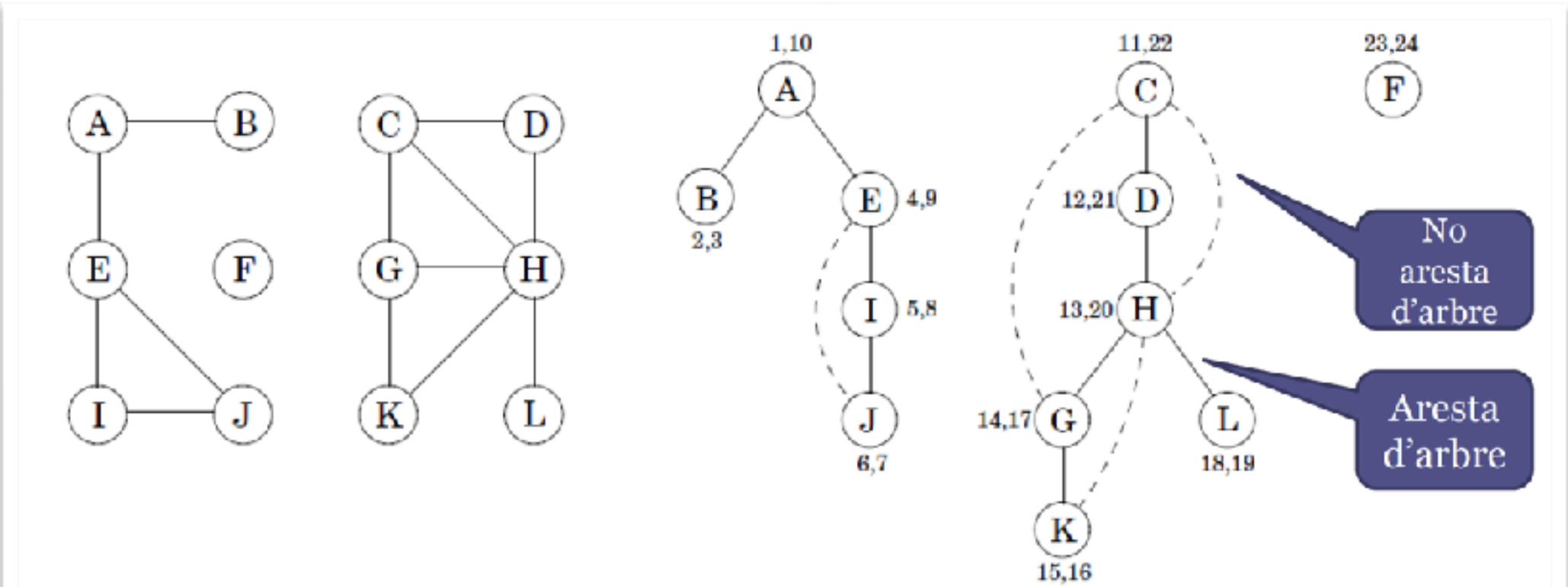
Alt nombre de nodes,
grau mitja del nodes petit

representation	space	add edge	edge between v and w?	iterate over vertices adjacent to v?
list of edges	E	1	E	E
adjacency matrix	V^2	1 *	1	V
adjacency lists	E + V	1	degree(v)	degree(v)

depth-first search

Algorismes sobre grafs

- **DFS** representa la connectivitat amb un **bosc d'arbres**



- Quina és la complexitat del **DFS**?

Exercici per a casa

Algorismes sobre grafs

```
procedure explore(G, v)
Input:    G = (V, E) is a graph; v ∈ V
Output:   visited(u) is set to true for all nodes u reachable from v

visited(v) = true
previsit(v)
for each edge (v, u) ∈ E:
   if not visited(u): explore(u)
postvisit(v)
```

Join at **www.kahoot.it** or with the **Kahoot! app**

with Game PIN:

184513

Go Full Screen

0
Players

Kahoot!

Start

Waiting for players...

UNIVERSITAT DE BARCELONA

DAS

2

56
Answers

- Matriu Adjacència == Llista Adjacència == |V+E|
- Matriu Adjacència == Llista Adjacència == |V2|
- Matriu Adjacència = |V+E| & Llista Adjacència = |V2|
- Matriu Adjacència = |V2| & Llista Adjacència = |V+E|

DFS complexitat

- Linked list o array = $\Theta(|E|^2)$
- Matriu d'adjacència = $\Theta(|V|^2)$
- Llista d'adjacència = $\Theta(|V|+|E|) \rightarrow \Theta(2|E|) \rightarrow \Theta(|E|)$

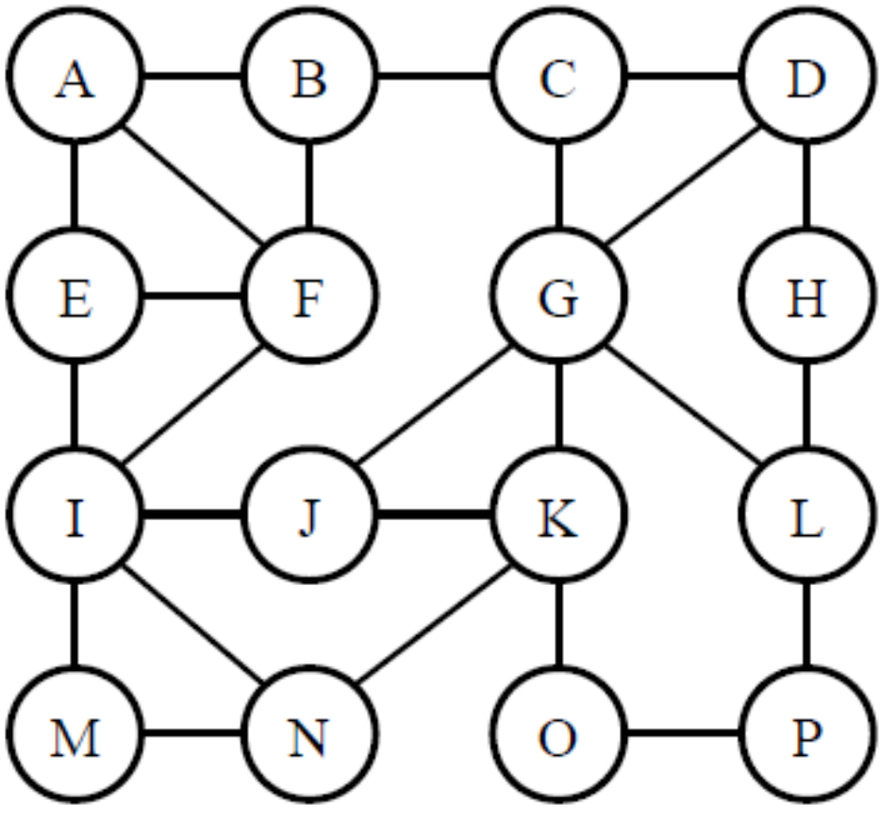
Algorismes sobre grafs

- Complexitat de DFS?
  - **TEOREMA**: la suma de tots els graus de tots els nodes és igual a 2 cops el nombre d'arestes del graf.

$$\Theta(|V|+|E|) \rightarrow \Theta(2|E|) \rightarrow \Theta(|E|)$$

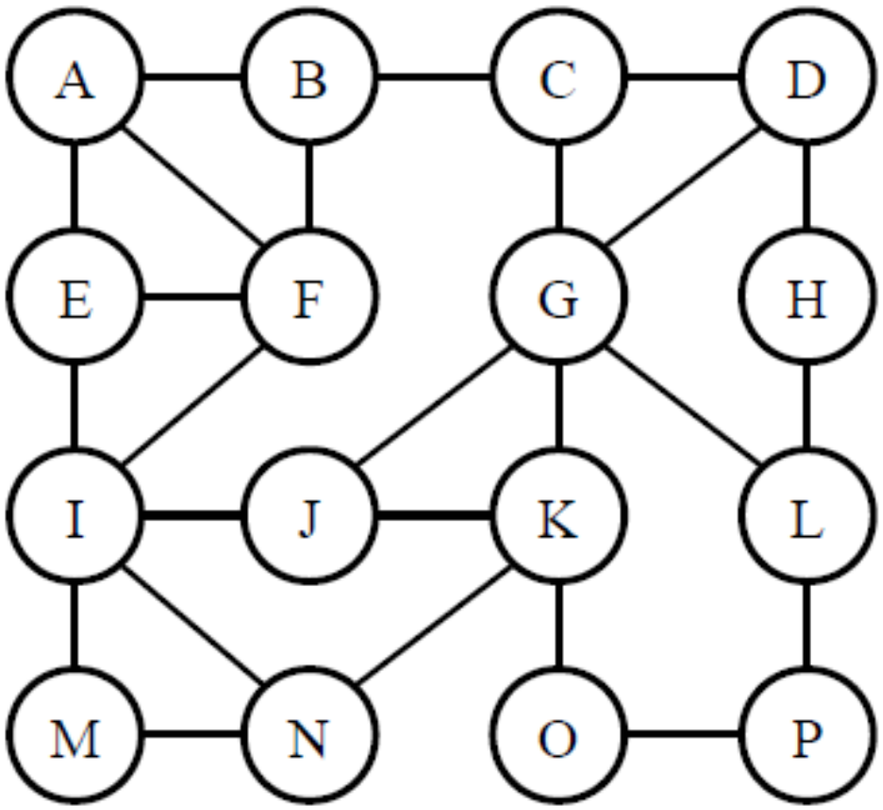
Algorismes sobre grafs

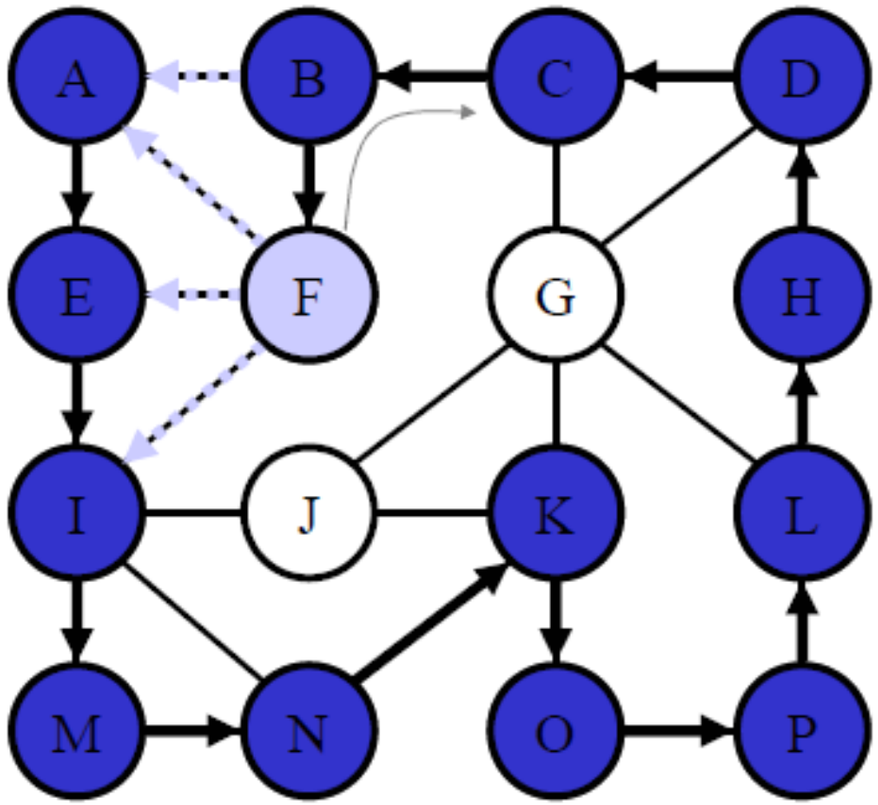
Exemple:



Algorismes sobre grafs

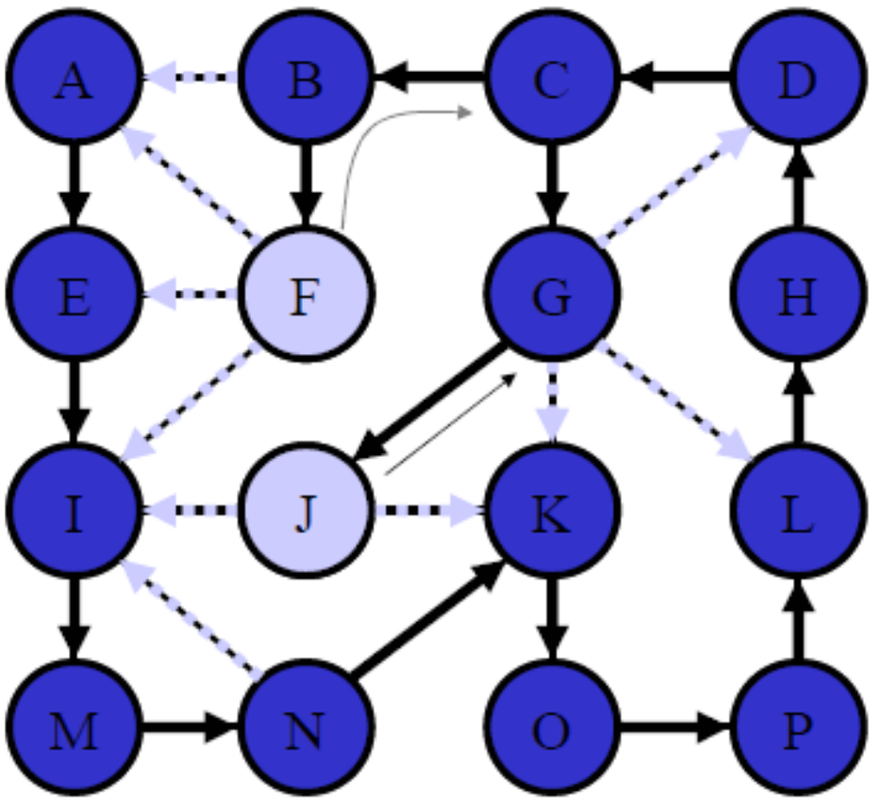
Exemple:

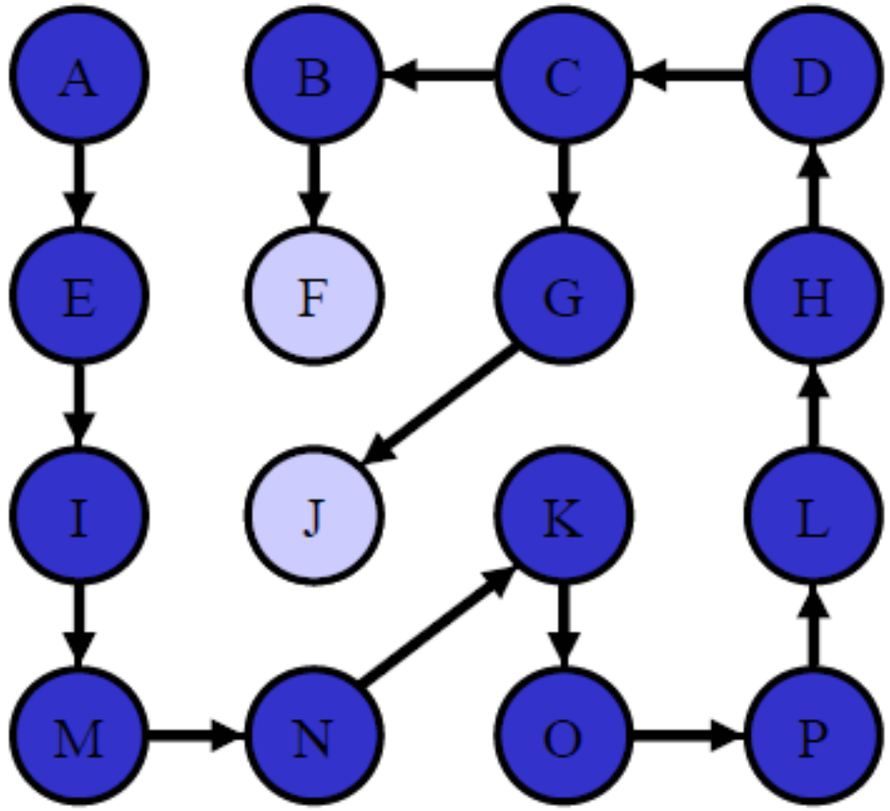




Algorismes sobre grafs

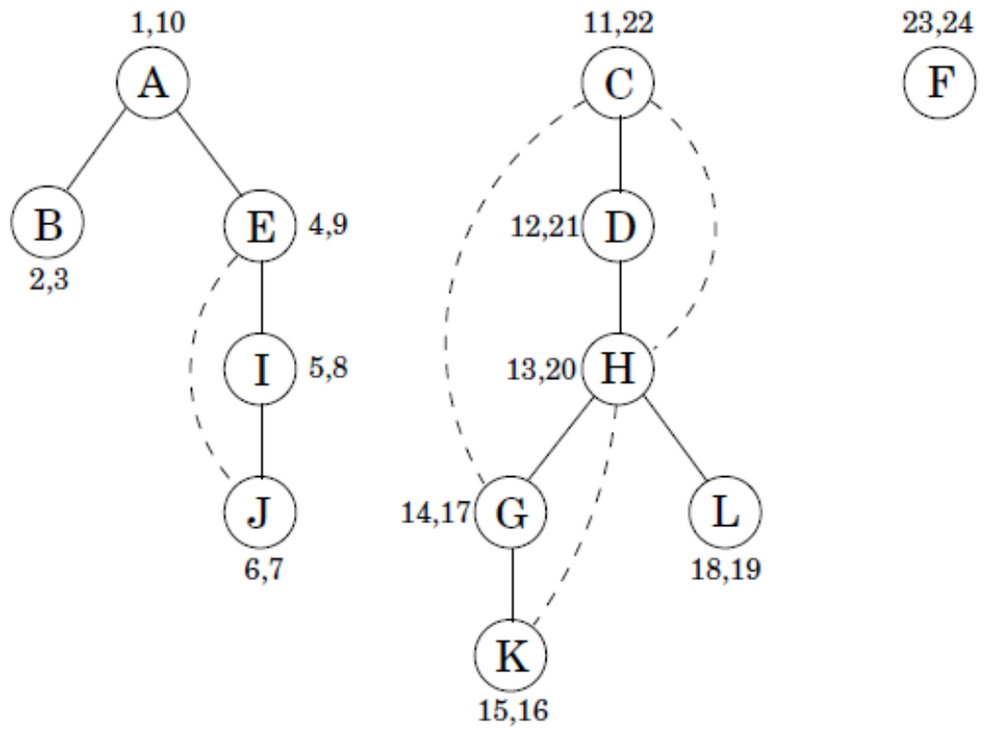
Exemple:





Algorismes sobre grafs

- DFS també ens soluciona un altre problema de grafs: els components connexos

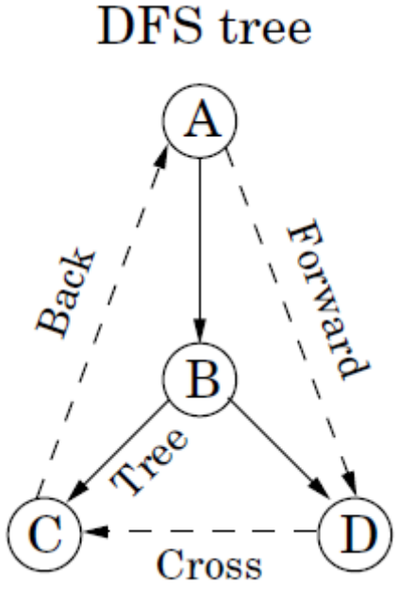


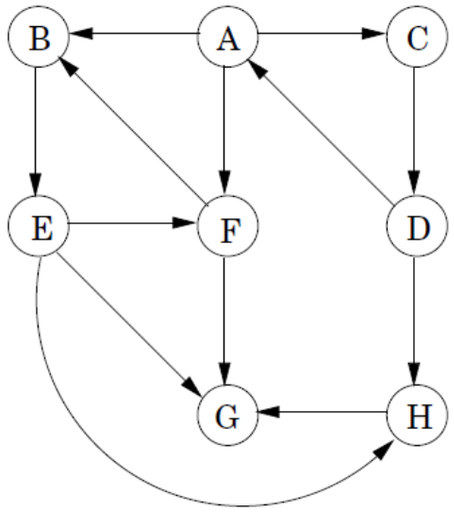
$\{A, B, E, I, J\}$ $\{C, D, G, H, K, L\}$ $\{F\}$

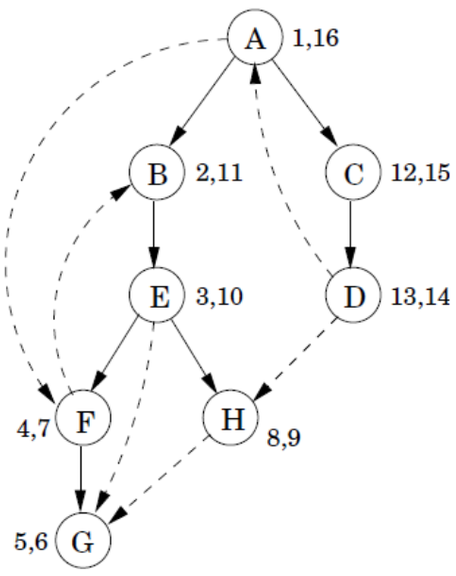
Cada crida a *explore* crea un nou arbre i es troba una **nova component connexa**

Algorismes sobre grafs

- Quins vèrtexs són accessibles des de quins?
- **Grafs dirigits**







Algorismes sobre grafs

- En els grafs **no dirigits**, una **component connexa** conté com a mínim un **cicle**

- En els grafs **dirigits**, un **cicle** ha de començar i acabar en el mateix vèrtex existint connectivitat, sinó és **acíclic**

$$v_0 \longrightarrow v_1 \longrightarrow v_2 \longrightarrow \cdots \longrightarrow v_k \longrightarrow v_0$$

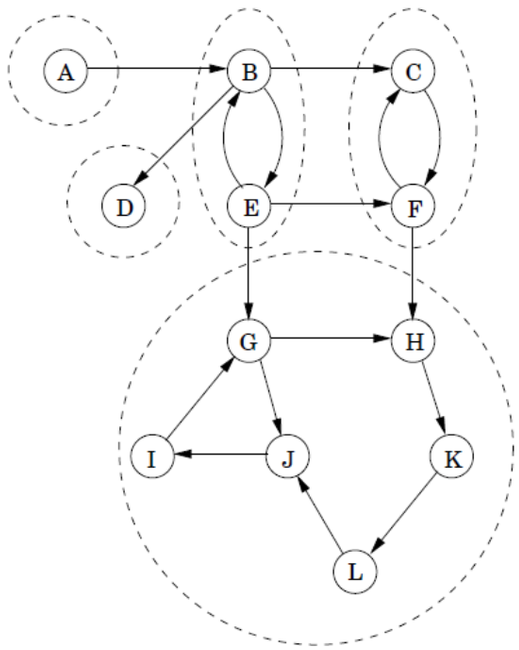
Algorismes sobre grafs

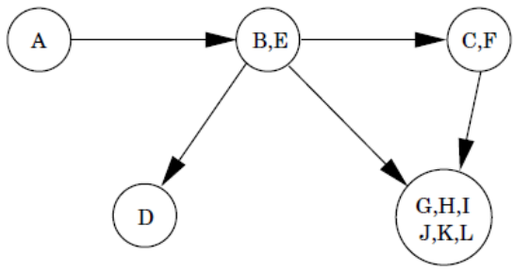
- Els grafs dirigits **acíclics** són molt comuns:
- Nosaltres modelem ó “intentem” modelar les nostres tasques quotidianes en un ordre determinat, una rere l’altre.

  - *Els grafs **acíclics** modelen **relacions** com jerarquies o dependències temporals*

Algorismes sobre grafs

- Connectivitat en grafs dirigits
- Hi ha d'haver connectivitat ***u*** → ***v*** i ***v*** → ***u***
- **Components forts connexes**





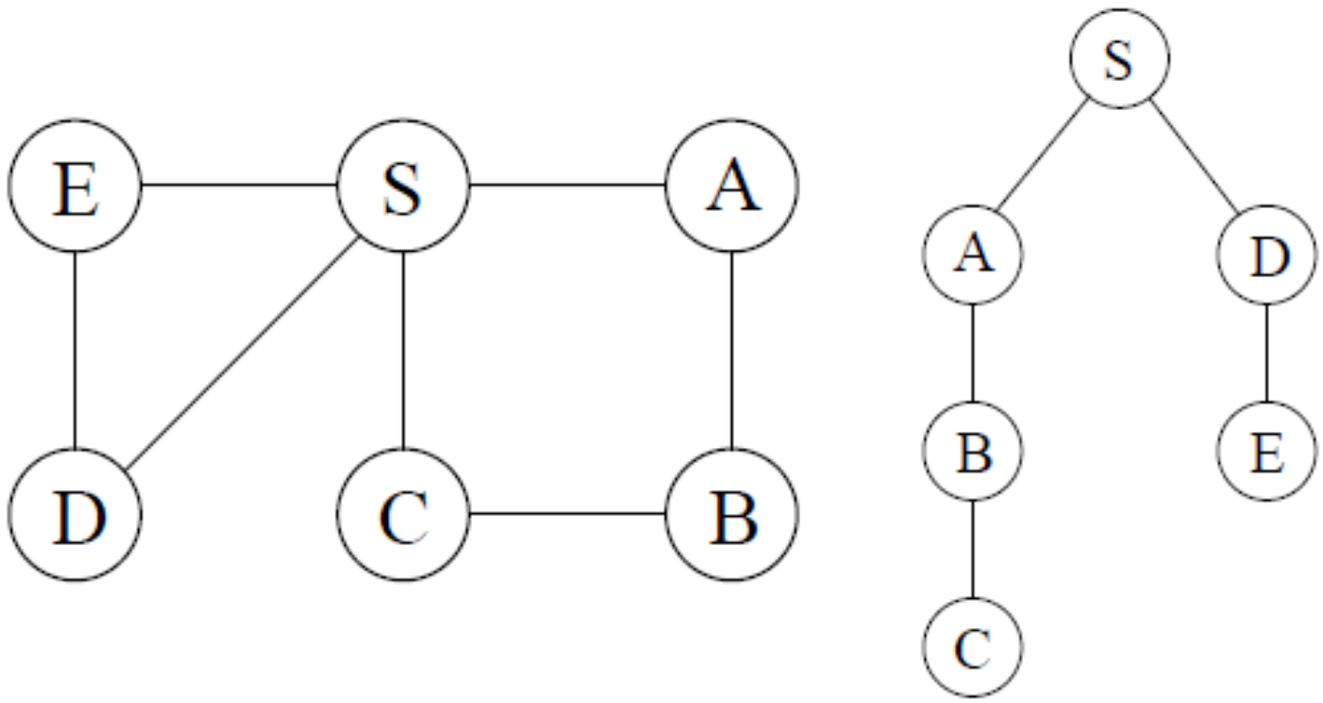
També els podem trobar amb complexitat lineal fent ús de l'algorisme DFS

Algorismes sobre grafs

- Fins ara hem parlat de **connectivitat**, però no hem analitzat el cost del camí que hem trobat entre els punts connectats.

- **DFS assegura el camí més curt entre 2 punts connectats en un graf no dirigit???**

Algorismes sobre grafs



Podem definir el **camí més curt entre 2 vèrtexs** com el número d'arestes fins arribar, o el número de vèrtexs que travessem, o la suma dels pesos de les arestes, dels vèrtexs, etc.

Aplicacions

Depth-first search application: flood fill

Challenge. Flood fill (Photoshop magic wand).

Assumptions. Picture has millions to billions of pixels.

input

floodFill mask

Aplicacions

Depth-first search application: flood fill

Challenge. Flood fill (Photoshop magic wand).

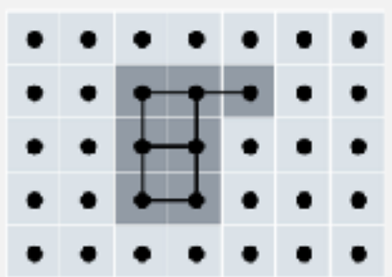
Assumptions. Picture has millions to billions of pixels.

input

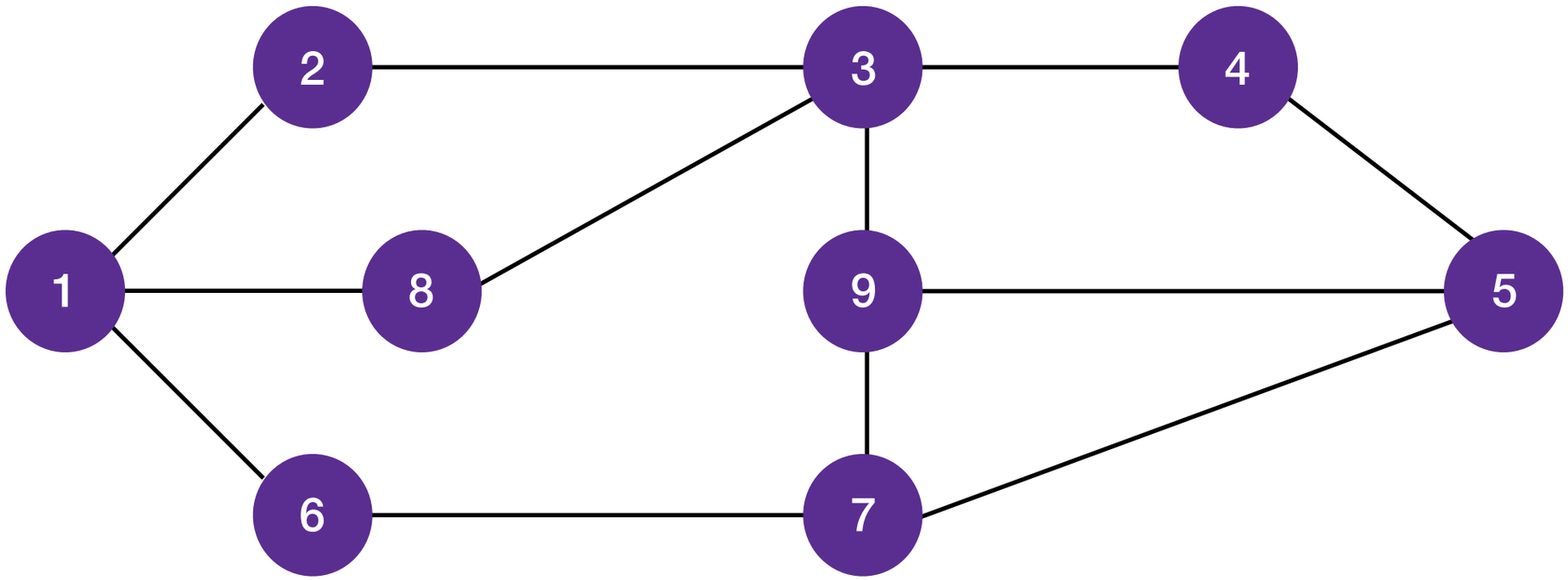
floodFill mask

Solution. Build a grid graph.

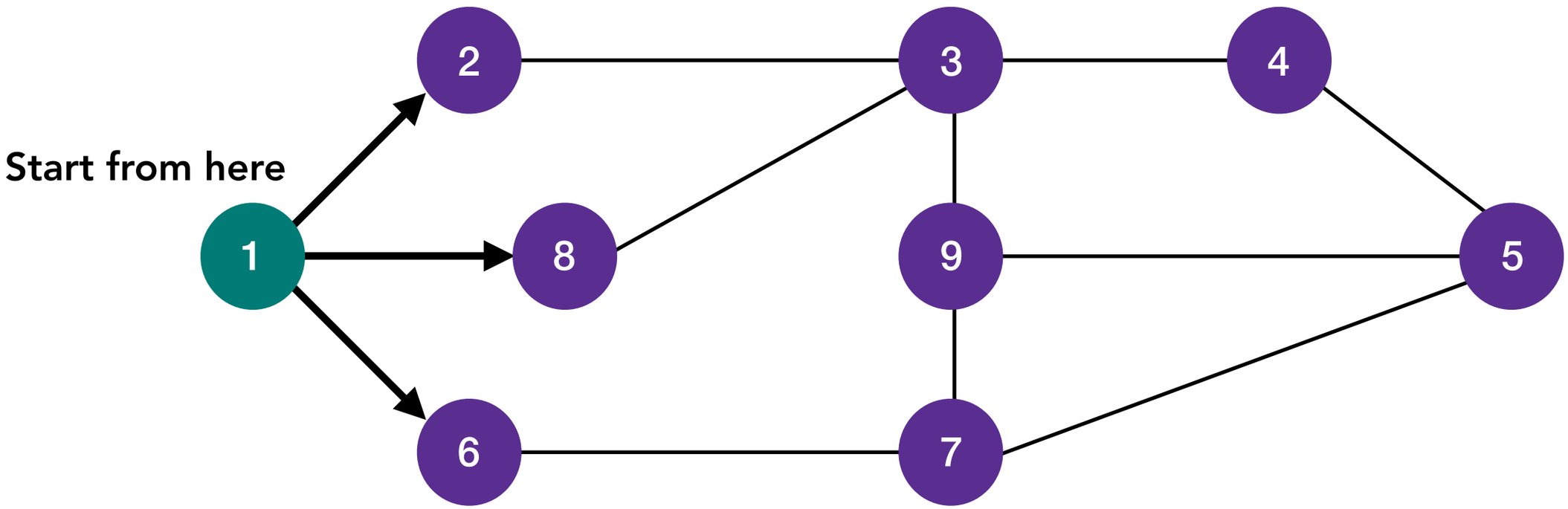
- Vertex: pixel.
- Edge: between two adjacent gray pixels.
- Blob: all pixels connected to given pixel.



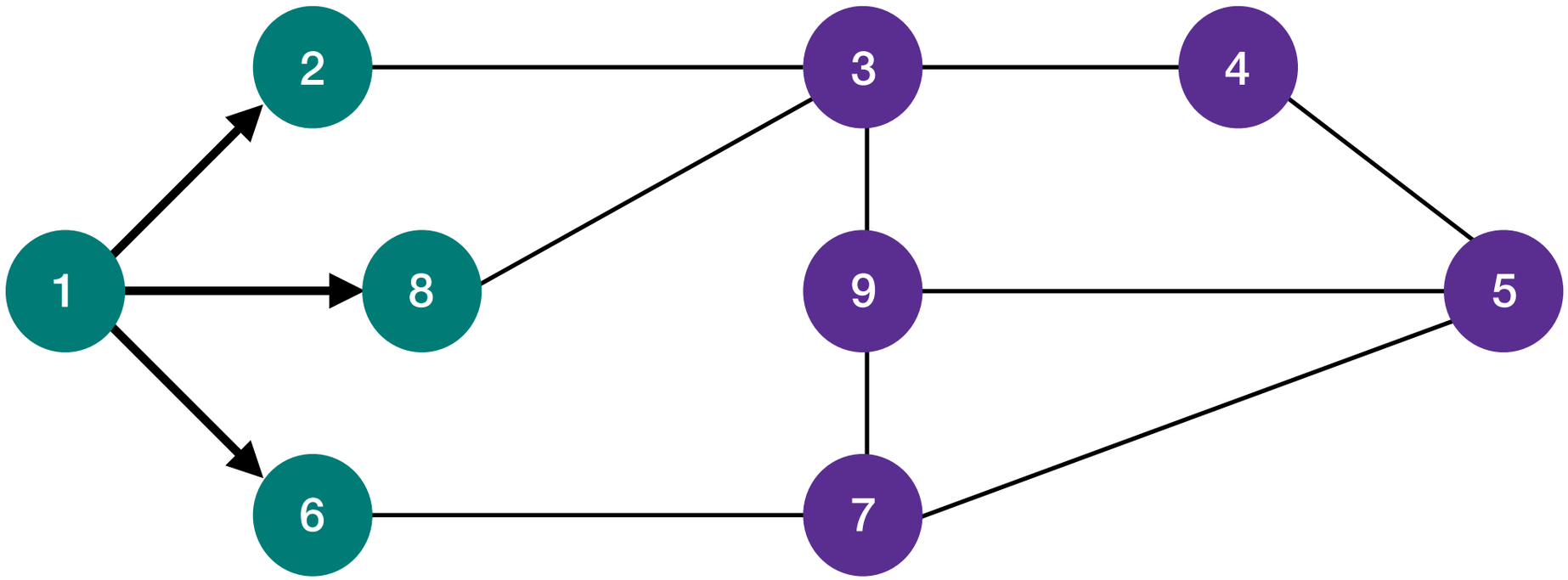
breadth-first search recorregut en amplada!



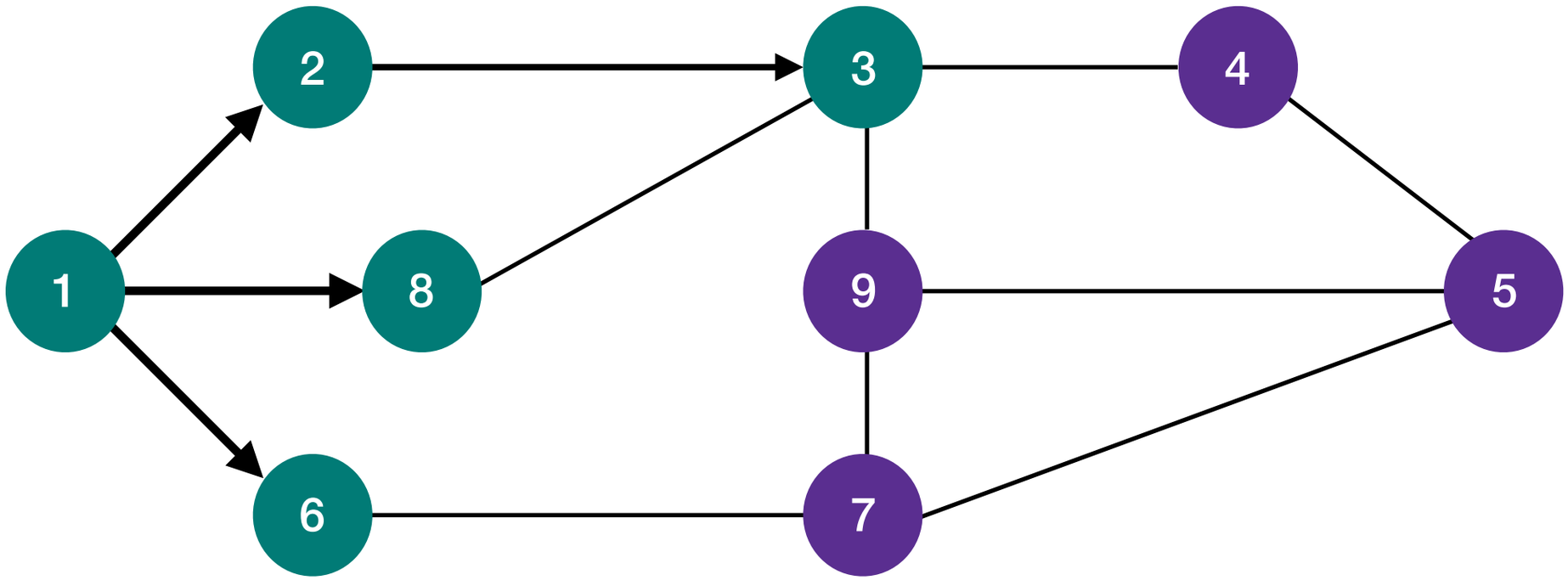
Output:



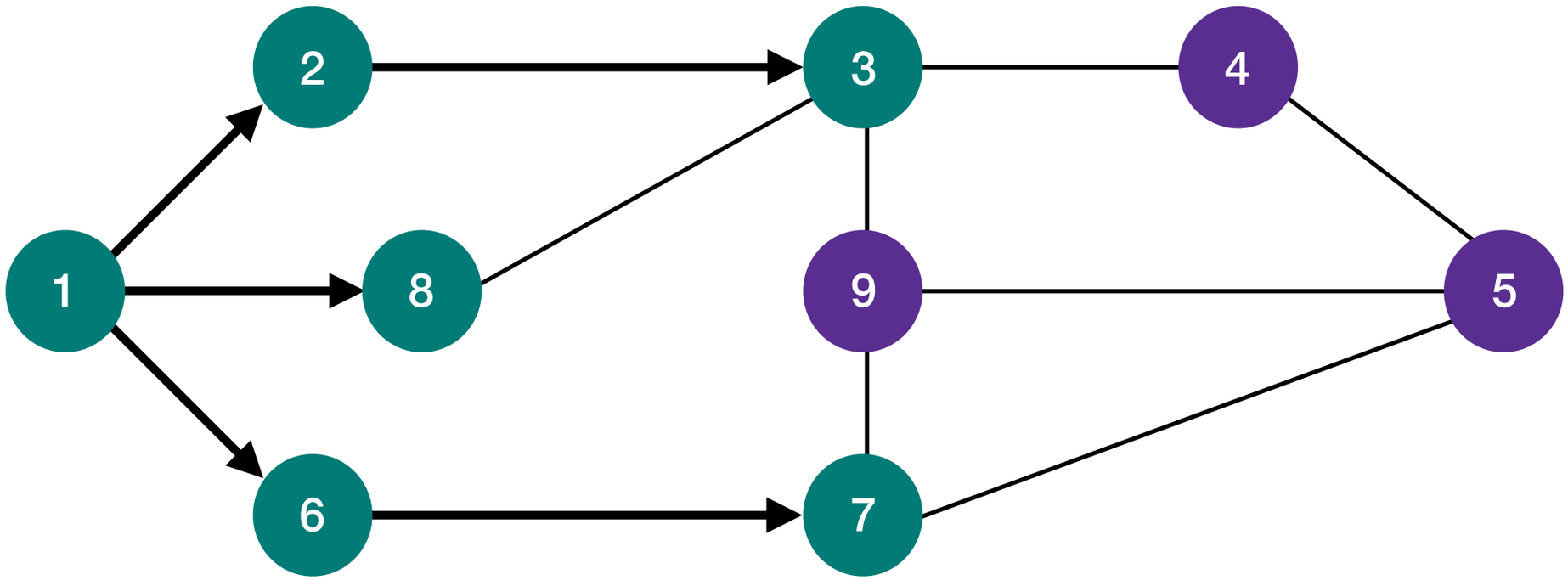
Output: 1



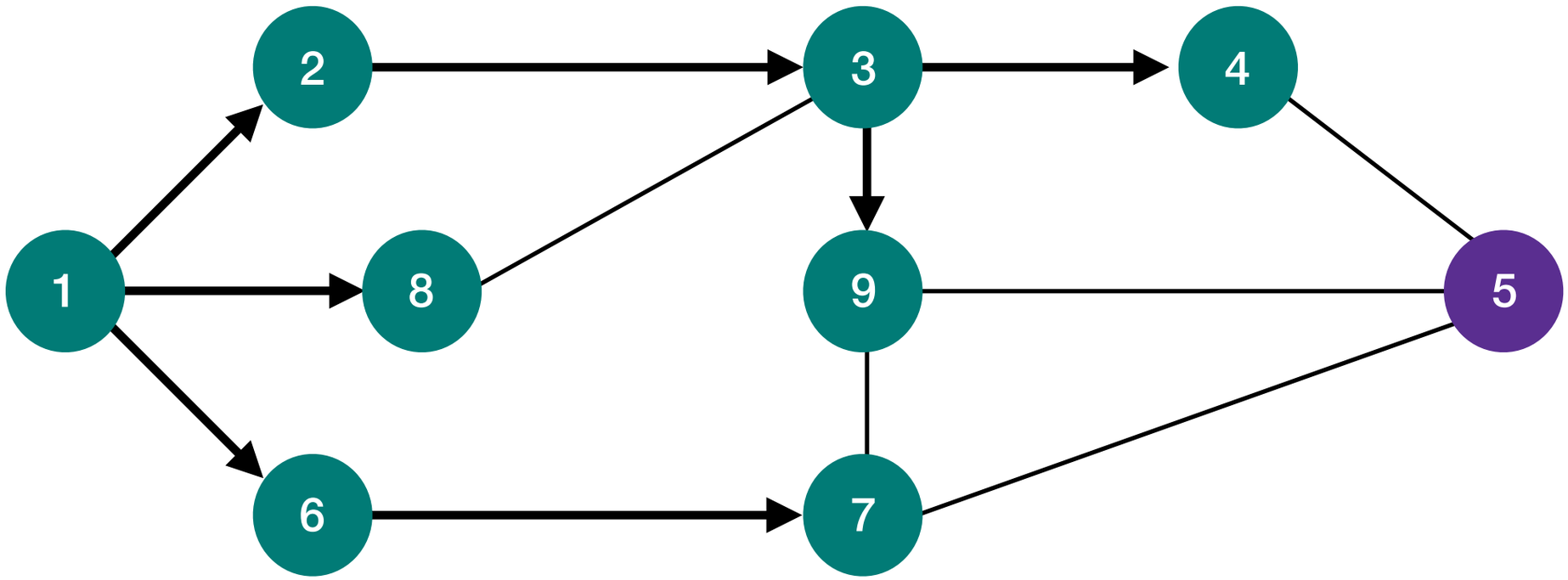
Output: 1 - 2 - 6 - 8



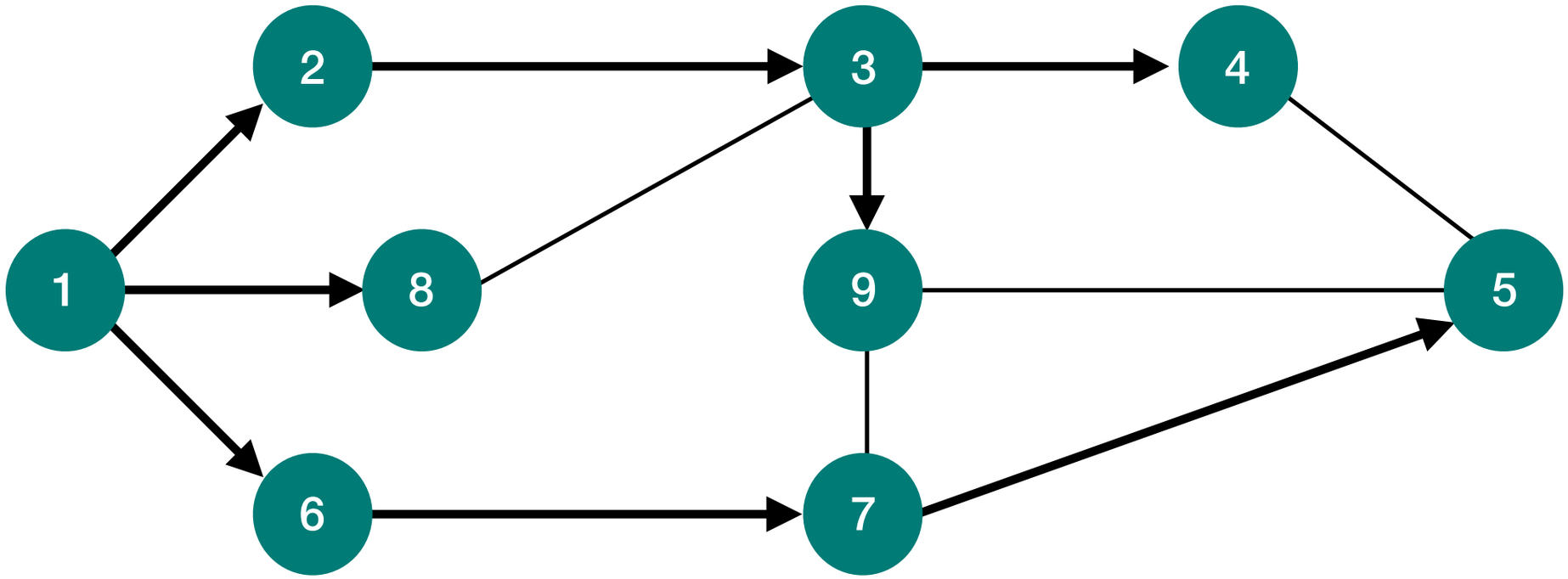
Output: 1 - 2 - 6 - 8 - 3



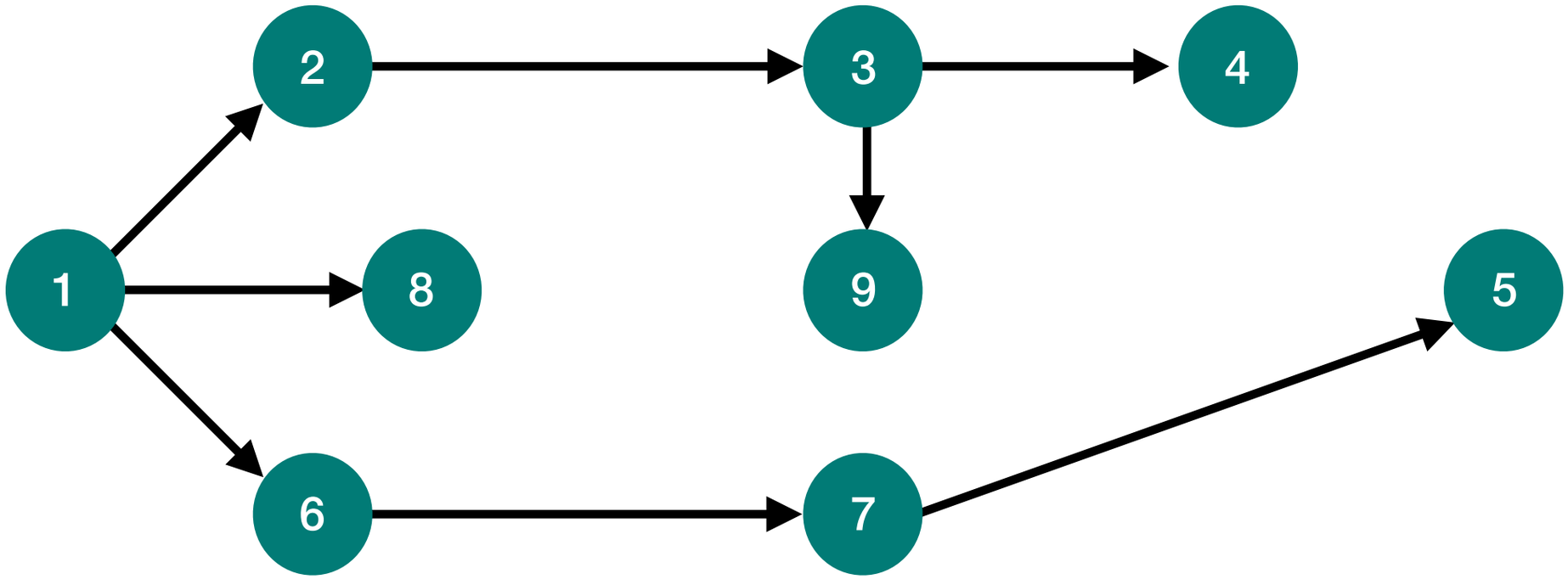
Output: 1 - 2 - 6 - 8 - 3 - 7



Output: 1 - 2 - 6 - 8 - 3 - 7 - 4 - 9



Output: 1 - 2 - 6 - 8 - 3 - 7 - 4 - 9 - 5


2
3
4
1
8
9
5
6
7

Alguna idea?

Algorismes sobre grafs

- **BFS** té un codi similar a **DFS**, però fa ús d'una **cua** en lloc d'una **pila**.

- Els arbres generats per BFS es diuen **arbres de camí mínim.**

- Si fem ús correcte del "cost del camí" a l'algorisme BFS trobem el camí mínim d'un vèrtex a la resta de vèrtex dins d'un graf!

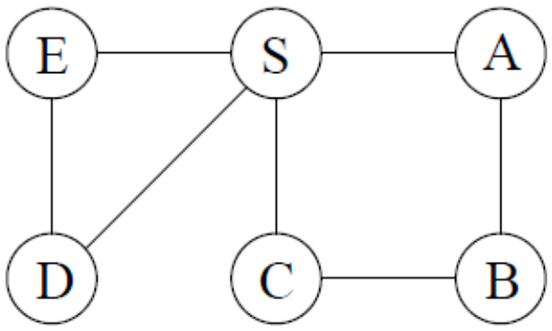
Algorismes sobre grafs

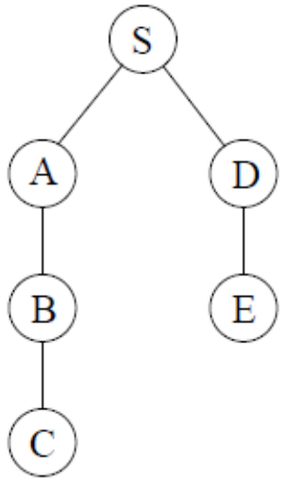
- Cerca en amplada (**Breadth-first search**): **BFS**

```
# visits all the nodes of a graph (connected component) using BFS
def bfs_connected_component(graph, start):
    # keep track of all visited nodes
    explored = []
    # keep track of nodes to be checked
    queue = [start]

    # keep looping until there are nodes still to be checked
    while queue:
        # pop shallowest node (first node) from queue
        node = queue.pop(0)
        if node not in explored:
            # add node to list of checked nodes
            explored.append(node)
            neighbours = graph[node]

            # add neighbours of node to queue
            for neighbour in neighbours:
                queue.append(neighbour)
    return explored
```

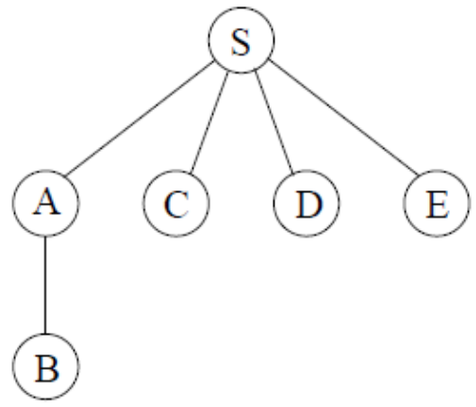
Algorismes sobre grafs

GRAF **DFS** **BFS**



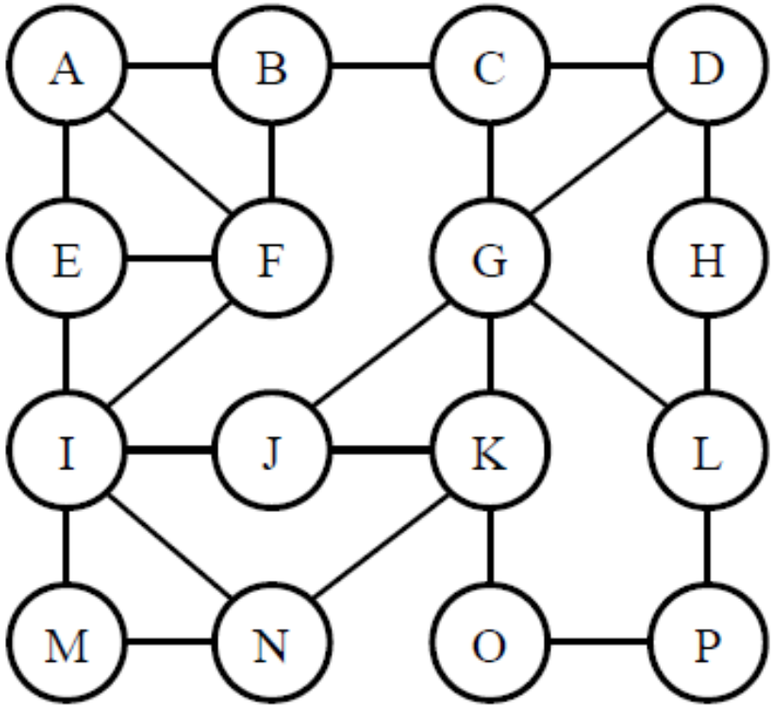


Order of visitation	Queue contents after processing node
	$[S]$
S	$[A\ C\ D\ E]$
A	$[C\ D\ E\ B]$
C	$[D\ E\ B]$
D	$[E\ B]$
E	$[B]$
B	$[\,]$

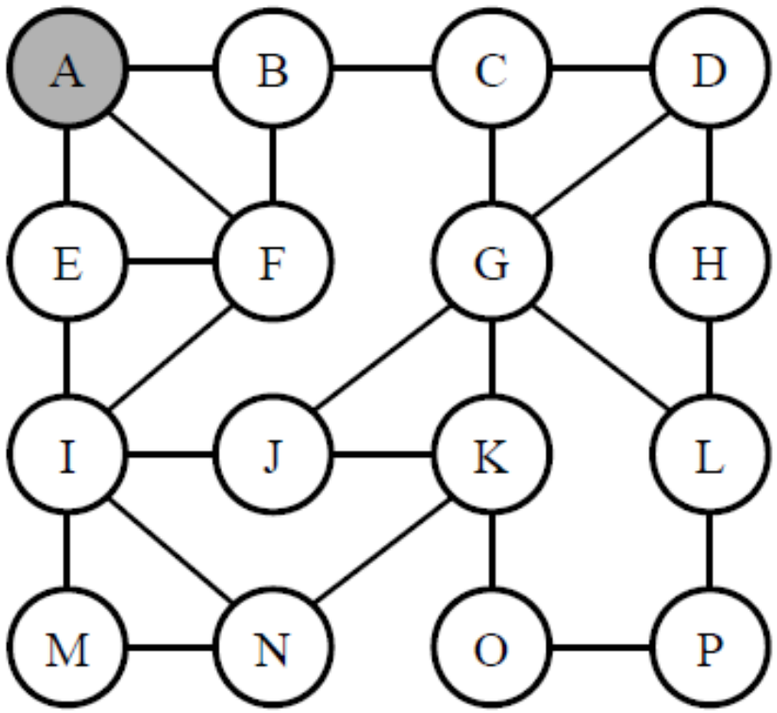


Algorismes sobre grafs

- Exemple

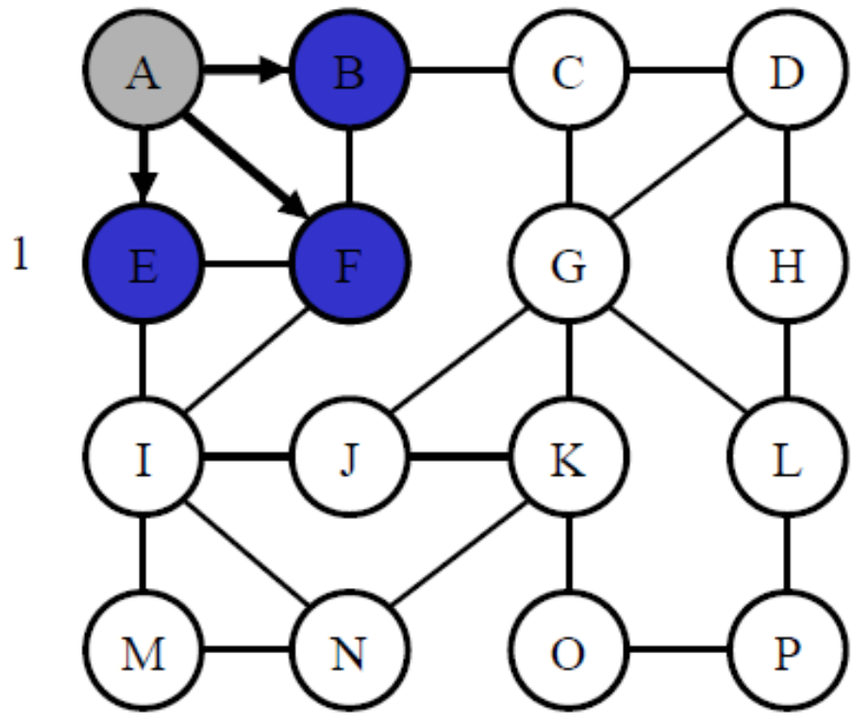


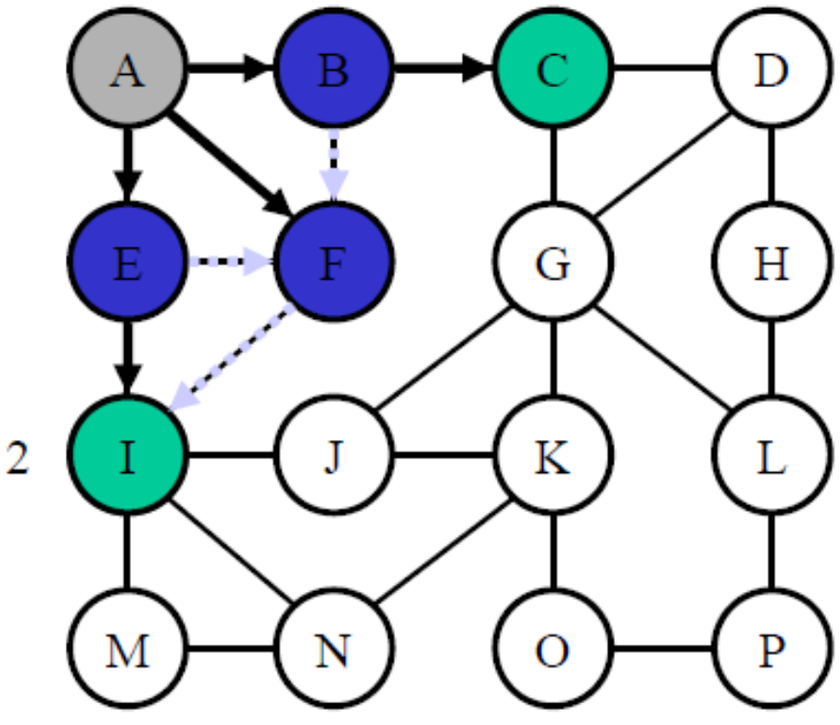
0



Algorismes sobre grafs

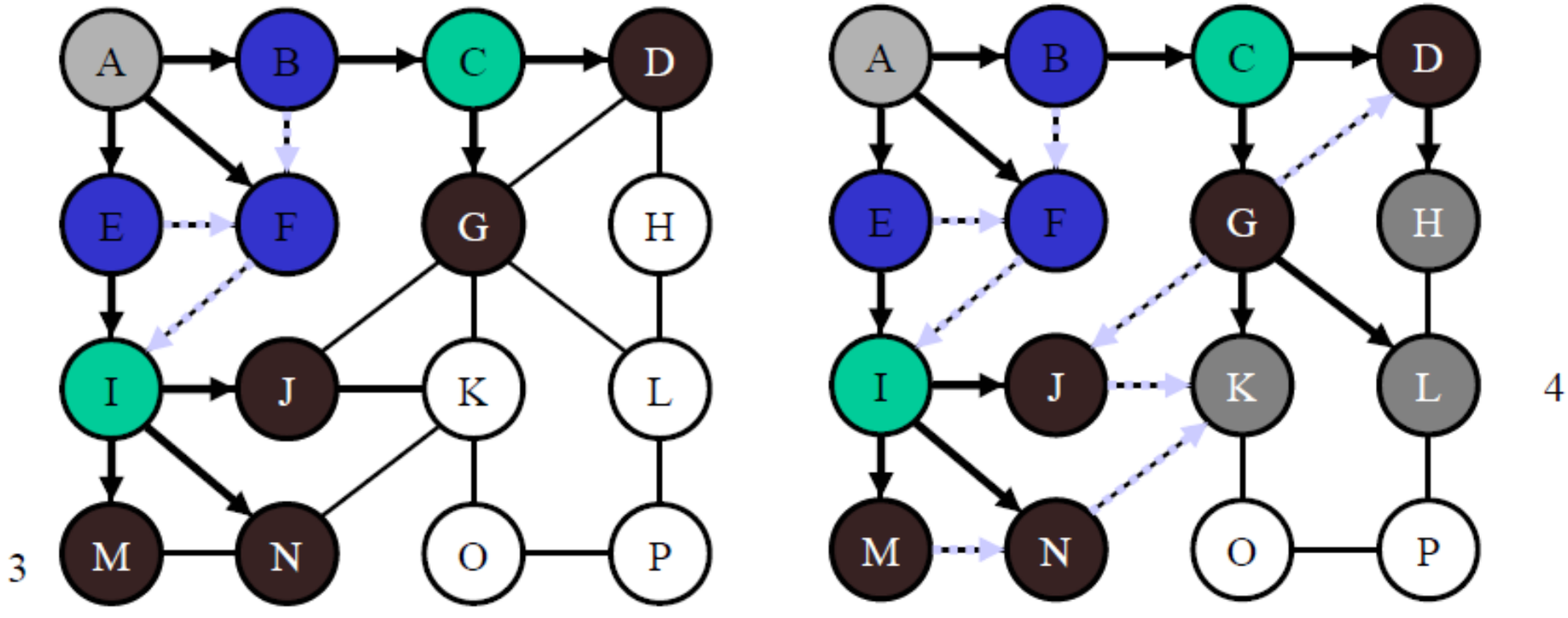
- Exemple





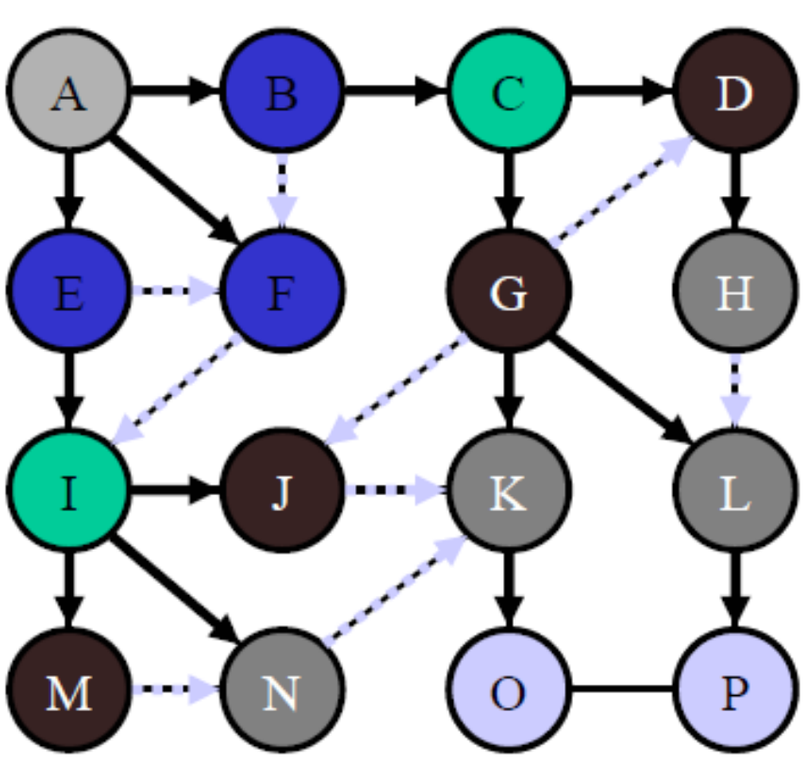
Algorismes sobre grafs

- Exemple

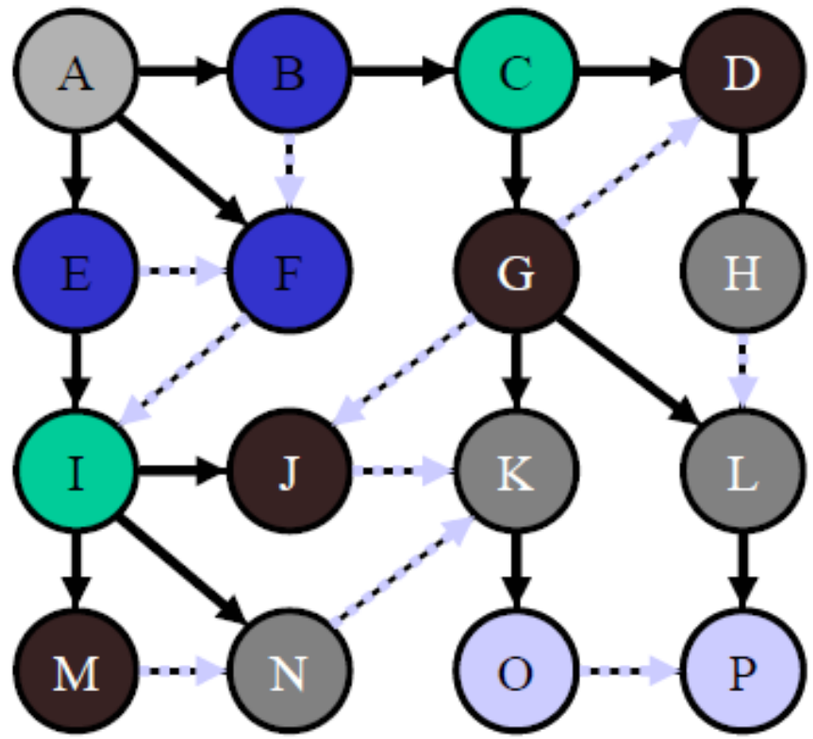


Algorismes sobre grafs

- Exemple

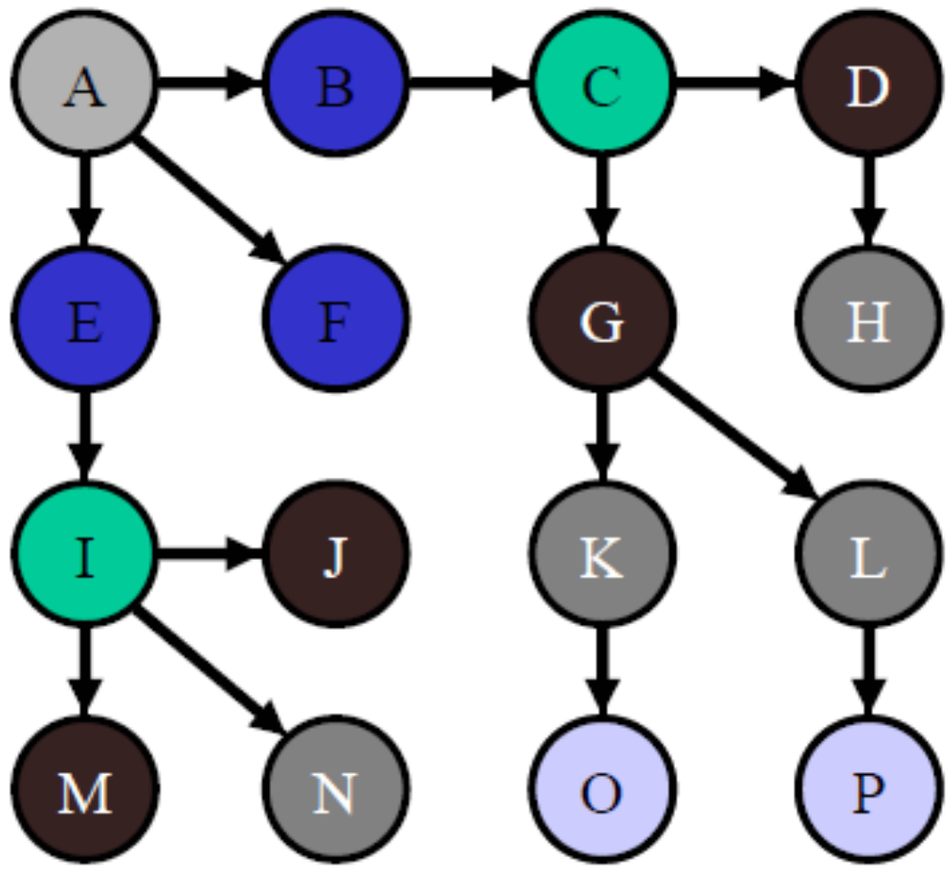


5



Algorismes sobre grafs

- Exemple



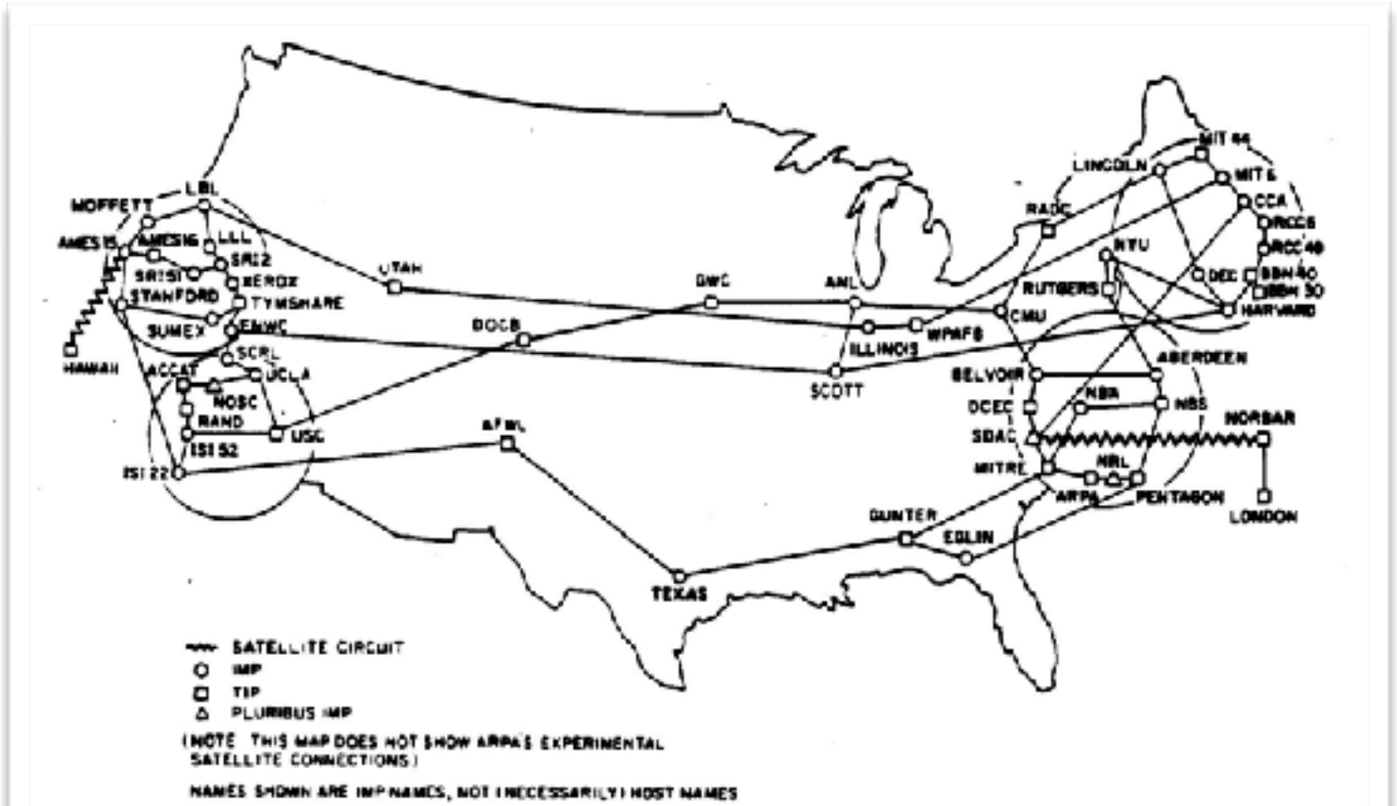
Algorismes sobre grafs

- BFS vs DFS
  - Una altra diferència és que BFS només té en compte els nodes que estan connectats a un node *s*, els altres són ignorats

→ només es genera un arbre de camins mínims

Applications

- El menor nombre de salts en una xarxa de comunicació.



ARPANET, July 1977

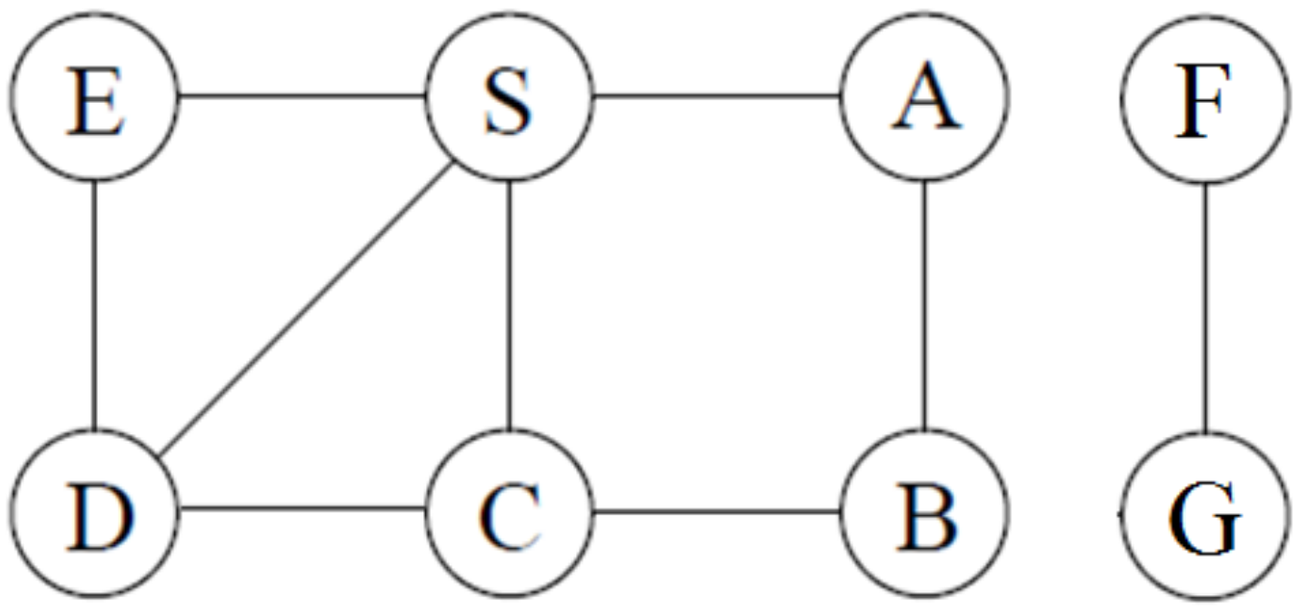
Aplicacions

- Ruta més curta de gràfics no ponderats
- Descobriu tots els nodes accessibles des d'un vèrtex inicial
- Trobar els nodes veïns a xarxes peer-2-peer com BitTorrent.
- Els rastrejadors usats pels motors de cerca per visitar enllaços d'una pàgina web i seguir fent el mateix de manera recursiva.
- Trobar les persones a una distància determinada a les xarxes socials.
- Identificar totes les ubicacions veïnes dels sistemes GPS.
- Cercar si hi ha un camí entre dos nodes d'un gràfic.
- Permet que els paquets emesos arribin a tots els nodes d'una xarxa.

Exercicis

1. Dibuixa els arbres DFS indicant els passos a cada node del següent graf.

2. Dibuixa començant amb explore(S) l'arbre BFS del mateix graf, indicant en cada iteració el contingut de la cua



Exercici

- Identifica el camí més curt entre dos nodes. Escriu el pseudo-codi.

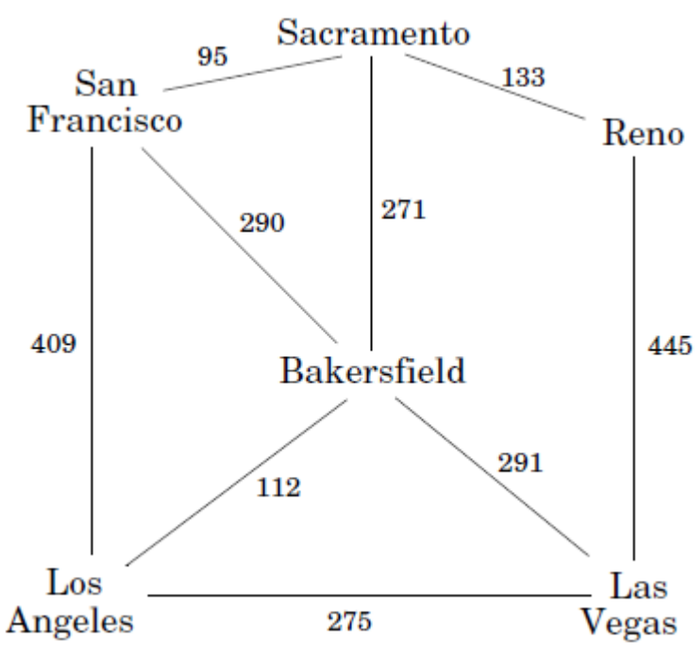
Grafs amb pesos

Algorismes sobre grafs

- **Pesos a les arestes**

  Exemple amb distàncies

$e \in E$	Aresta
l_e	Longitud
$e = (u, v)$	Notació d'aresta I
$l(u, v)$	Notació d'aresta II
l_{uv}	Notació d'aresta III

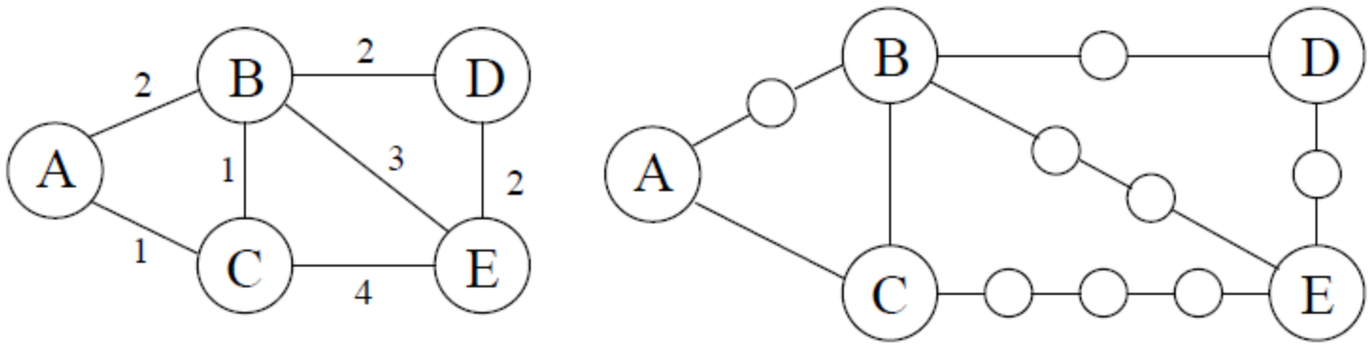


Algorismes sobre grafs

- De moment suposem que tots els pesos són positius >= 0
- BFS troba camins mínims on les arestes tenen un cost unitari.
- Cóm ho fem general per a qualsevol graf G=(V,E) amb l_e enters positius?
  - **Algorisme de Dijkstra**

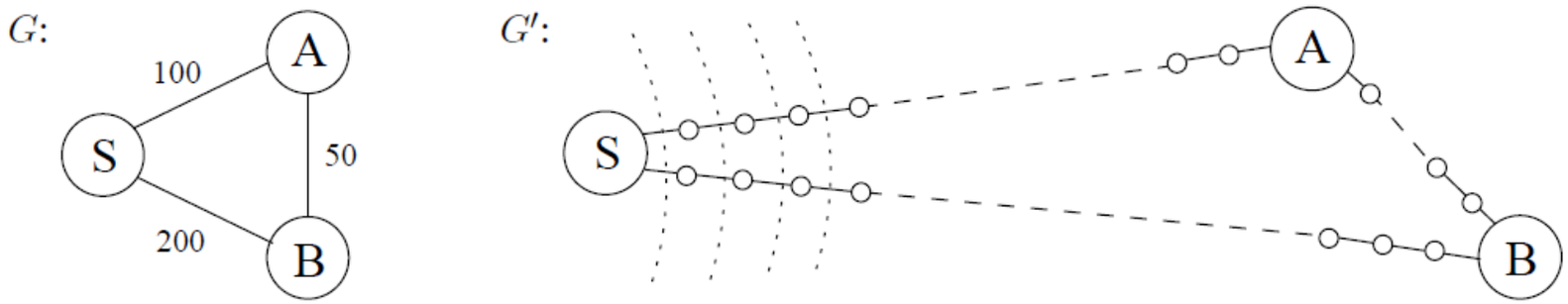
Algorismes sobre grafs

- Algorisme de **Dijkstra**
  - Una versió per fer ús de BFS
  - Dividir les longituds en valors unaris incloent vèrtexs extra



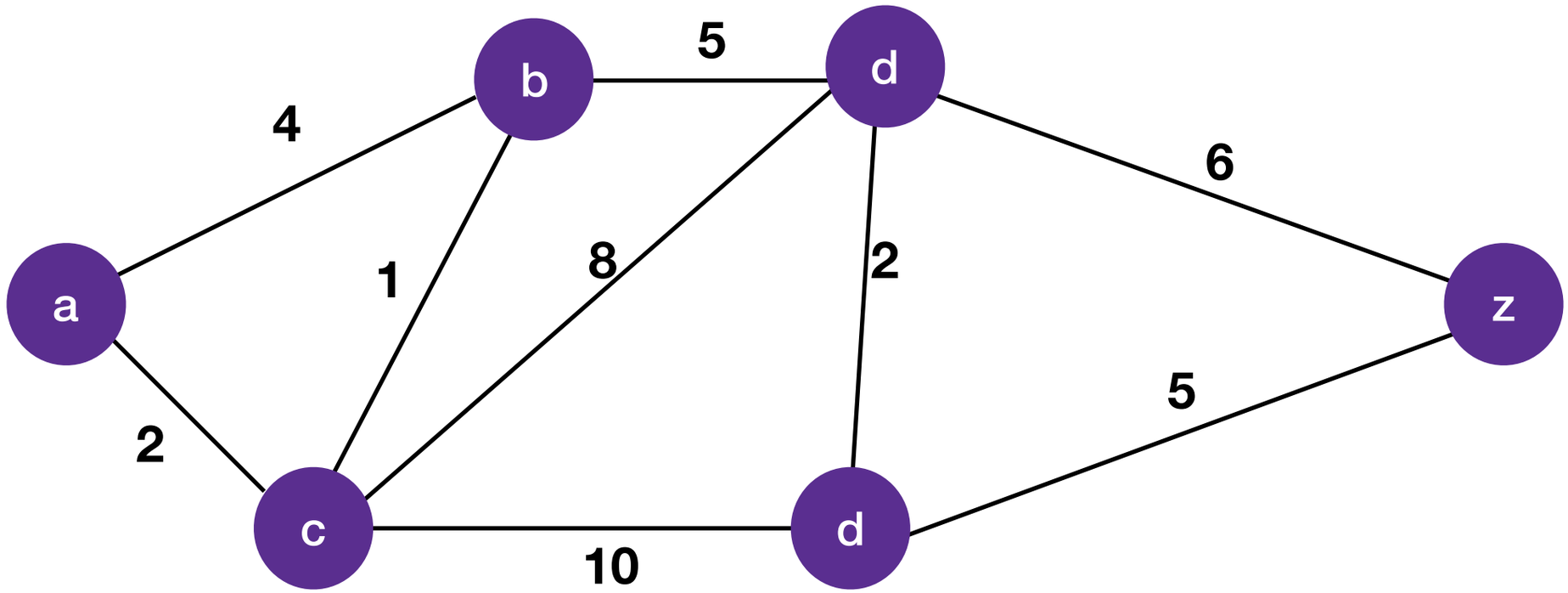
Algorismes sobre grafs

- Un **problema evident**..

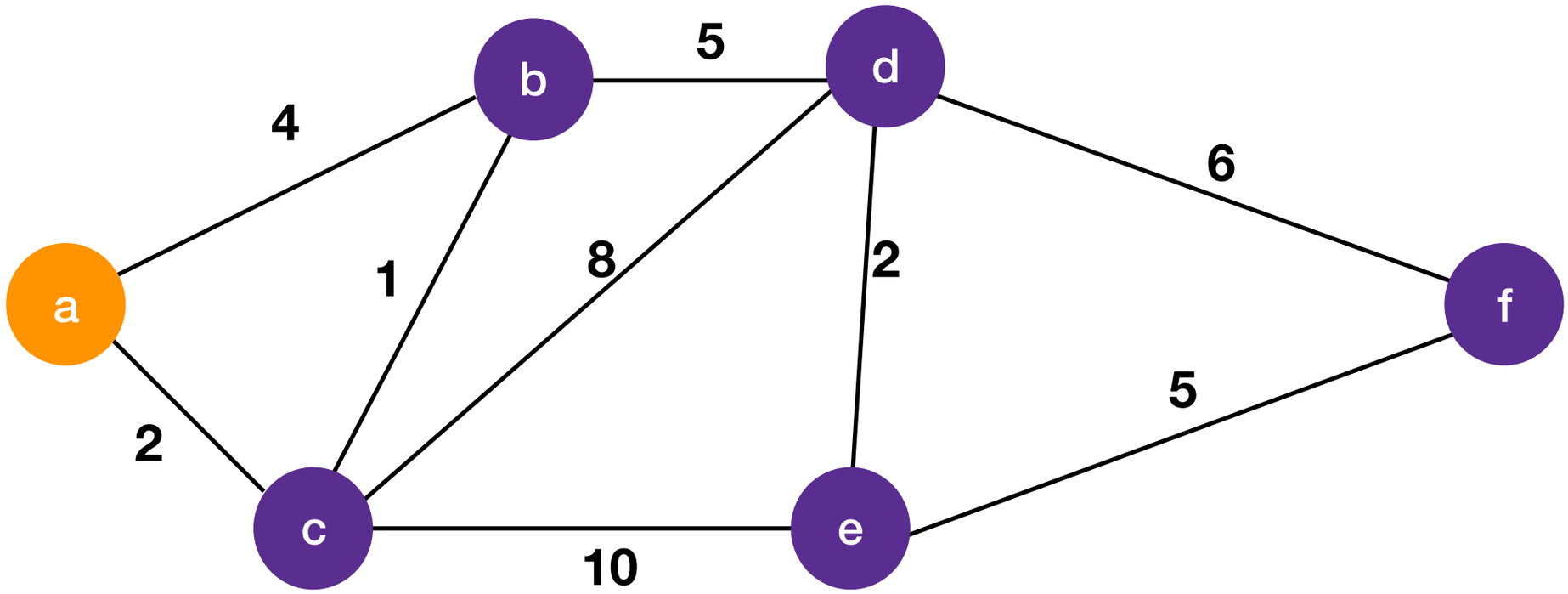


- Pensem millor en posar una “alarma” a cada node i l’actualitzem a mida que arribem fent ús de DFS. Els valors de les alarmes podrien ser els costos de les arestes!!!

Algorismes sobre grafs

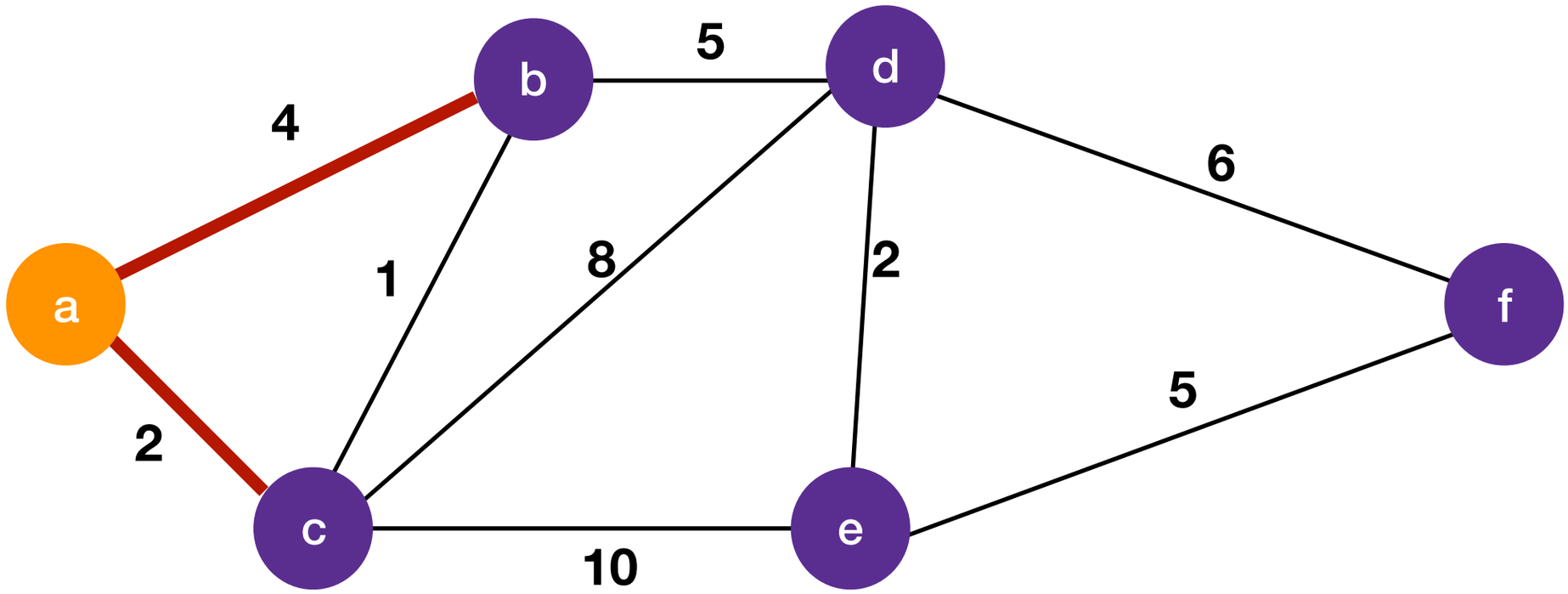


Algorismes sobre grafs



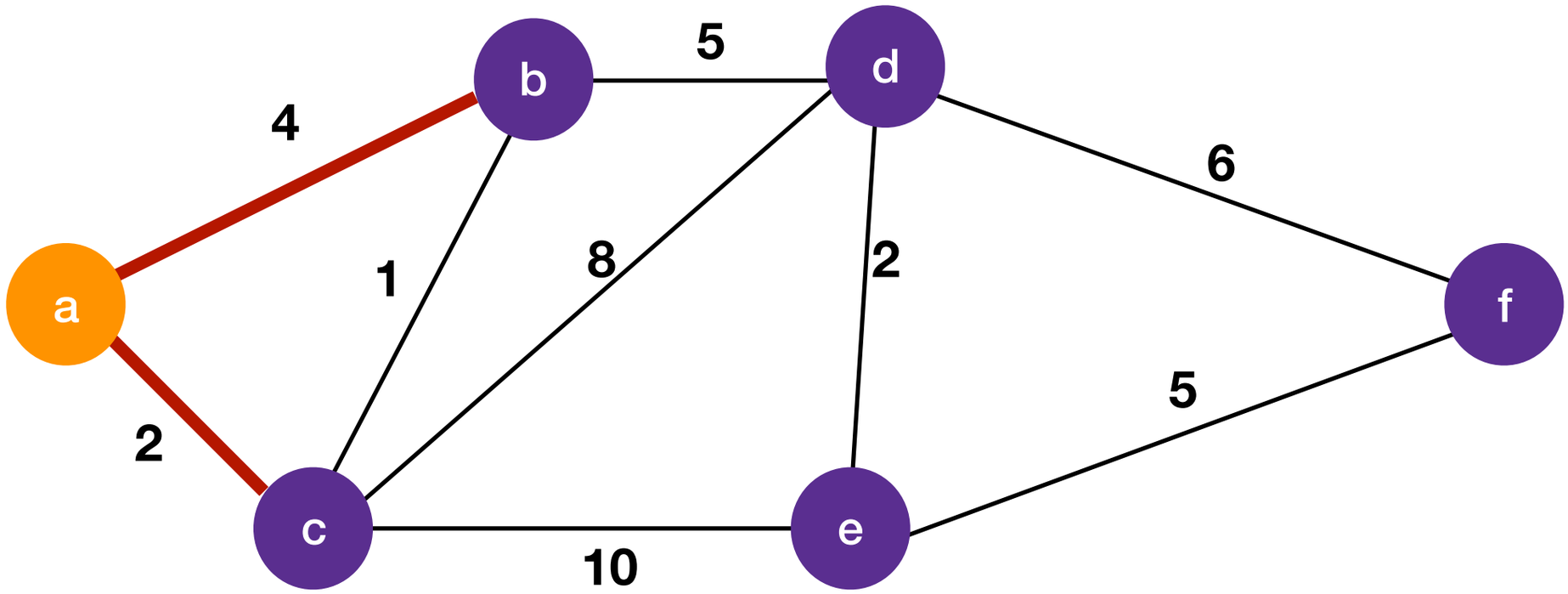
Node	Estat	Cami més curt desde A	Node previ
a	**Node actual**	0	
b		∞	
c		∞	
d		∞	
e		∞	
f		∞	

Algorismes sobre grafs



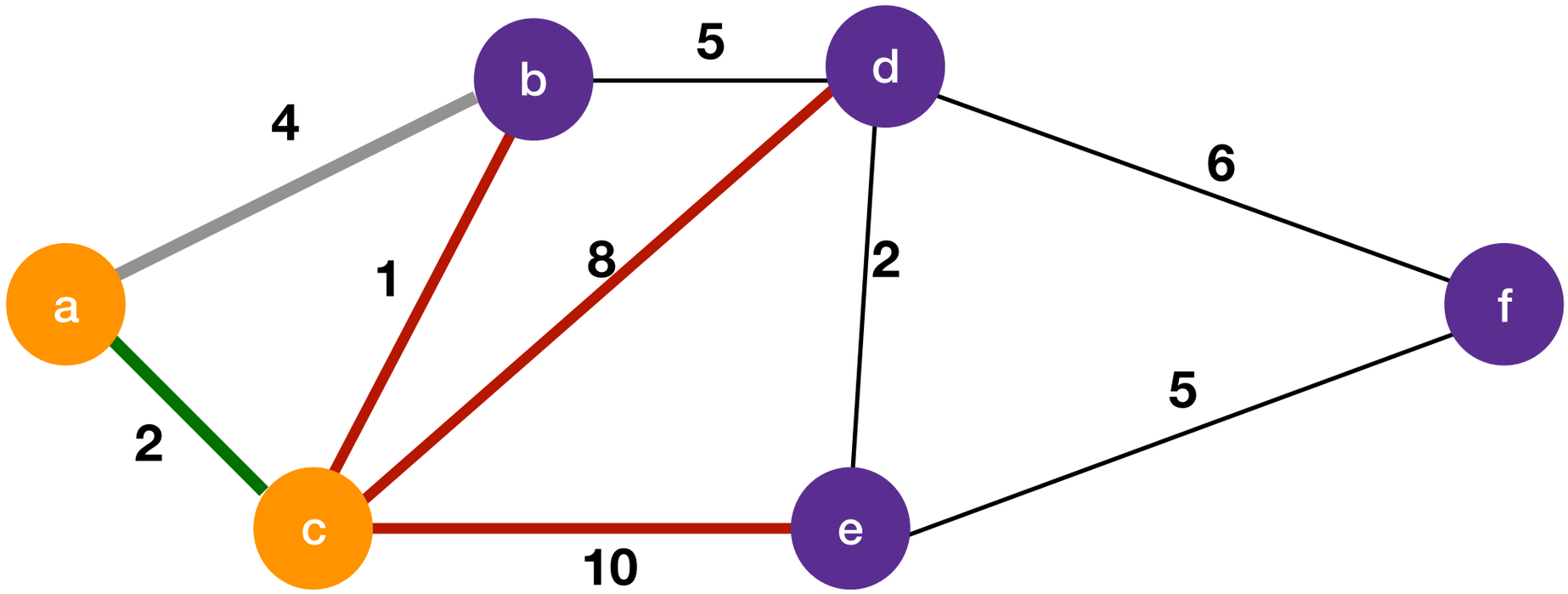
Node	Estat	Cami més curt desde A	Node previ
a	**Node actual**	0	
b		~~∞~~ **4**	**A**
c		~~∞~~ **2**	**A**
d		∞	
e		∞	
f		∞	

Algorismes sobre grafs



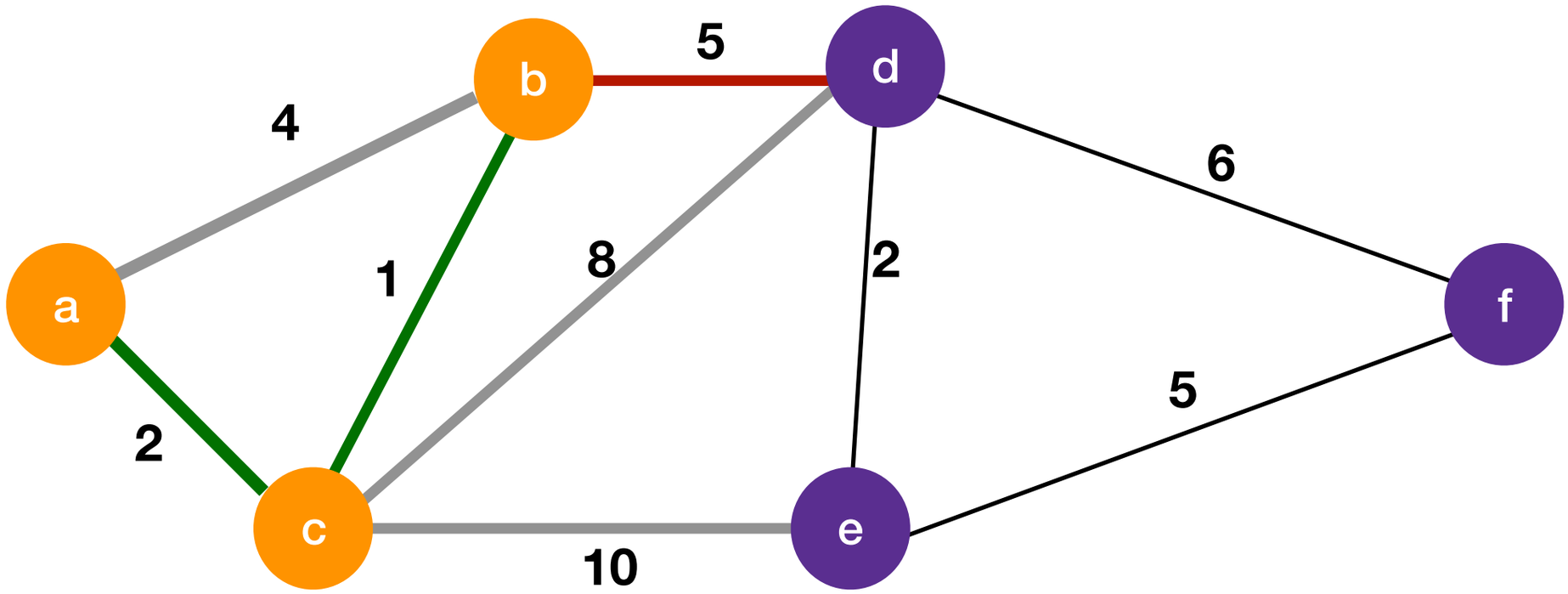
Node	Estat	Cami més curt desde A	Node previ
a	**visitat**	0	
b		**4**	**A**
c	**Node actual**	**2**	**A**
d		∞	
e		∞	
f		∞	

Algorismes sobre grafs



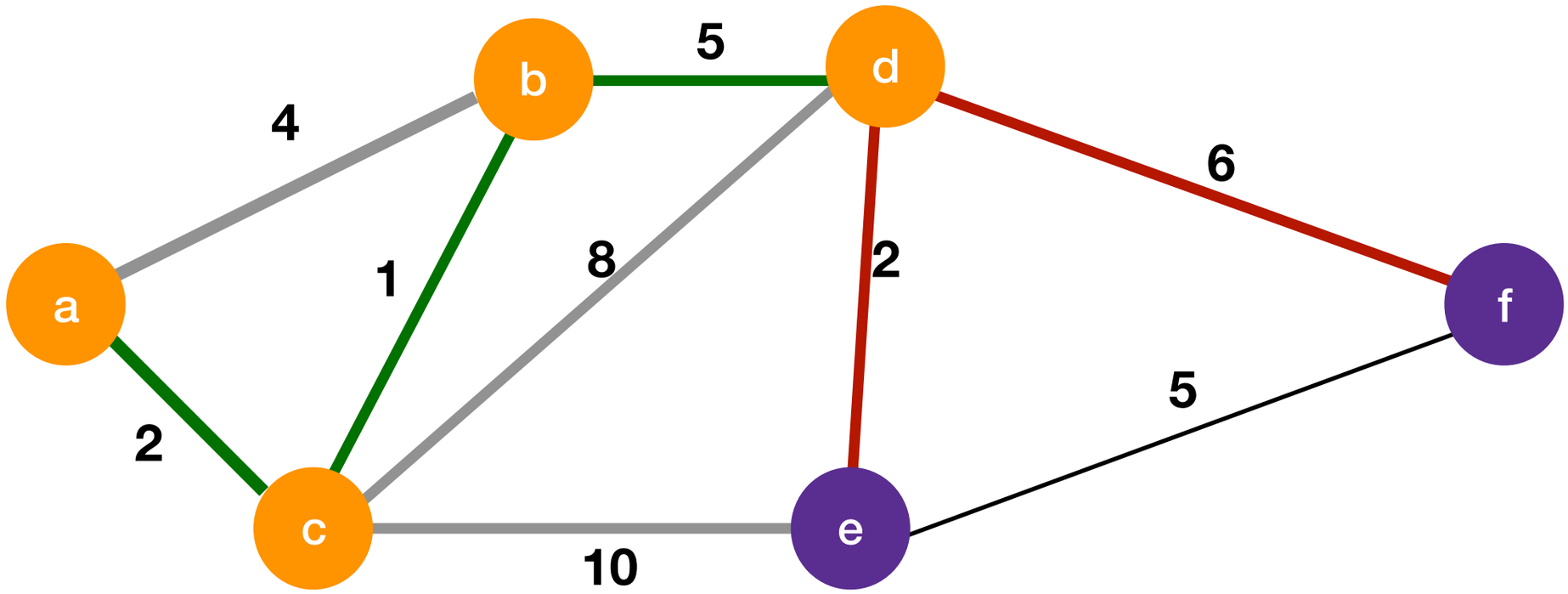
Node	Estat	Cami més curt desde A	Node previ
a	**visitat**	0	
b		~~4~~ 2+1 = 3	~~A~~ C
c	**Node actual**	2	A
d		∞ 2+8 = 10	C
e		∞ 2+10 = 12	C
f		∞	

Algorismes sobre grafs



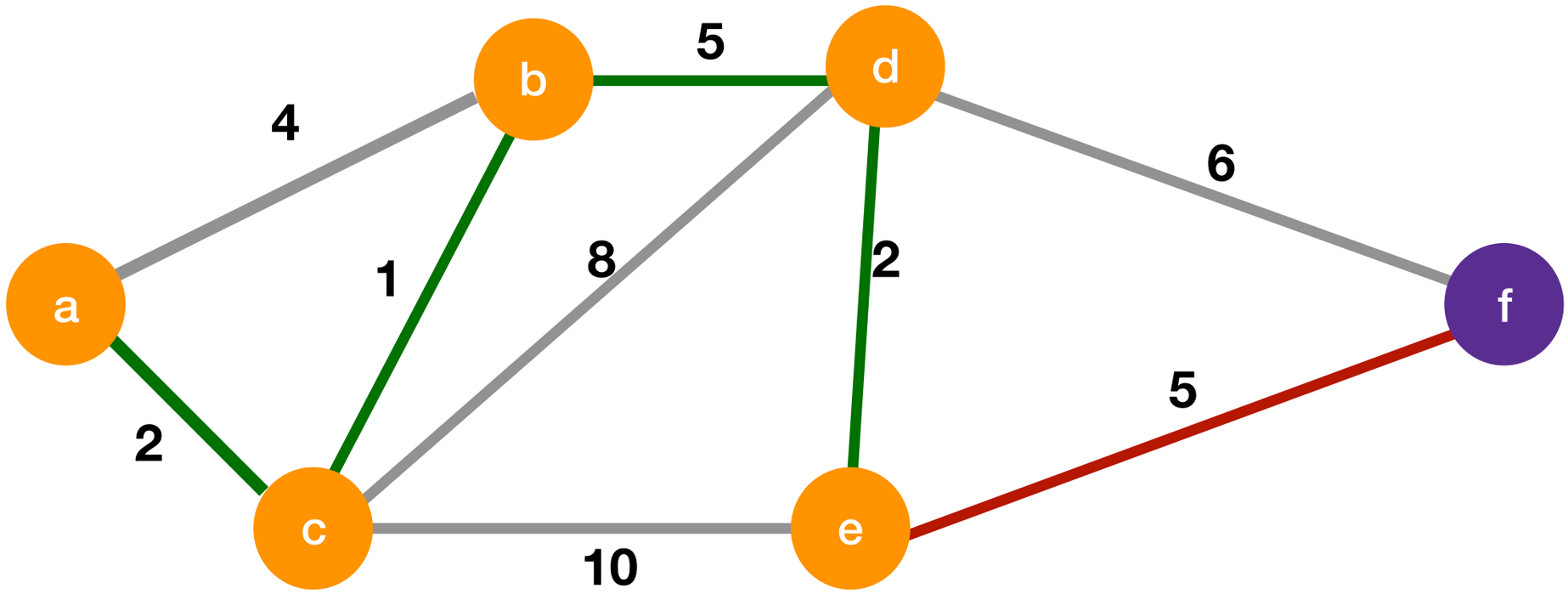
Node	Estat	Cami més curt desde A	Node previ
a	**visitat**	0	
b	**Node actual**	**3**	**C**
c	**visitat**	**2**	**A**
d		~~**10**~~ **3+5 = 8**	~~**C**~~ **B**
e		**12**	**C**
f		∞	

Algorismes sobre grafs



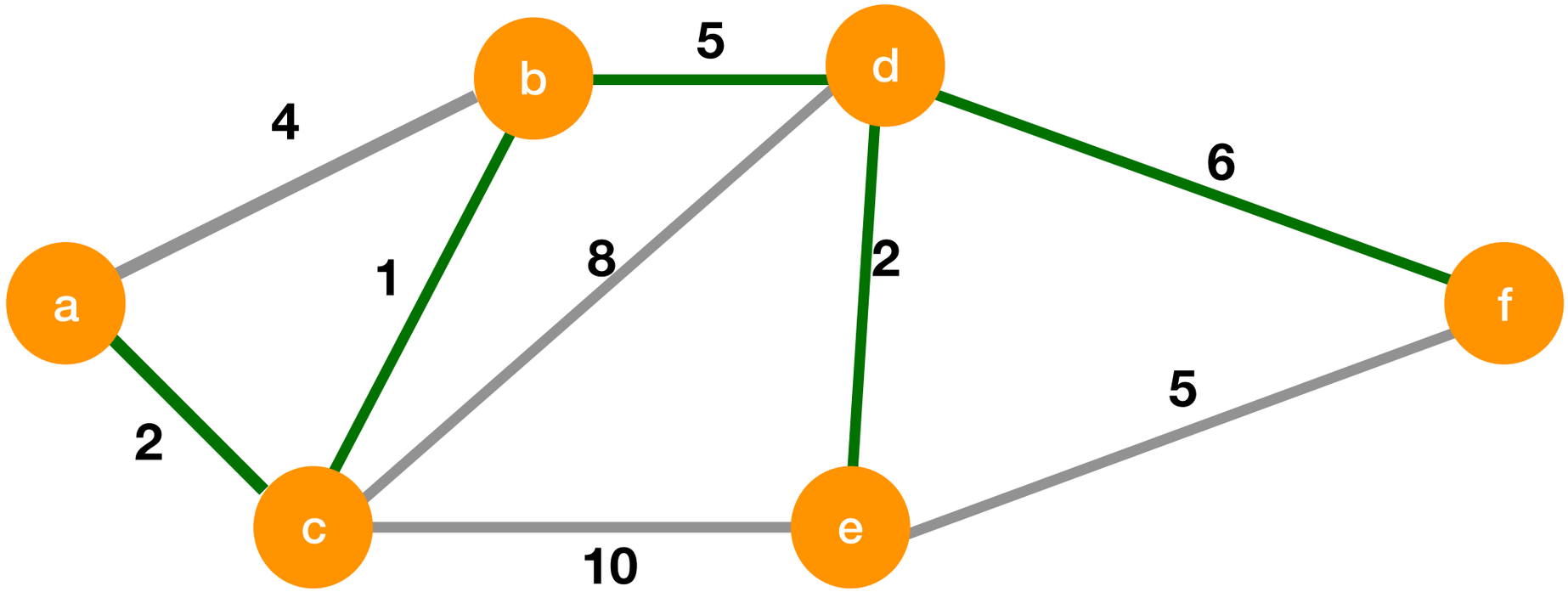
Node	Estat	Cami més curt desde A	Node previ
a	**visitat**	0	
b	**visitat**	**3**	**C**
c	**visitat**	**2**	**A**
d	**Node actual**	**8**	**B**
e		~~12~~ **8+2 = 10**	~~C~~ **D**
f		~~∞~~ **8+6 = 14**	**D**

Algorismes sobre grafs



Node	Estat	Cami més curt desde A	Node previ
a	**visitat**	0	
b	**visitat**	**3**	**C**
c	**visitat**	**2**	**A**
d	**visitat**	**8**	**B**
e	**Node actual**	**10**	**D**
f		**14**	**D**

Algorismes sobre grafs



Node	Estat	Cami més curt desde A	Node previ
a	**visitat**	0	
b	**visitat**	**3**	**C**
c	**visitat**	**2**	**A**
d	**visitat**	**8**	**B**
e	**Node actual**	**10**	**D**
f		**14**	**D**

Algorismes sobre grafs

- Set an alarm clock for node s at time 0.
- Repeat until there are no more alarms:
  Say the next alarm goes off at time T, for node u. Then:
  - The distance from s to u is T.
  - For each neighbor v of u in G:
    * If there is no alarm yet for v, set one for time $T + l(u, v)$.
    * If v's alarm is set for later than $T + l(u, v)$, then reset it to this earlier time.

Ara ens queda implementar el sistema d'alarmes

Dijkstra

- Algorisme de **Dijkstra**
  - Cua amb prioritats → generalment **Heap**
  - Manté un conjunt d'elements (nodes) amb les valors numèrics associats com a claus (temps de l'alarma) i suporta les següents operacions:

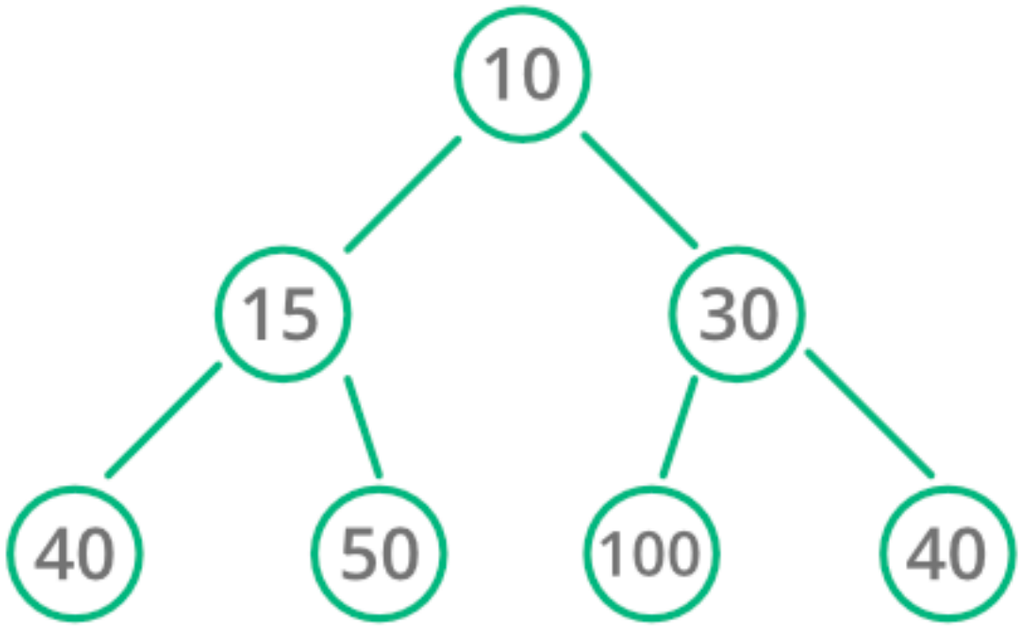
Inserció: *inclou un nou element al conjunt.*

Decrementar-clau: *Decrementa el valor de la clau d'un element particular. La cua amb prioritats normalment no canvia el valor de les claus, el que fa és notificar a la cua que el valor d'una certa clau ha estat decrementat.*

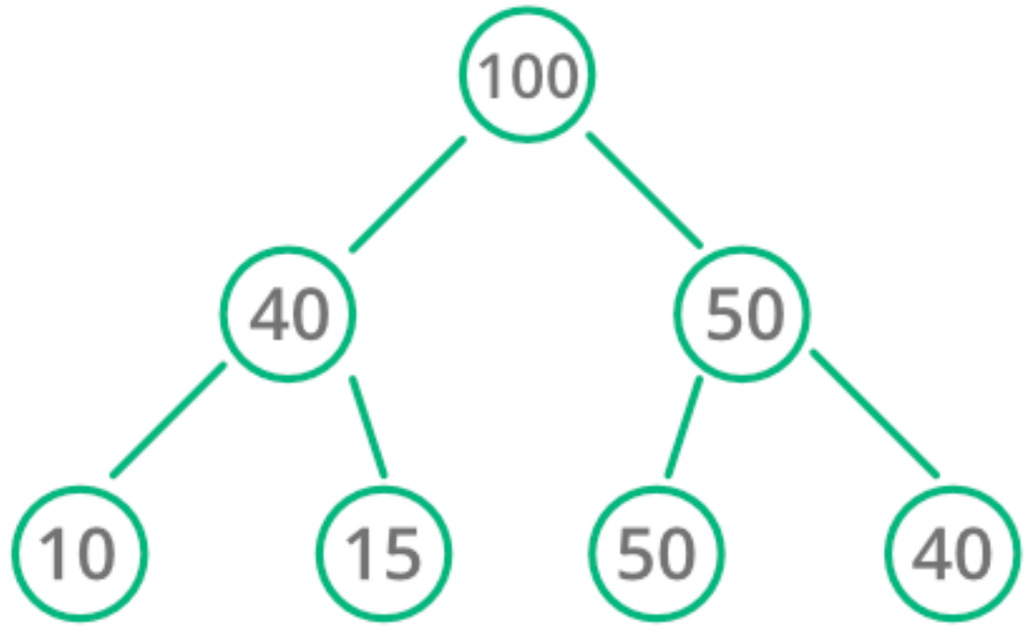
Eliminar-min: *Retorna l'element amb la menor clau i l'elimina del conjunt.*

Fer-cua: *Construeix una cua amb prioritats amb els elements donats i els seus valors de clau associats.*

Heap Data Structure



Min Heap



Max Heap

Dijkstra

Inserció: *inclou un nou element al conjunt.*

Decrementar-clau: *Decrementa el valor de la clau d'un element particular. La cua amb prioritats normalment no canvia el valor de les claus, el que fa és notificar a la cua que el valor d'una certa clau ha estat decrementat.*

Eliminar-min: *Retorna l'element amb la menor clau i l'elimina del conjunt.*

Fer-cua: *Construeix una cua amb prioritats amb els elements donats i els seus valors de clau associats.*

- **Inserir** i **decrementar** clau ens permet fixar les alames, mentres que **eliminar-min** ens diu quina és la pròxima alarma a tenir en compte.

Dijkstra

```
procedure dijkstra(G, l, s)
Input:    Graph G = (V, E), directed or undirected;
          positive edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil
dist(s) = 0

H = makequeue(V)   (using dist-values as keys)
while H is not empty:
    u = deletemin(H)
    for all edges (u, v) ∈ E:
        if dist(v) > dist(u) + l(u, v):
            dist(v) = dist(u) + l(u, v)
            prev(v) = u
            decreasekey(H, v)
```

Dijkstra

- **dist(u)** es refereix als valors actuals d'alarma del node u. Un valor infinit significa que encara no hem posat valor a l'alarma.

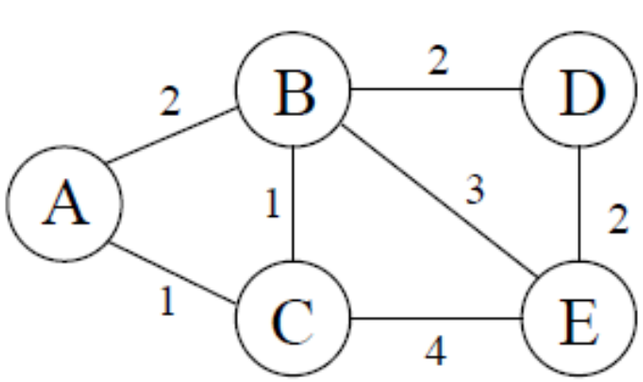
- L'array **prev**: guarda informació del node immediat abans del node actual u dins la ruta més curta entre s i u.

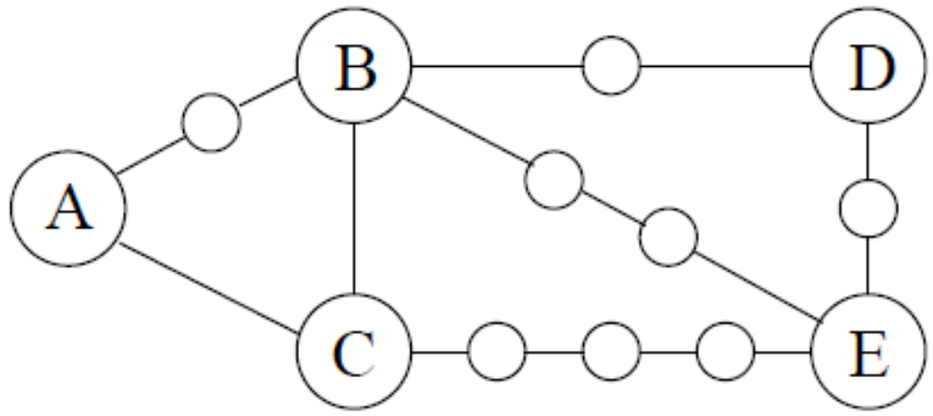
- Si retornem fent ús dels valor d'aquests punters podem reconstruir els camins més curts de forma senzilla.

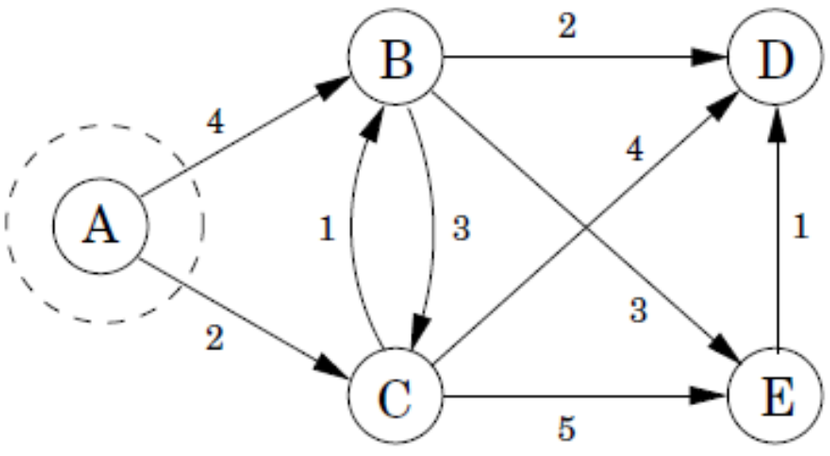
- Podem veure que la diferència principal entre l'algorisme Dijkstra I BFS és que el primer usa una cua amb prioritats en lloc d'una cua regular, de forma que prioritza nodes en funció dels costos de les arestes.

Dijkstra

- Algorisme de **Dijkstra:** exemple graf dirigit



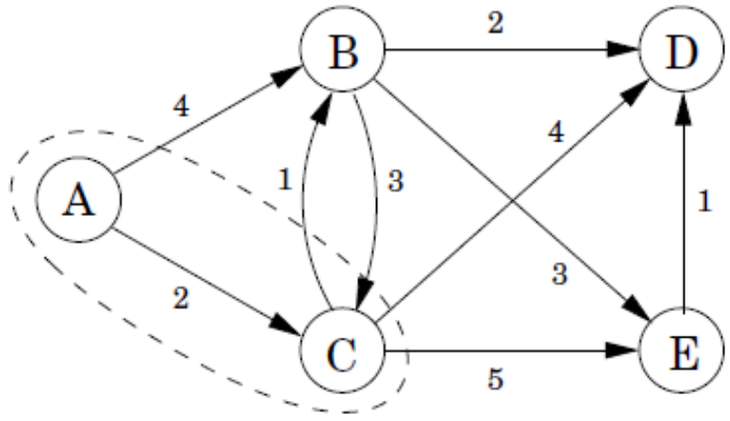




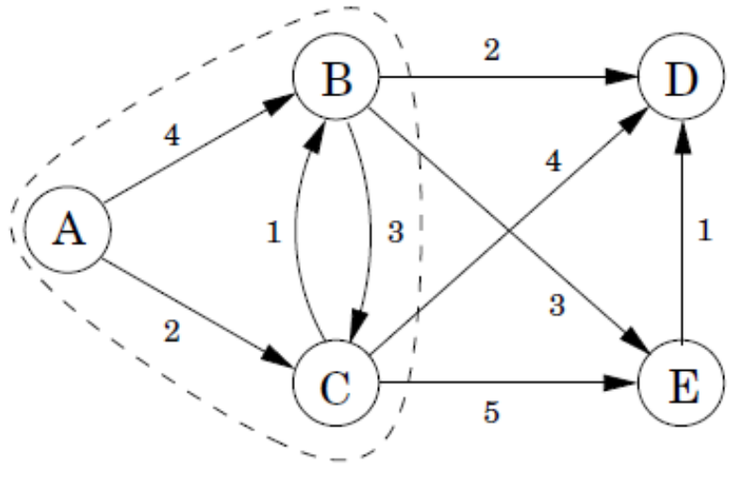
A: 0	D: ∞
B: 4	E: ∞
C: 2	

Dijkstra

- Algorisme de **Dijkstra:** exemple graf dirigit



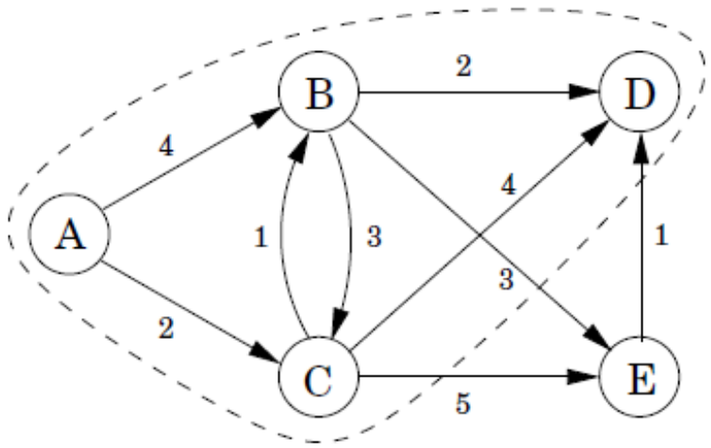
A: 0	D: 6
B: 3	E: 7
C: 2	



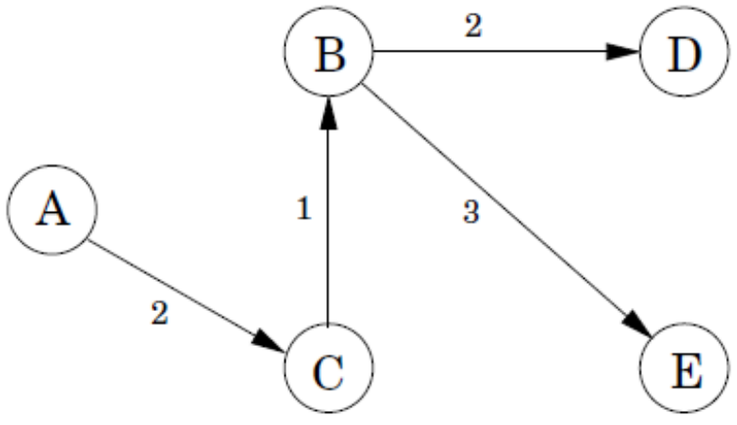
A: 0	D: 5
B: 3	E: 6
C: 2	

Dijkstra

- Algorisme de **Dijkstra:** exemple graf dirigit



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



Dijkstra

- Algorisme de **Dijkstra:** exemple graf **NO** dirigit