



Grafs II

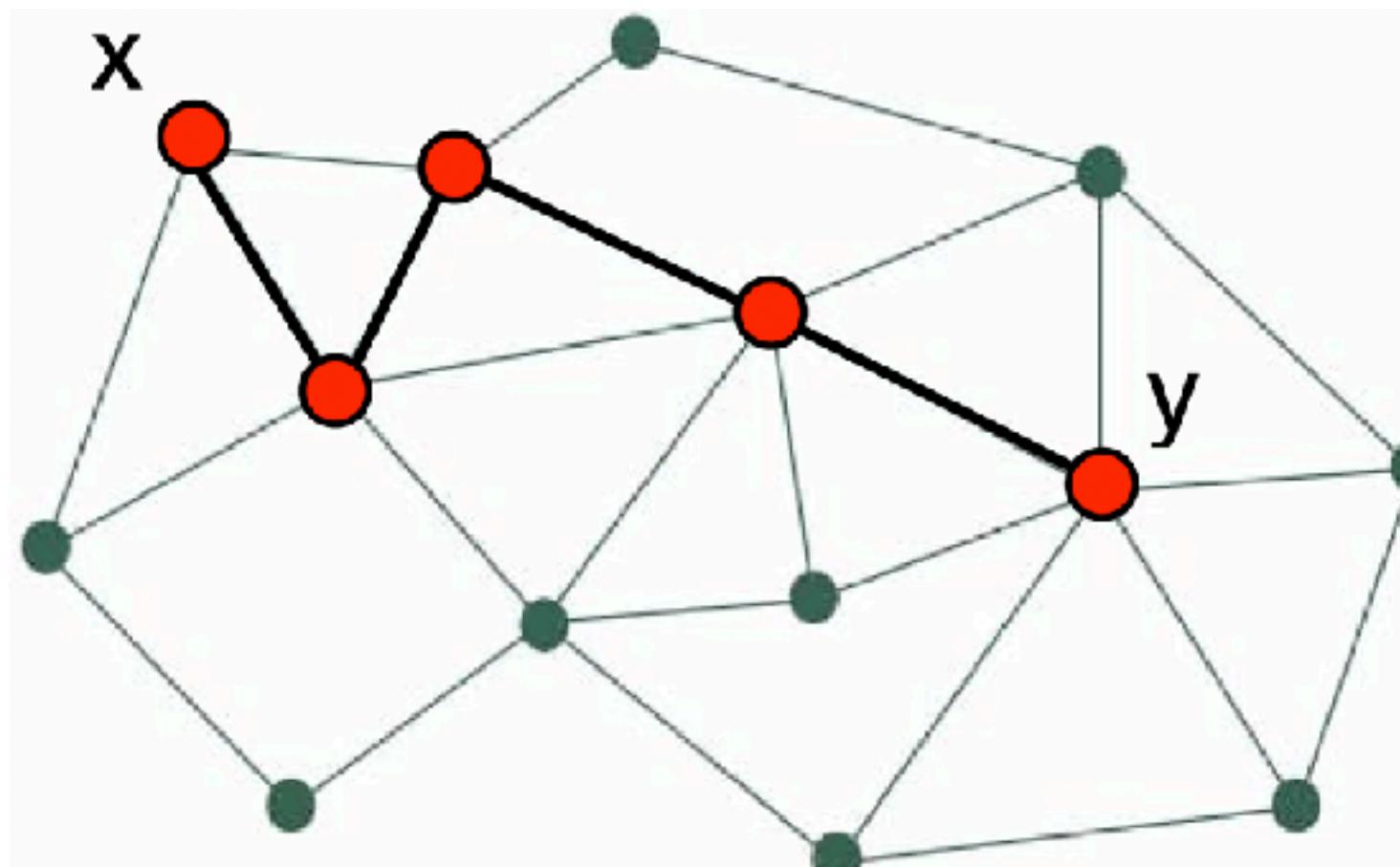
Algorísmica Avançada | Enginyeria Informàtica

Santi Seguí | 2021-2022

Recorreguts

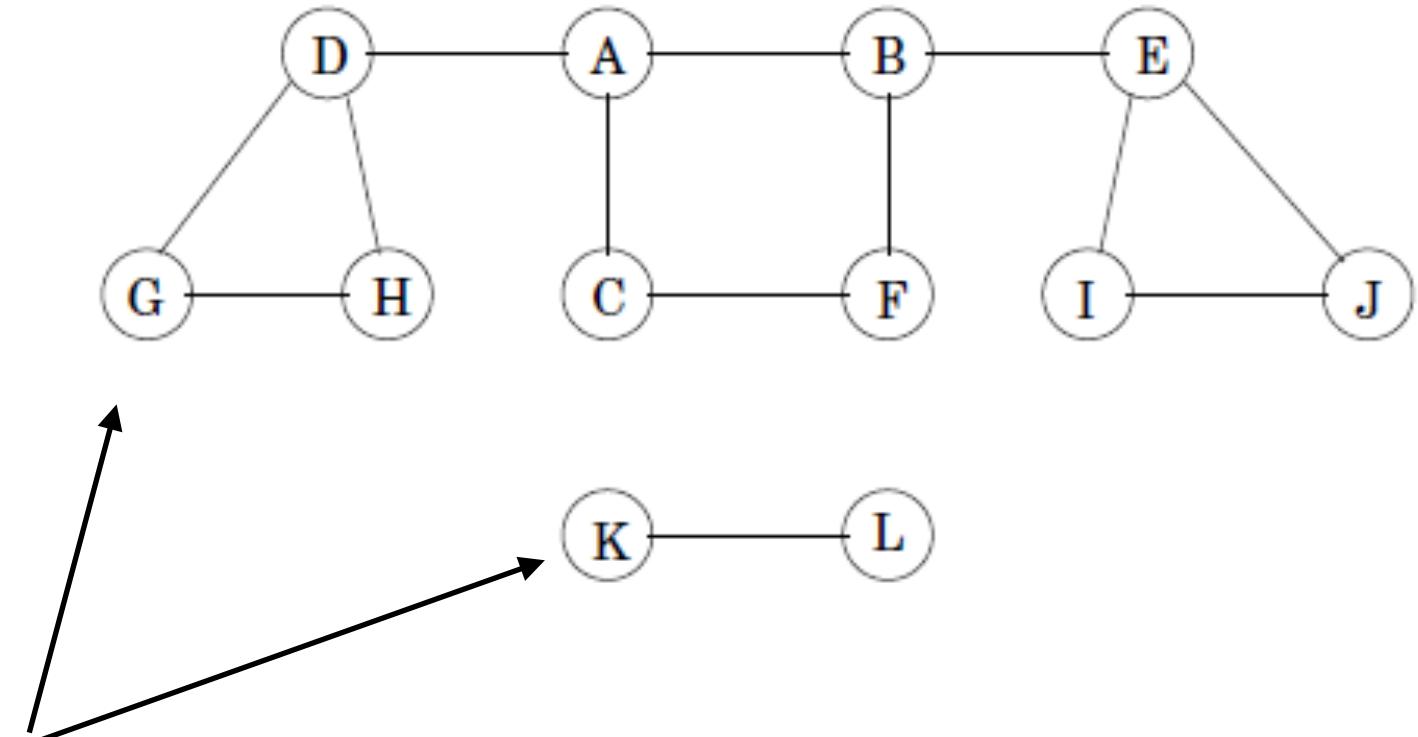
Recorreguts

- Un recorregut entre dos vèrtex x i y és una successió de vèrtexs v_o, v_1, \dots, v_k que compleix:
 - $v_o = x; v_k = y$
 - $\{v_{i-1}, v_i\}$ és una aresta del graf G per a tot i entre 1 i k.
 - La longitud del recorregut és k
 - Un **camí** és un recorregut sense vèrtex repetits. Un cicle és un camí tancat, és a dir, el primer vèrtex i l'últim coincideixen.

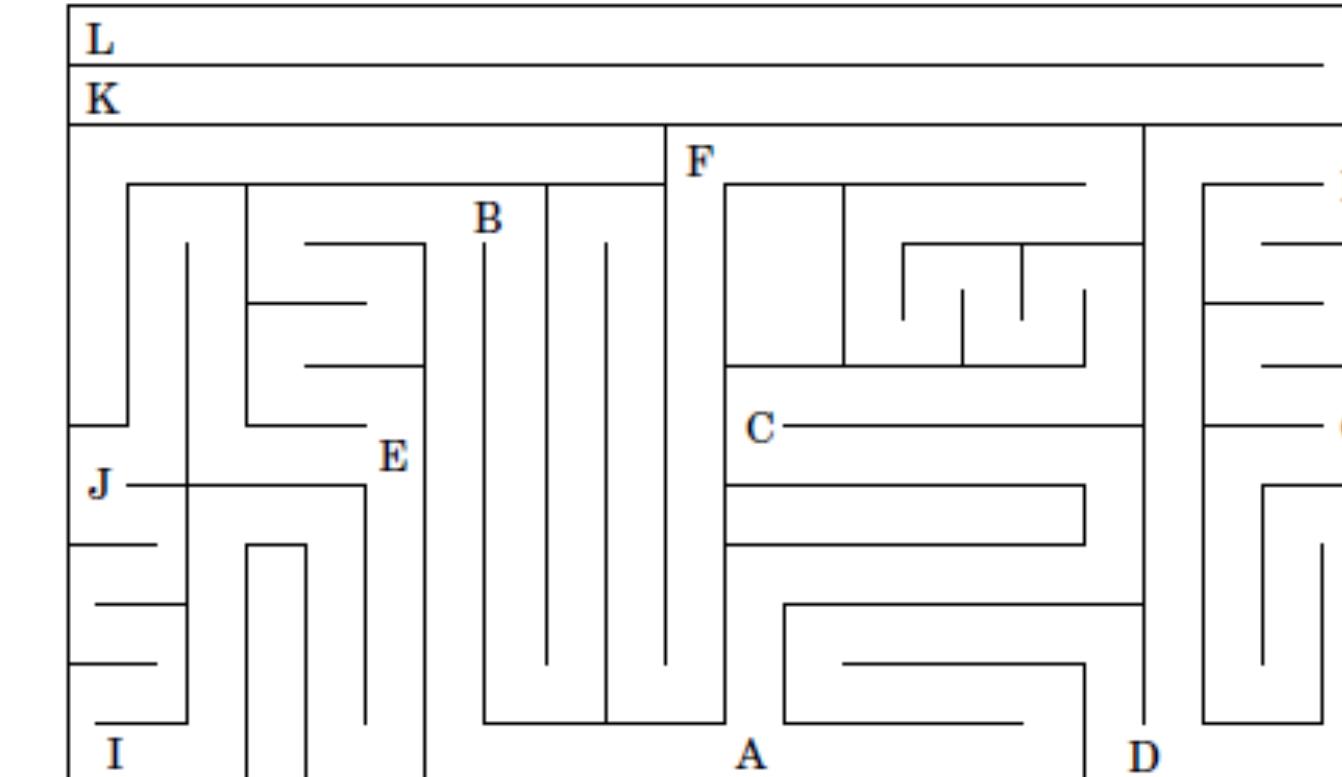


Recorreguts

- Algunes preguntes:
 - Quins vèrtexs són accessibles des de quins?
 - Existeix un camí des de el node A al node J?
 - Quin és el camí més curt des de el node A al node J?



Components connexes



Ho podem veure com un laberint

Recorreguts amb grafs

- Recorregut per profunditat - **Depth Frist Search (DFS)**
- Recorregut per amplada - **Breath Frist Search (BFS)**

Algorismes sobre grafs

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

```
procedure explore( $G, v$ )
```

Input: $G = (V, E)$ is a graph; $v \in V$

Output: $\text{visited}(u)$ is set to true for all nodes u reachable from v

```
visited( $v$ ) = true
```

```
previsit( $v$ )
```

```
for each edge  $(v, u) \in E$ :
```

```
    if not visited( $u$ ): explore( $u$ )
```

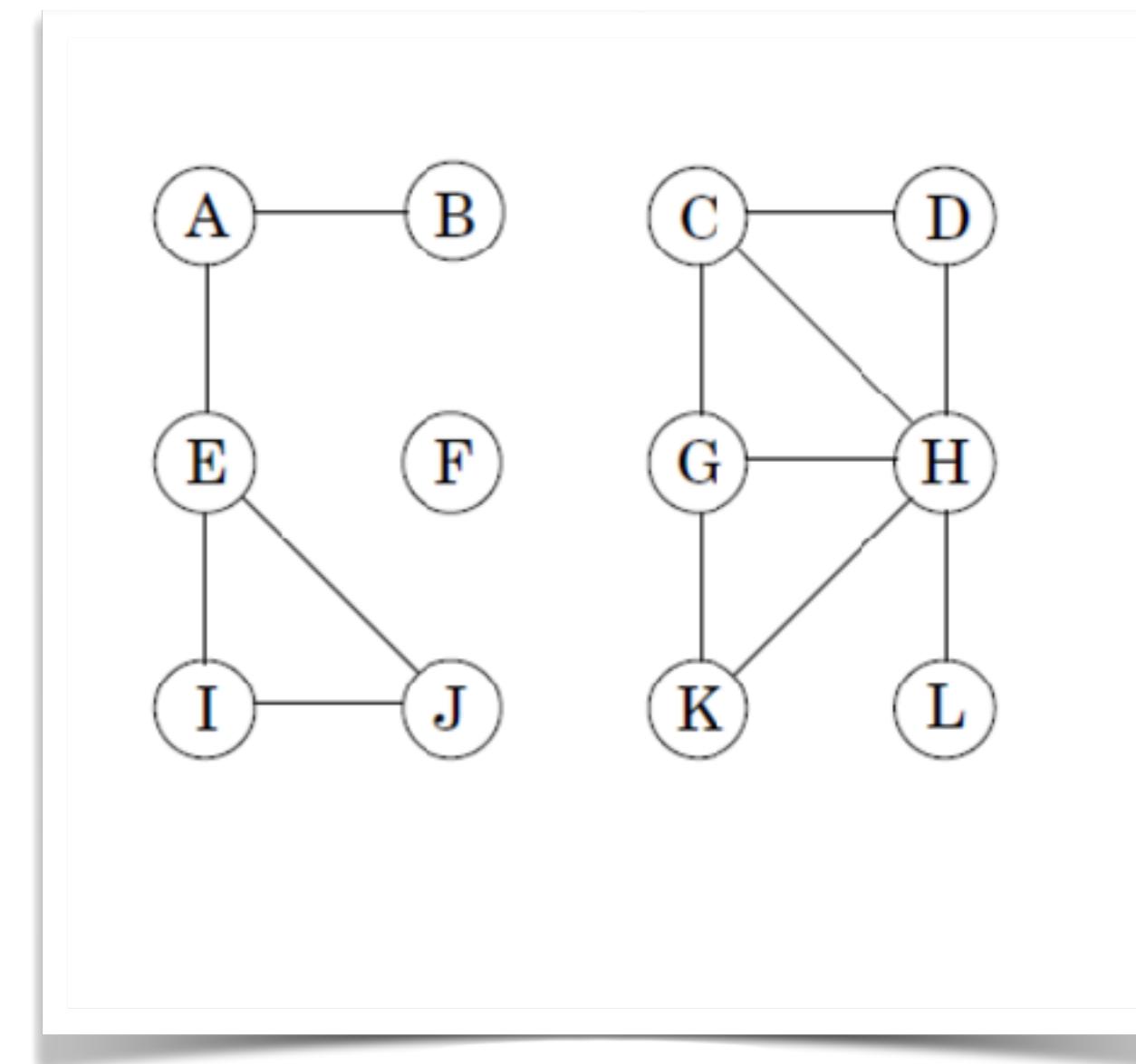
```
postvisit( $v$ )
```

-Recorregut amb profunditat-
-depth-first search-

DFS

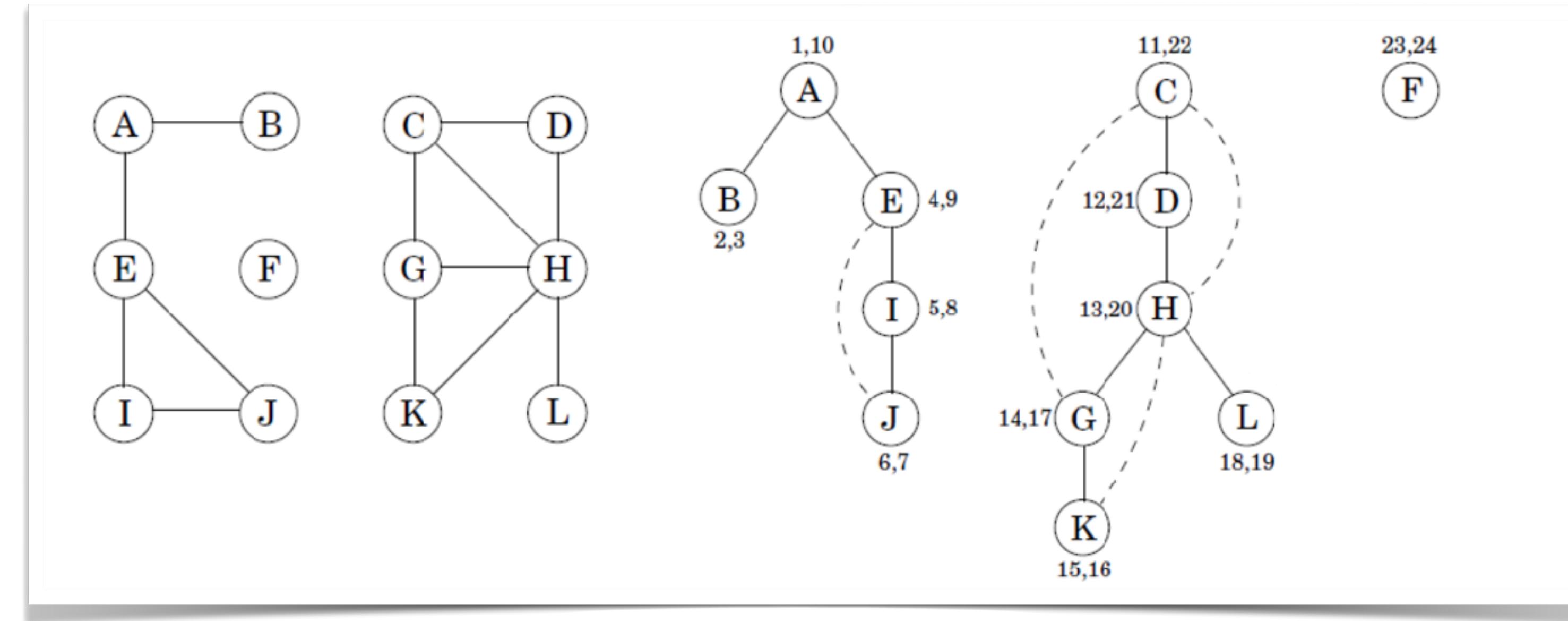
Recorregut amb profunditat (DFS)

- El recorregut per profunditat en grafs és similar al recorregut amb profunditat amb arbres. L'única diferència aquí, es que els **grafs poden tindre cicles**.



Exploració dels grafs (DFS)

- **Exemple**



DFS - Implementació Recursiva

```
# Using a Python dictionary to act as an adjacency list
graph = {
    'A' : ['B','C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

visited = set() # Set to keep track of visited nodes.

def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
dfs(visited, graph, 'A')
```

DFS - Implementació Iterativa

```
def dfs(graph, node):
    visited = [node]
    stack = [node]
    while stack:
        node = stack[-1]
        if node not in visited:
            visited.append(node)
            remove_from_stack = True
        for next in graph[node]:
            if next not in visited:
                stack.append(next)
                remove_from_stack = False
                break
        if remove_from_stack:
            stack.pop()
    return visited

print(dfs(graph1, 'A'))
```

Complexitat DFS?

```
# Using a Python dictionary to act as an adjacency list
graph = {
    'A' : ['B','C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

visited = set() # Set to keep track of visited nodes.

def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
dfs(visited, graph, 'A')
```

DFS complexitat

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

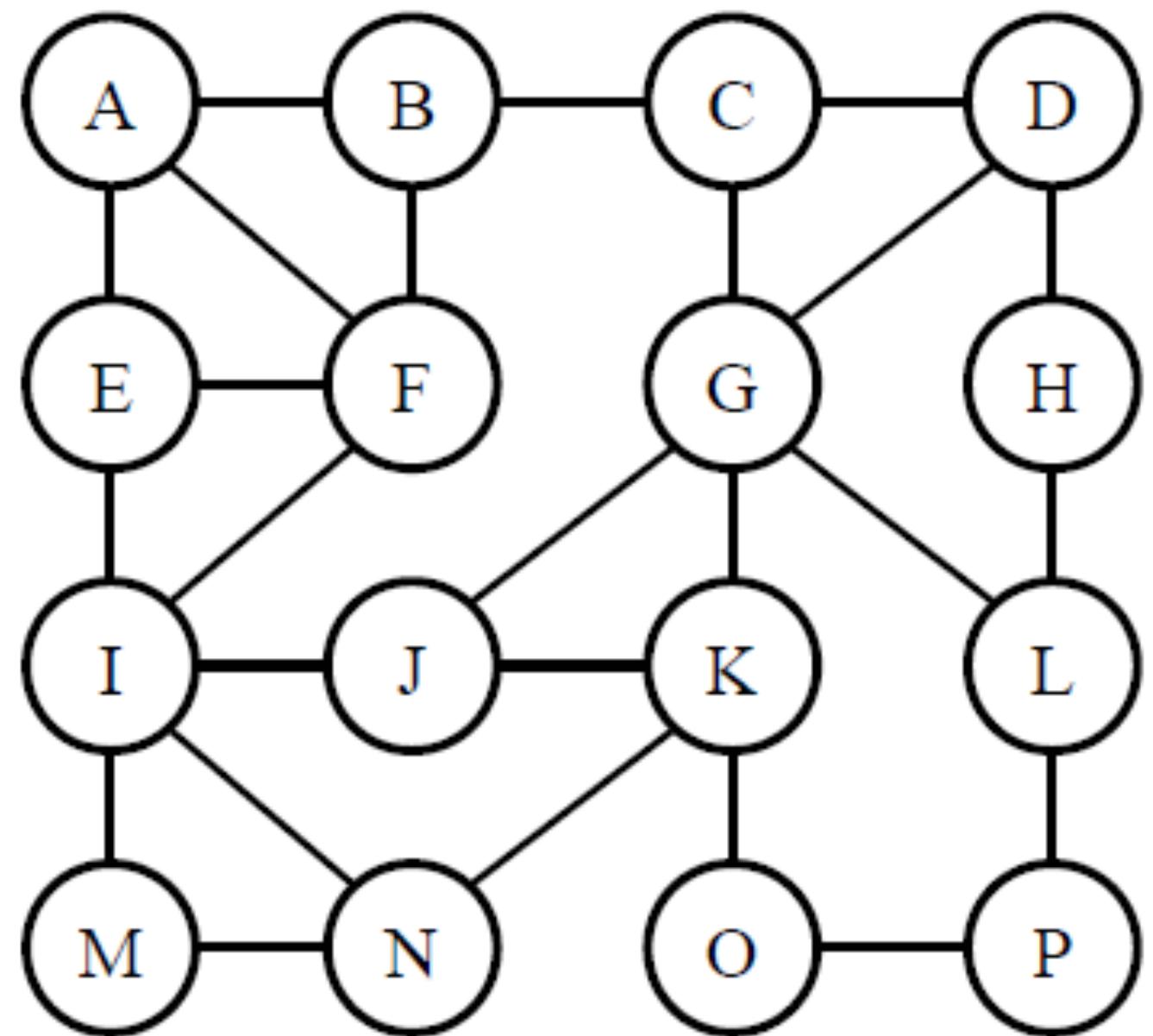
Exploració dels grafs (DFS)

- 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.

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

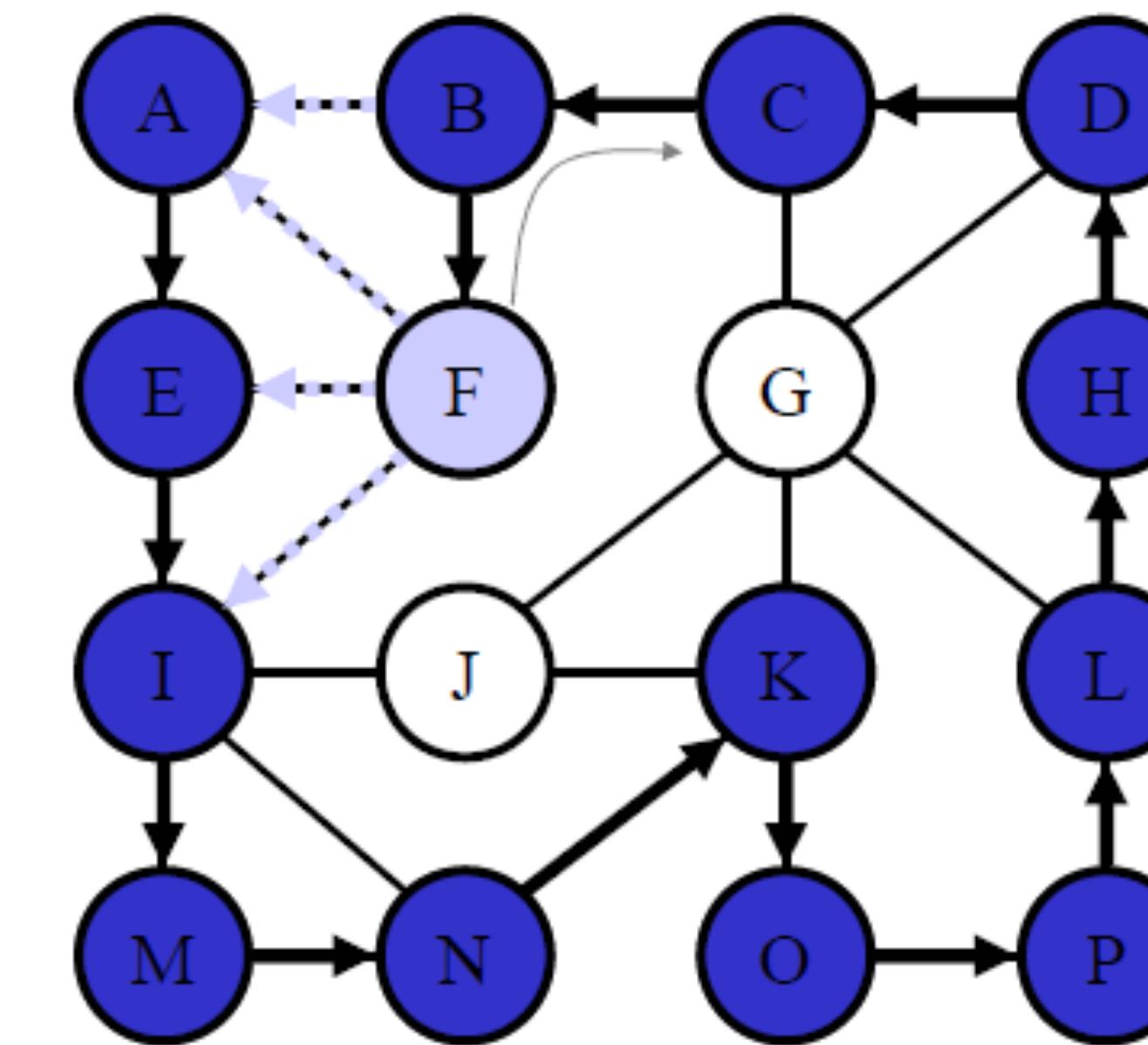
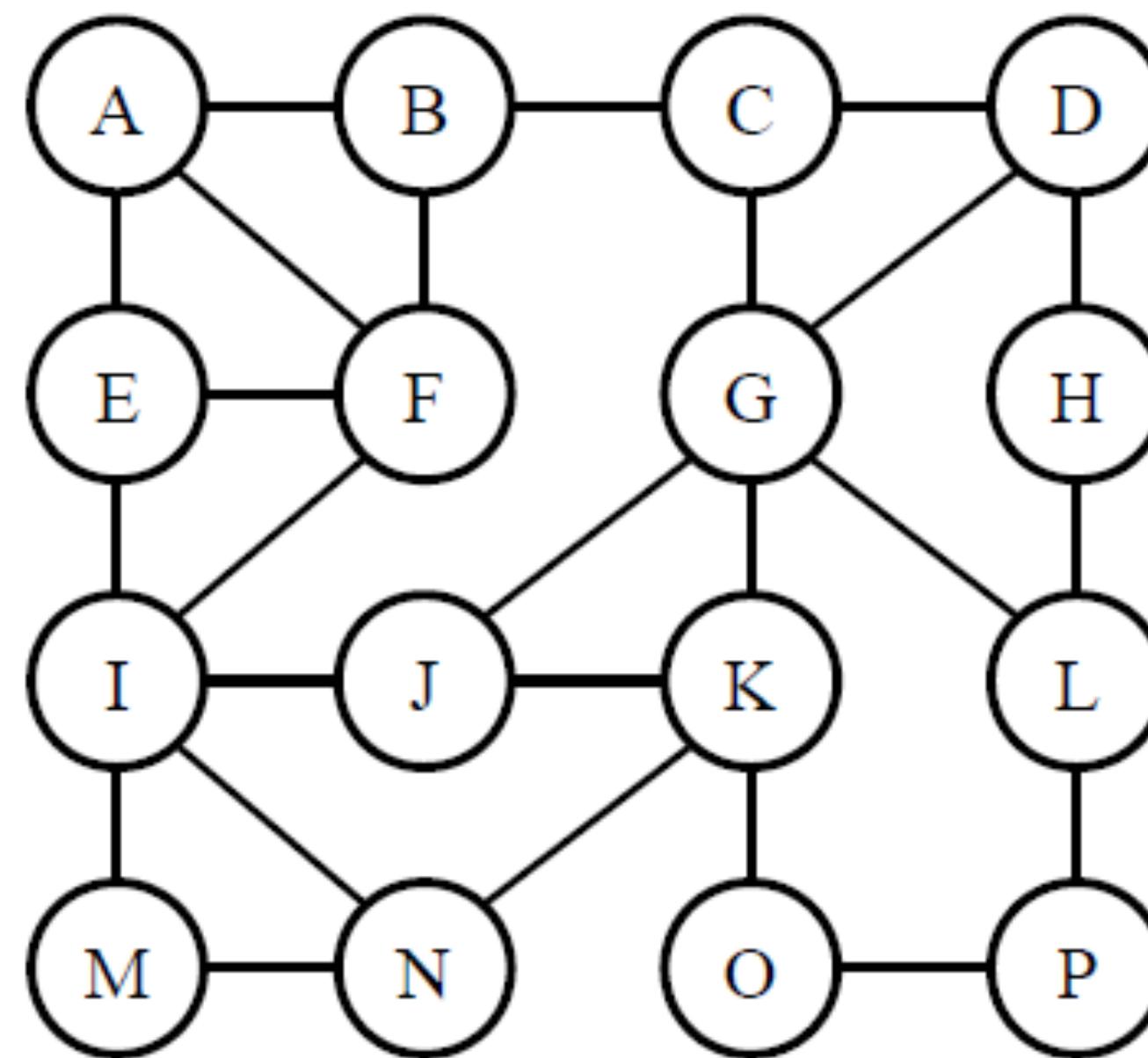
Algorismes sobre grafs

Exemple:



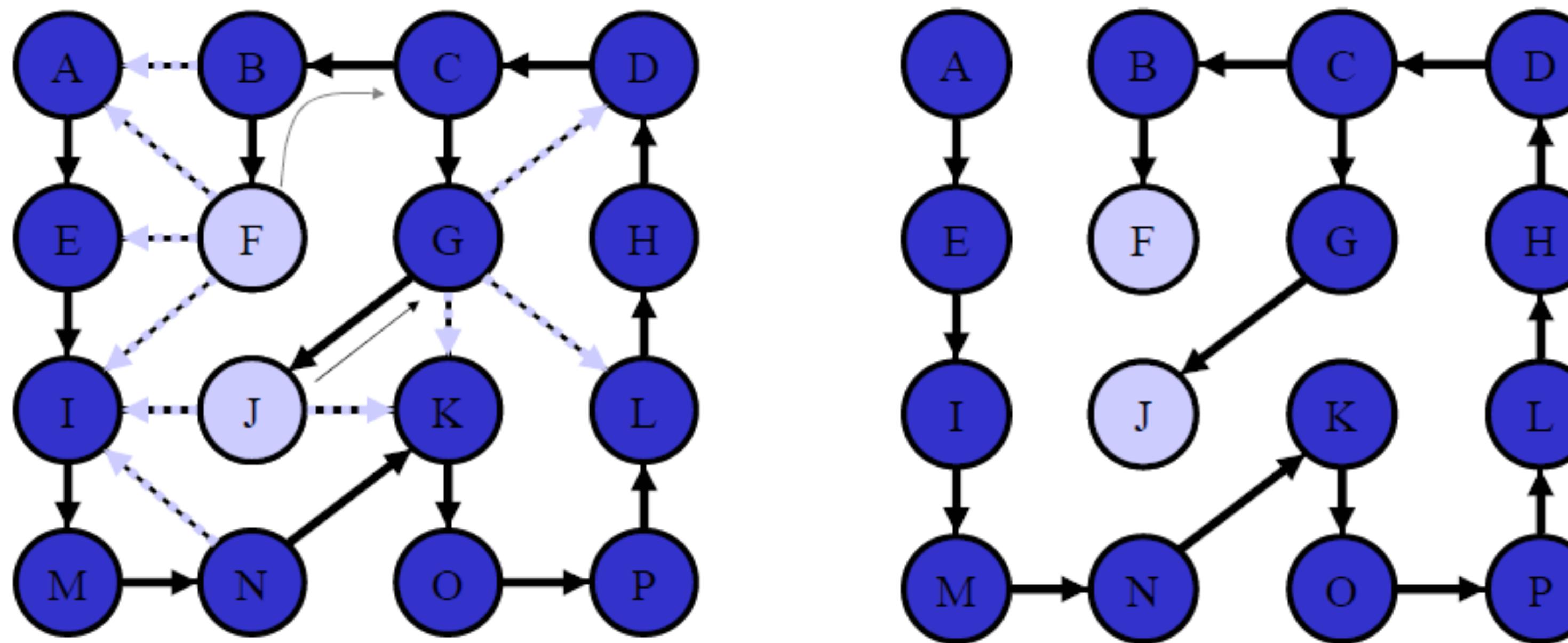
Algorismes sobre grafs

Exemple:



Algorismes sobre grafs

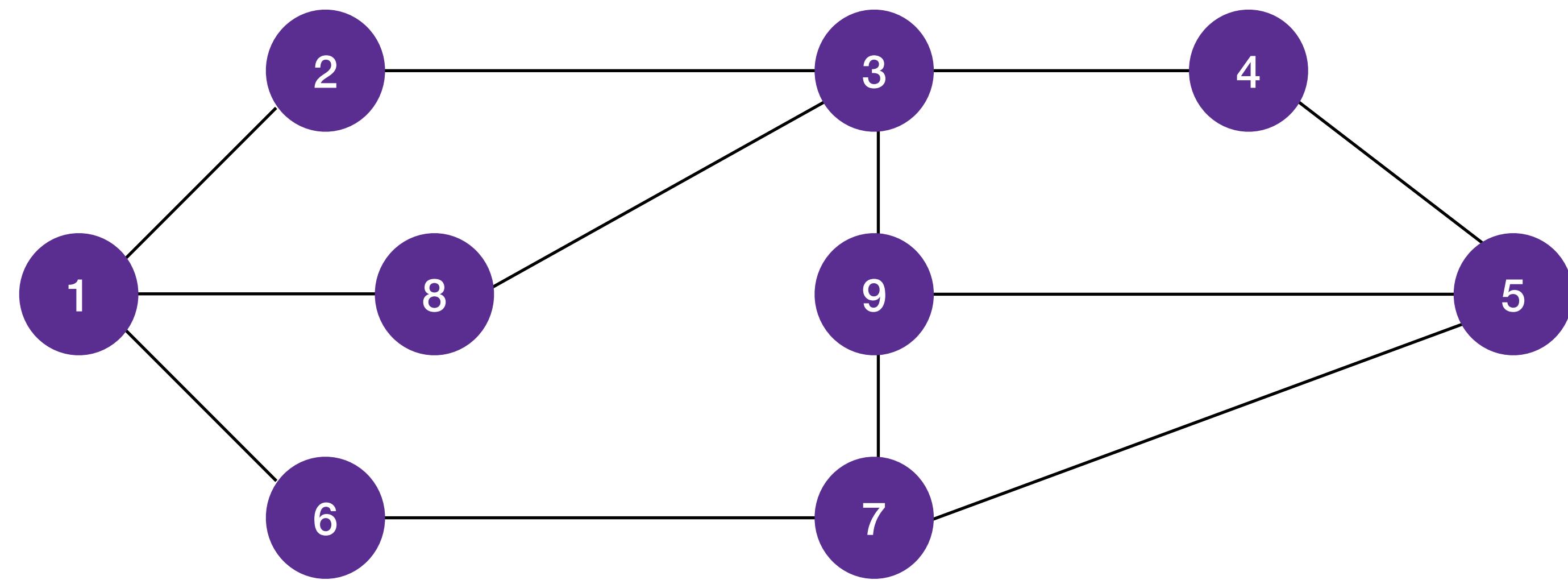
Exemple:



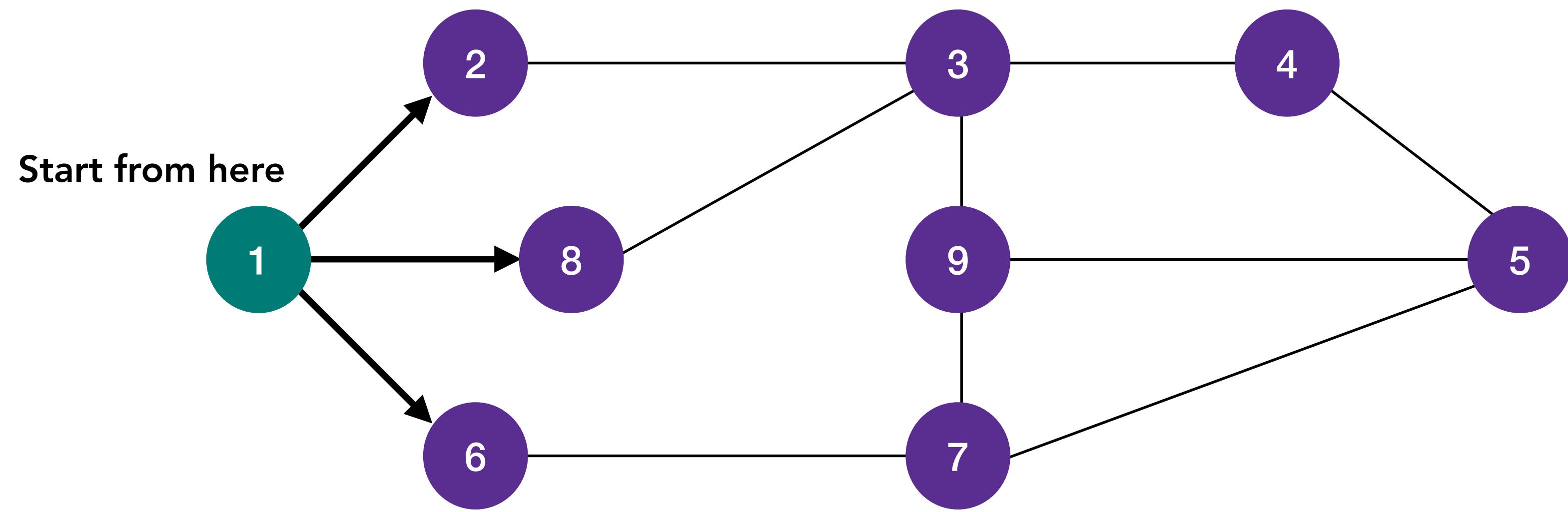
breath-first search

-Recorregut amb amplada-

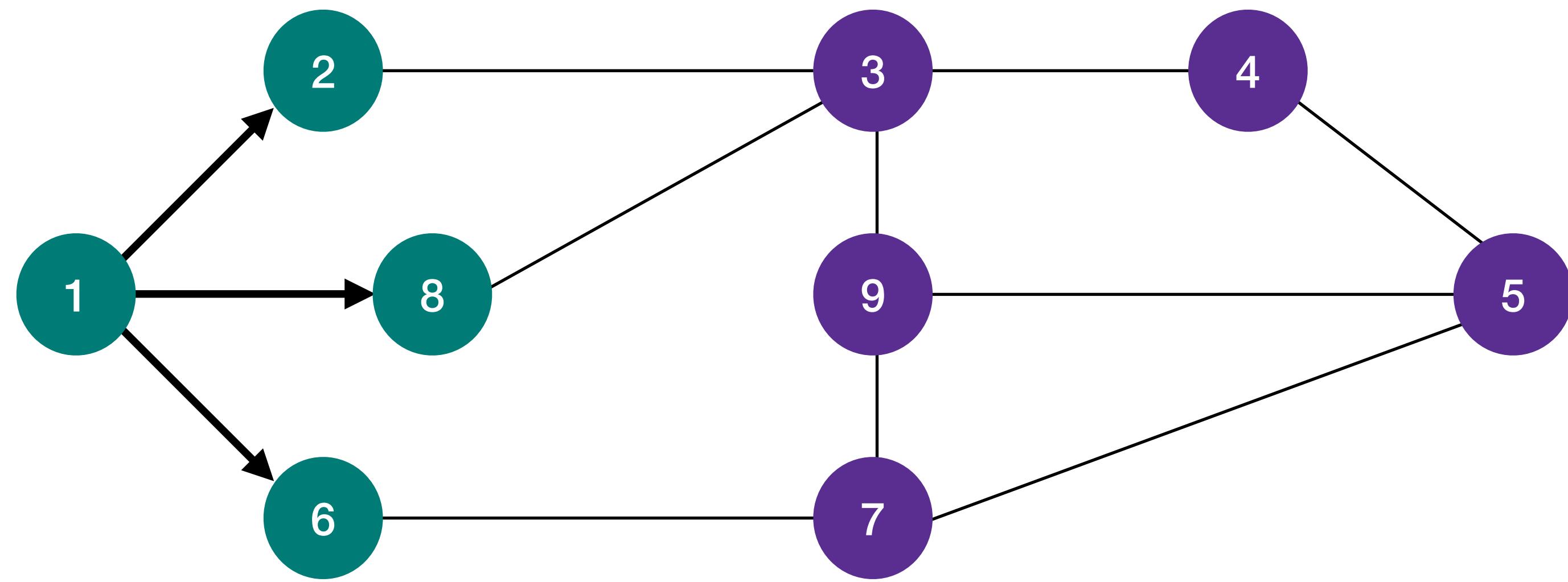
BFS



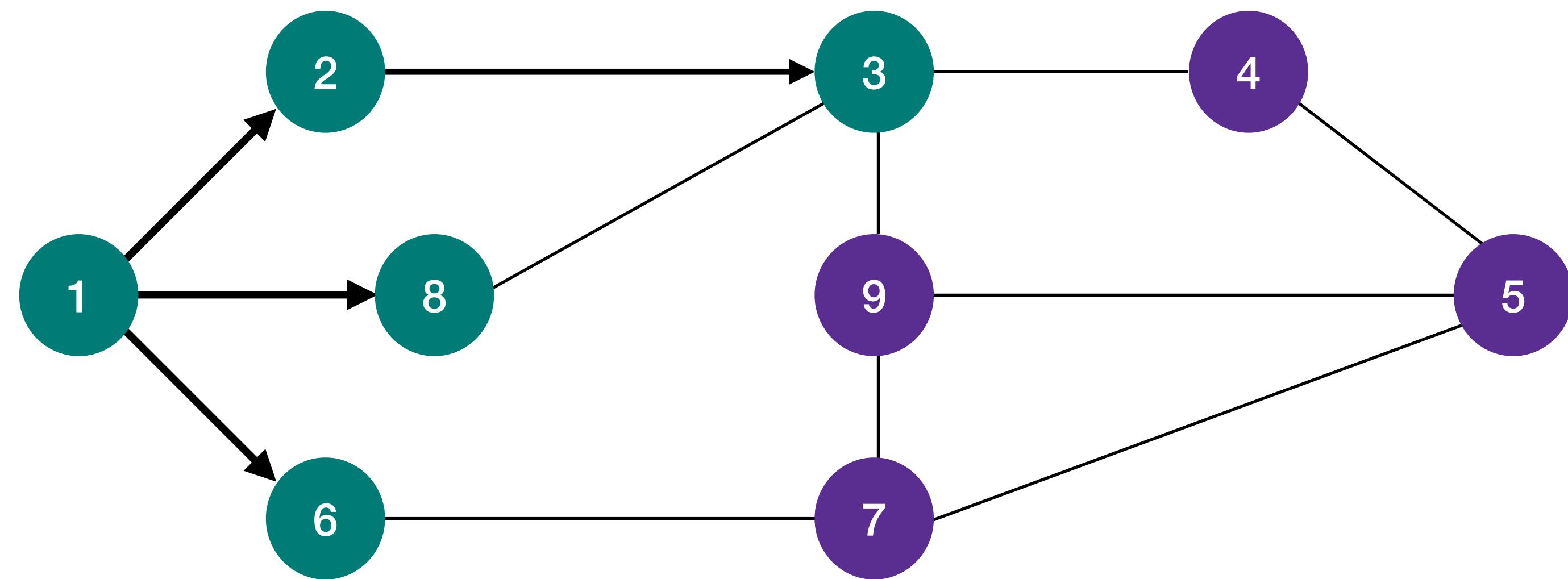
Output:



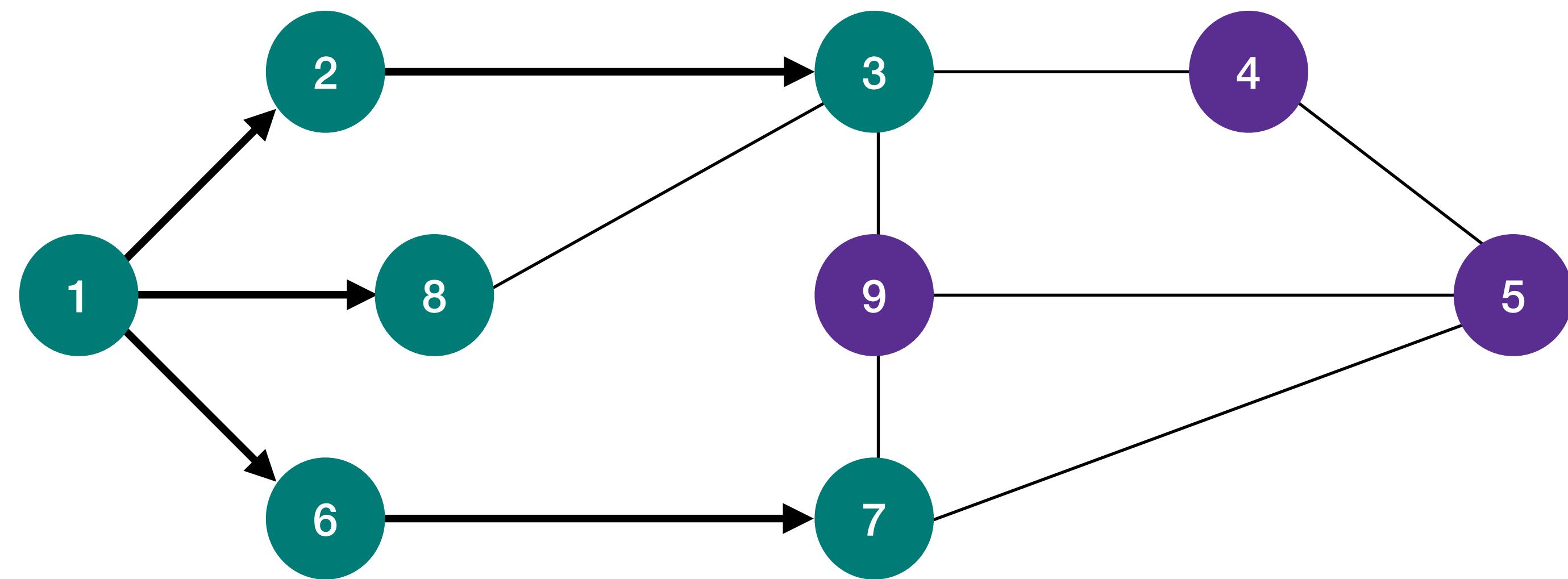
Output: 1



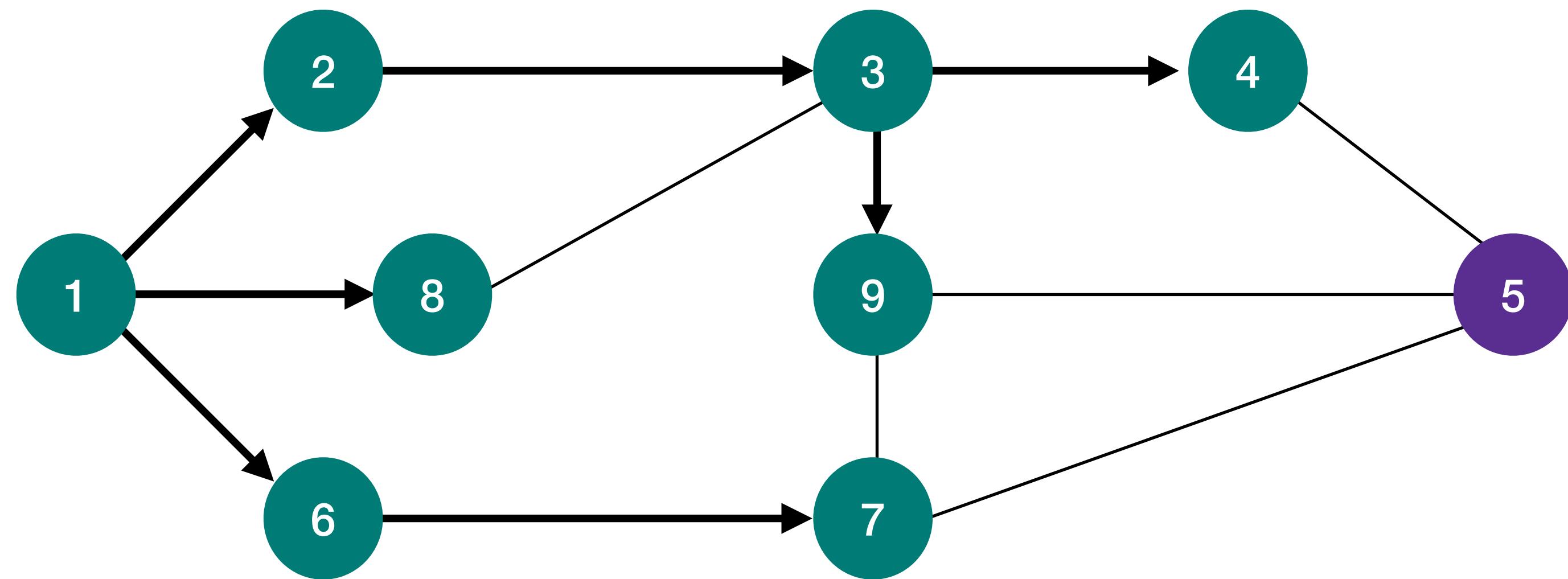
Output: 1 - 2 - 6 - 8



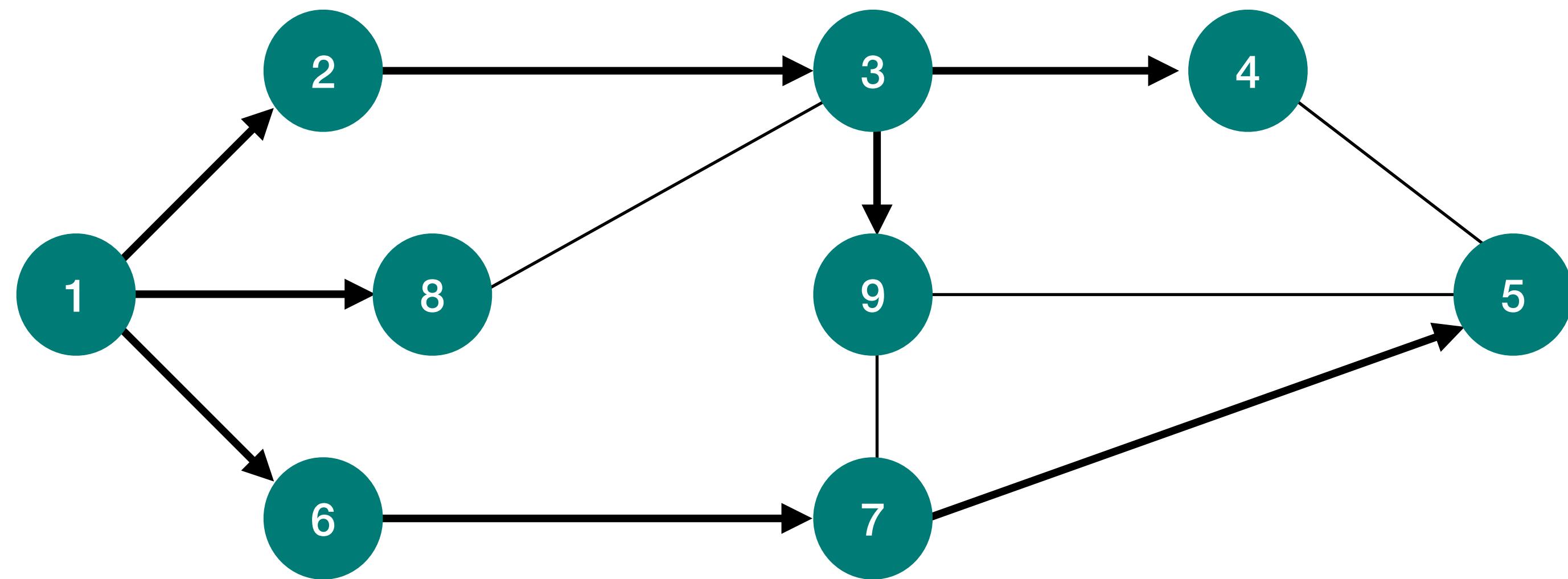
Output: 1 - 2 - 6 - 8 - 3



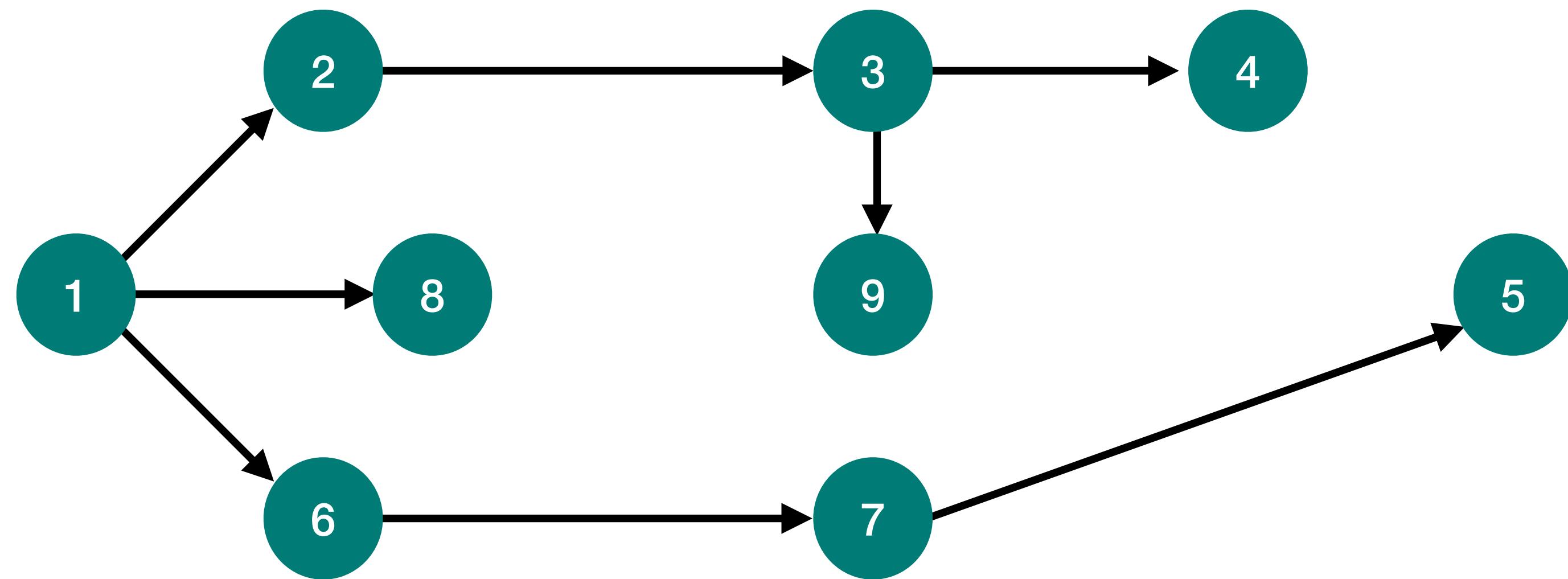
Output: 1 - 2 - 6 - 8 - 3 - 7



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



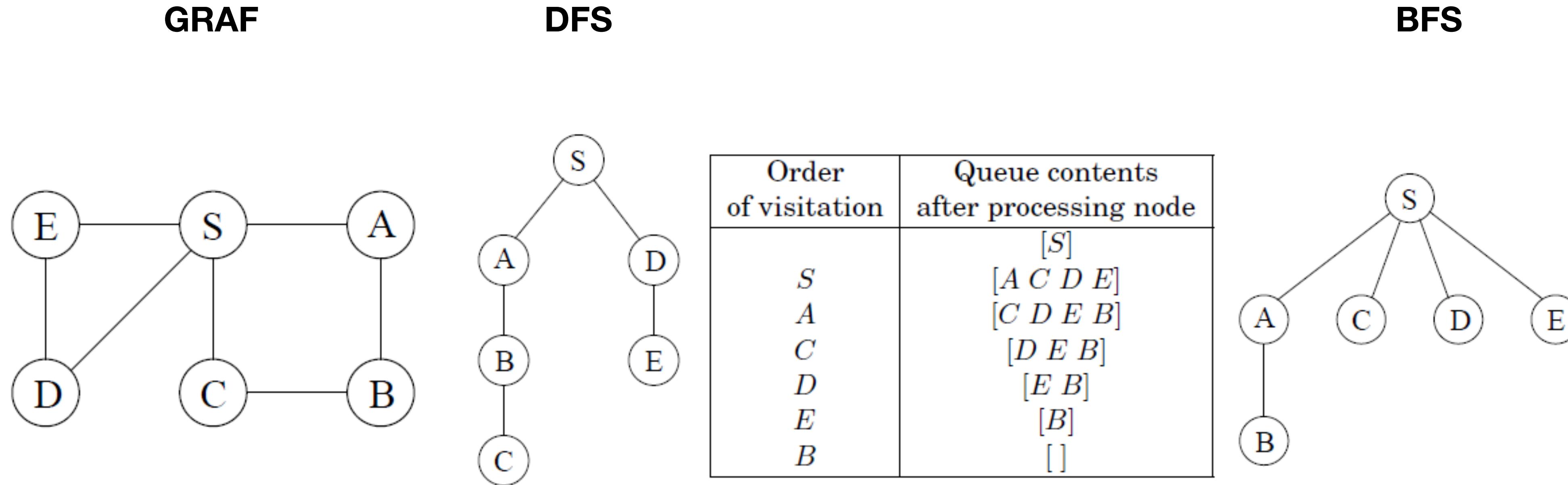
Output: 1 - 2 - 6 - 8 - 3 - 7 - 4 - 9 - 5



Com podem implementar-ho?
Alguna idea?

Recorregut amb profunditat (BFS)

- **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!



Recorregut amb profunditat (BFS)

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

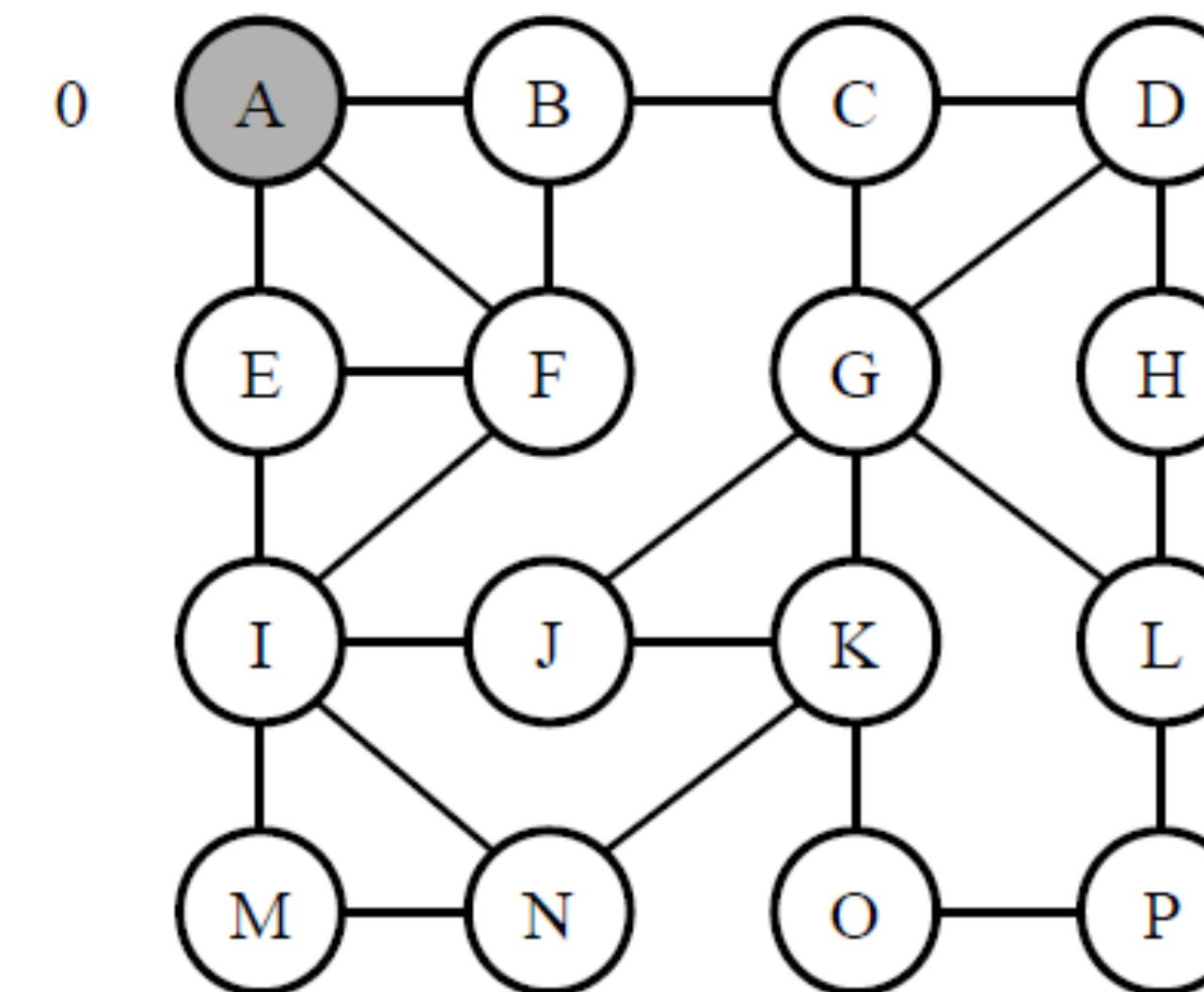
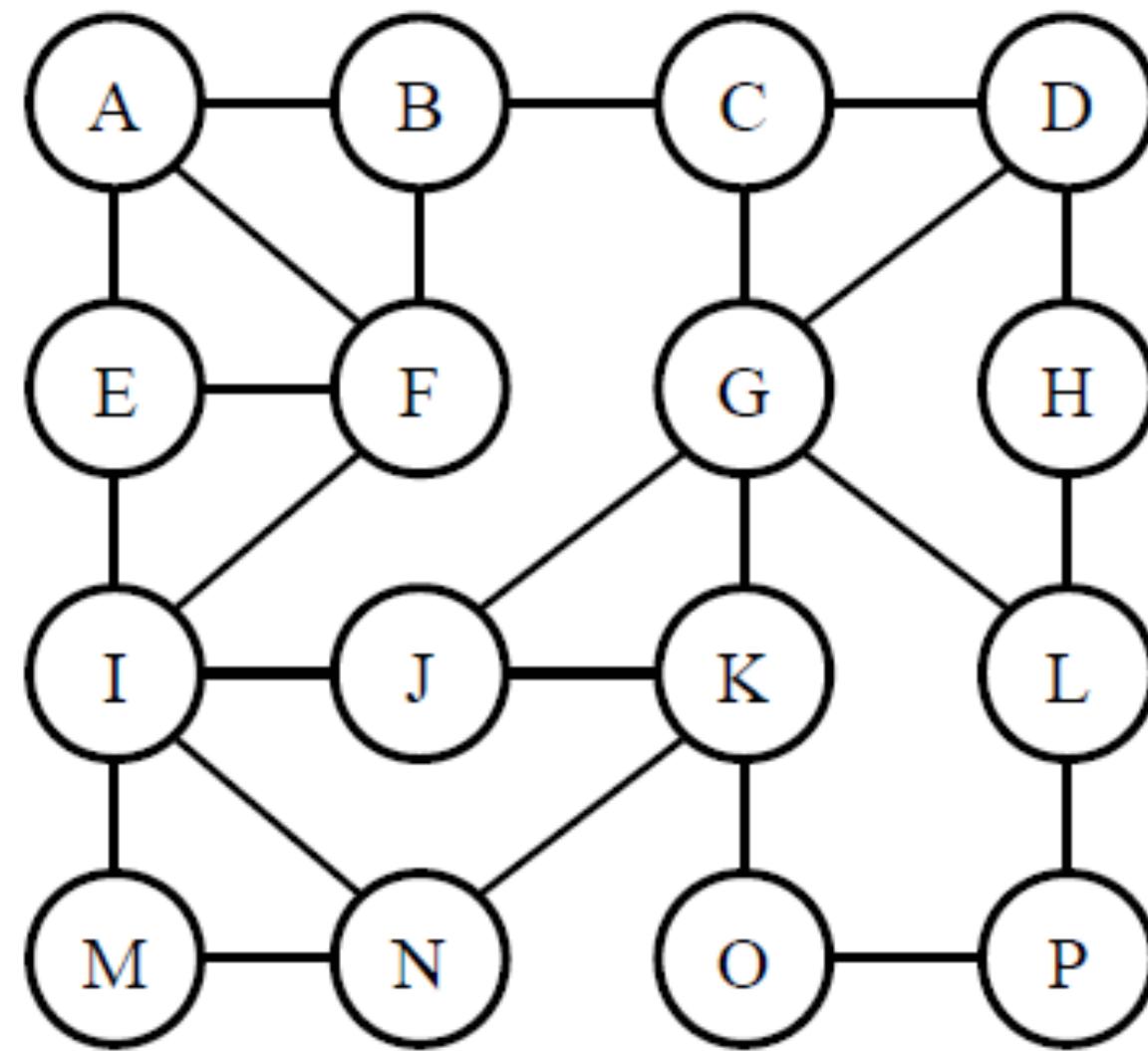
```
# visits all the nodes of a graph (connected component) using BFS
def bfs_connected_component(graph, start):
    # keep track of all visited nodes
    explored = []
    # keep track of nodes to be checked
    queue = [start]

    # 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
```

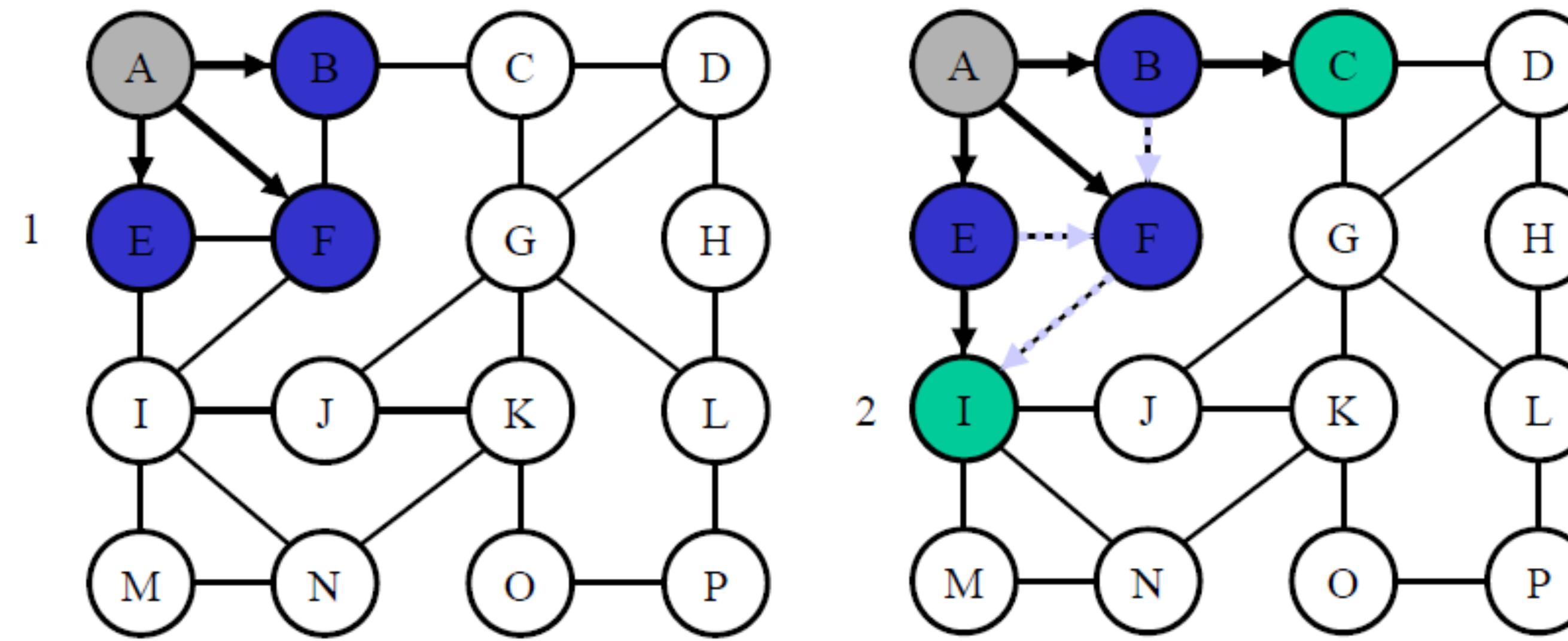
Algorismes sobre grafs

- Exemple



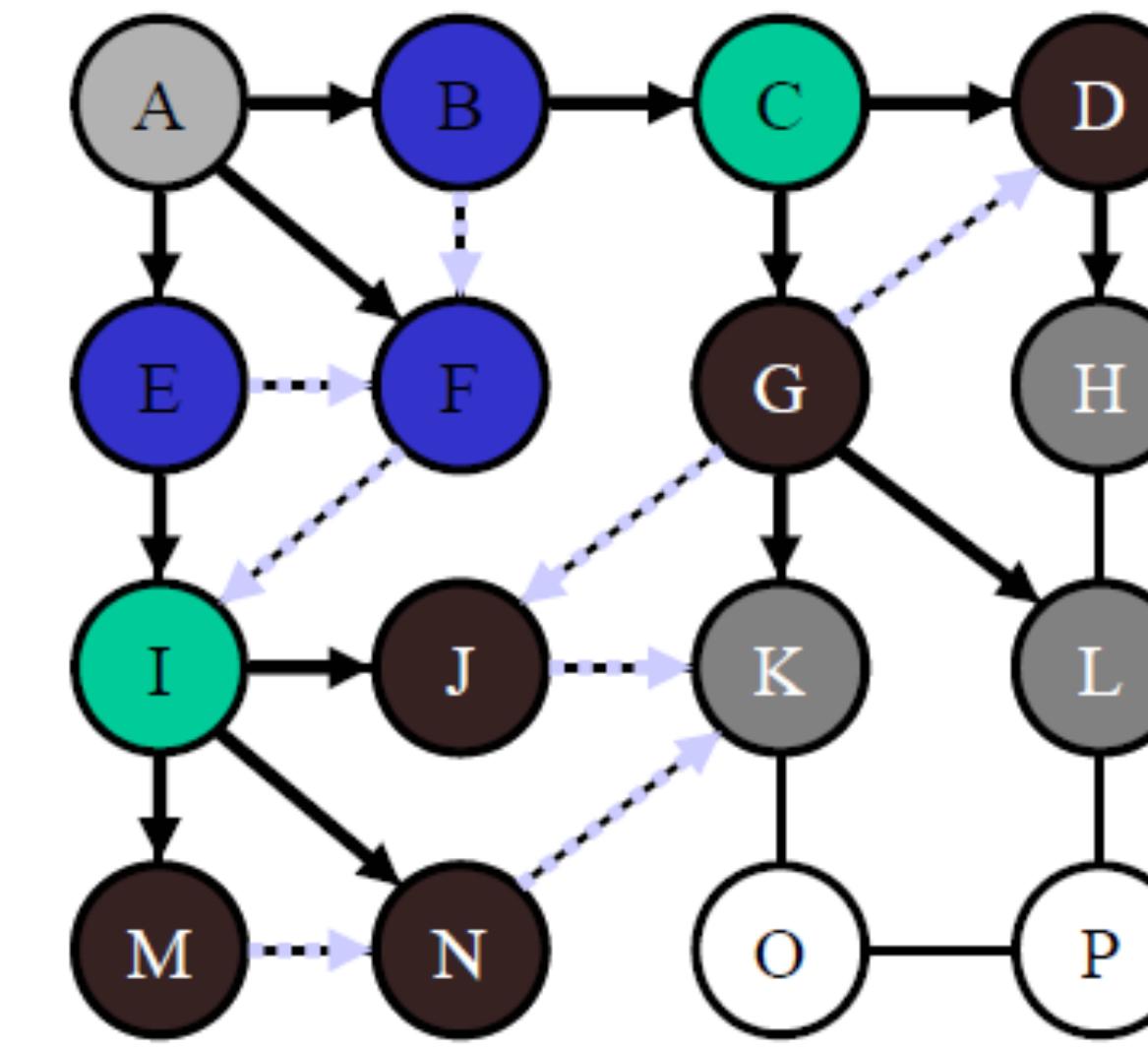
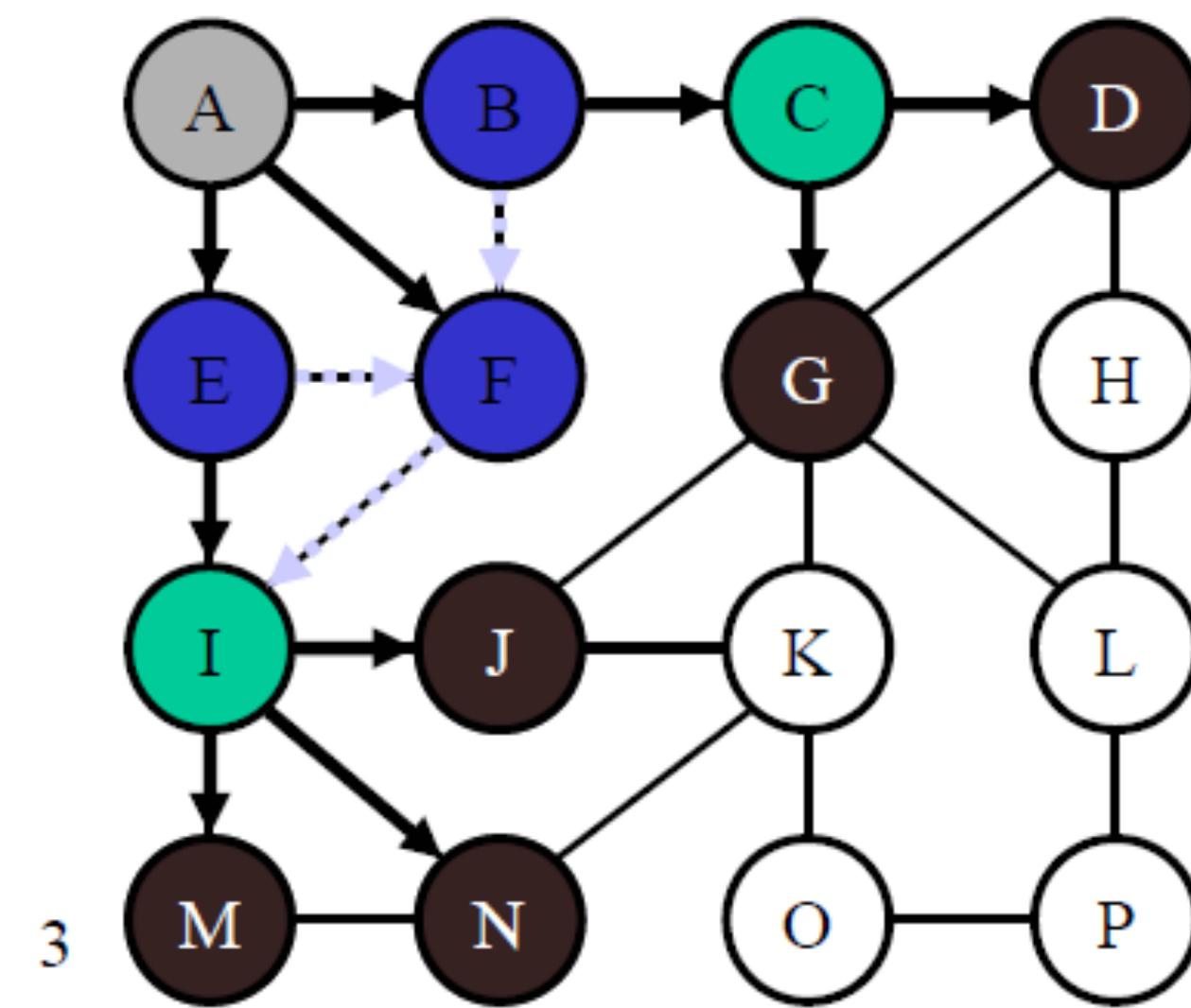
Algorismes sobre grafs

- Exemple



Algorismes sobre grafs

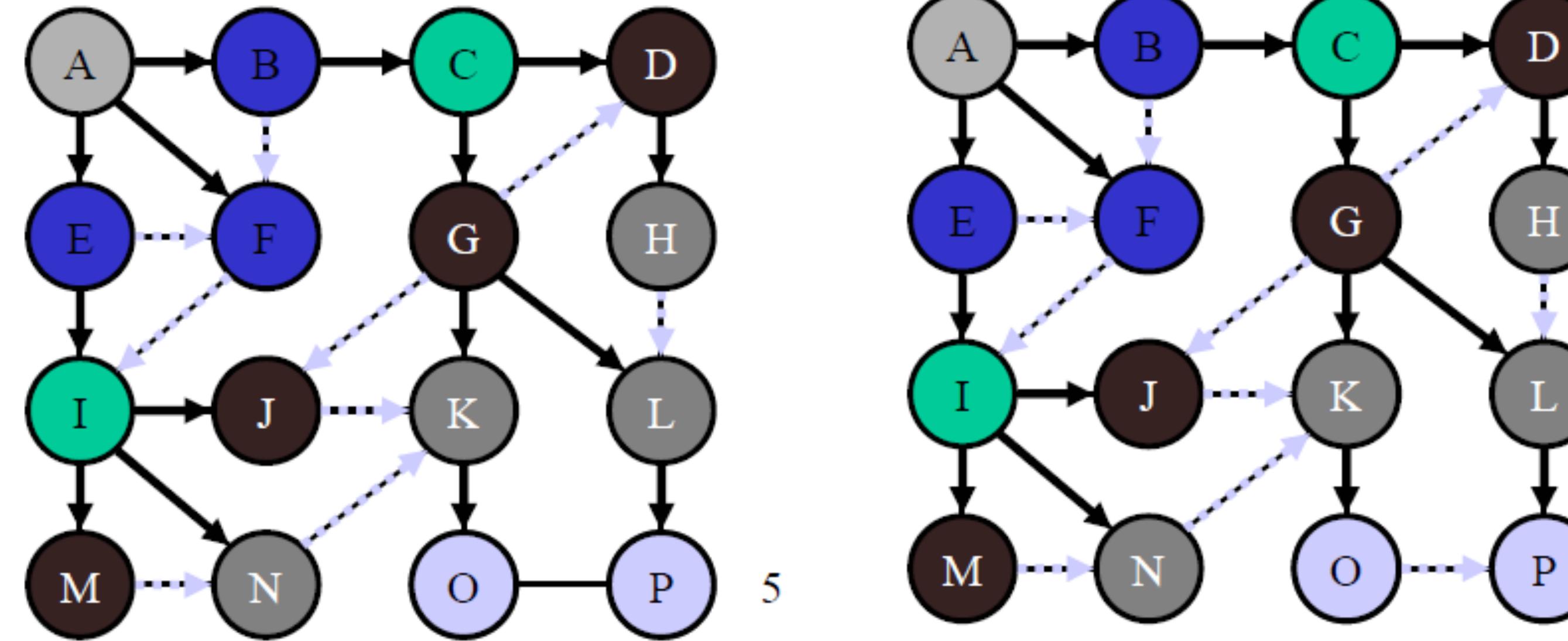
- Exemple



4

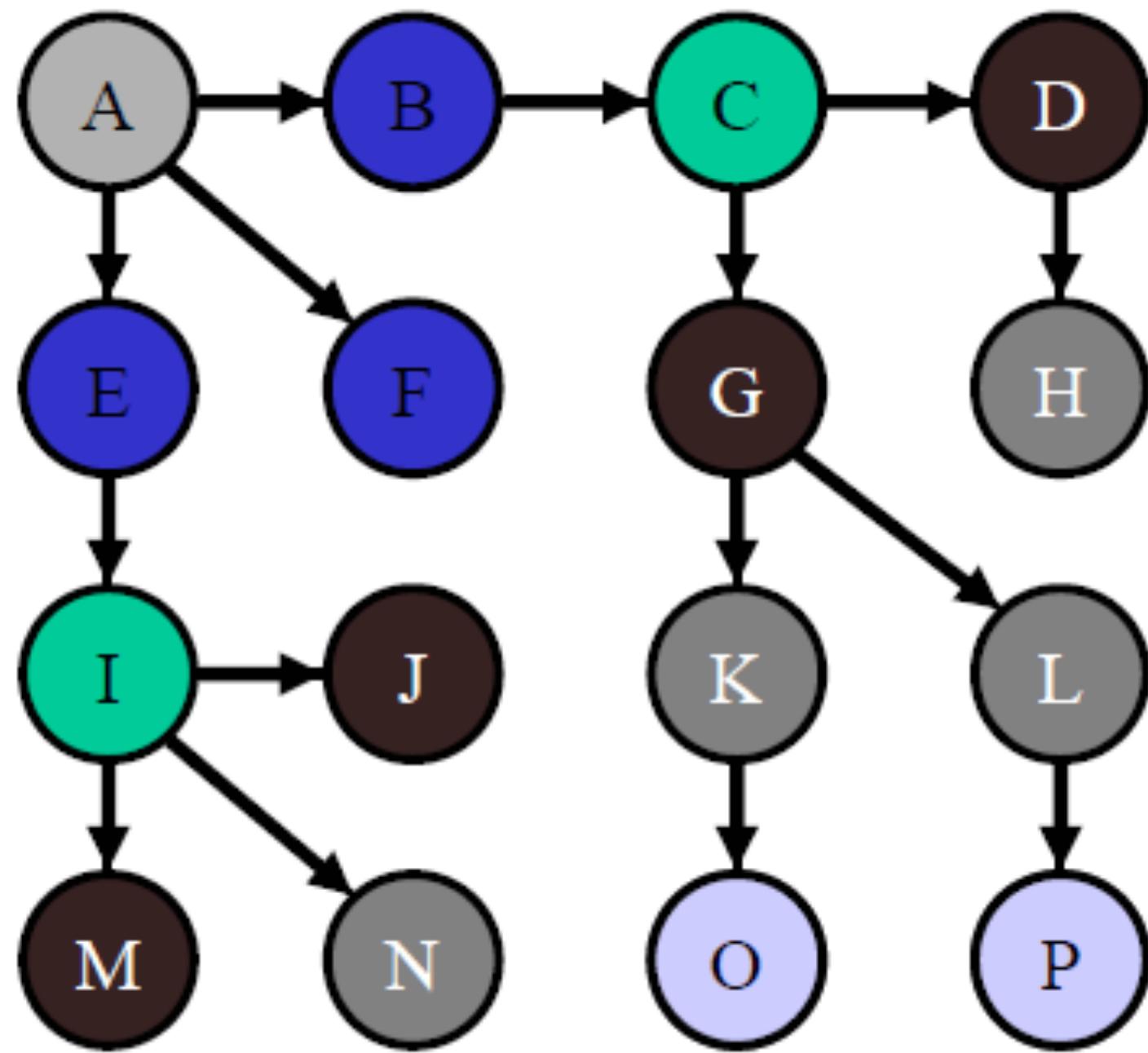
Algorismes sobre grafs

- Exemple



Algorismes sobre grafs

- Exemple

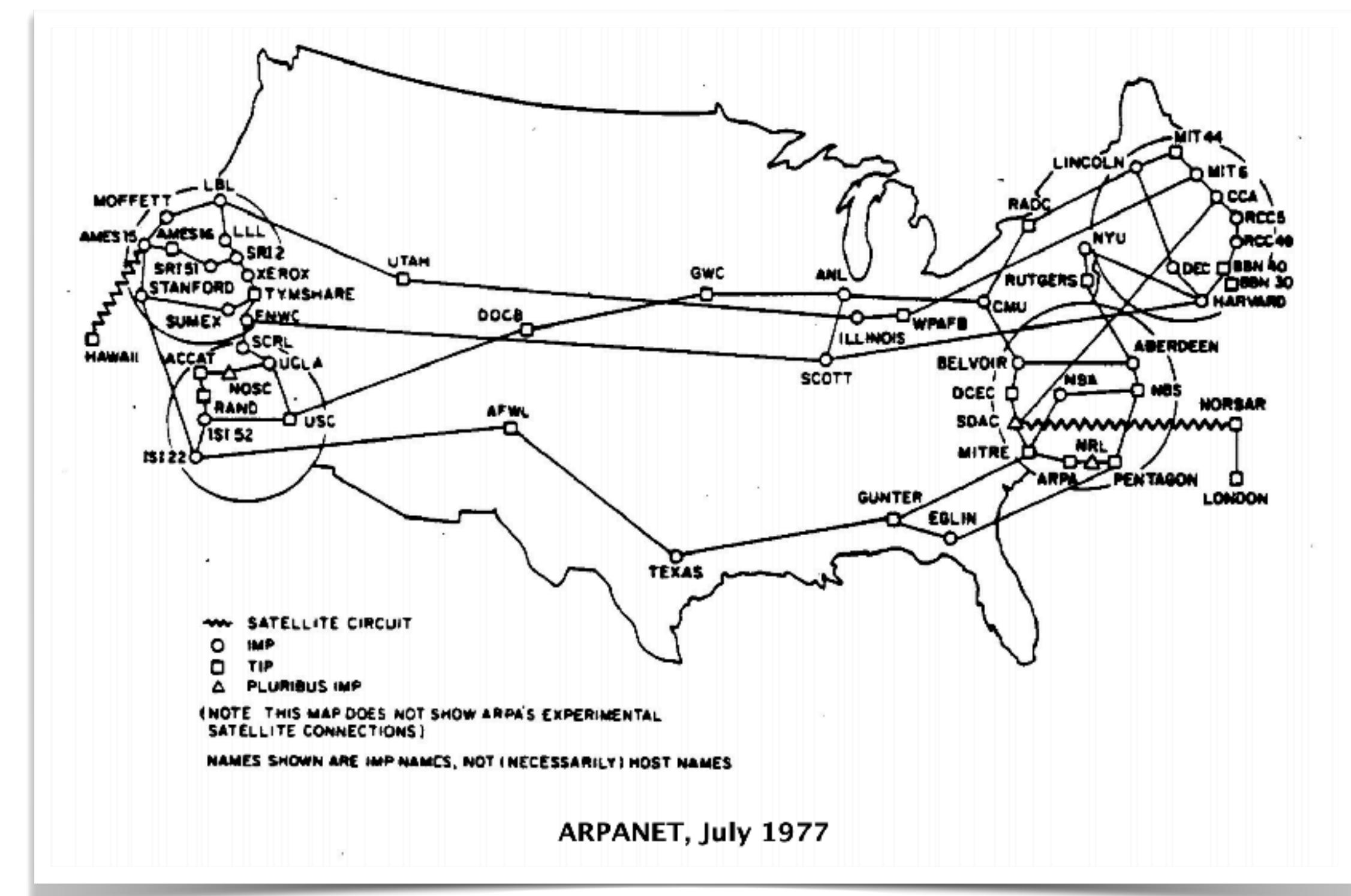


Algorismes sobre grafs

- 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

Aplicaciones

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

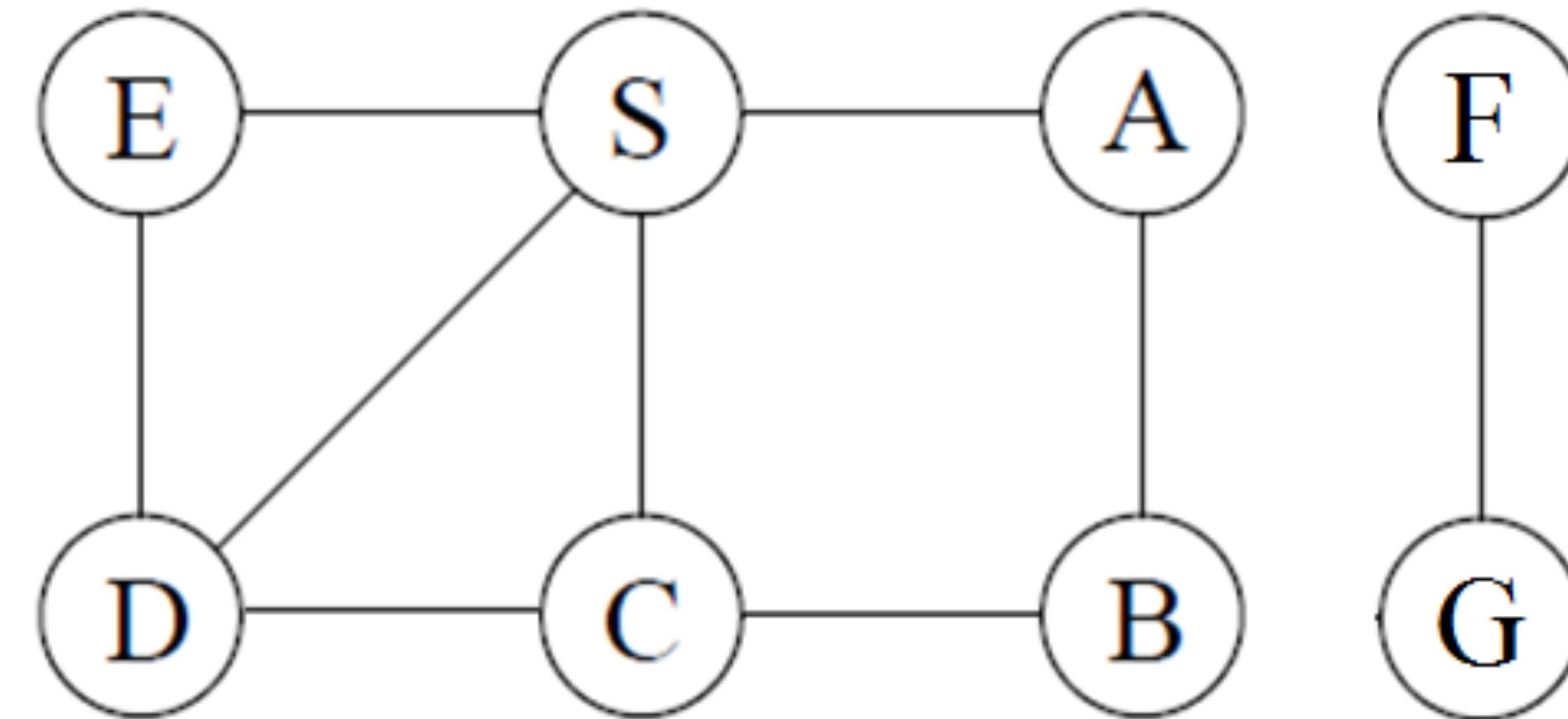


Aplicacions dels recorreguts

- Ruta més curta en grafs 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 graf.
- Permet que els paquets emesos arribin a tots els nodes d'una xarxa.

Exercicis

1. Dibuixa els arbres DFS obtinguts per a començant a cada node del següent graf
2. Dibuixa l'arbre BFS del mateix graf que obtenuï si comenceu pel node **S**. Indiqueu el contingut de la cua a cada iteració del mètode



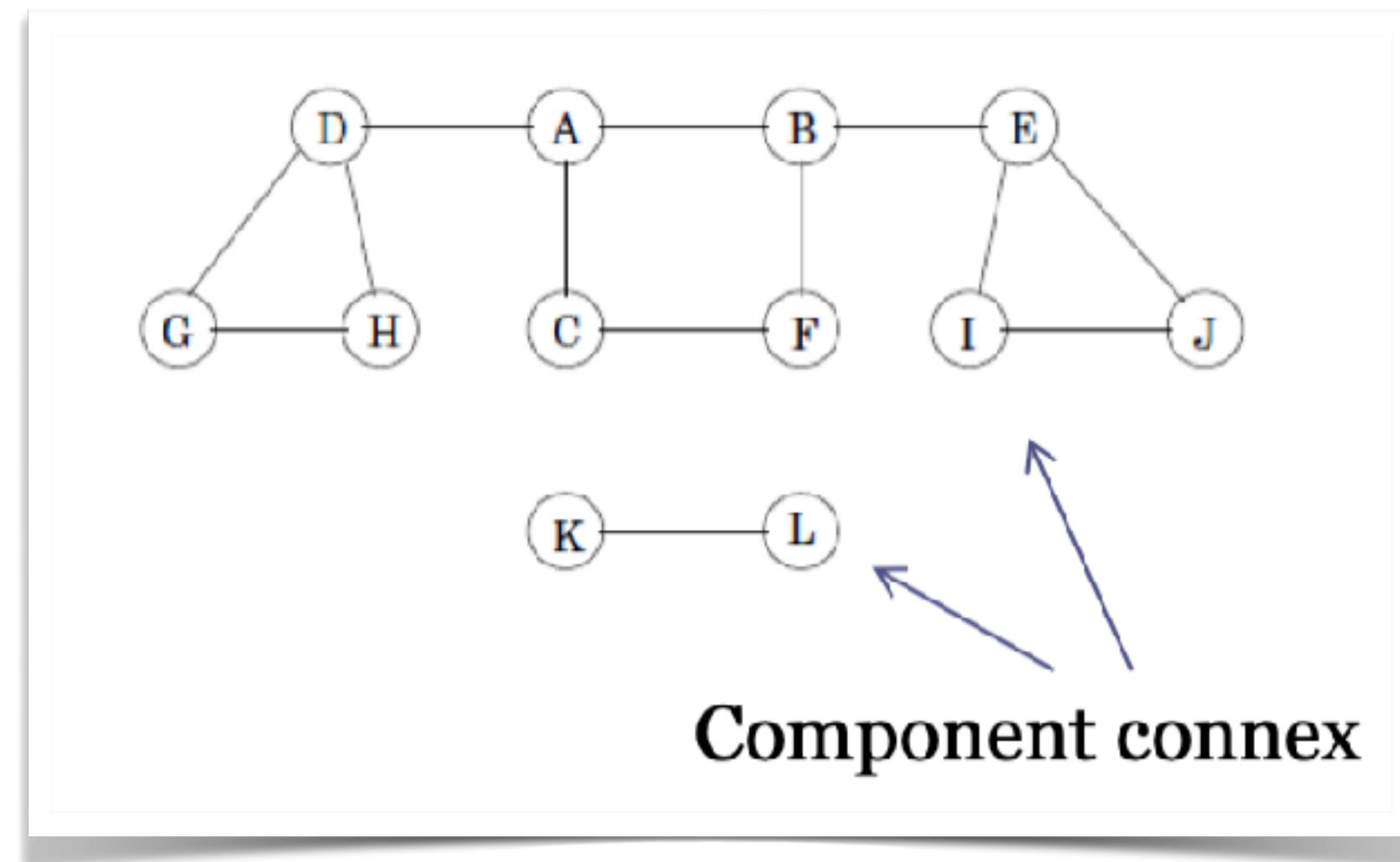
Exercici

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

Components Connexes

Components Connexes

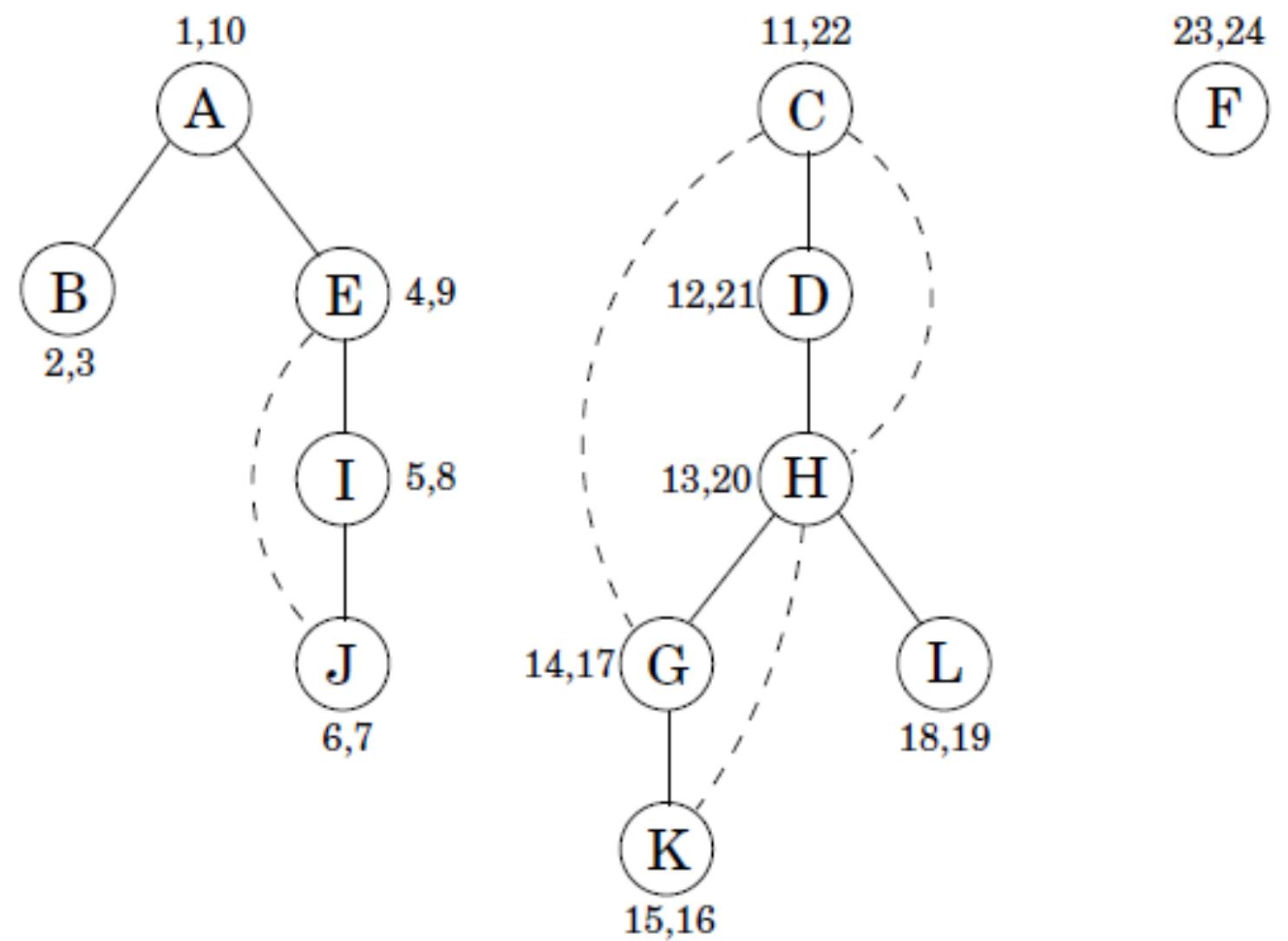
- Diem que un **graf no dirigit és connex** si és possible formar un camí des de qualsevol vèrtex a qualsevol altre en el graf. Una **component connexa** d'un graf no dirigit és un subgraf connex maximal. Cada vèrtex i cada aresta pertany a una única component connexa.



Exemple: Graf no dirigit no connex, 12 vèrtexs, 13 arestes i dues components connexes

Components Connexes

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



$\{A, B, E, I, J\}$

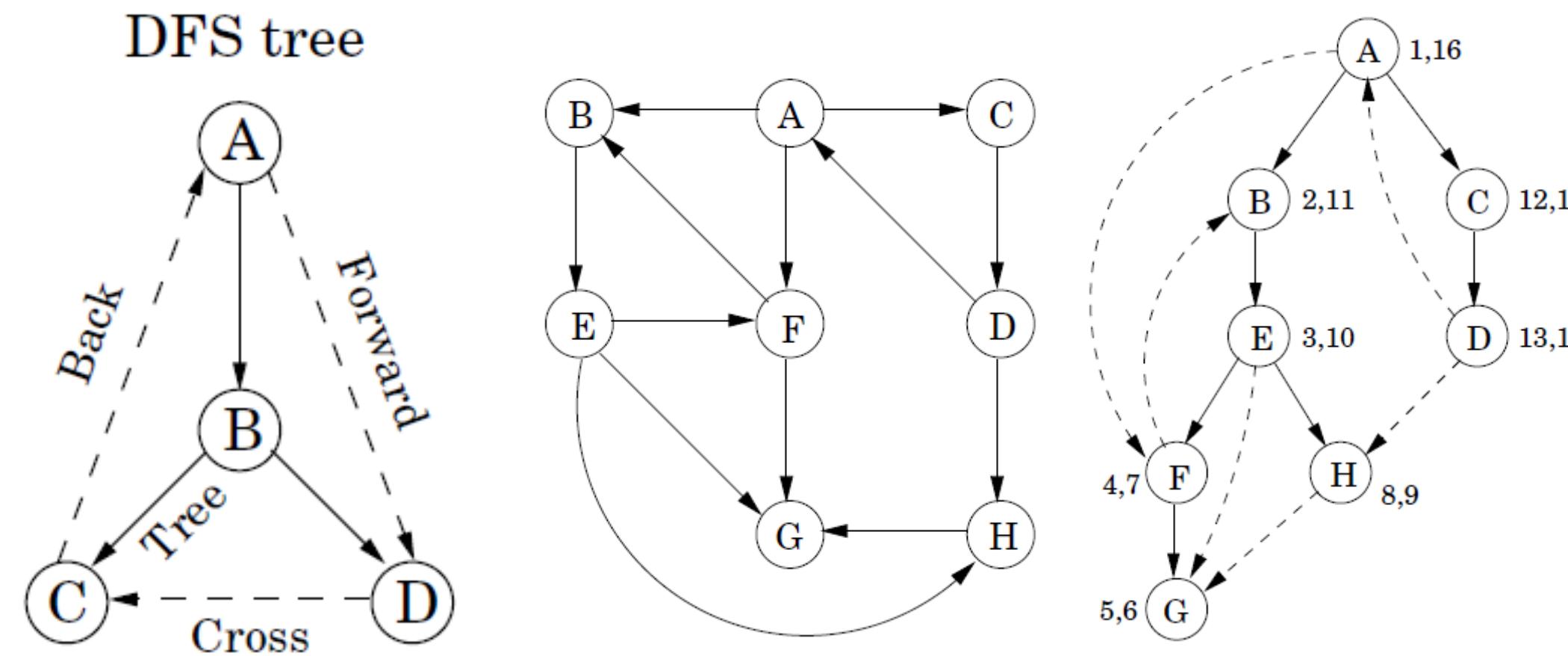
$\{C, D, G, H, K, L\}$

$\{F\}$

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

Graf fortament connex

- Un **graf dirigit** s'anomena **fortament connex** quan, donats dos vèrtexs qualssevol u, v , conté un camí dirigit de u a v i un camí dirigit de v a u . Una component connexa forta és un subgraf maximal fortament connex.

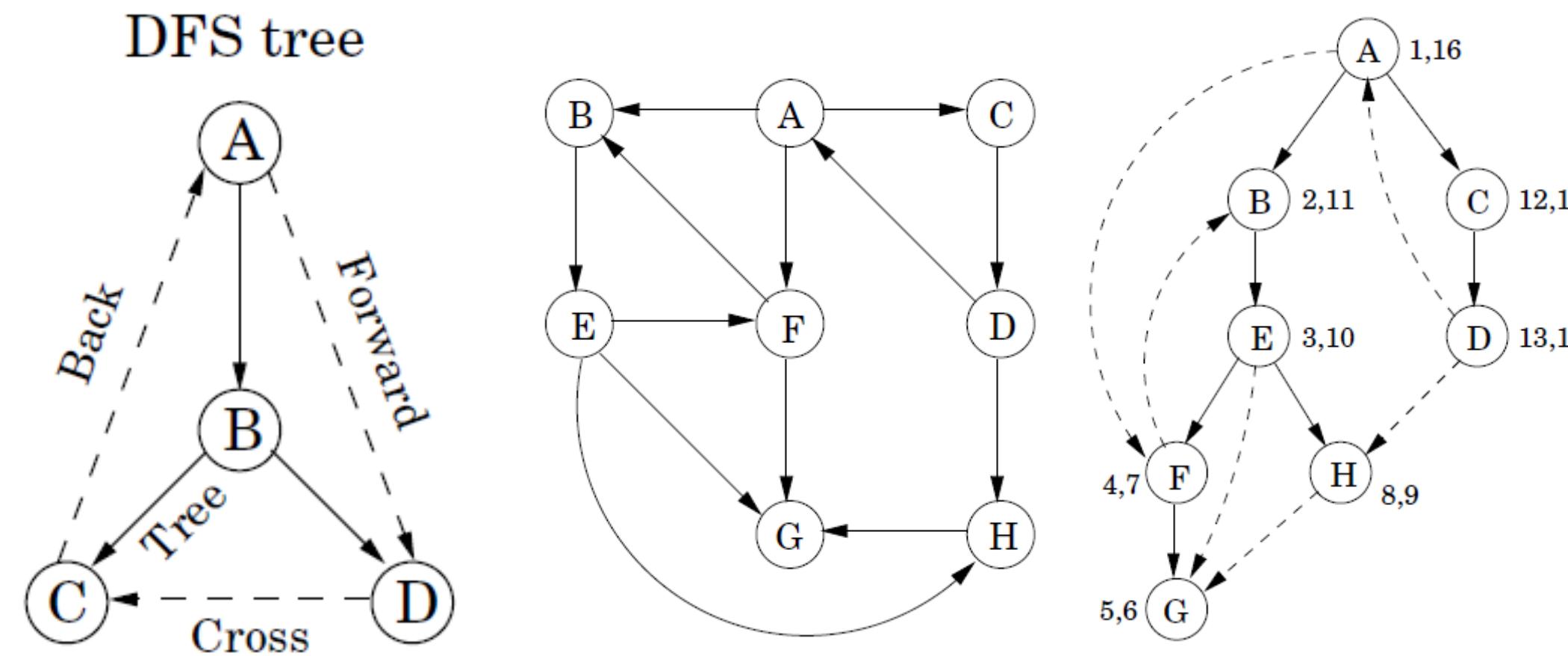


Graf feblement connex

- Un **graf dirigit** s'anomena **feblement connex** quan és connex com a graf no dirigit. És a dir, quan en substituir totes les seves arestes (dirigides) per arestes no dirigides obtenim un graf no dirigit connex.

Components Connexes

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

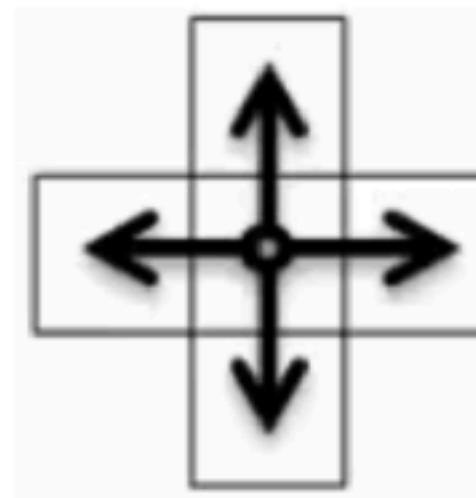


Algunes Aplicacions

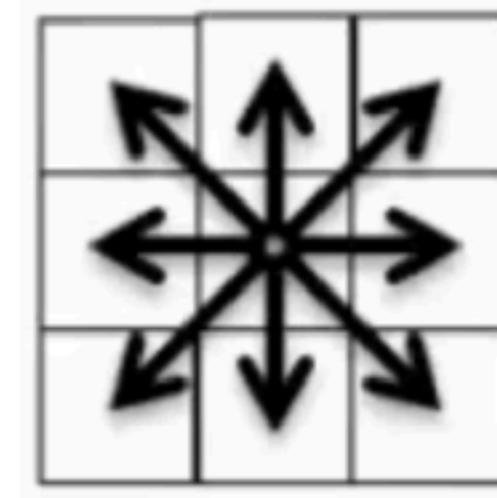
Algoritme **FloodFill**

Flood Fill

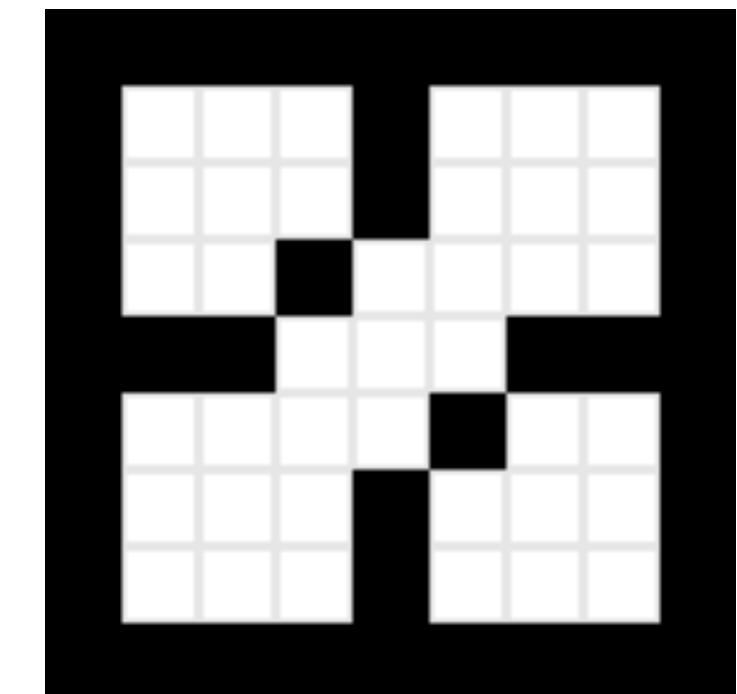
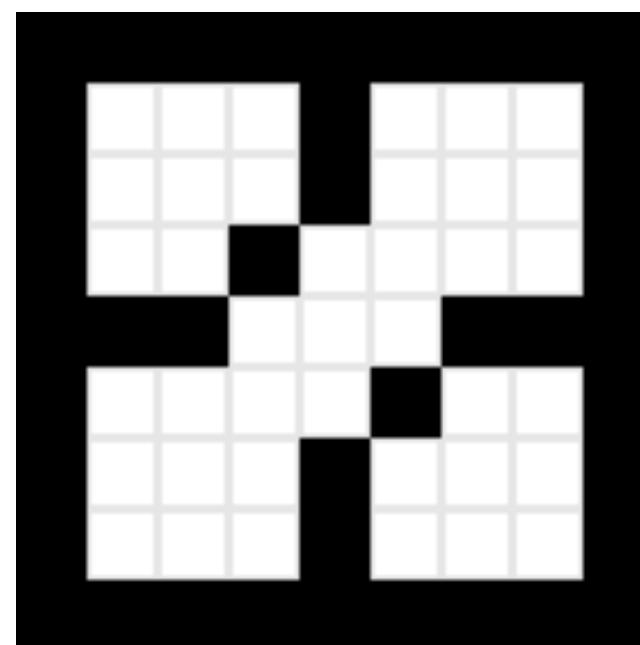
- L'algorisme **Flood Fill** ens permet determinar l'àrea connectada a un node donada una matriu multidimensional.
Una imatge de $N \times M$ píxels pot ser representada com un graf de $N \times M$ nodes on cada pixel és un node del graf. Dos nodes estan connectats mitjançant una aresta si aquests són adjacents. Podem tindre connectivitat a 4 (horitzontal i vertical) o connectivitat a 8 (horitzontal, vertical i diagonals).



Connectivitat a 4



Connectivitat a 8



Aplicaciones

Depth-first search application: flood fill

Challenge. Flood fill (Photoshop magic wand).

Assumptions. Picture has millions to billions of pixels.

input

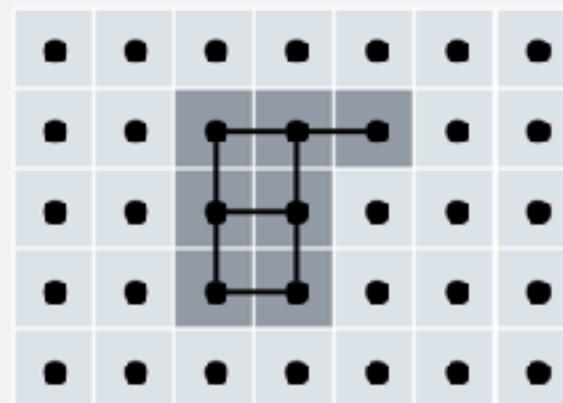


floodFill mask



Solution. Build a **grid graph**.

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

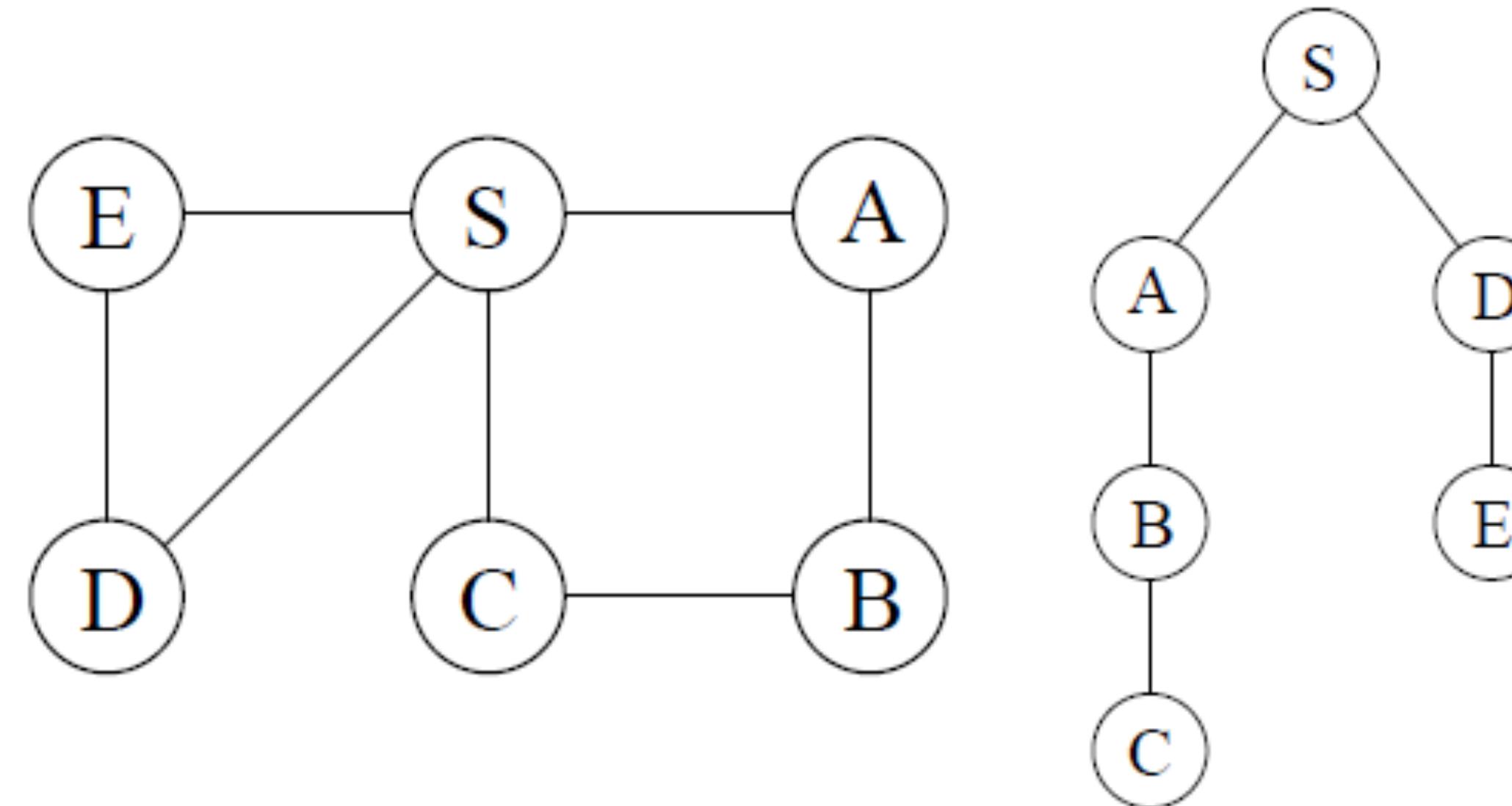


Shortest Path

Camí més curt entre dos
nodes

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???**



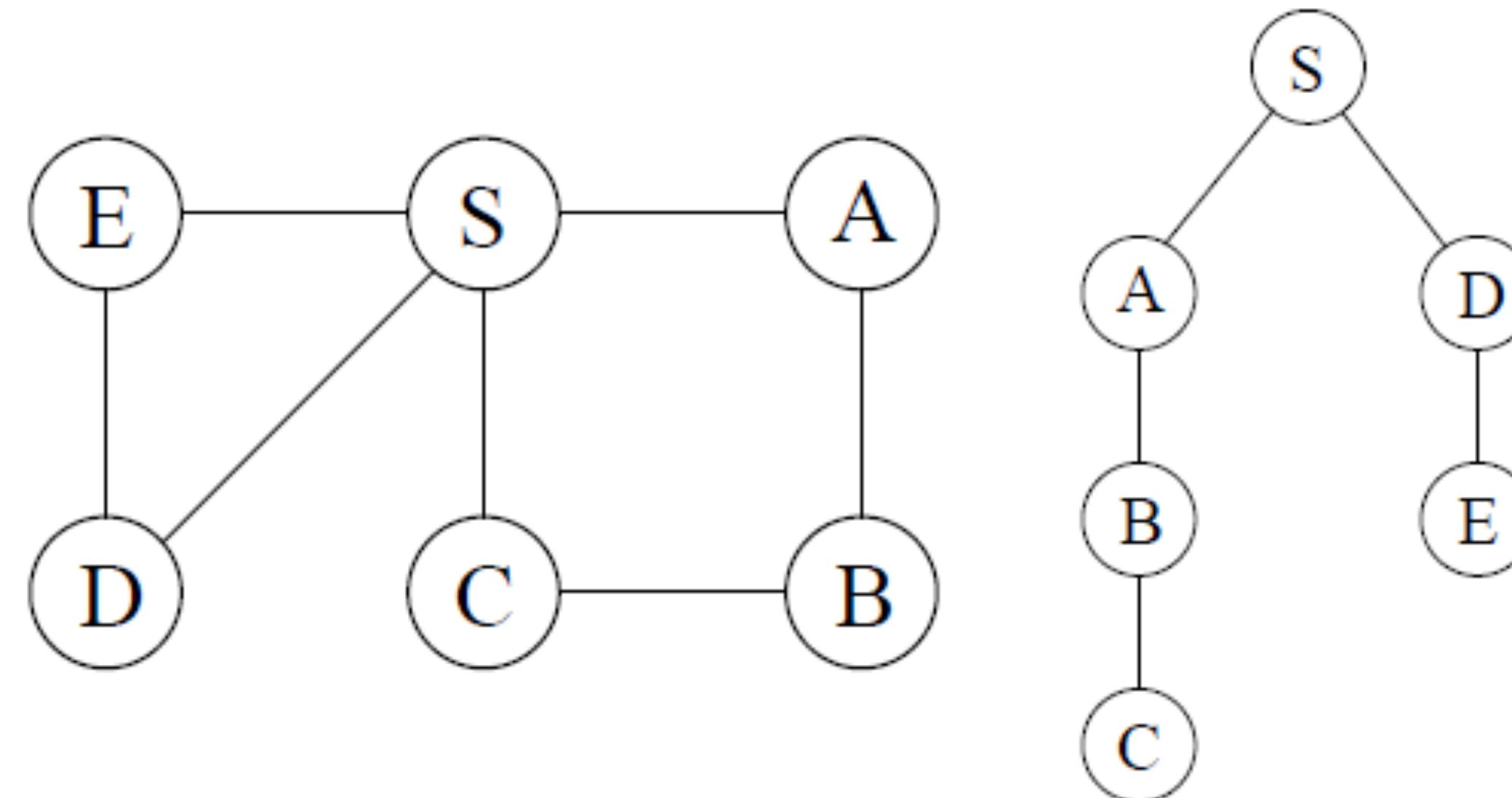
Podem definir el **camí més curt entre 2 vèrtexs** com el número d'arestes fins arribar, o el número de vèrtexs que travesssem, o la suma dels pesos de les arestes, dels vèrtexs, etc.

Shortest Path

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

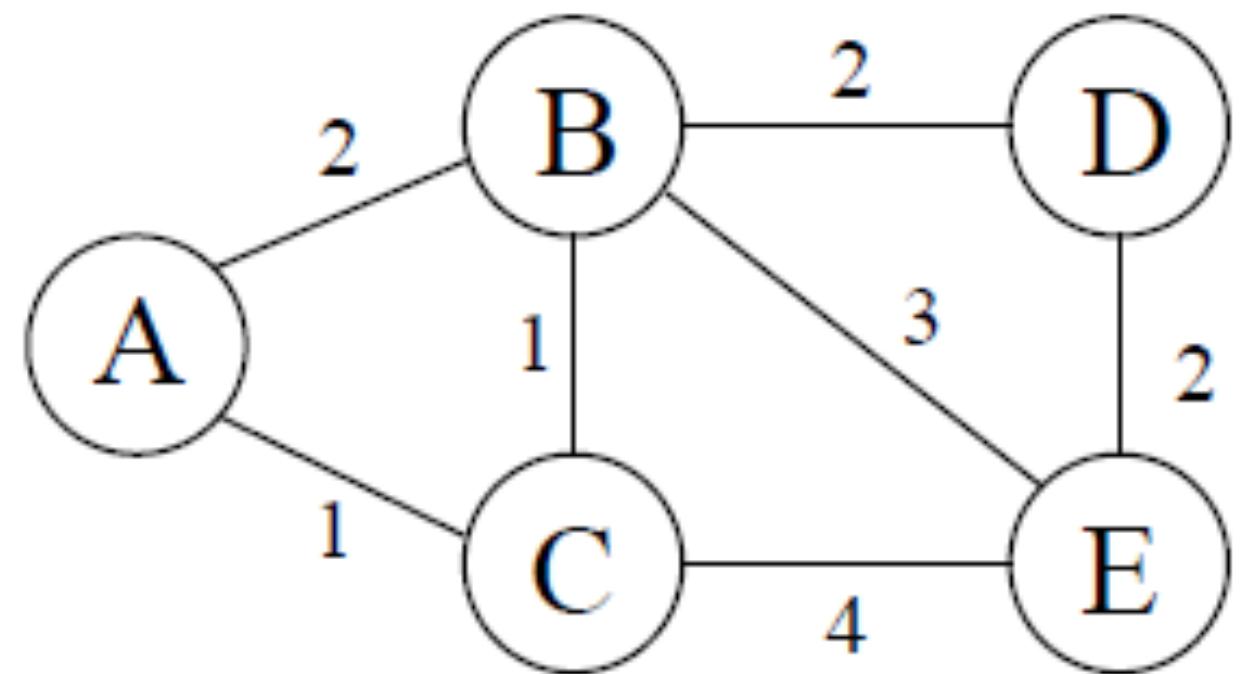
Shortest Path

- Pensem primer una versió una versió per fer ús de BFS



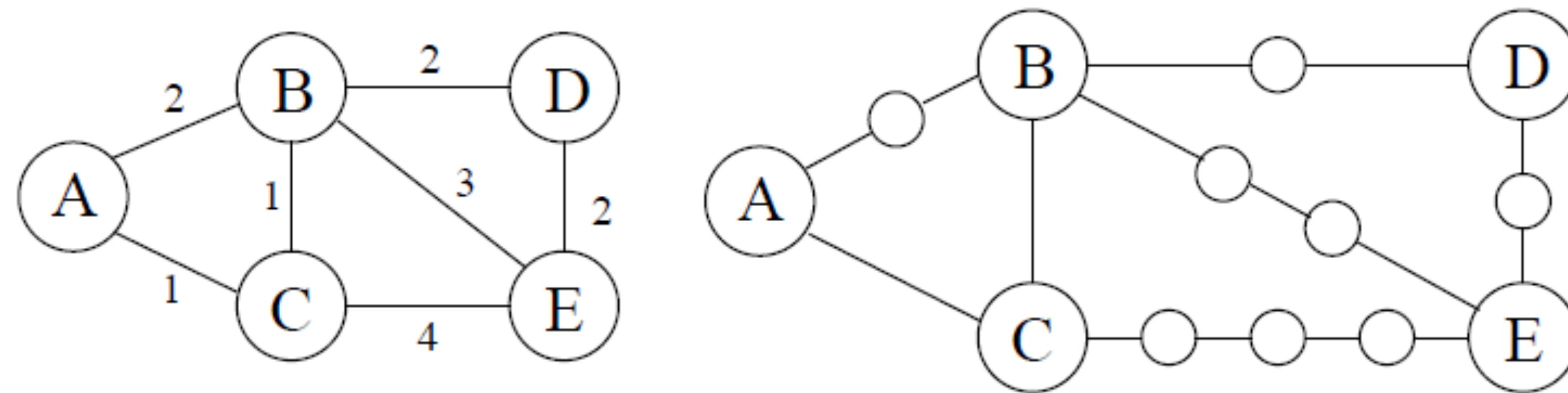
Shortest Path

- En cas que les arestes tinguin pesos. Com ho podem fer?
- Pensem primer una versió una versió per fer ús de BFS



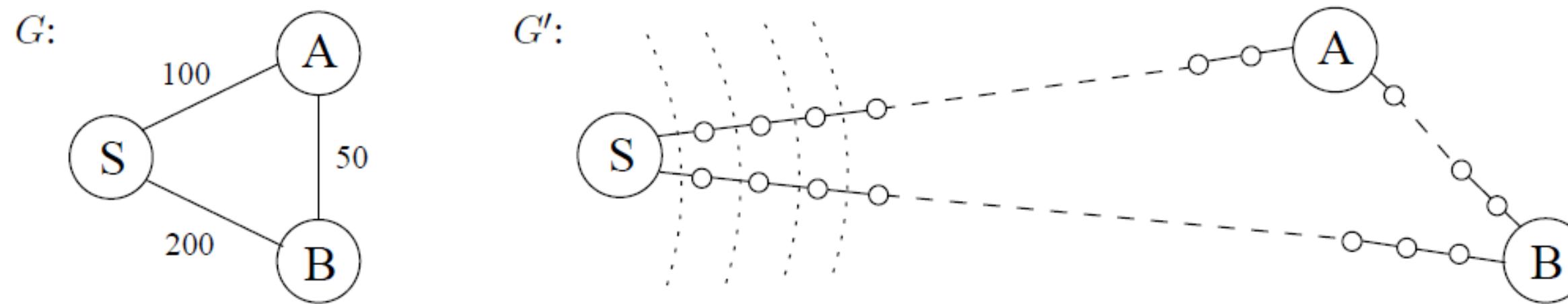
Shortest Path

- En cas que les arestes tinguin pesos. Com ho podem fer?
 - Pensem primer una versió 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 arrestes!!!

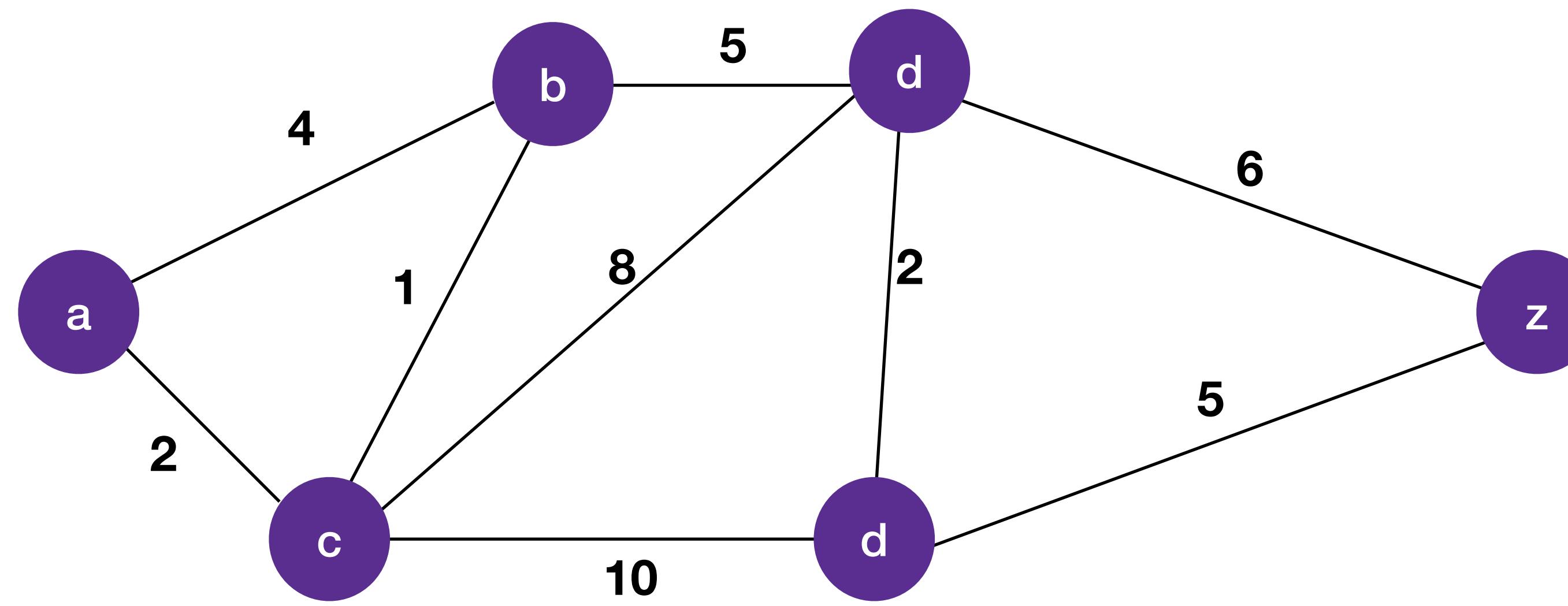
Dijkstra

Algoritme Dijkstra



- L'algoritme **Dijkstra** ens permet determinar el camí més curt donat un vèrtex origen a la resta de vèrtexs en un graf amb pesos positius a les arestes.
- El seu nom li ve d' [Edsger Dijkstra](#), qui el va descriure per primera vegada el 1959.
- La idea subjacent en aquest algorisme consisteix a anar explorant tots els camins més curts que parteixen del vèrtex origen i que porten a tots els altres vèrtexs, quan s'obté el camí més curt des del vèrtex origen, a la resta de vèrtexs que formen el graf, l'algorisme s'atura.
- L'algorisme **no funciona** si tenim **costos negatius** a les arestes.
- En funció de com implementem el mètode i les estructures de dades utilitzades, la complexitat en temps és $O(E \cdot \log(V))$

Algorismes sobre grafs



Algorismes sobre grafs

- 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!!!

- Definim una alarma per al node **s** al temps **0**
- Repetim mentres existeixin alarmes pendents
 - Diguem que la següent alarma es dispara al temps **T**, per al node **u**:
 - Fixem la **distància** del node **s** a **u** a cost **T**
 - Per a cada veí **v** del node **u** del graf **G**:
 - Si no tenim una alarma fixada pel node **v**, fixem una nova alarma per aquest node al temps **T+ l(u,v)**
 - Si l’alarma del node **v** esta programada per disparar-se més tard que **T+ l(u,v)**, la reiniciem perquè es dispari al temps **T+ l(u,v)**

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:

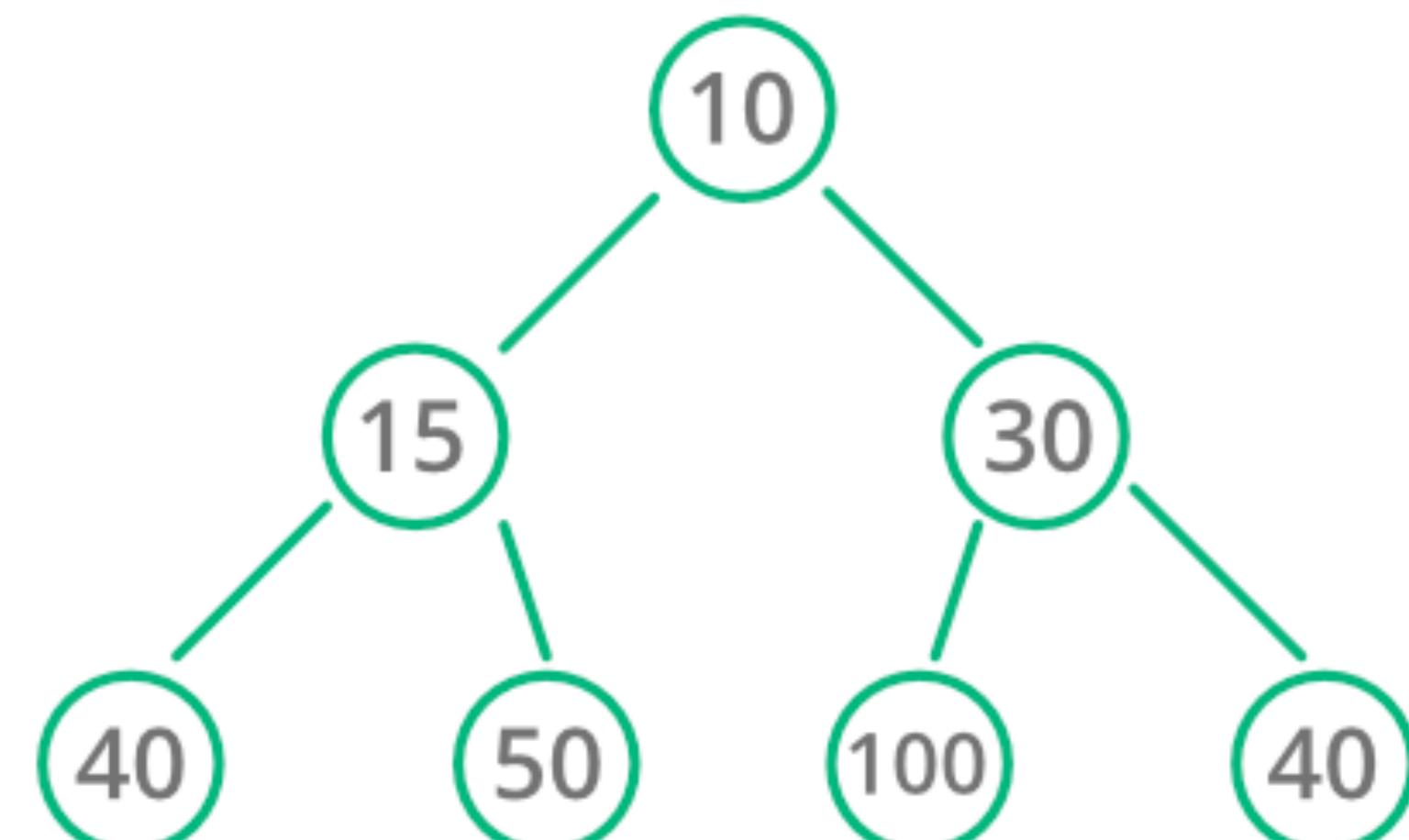
Inserció: inclou un nou element al conjunt.

Decrementar-clau: Disminuir 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 disminuït.

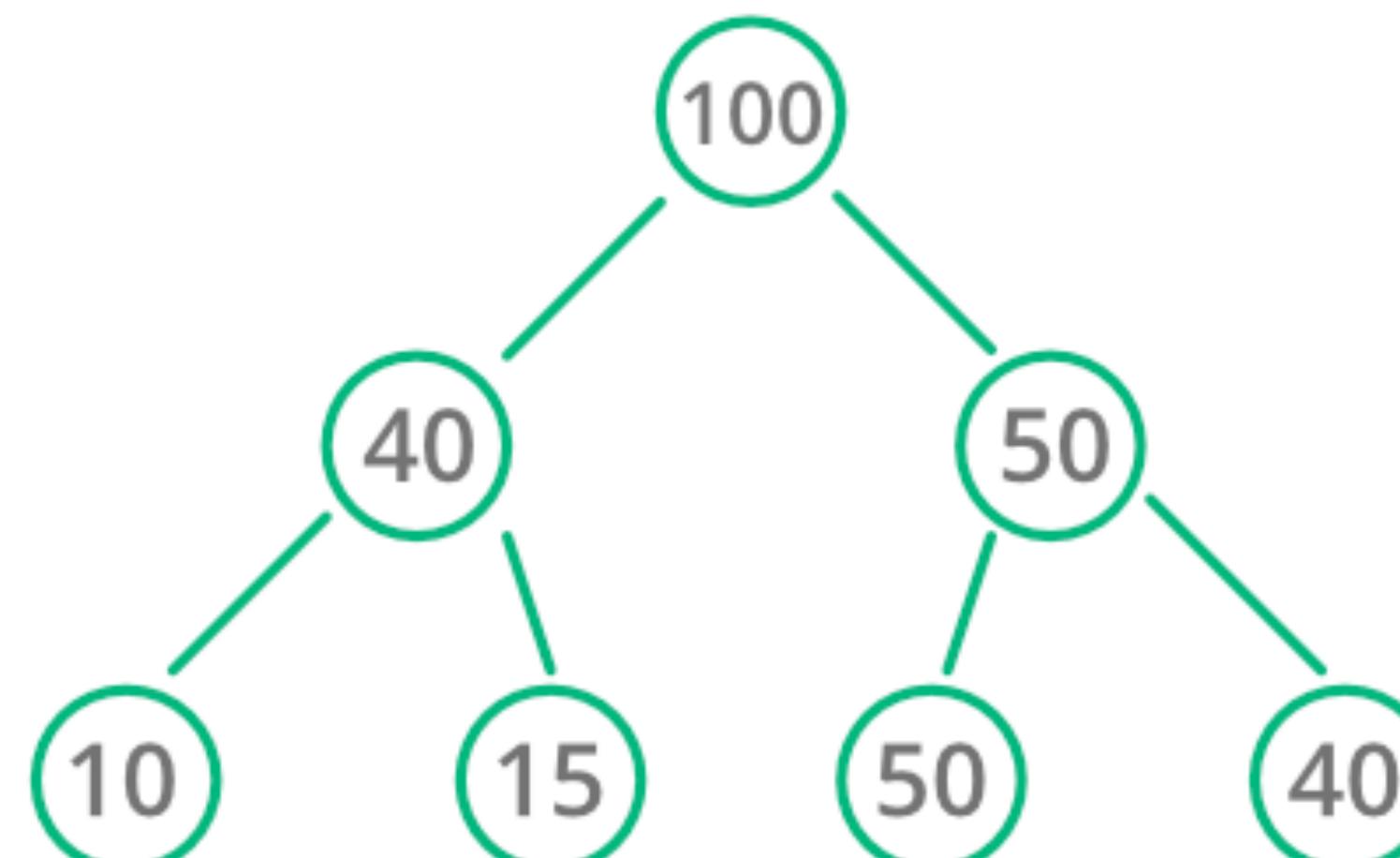
Eliminar-min: Retorna l'element amb la menor clau i l'elimina del conjunt.

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

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

Inserció: inclou un nou element al conjunt.

Disminuir-clau: Disminuir 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 disminuït.

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.

Dijkstra

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

for all $u \in V$:

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

$\text{prev}(u) = \text{nil}$

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

$H = \text{makequeue}(V)$ (using dist -values as keys)

while H is not empty:

$u = \text{deletemin}(H)$

for all edges $(u, v) \in E$:

if $\text{dist}(v) > \text{dist}(u) + l(u, v)$:

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

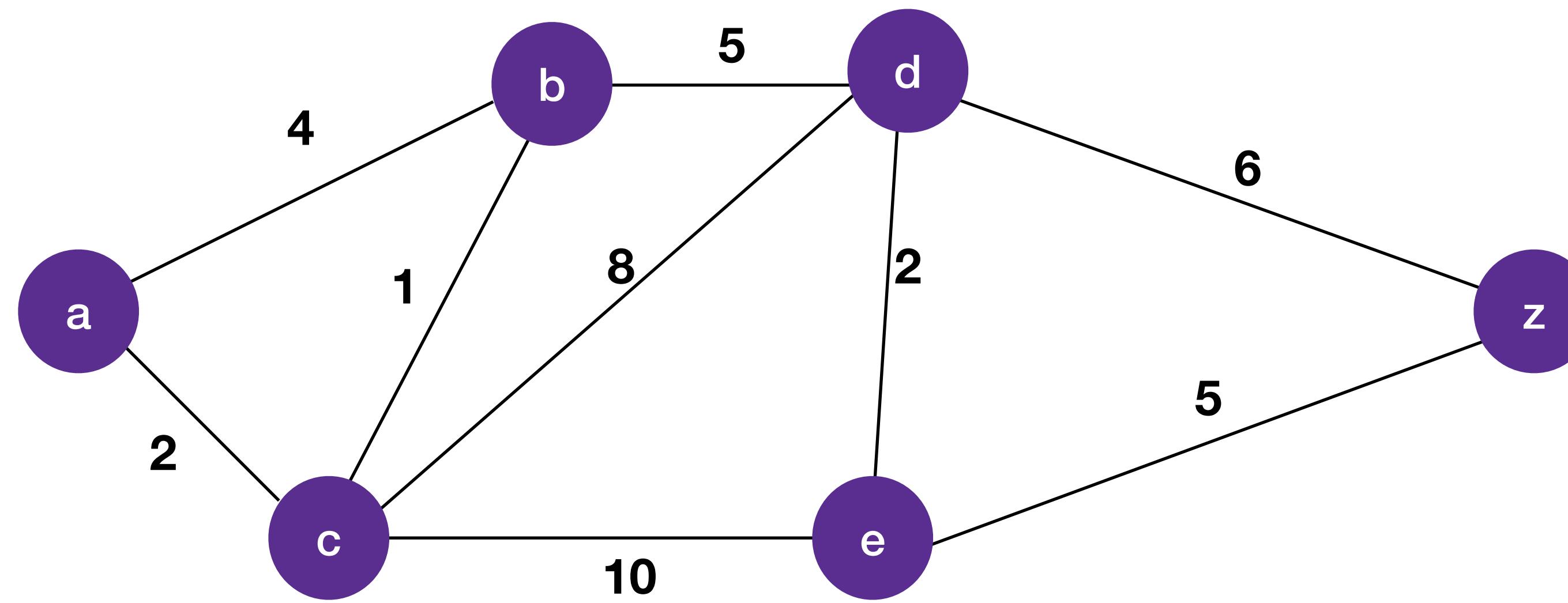
$\text{prev}(v) = u$

$\text{decreasekey}(H, v)$

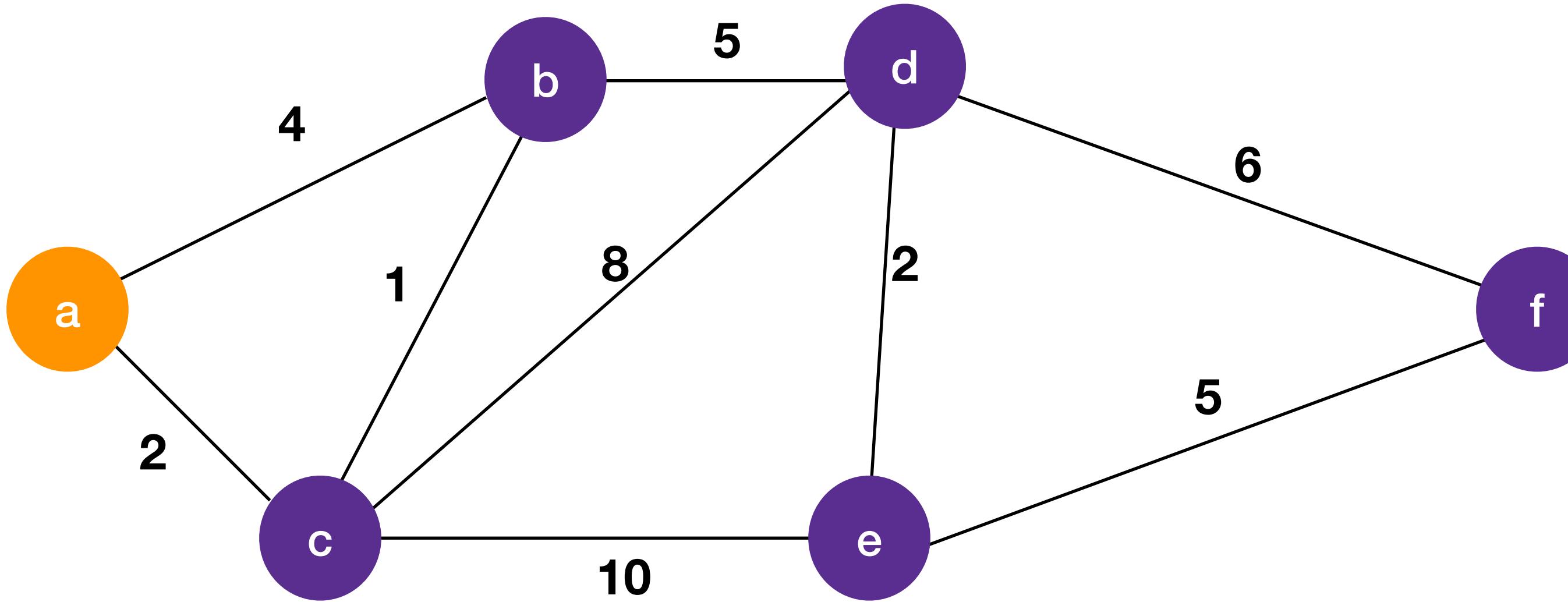
Dijkstra

- **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.

Algorismes sobre grafs

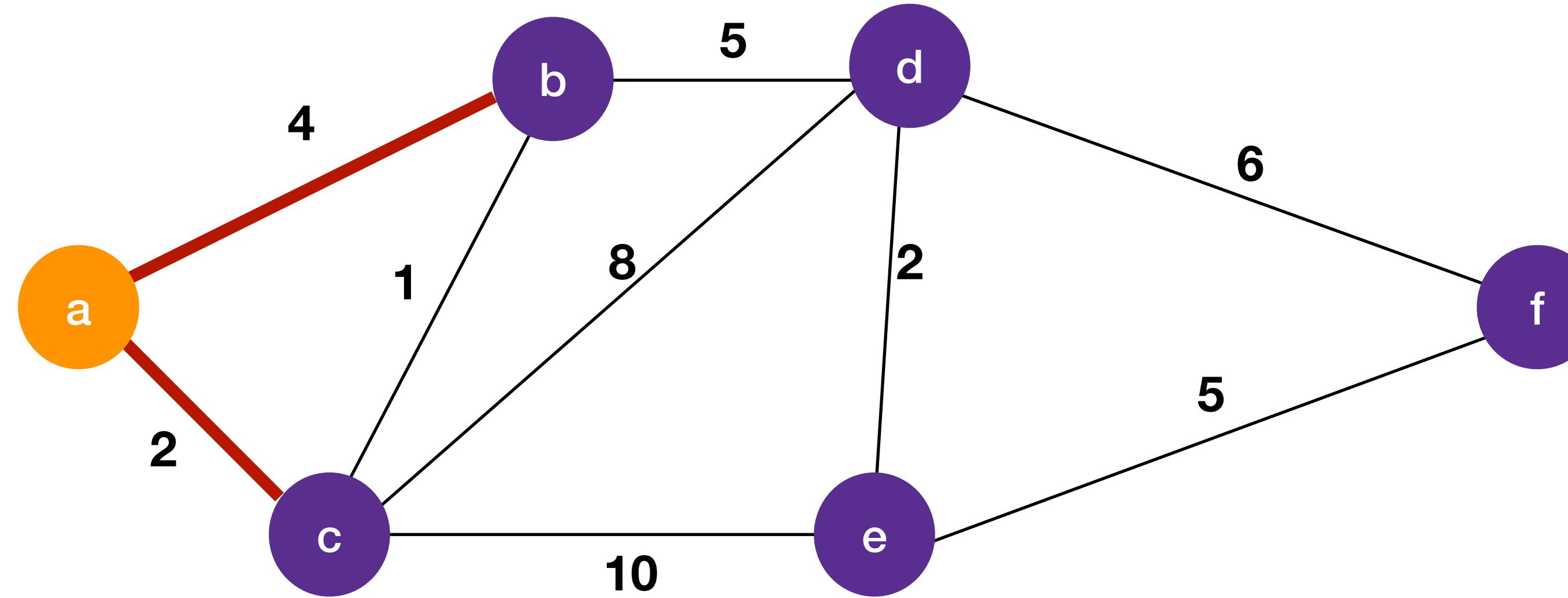


Algorismes sobre grafs



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

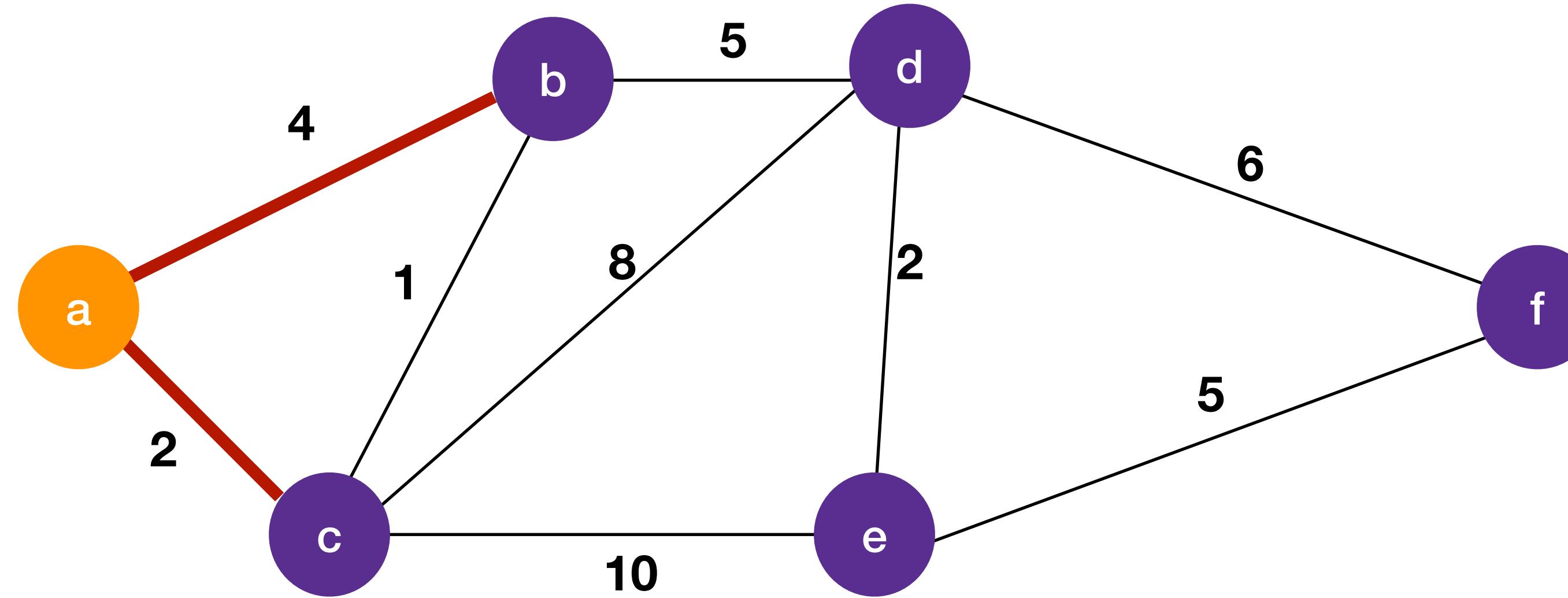
Algorismes sobre grafs



Node	Estat	Cami més curt des de A	Node previ
a	Node actual	0	
b		∞ 4	A
c		∞ 2	A
d		∞	
e		∞	
f		∞	

veïns de A
no visitats

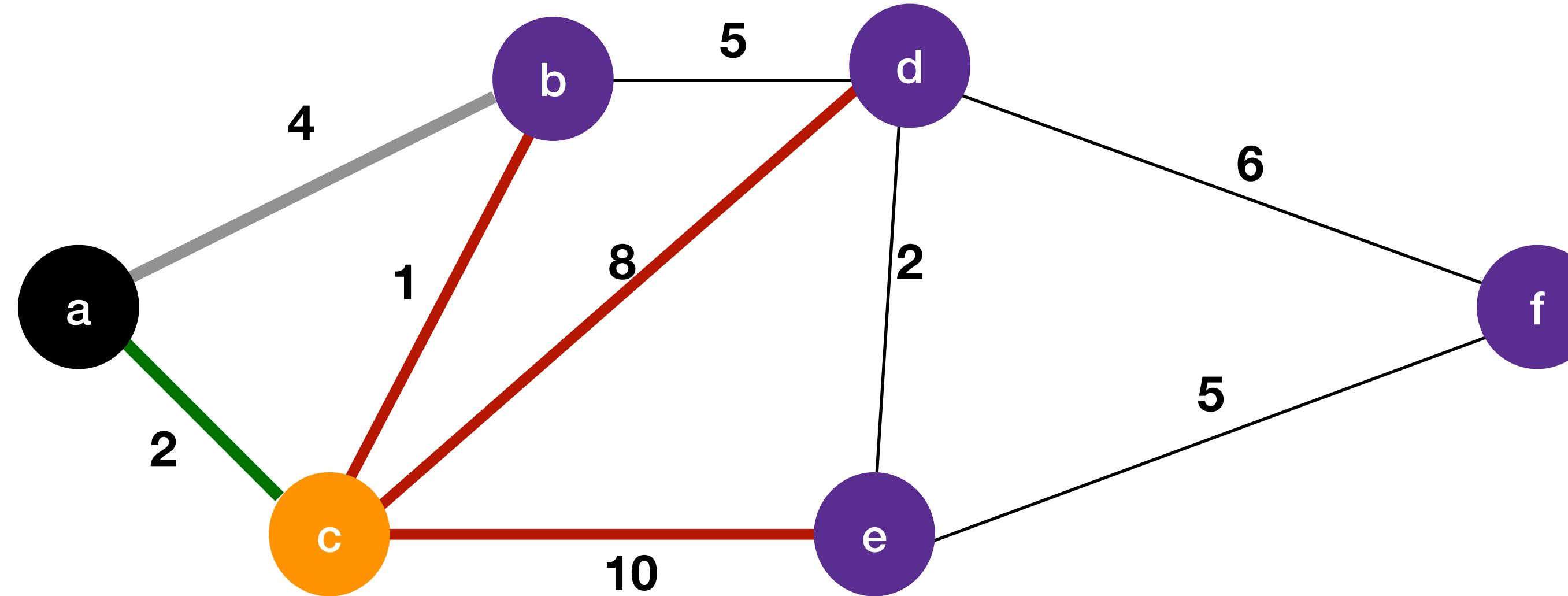
Algorismes sobre grafs



veïns de A
no visitats

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

Algorismes sobre grafs

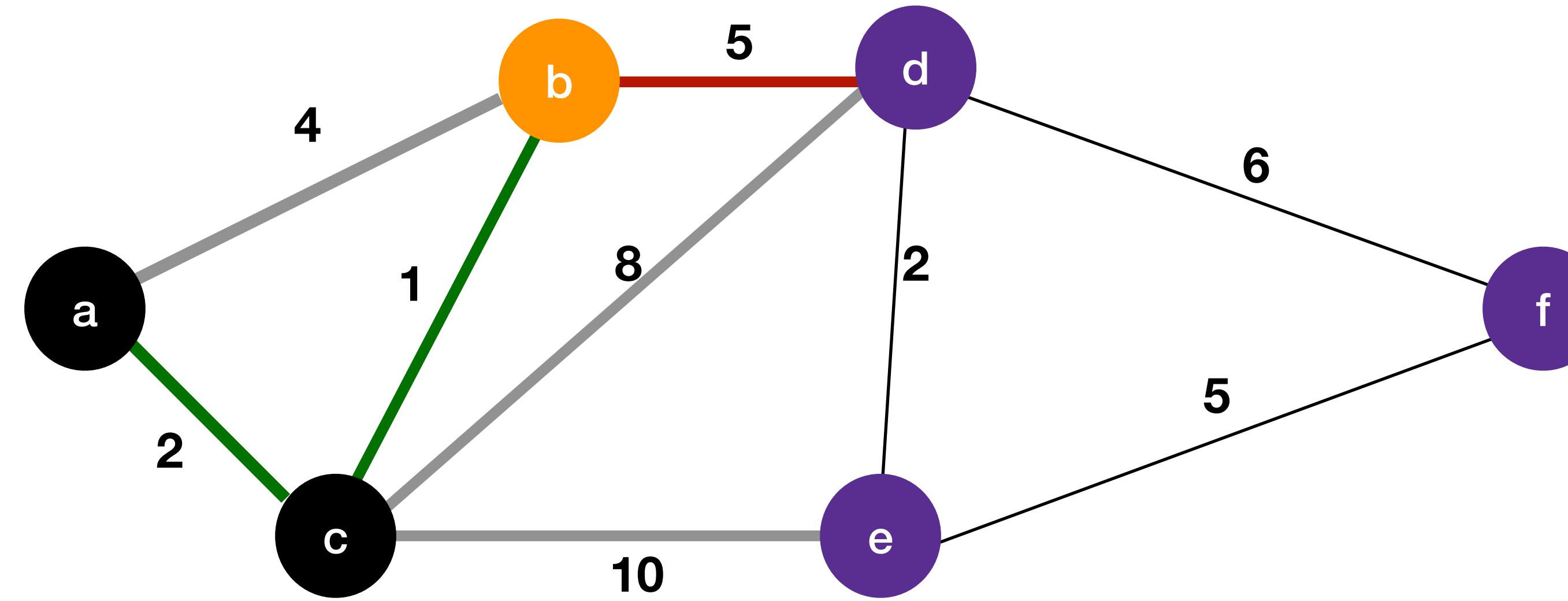


Node	Estat	Cami més curt des de A	Node previ
a	visitat	0	
b		$4 + 1 = 3$	A C
c	Node actual	2	A
d		$\infty + 8 = 10$	C
e		$\infty + 10 = 12$	C
f		∞	

veïns de C: b, e

no visitats: d, e

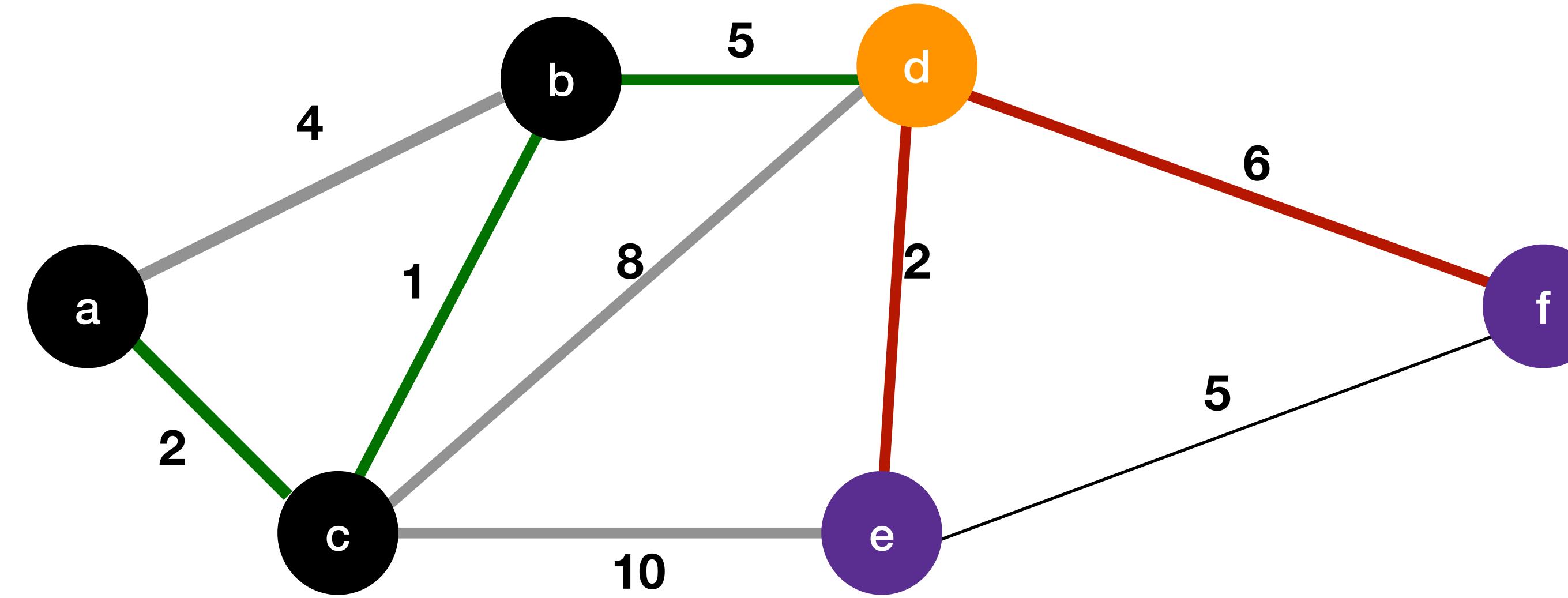
Algorismes sobre grafs



veïns de B
no visitats

Node	Estat	Cami més curt des de 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		∞	

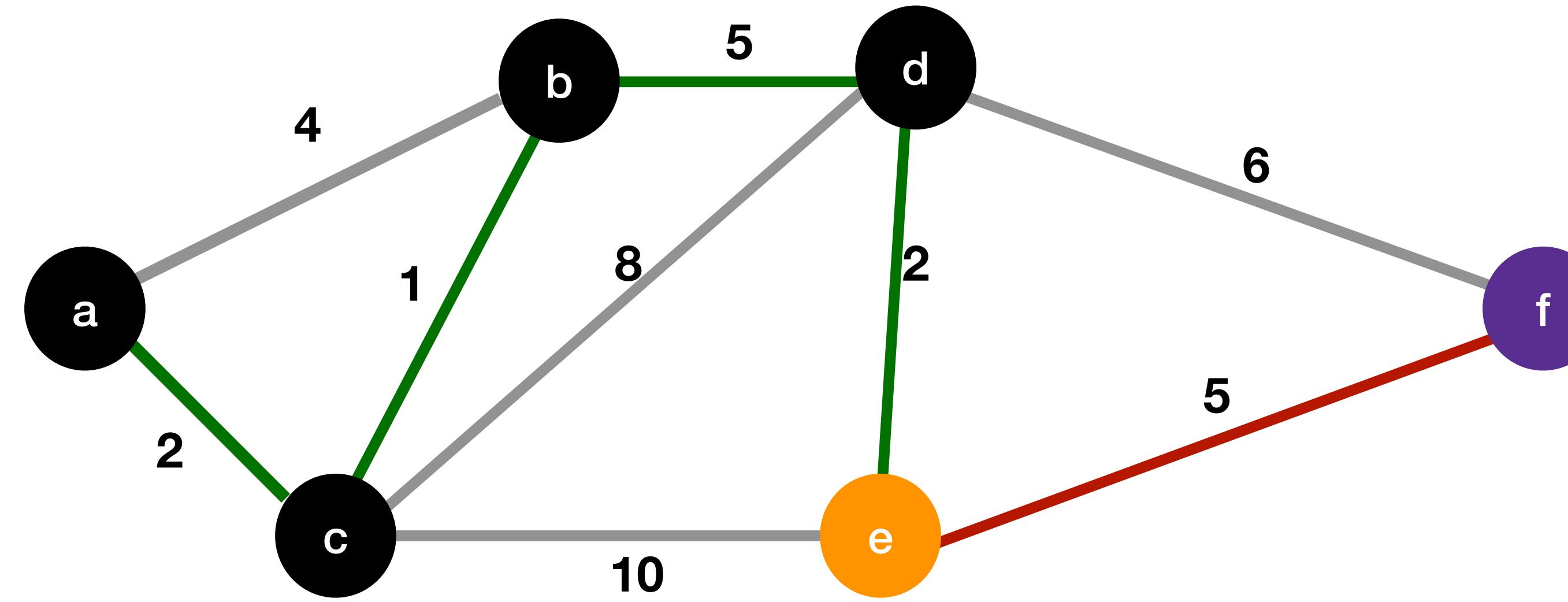
Algorismes sobre grafs



veïns de D
no visitats

Node	Estat	Cami més curt des de 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		$\infty - 8 + 6 = 14$	D

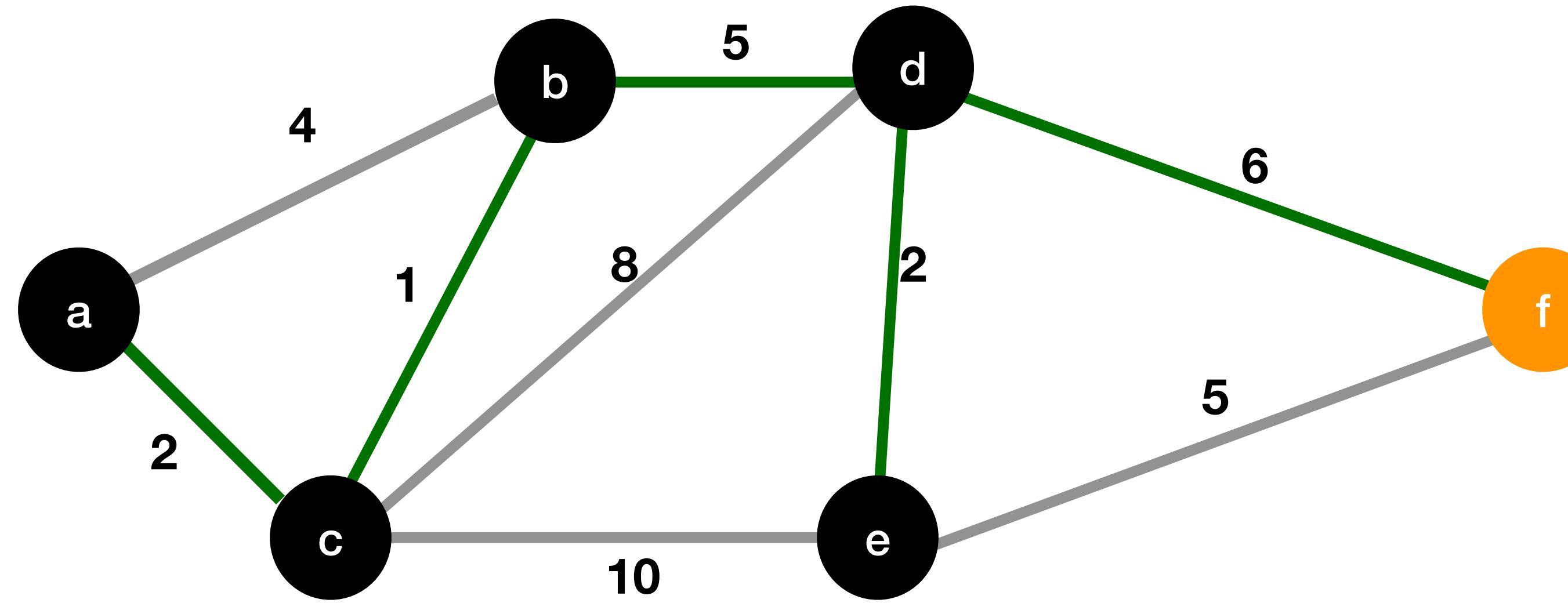
Algorismes sobre grafs



veïns de E
no visitats →

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

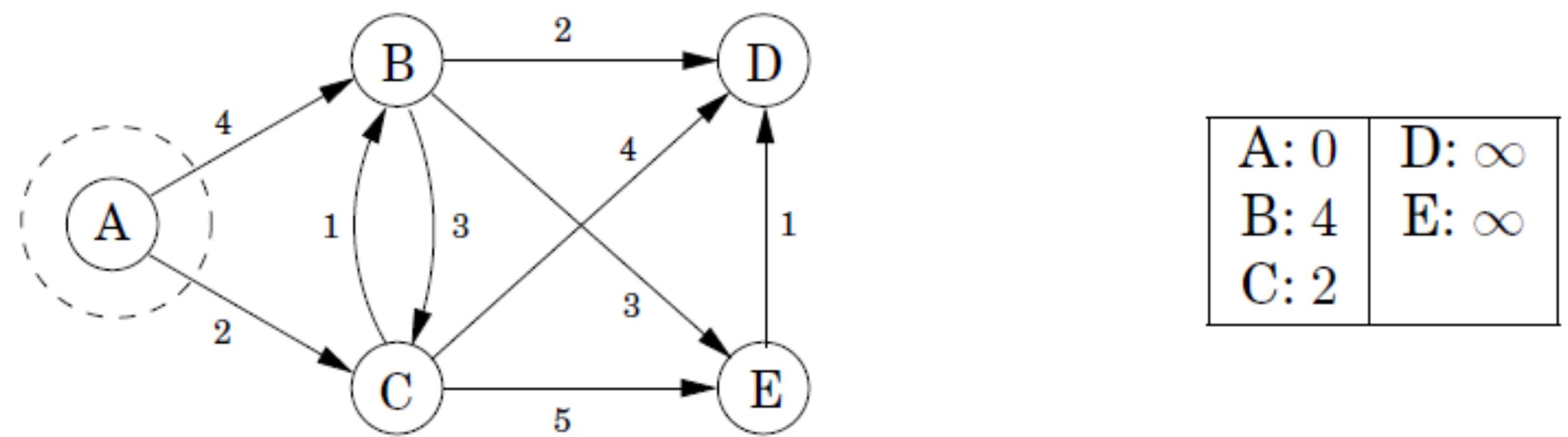
Algorismes sobre grafs



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

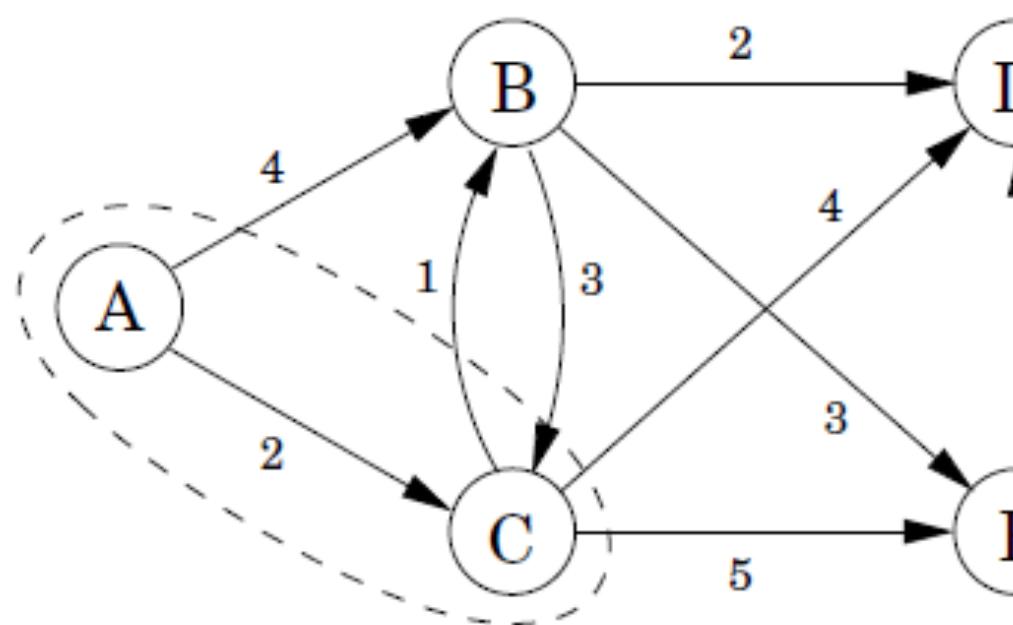
Dijkstra

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

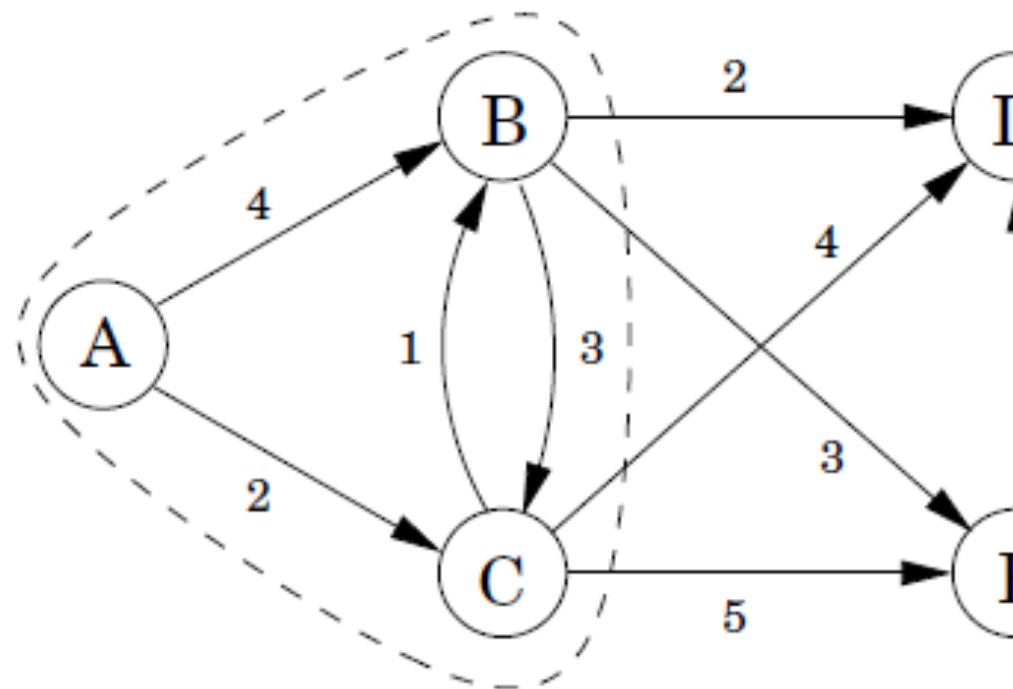


Dijkstra

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



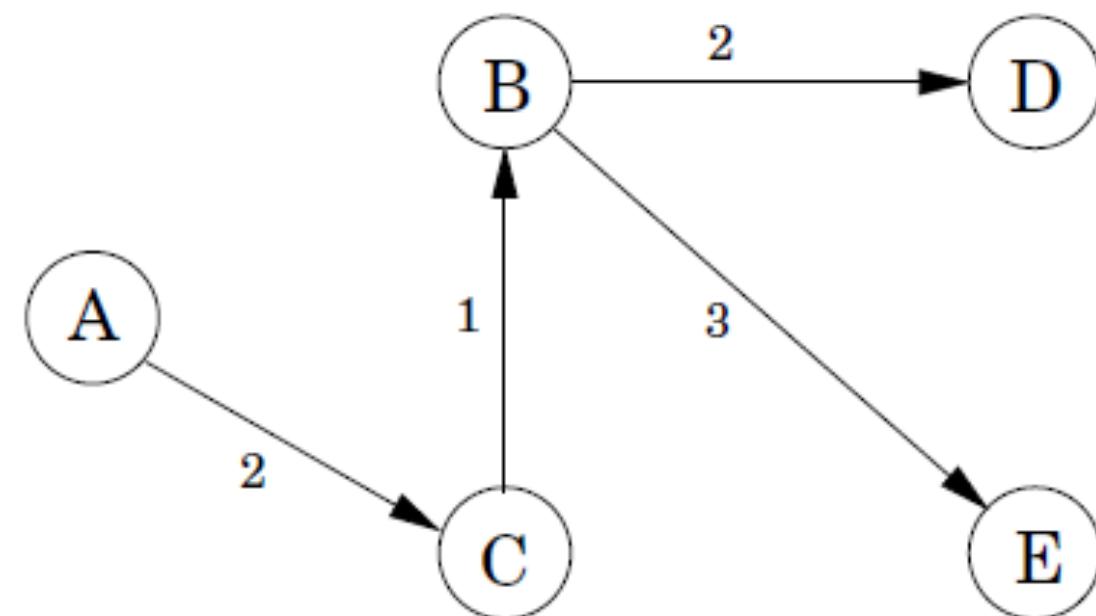
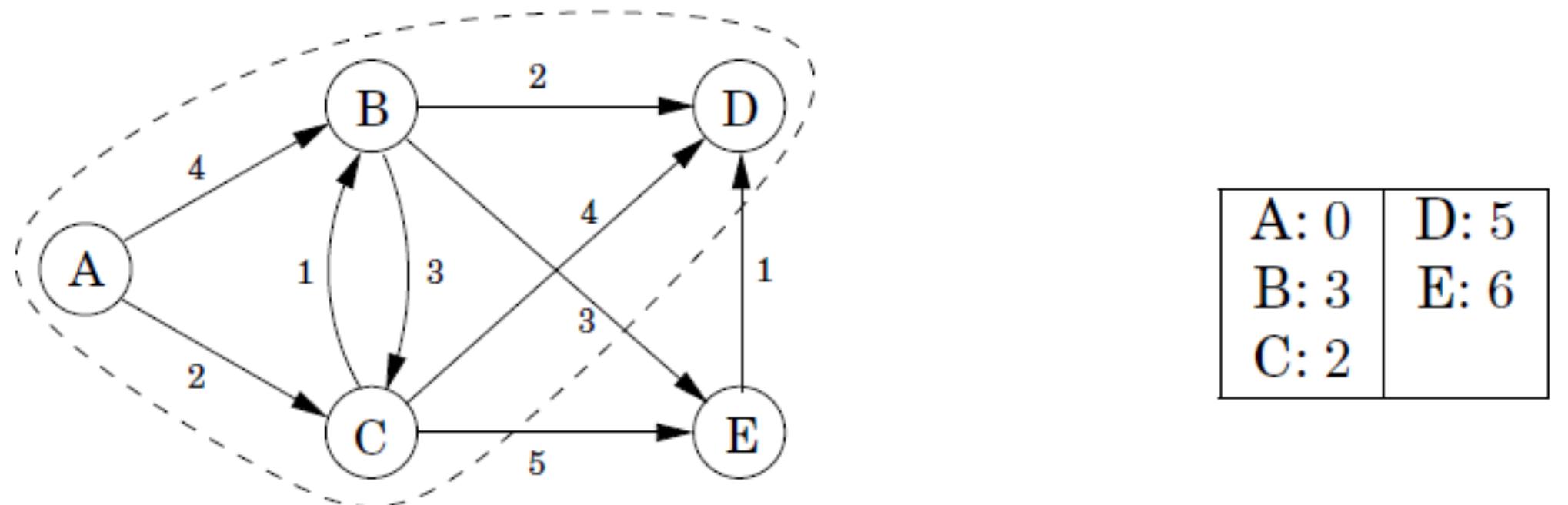
A: 0	D: 6
B: 3	E: 7
C: 2	



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

Dijkstra

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



Dijkstra

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

