Les graphes en Recherche Opérationnelle

Damien Leprovost

Laboratoire LIMICS
Inserm – UPMC – Paris 13
http://www.damien-leprovost.fr

Objectifs du cours

- **Comprendre l'intérêt de la recherche opérationnelle**

Objectifs du cours

- **Comprendre l'intérêt de la recherche opérationnelle**
- **Maîtriser les bases utiles de la théorie des graphes**

Objectifs du cours

- **Comprendre l'intérêt de la recherche opérationnelle**
- **Maîtriser les bases utiles de la théorie des graphes**
- **Savoir modéliser un problème abstrait en un algorithme concret**

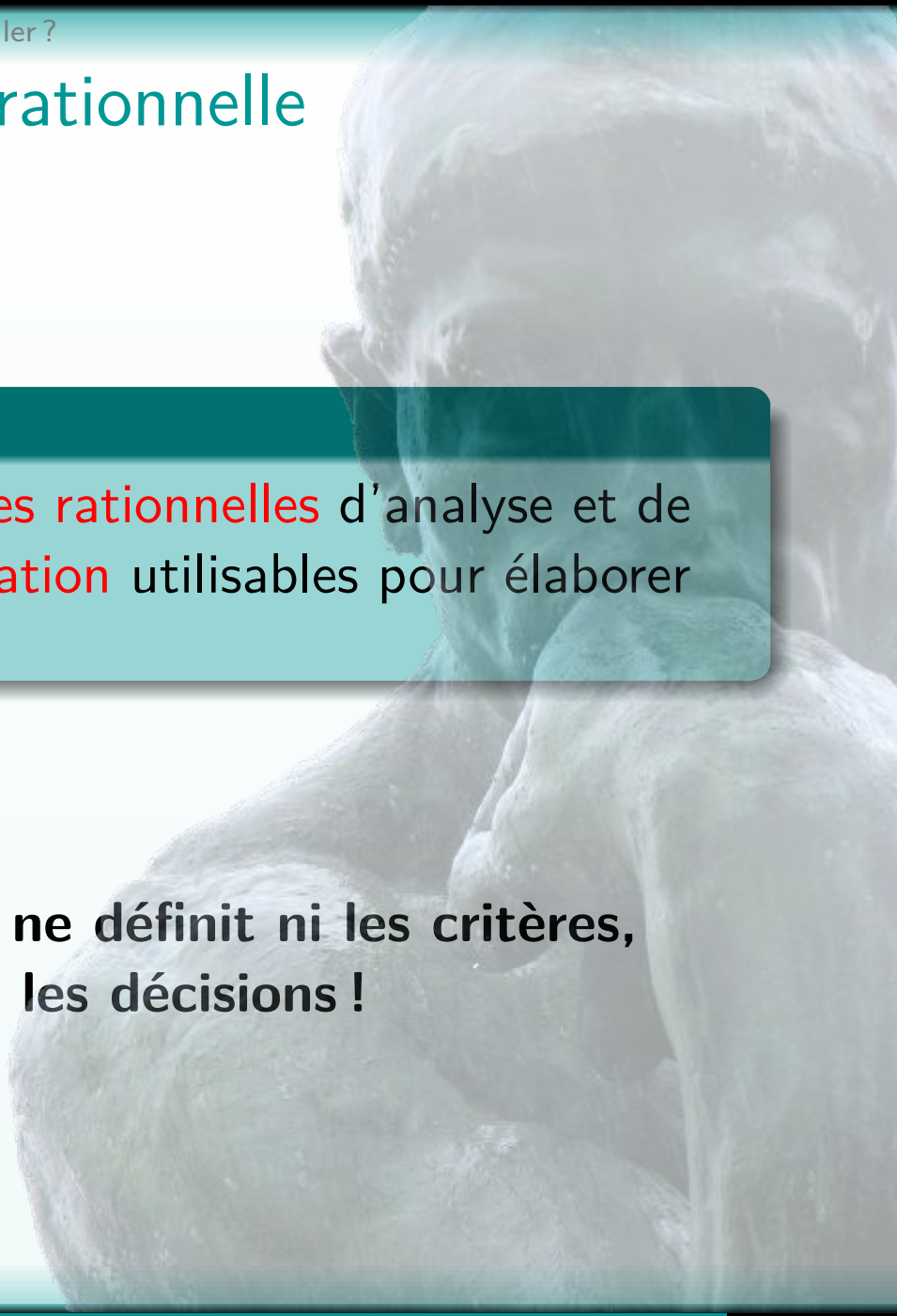
Objectifs du cours

- **Comprendre l'intérêt de la recherche opérationnelle**
- **Maîtriser les bases utiles de la théorie des graphes**
- **Savoir modéliser un problème abstrait en un algorithme concret**
- **. . . Et savoir le résoudre !**

Plan du cours

Plan du cours

Définition de la recherche opérationnelle



Définition

Ensemble de méthodes et techniques rationnelles d'analyse et de synthèse des phénomènes d'organisation utilisables pour élaborer de meilleures décisions.

La recherche opérationnelle ne définit ni les critères, ni les objectifs, ni les décisions !

Origines

- Seconde Guerre mondiale (état-major britannique)
  - Patrick Blackett, physicien (1940)
  - Application aux *opérations* militaires : positionnement des radars, ravitaillements

- En réalité beaucoup plus ancienne
  - *Espérance mathématique*, de Blaise Pascal et Pierre de Fermat (1654)
  - *Décision dans l'incertain* de Jacques Bernoulli (1713)

Domaine d'applications

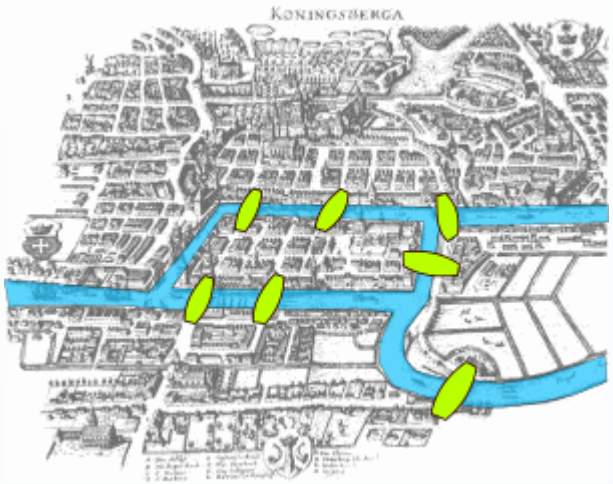
- Les domaines où le sens commun est mis en défaut.
- Notamment :
  - les problèmes combinatoires ;
  - les domaines de l'aléatoire ;
  - les situations de concurrence.

Plan du cours

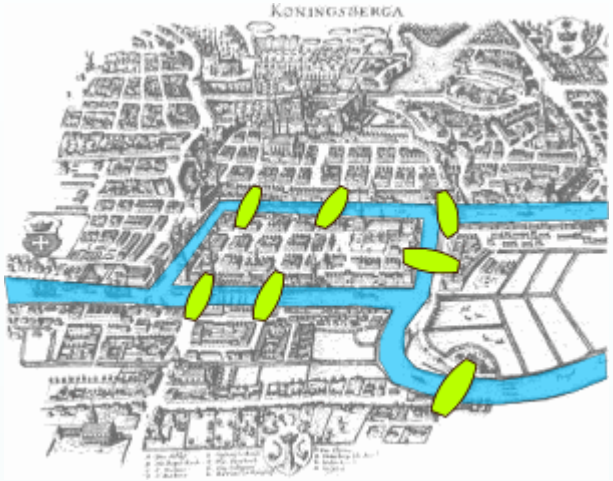
Définition des graphes

- Un graphe est tout d'abord :
  - un ensemble d'éléments ;
  - un ensemble de relations entre ces éléments.
- Deux grandes familles :
  - les graphes non-orientés ;
  - les graphes orientés.
- Nombreuses conceptualisations et modélisations possibles

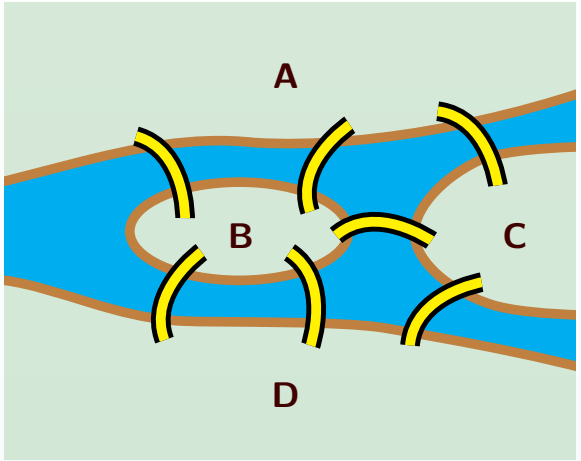
Problème des sept ponts de Königsberg (Euler, 1735)



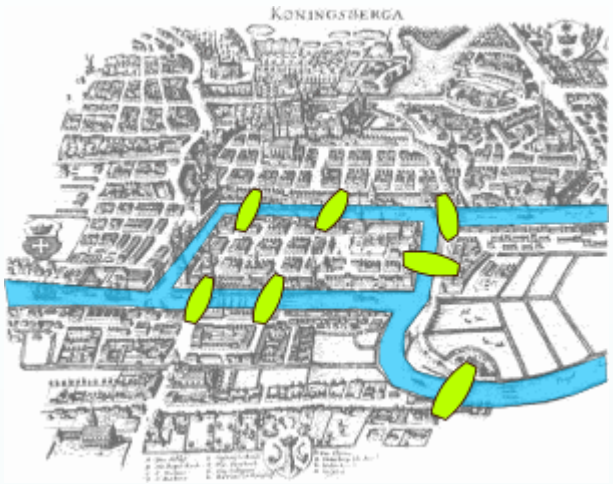
Problème des sept ponts de Königsberg (Euler, 1735)



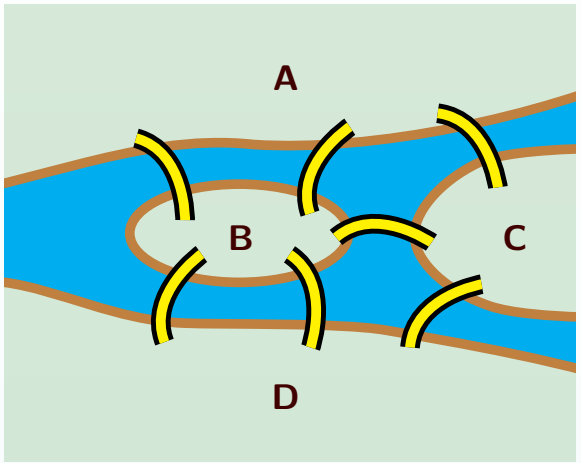
$\Leftrightarrow$



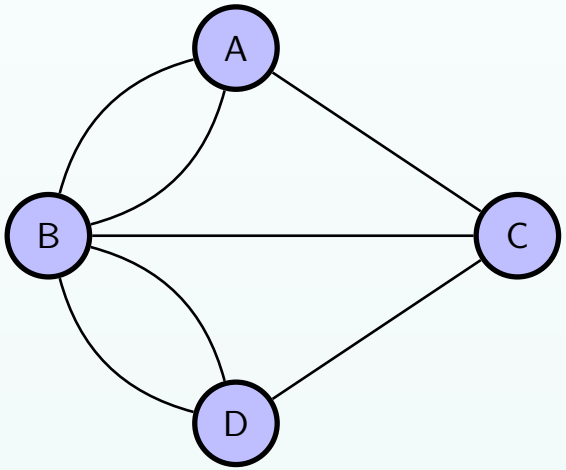
Problème des sept ponts de Königsberg (Euler, 1735)



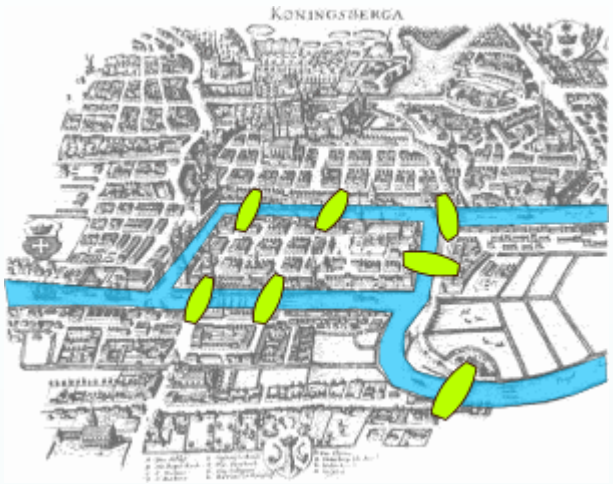
$\Leftrightarrow$



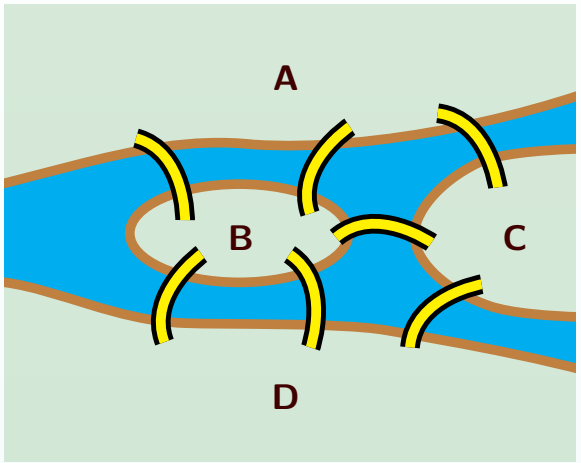
Représentation sous forme de graphe :



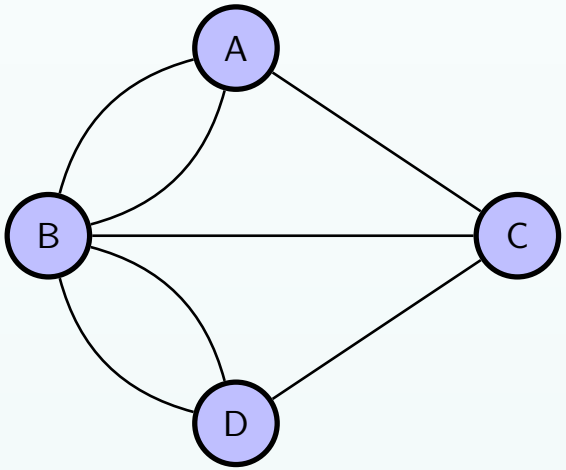
Problème des sept ponts de Königsberg (Euler, 1735)



$\Leftrightarrow$



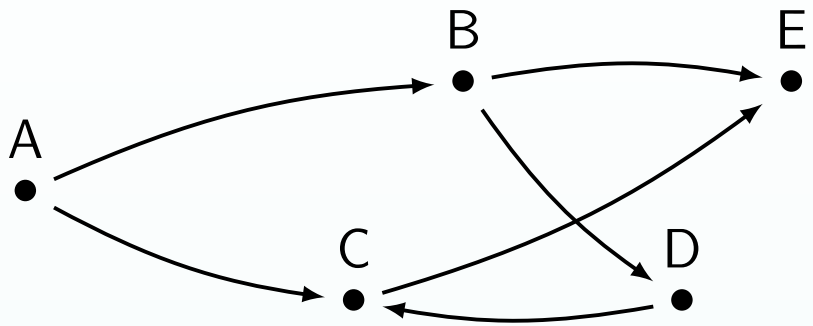
Représentation sous forme de graphe :



$\Leftrightarrow$

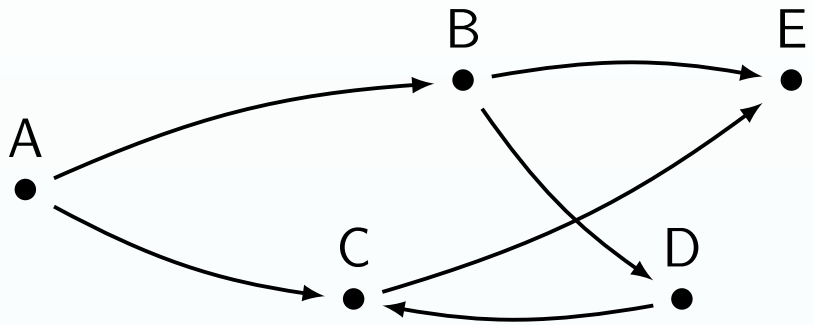
$$G = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & - & 2 & 1 & - \\ B & 2 & - & 1 & 2 \\ C & 1 & 1 & - & 1 \\ D & - & 2 & 1 & - \end{array}$$

Vocabulaire des graphes orientés



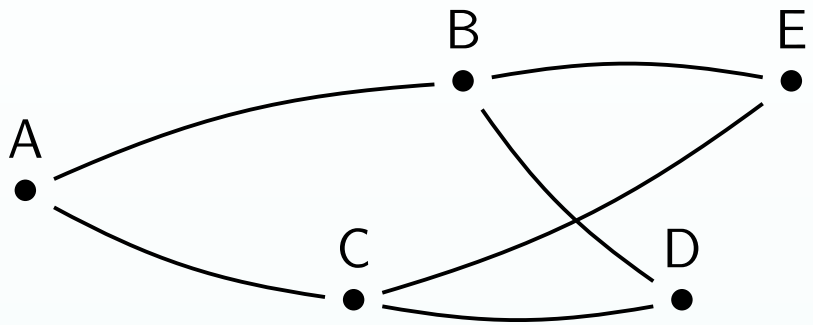
- Ensemble de sommets $X = \{A, B, C, D, E\}$
- Arcs : Couples orientés de sommets, sous-ensembles de X
  - $U \subset X \times X$, relation binaire sur X
  - $U = \{(A, B), (A, C), (B, D), (B, E), (C, E), (DC)\}$
  - $G = (X, U)$ est une notation possible de G

Vocabulaire des graphes orientés



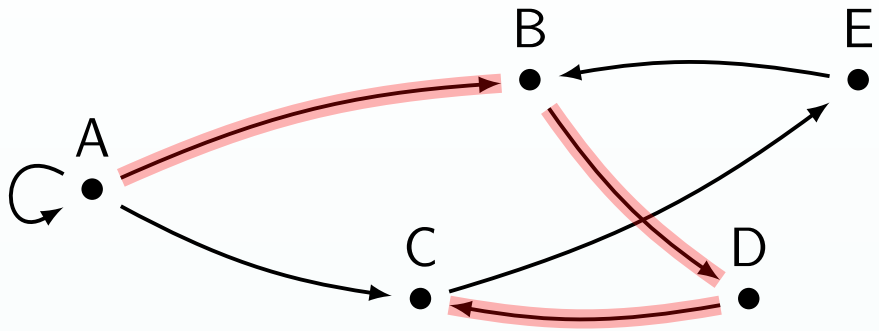
- Application Γ^+, nommée application successeur.
  - $\Gamma^+(A) = \{B, C\}$; A est *entrée du graphe*
  - $\Gamma^+(E) = \emptyset$; E est *sortie du graphe*
  - $G = (X, \Gamma^+)$ et $G = (X, \Gamma^-)$ sont deux écritures possibles de G

Vocabulaire des graphes non-orientés



- Couples non-orientés de sommets nommés arrêtes.
  - $[A, B] \Leftrightarrow [B, A]$
- À tout graphe orienté correspond un unique graphe non-orienté.
  - « Désorientation » :
    $U = \{(A, C), (B, D), (D, B)\} \Rightarrow U = \{[A, C], [B, D]\}$

Vocabulaire des graphes

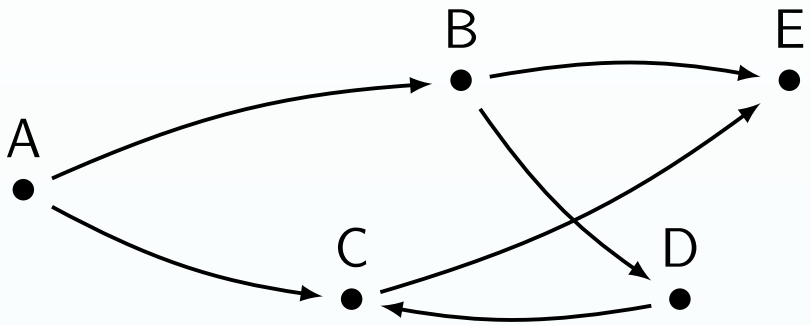


- Chemin : séquence d'arcs consécutifs
- Circuit : chemin se fermant sur lui-même
  - Le circuit est *simple* si les arcs sont uniques
  - Le circuit est *élémentaire* si les sommets sont uniques
  - Un circuit de longueur 1 est une boucle
- Chaîne : séquence d'arrêtes consécutives
  - Un graphe est **connexe** s'il existe au moins une chaîne entre tous les sommets
- Cycle : chaîne se fermant sur elle-même

Vocabulaire des graphes

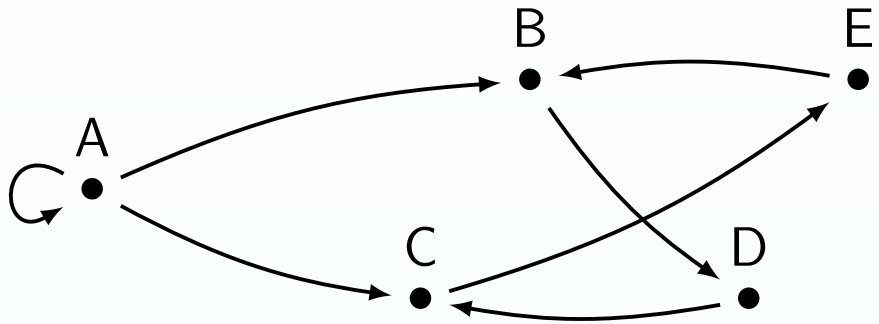
- Cycles remarquables :
  - Un cycle est dit eulérien s'il emprunte exactement une fois chaque arrête
    - Il s'agit d'un *cycle simple* sur l'ensemble du graphe
  - Un cycle est dit hamiltonien s'il emprunte exactement une fois chaque sommet
    - Il s'agit d'un *cycle élémentaire* sur l'ensemble du graphe
    - Nécessairement un *cycle simple*, mais pas nécessairement eulérien
    - Problème complexe !

Vocabulaire des graphes



- Demi-degré extérieur de x : $d_x^+ = \text{card } \Gamma^+(x)$ (boucles exclues)
- Demi-degré intérieur de x : $d_x^- = \text{card } \Gamma^-(x)$ (boucles exclues)
- Degré de x : nombre d'arrêtes ayant une extrémité en x.

Représentations matricielles

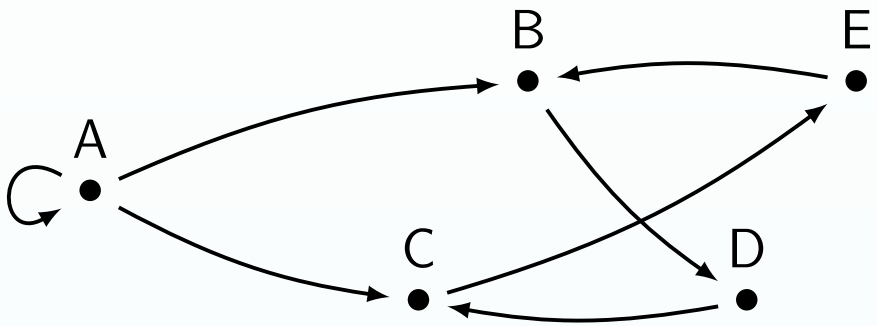


Définition d'une matrice d'adjacence

$$M = \begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 1 & 1 & 1 & 0 & 0 \\ B & 0 & 0 & 0 & 1 & 0 \\ C & 0 & 0 & 0 & 0 & 1 \\ D & 0 & 0 & 1 & 0 & 0 \\ E & 0 & 1 & 0 & 0 & 0 \end{array} \Rightarrow \begin{array}{l} \Gamma^+(A) = \{A, B, C\} \\ \Gamma^+(B) = \{D\} \\ \Gamma^+(C) = \{E\} \\ \Gamma^+(D) = \{C\} \\ \Gamma^+(E) = \{B\} \end{array}$$

- Exemple d'utilisation : détermination des degrés

Représentations matricielles



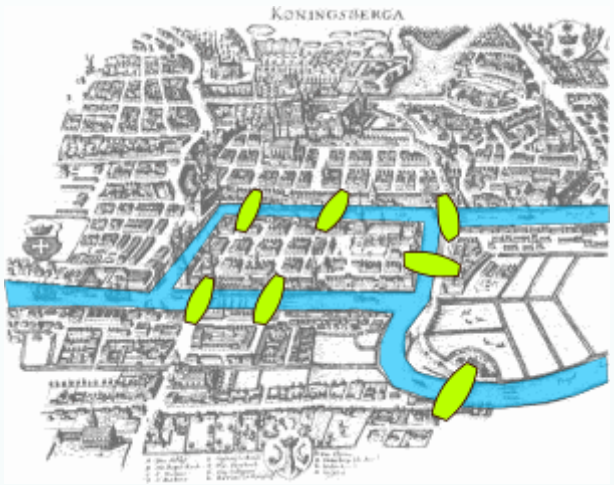
- Autre exemple d'utilisation : $M^k =$ cardinaux des chemins uniques de longueur k

$$M = \begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 1 & 1 & 1 & 0 & 0 \\ B & 0 & 0 & 0 & 1 & 0 \\ C & 0 & 0 & 0 & 0 & 1 \\ D & 0 & 0 & 1 & 0 & 0 \\ E & 0 & 1 & 0 & 0 & 0 \end{array} \qquad M^3 = \begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 1 & 2 & 2 & 1 & 1 \\ B & 0 & 0 & 0 & 0 & 1 \\ C & 0 & 0 & 0 & 1 & 0 \\ D & 0 & 1 & 0 & 0 & 0 \\ E & 0 & 0 & 1 & 0 & 0 \end{array}$$

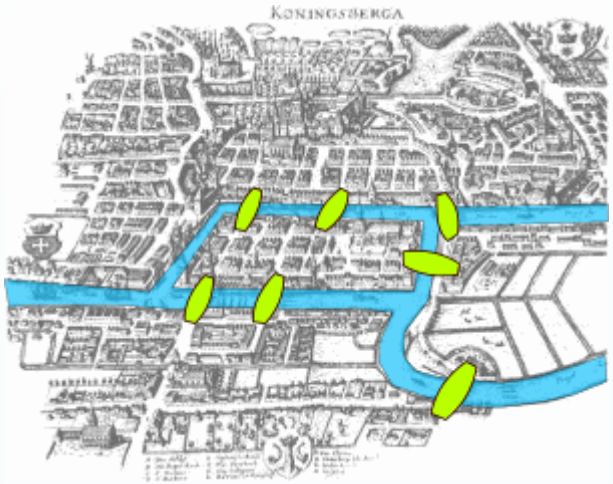
Utilité en recherche opérationnelle

- Représenter tout sorte de situation dans les phénomènes d'organisation
- Modéliser, par exemple :
  - Réseaux de transports
    - Utilisation de la loi de Kirchhoff (loi des nœuds)
  - Systèmes de relations
    - Utilisation de la loi de transitivité
    - Problématiques d'ordonnancement
  - Systèmes d'états et de transitions
    - Chaînes et processus de Markov
    - Réseaux de Petri

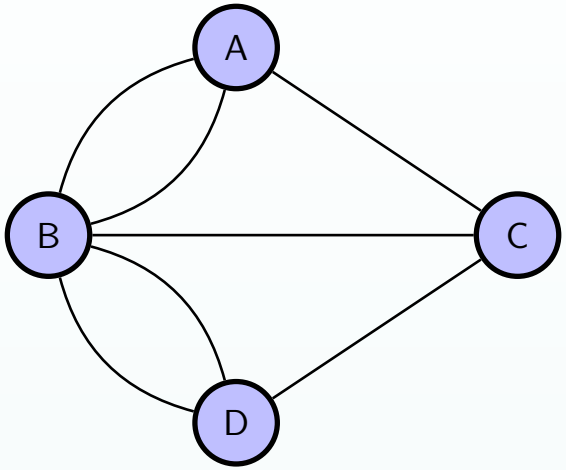
Test : Êtes-vous Euler ?



Test : Êtes-vous Euler ?

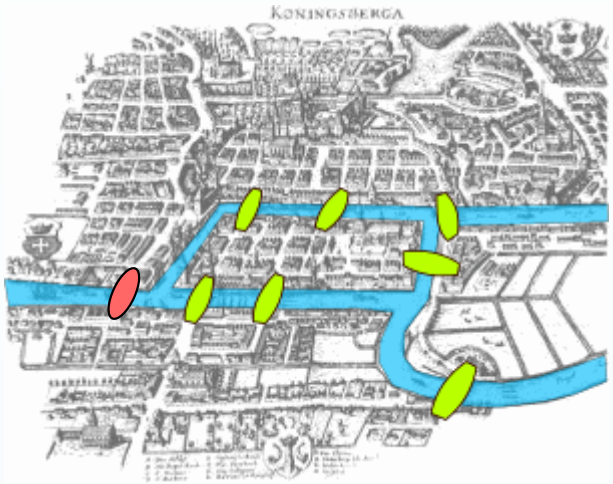


$\Rightarrow$

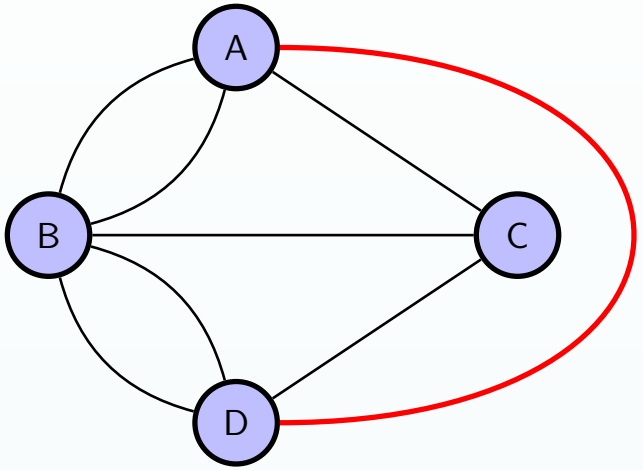


$$G = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & - & 2 & 1 & - \\ B & 2 & - & 1 & 2 \\ C & 1 & 1 & - & 1 \\ D & - & 2 & 1 & - \end{array} \Rightarrow \begin{array}{l} \Gamma(A) = 3 \\ \Gamma(B) = 5 \\ \Gamma(C) = 3 \\ \Gamma(D) = 3 \end{array}$$

Test : Êtes-vous Euler ?

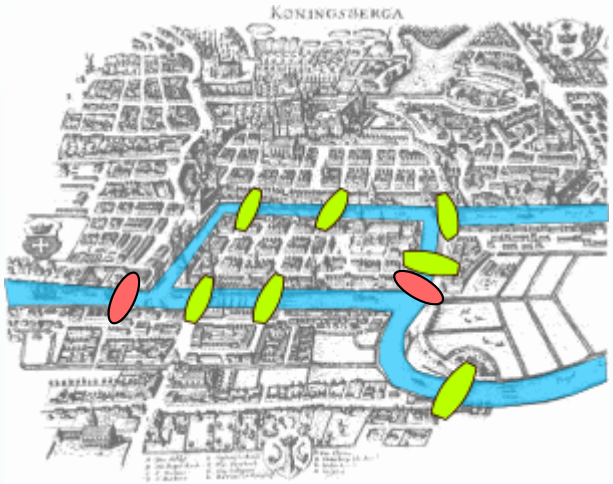


$\Rightarrow$

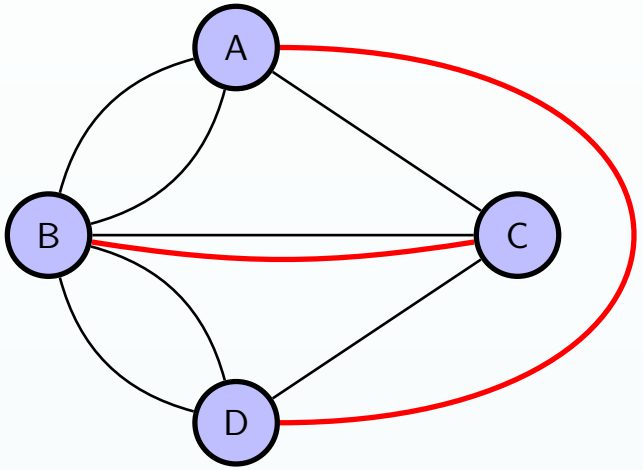


$$G = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & - & 2 & 1 & - \\ B & 2 & - & 1 & 2 \\ C & 1 & 1 & - & 1 \\ D & - & 2 & 1 & - \end{array} \Rightarrow \begin{array}{l} \Gamma(A) = 4 \\ \Gamma(B) = 5 \\ \Gamma(C) = 3 \\ \Gamma(D) = 4 \end{array}$$

Test : Êtes-vous Euler ?



$\Rightarrow$



$$G = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & - & 2 & 1 & - \\ B & 2 & - & 1 & 2 \\ C & 1 & 1 & - & 1 \\ D & - & 2 & 1 & - \end{array} \Rightarrow \begin{array}{l} \Gamma(A) = 4 \\ \Gamma(B) = 6 \\ \Gamma(C) = 4 \\ \Gamma(D) = 4 \end{array}$$

Plan du cours

2 L'algorithmique des graphes

- Définition d'un algorithme
- Complexité
- Premiers algorithmes : parcours de graphe
- La programmation dynamique

Définition d'un algorithme[1]

Définition

Un algorithme est une suite finie et non ambigüe d'opérations ou d'instructions permettant de résoudre un problème.

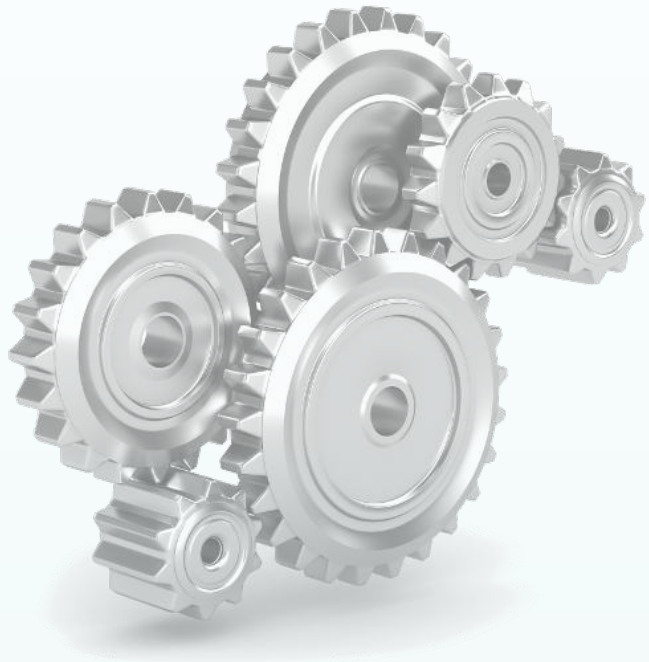
- Propriétés de Knuth :
  - Les entrées
  - Les sorties
  - La finitude
  - La définition
  - Le rendement

1. Du nom du mathématicien perse *Al-Khwârizmî* (∼ 780 – 850)

Plan du cours

Complexité

- Évaluer et comparer l'efficacité des algorithmes
- Deux critères principaux :
  - Temps de calcul
  - Espace mémoire
- Le temps de calcul est toujours limité !



La notation O

- O classe de fonction « bornant » supérieurement.
- $\exists c > 0, \exists n_0$ tels que $\forall n \geq n_0, g(n) \leq c \cdot f(n) \Leftrightarrow g \in O(f)$
- $O(1)$ ensemble des fonctions majorées par une constante partir d'un certain rang
- $O(c \cdot f) = O(f)$: La complexité s'entend à une constante multiplicative près (comportement asymptotique)
  - $O(100000n^2) = O(0,01n^2) = 0(n^2)$
  - $\forall P(n) = a_0 + a_1 n + a_2 n^2 + \cdots + a_p n^p, P \in O(n^p)$

Temps d'exécution des fonctions usuelles [2]

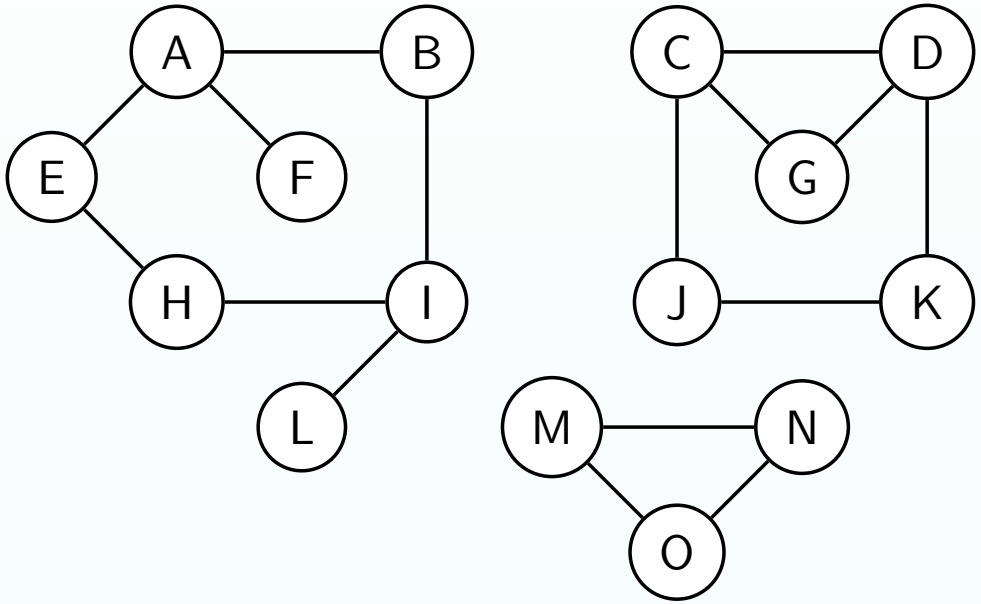
$f(n) =$	$n = 10$	$n = 100$	$n = 1000$	$n = 10^6$	$n = 10^9$
$\log n$	10^{-9} s	$2 \cdot 10^{-9}$ s	$3 \cdot 10^{-9}$ s	$6 \cdot 10^{-9}$ s	$9 \cdot 10^{-9}$ s
n	10^{-8} s	10^{-7} s	10^{-6} s	10^{-3} s	1 s
$n \log n$	10^{-8} s	$2 \cdot 10^{-7}$ s	$3 \cdot 10^{-6}$ s	$6 \cdot 10^{-3}$ s	9 s
n^2	10^{-7} s	10^{-5} s	10^{-3} s	1000 s	32 ans
n^3	10^{-6} s	10^{-3} s	1 s	32 ans	$3 \cdot 10^4$ Ma
2^n	10^{-6} s	$3 \cdot 10^8$ Ma	10^{273} Ma	–	–

- $O(1) \subset O(\log\ n) \subset O(\sqrt{n}) \subset O(n) \subset O(n\ \log\ n) \subset O(n^2) \subset O(2^n) \subset O(e^n) \subset O(n^n)$

1. Sur la base d'un milliard d'opérations par seconde (1Ghz)

Plan du cours

Parcourir un graphe



1: initialement tous les sommets sont non-marqués

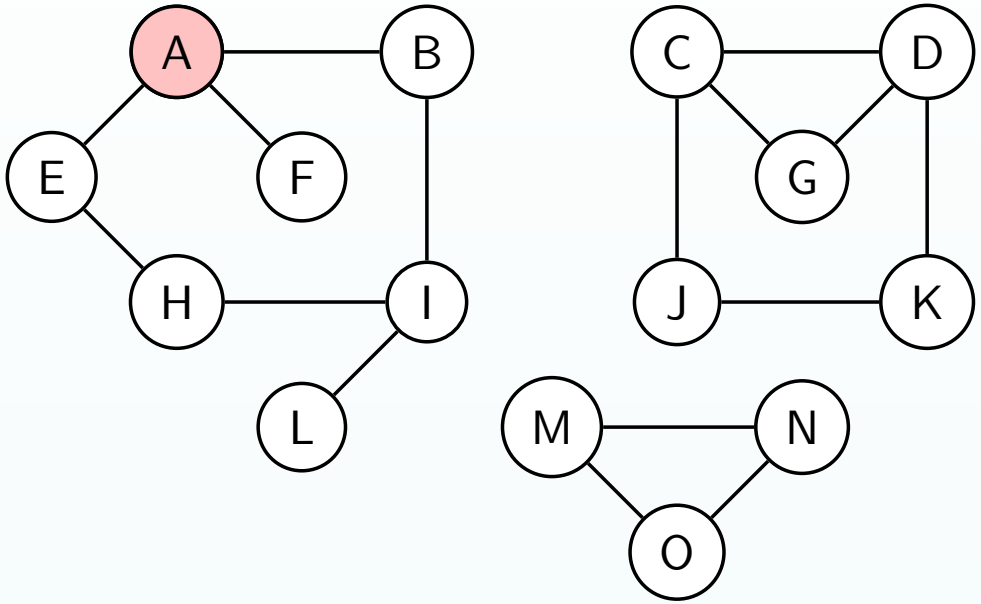
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre A

1: initialement tous les sommets sont non-marqués

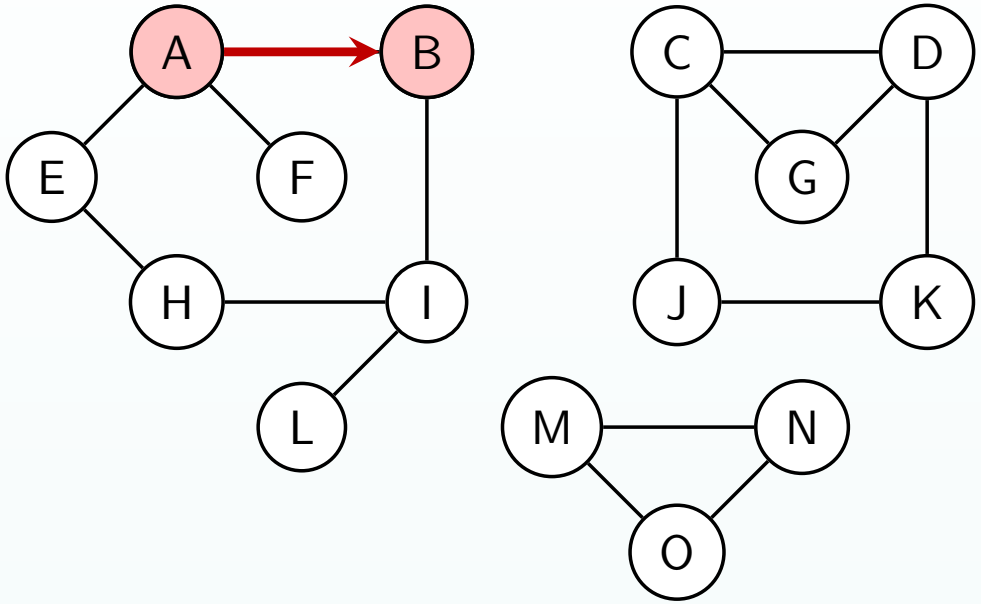
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre B

1: initialement tous les sommets sont non-marqués

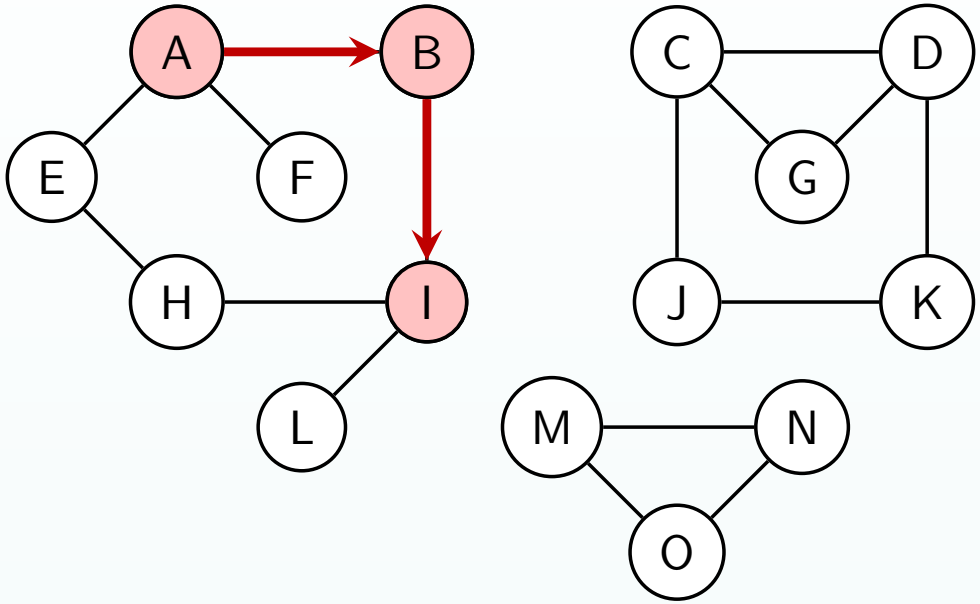
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre I

1: initialement tous les sommets sont non-marqués

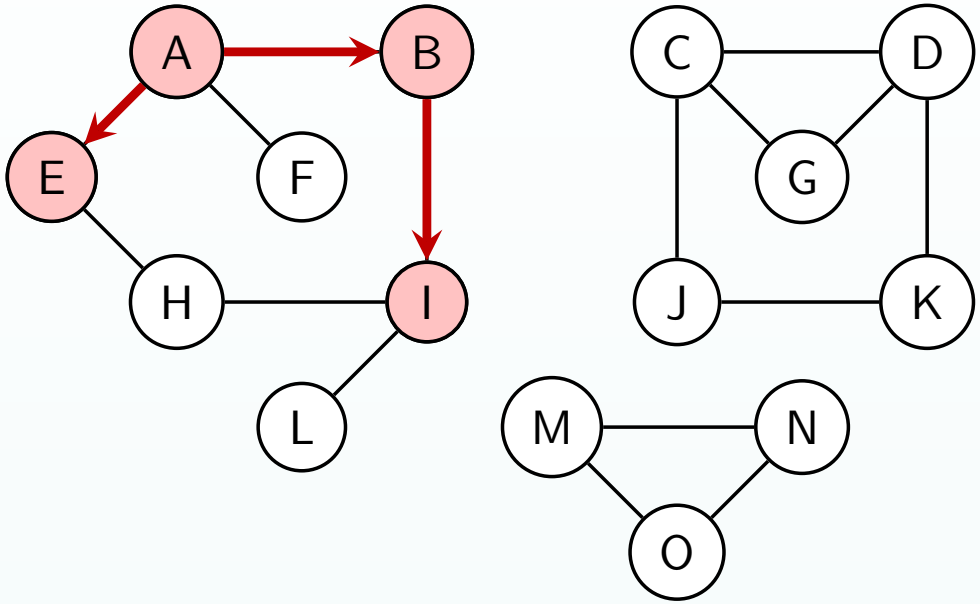
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre E

1: initialement tous les sommets sont non-marqués

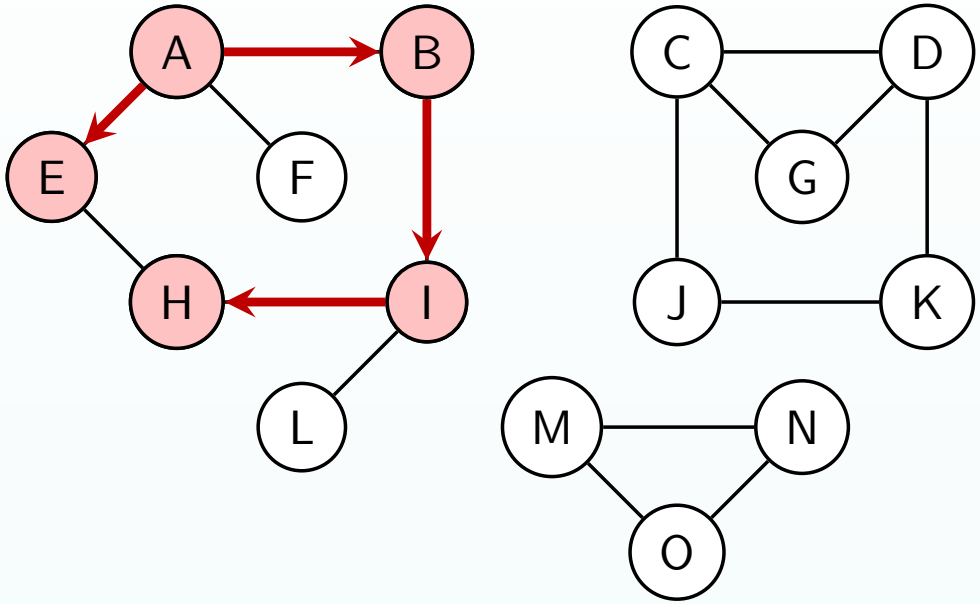
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre H

1: initialement tous les sommets sont non-marqués

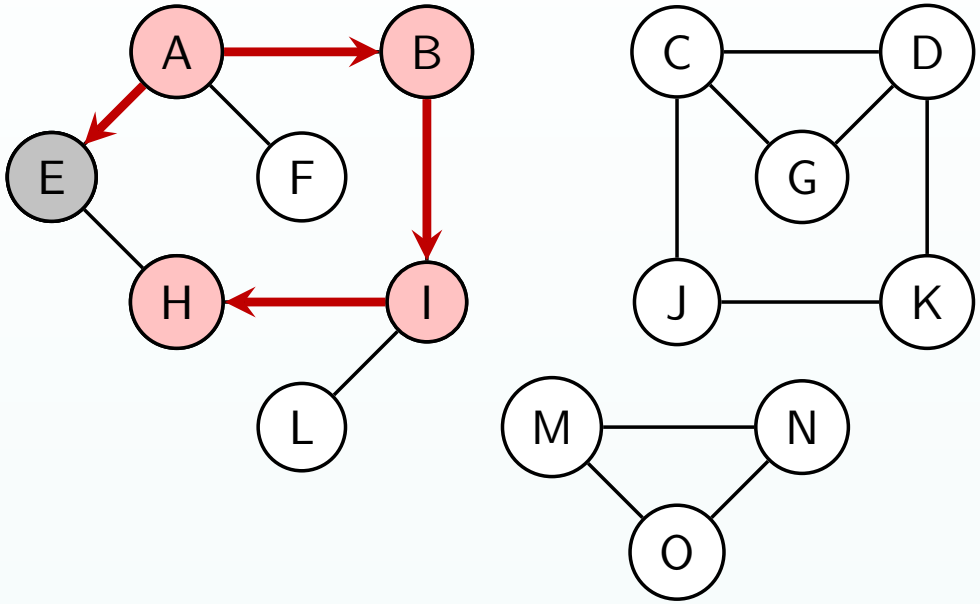
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ferme E

1: initialement tous les sommets sont non-marqués

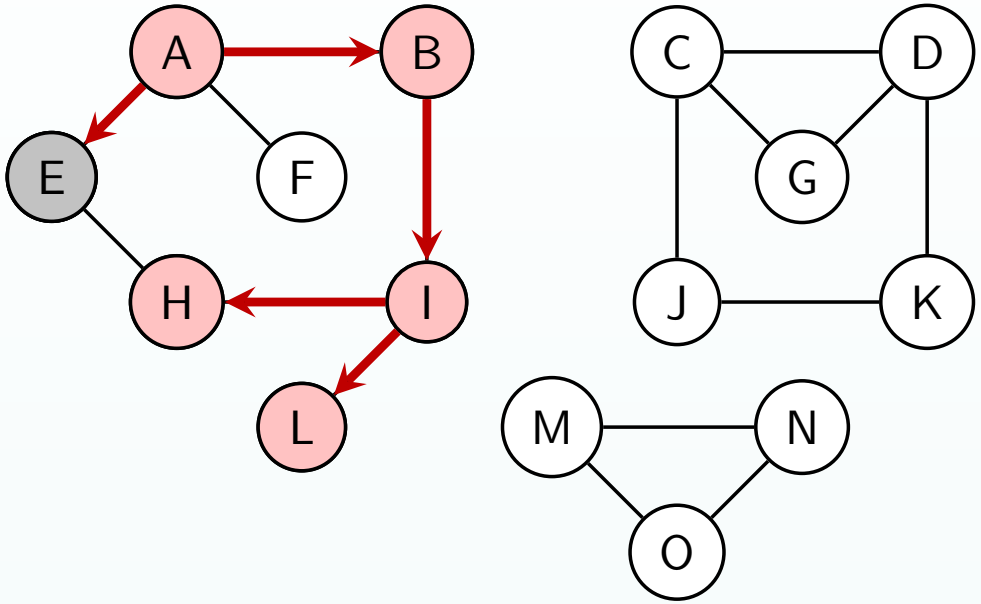
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre L

1: initialement tous les sommets sont non-marqués

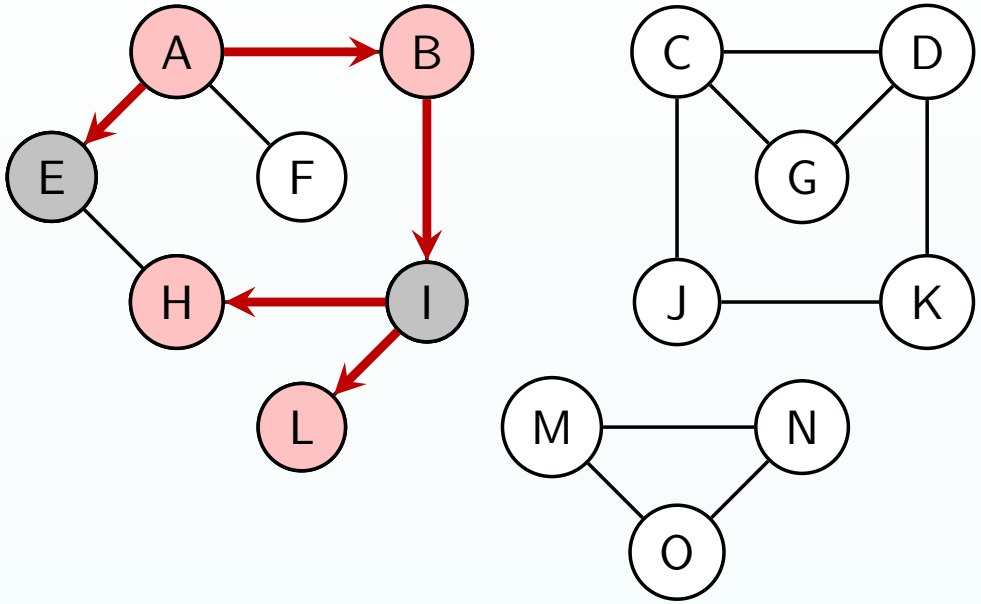
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ferme I

1: initialement tous les sommets sont non-marqués

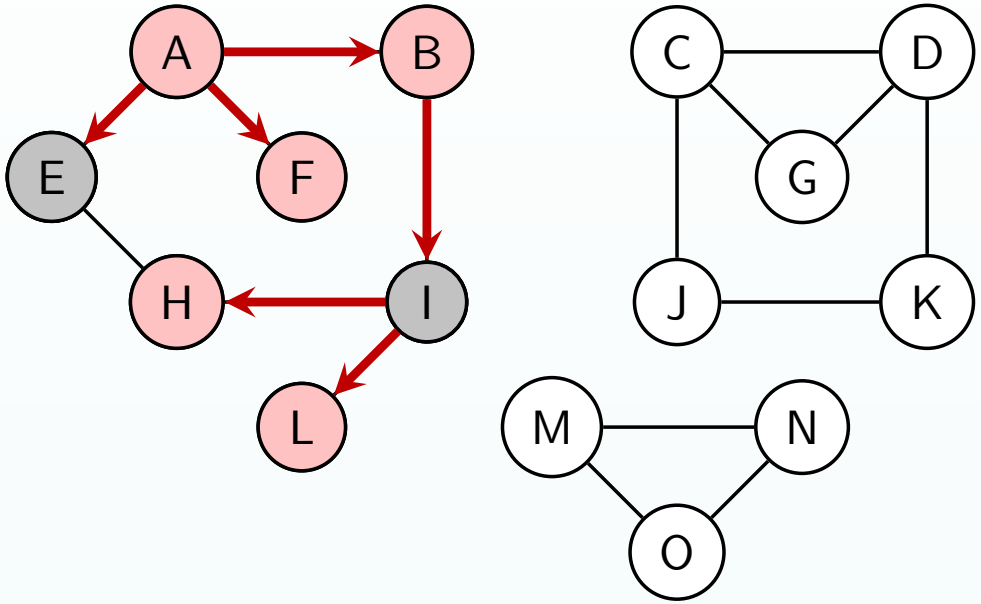
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre F

1: initialement tous les sommets sont non-marqués

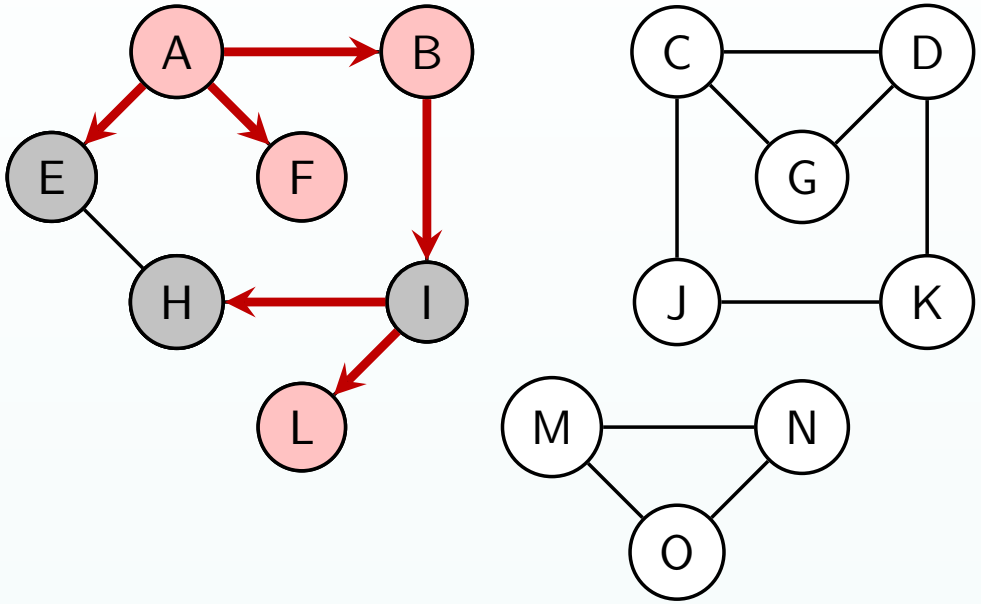
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ferme H

1: initialement tous les sommets sont non-marqués

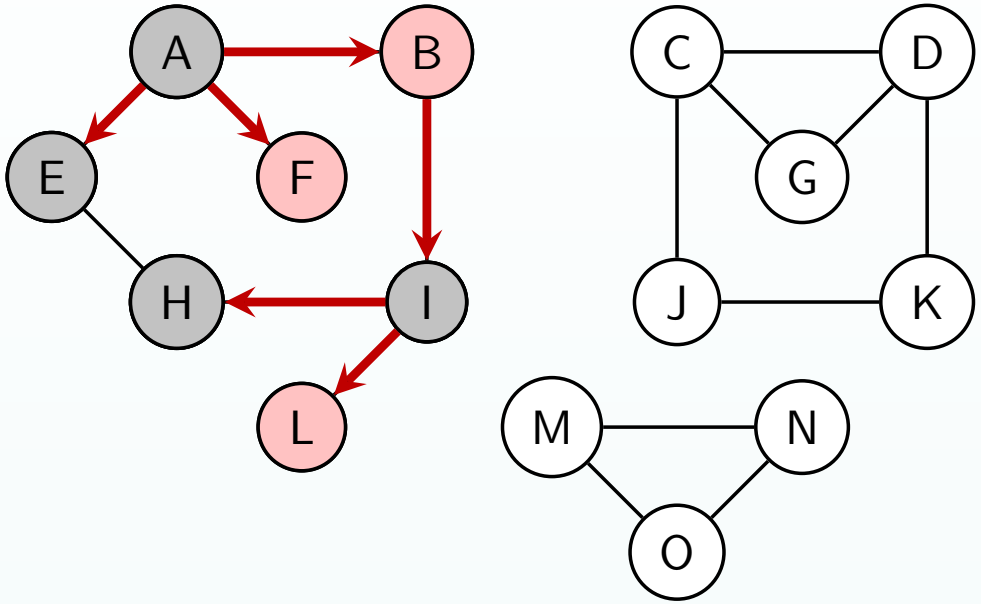
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ferme A

1: initialement tous les sommets sont non-marqués

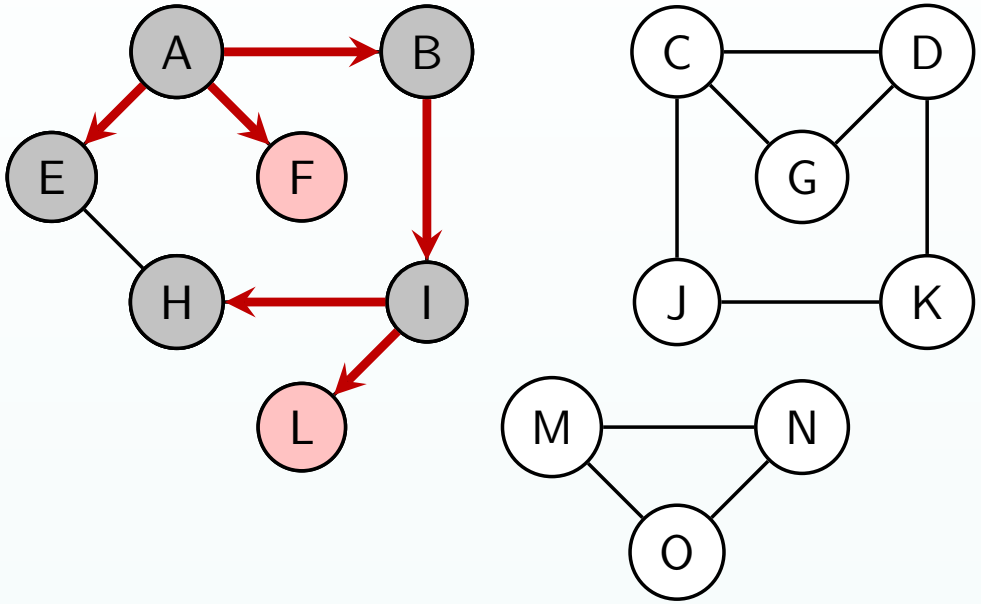
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ferme B

1: initialement tous les sommets sont non-marqués

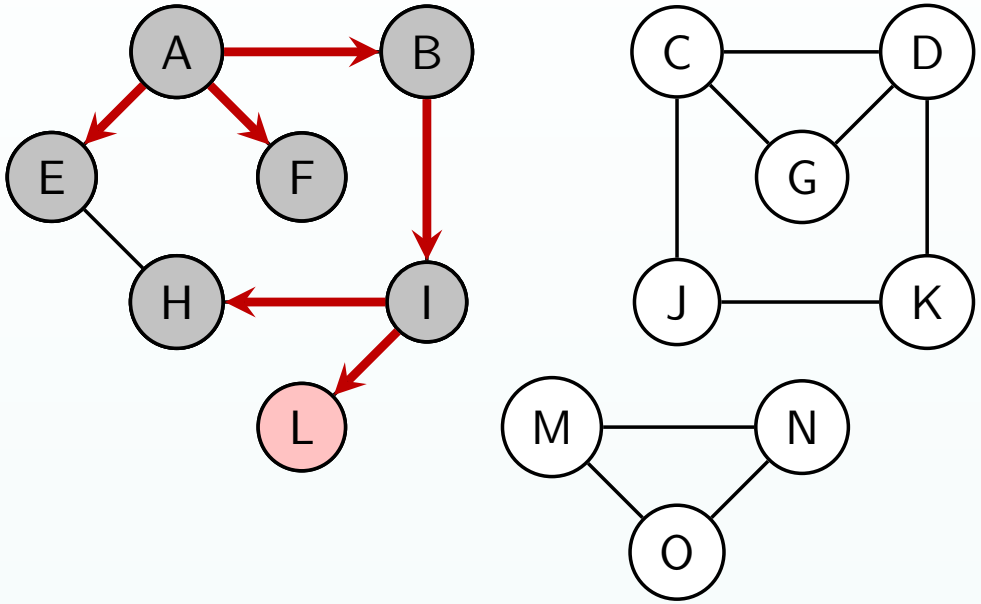
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ferme F

1: initialement tous les sommets sont non-marqués

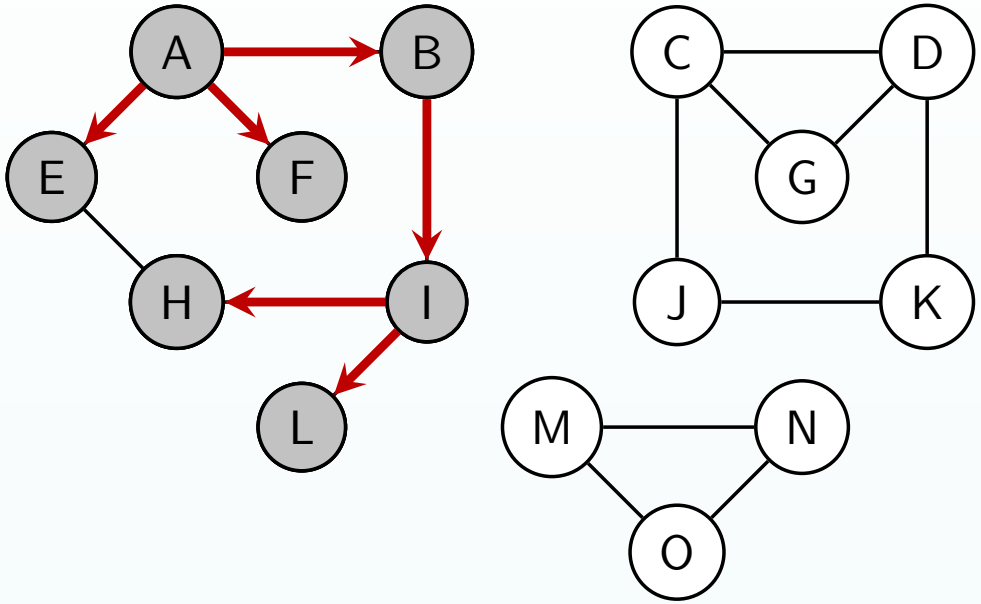
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ferme L

1: initialement tous les sommets sont non-marqués

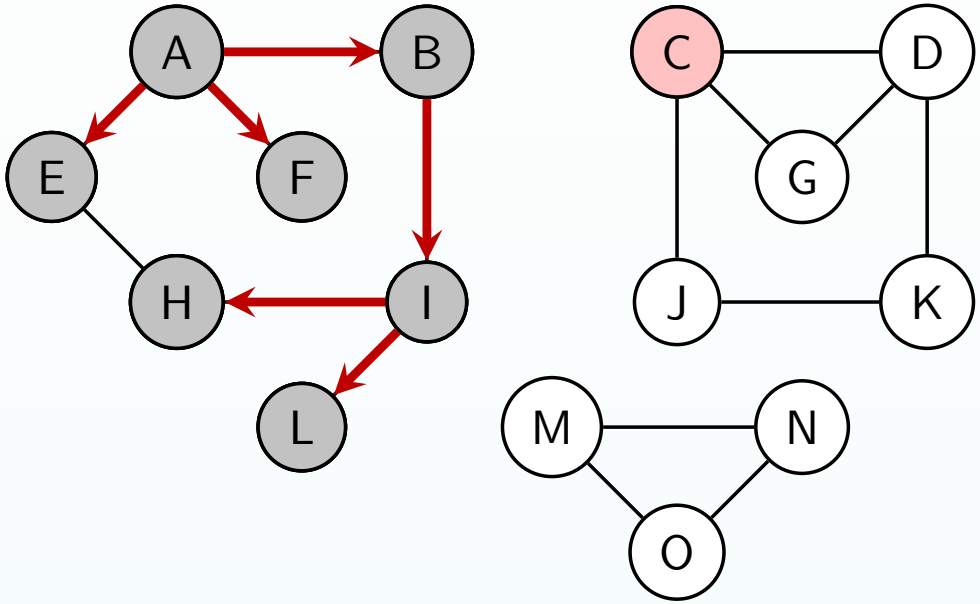
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre C

1: initialement tous les sommets sont non-marqués

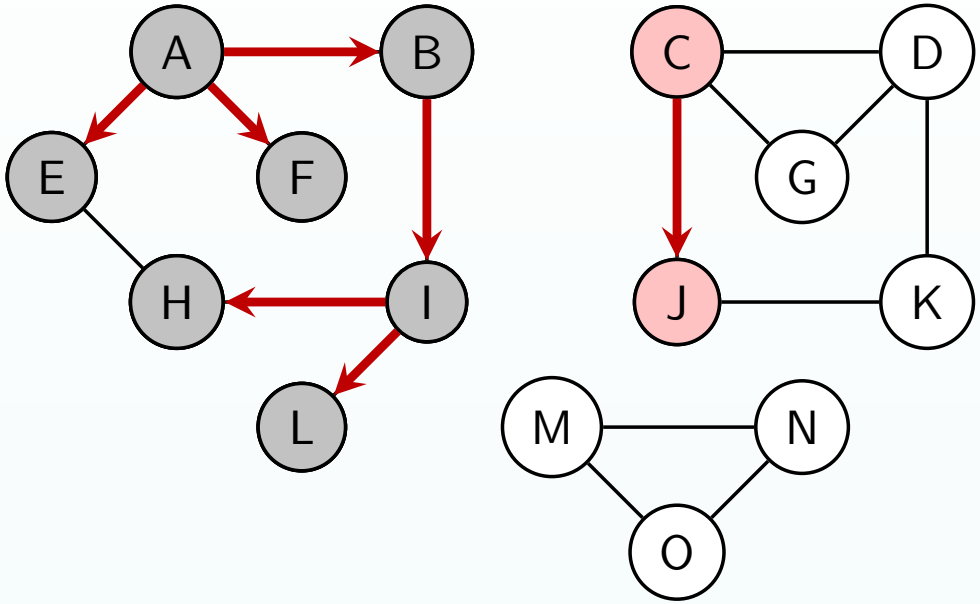
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre J

1: initialement tous les sommets sont non-marqués

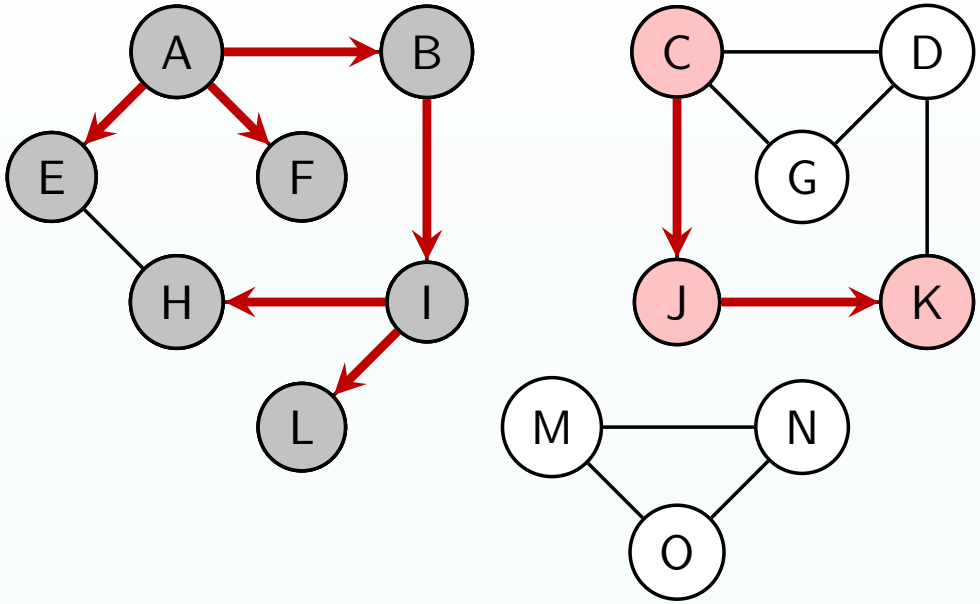
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre K

1: initialement tous les sommets sont non-marqués

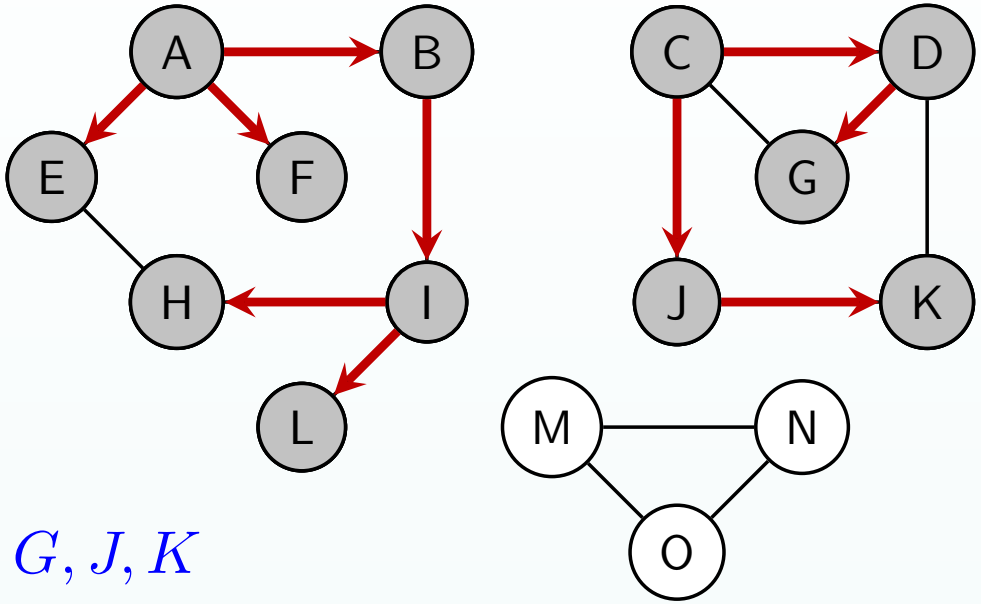
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre D, G
On ferme C, D, G, J, K

1: initialement tous les sommets sont non-marqués

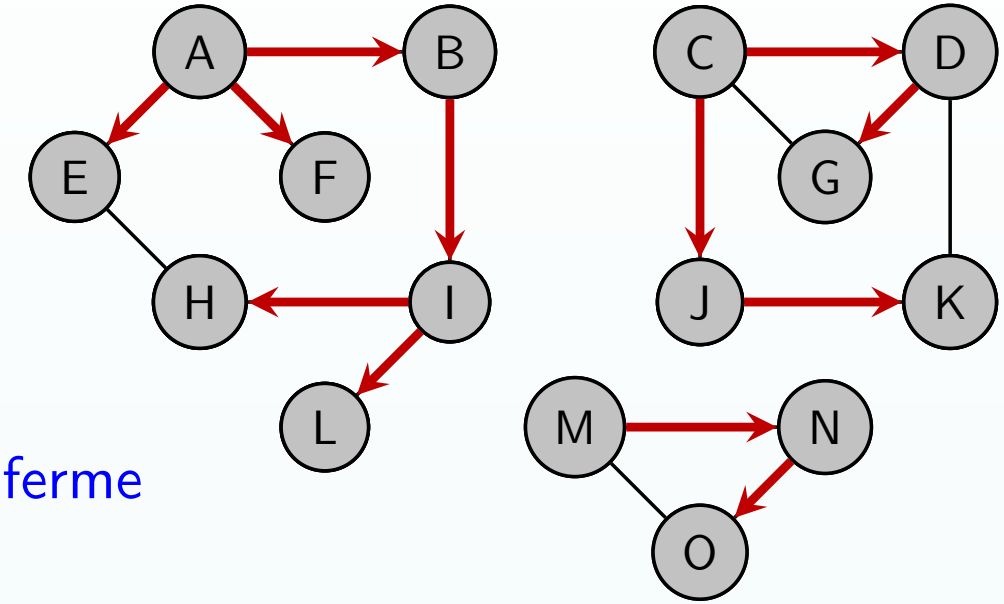
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



On ouvre puis ferme
M, N, O

1: initialement tous les sommets sont non-marqués

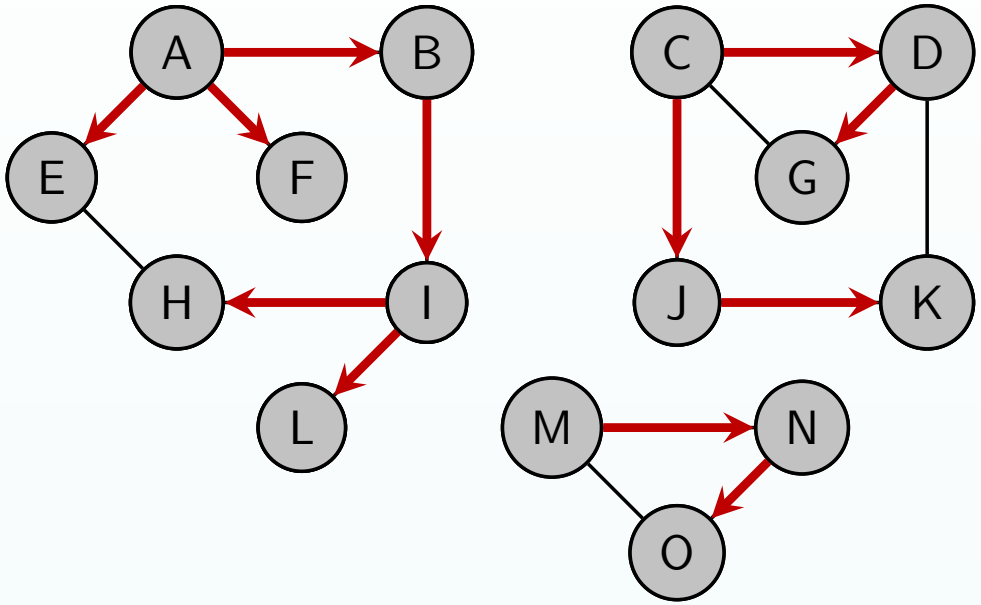
2: tant qu'il existe un sommet non marqué s, ouvrir s

3: tant que cela est possible executer l'une des instructions 4 ou 5

4: ouvrir un sommet y non-marqué s'il est adjacent à un sommet x ouvert

5: fermer un sommet x si tous ses sommets adjacents sont ouverts ou fermés

Parcourir un graphe



Quelle complexité pour le parcours ?

- Avec n sommets et m arcs, $O(n+m) \Leftrightarrow O(\max(n+m))$
- Dans un graphe fortement connexe, $m \gg n \Rightarrow O(m)$

Parcours de graphe en profondeur : calcul de connexité

- Objectif : Déterminer la connexité d'un graphe, ou son nombre de composantes connexes
- Méthode : Le parcours de graphe en profondeur

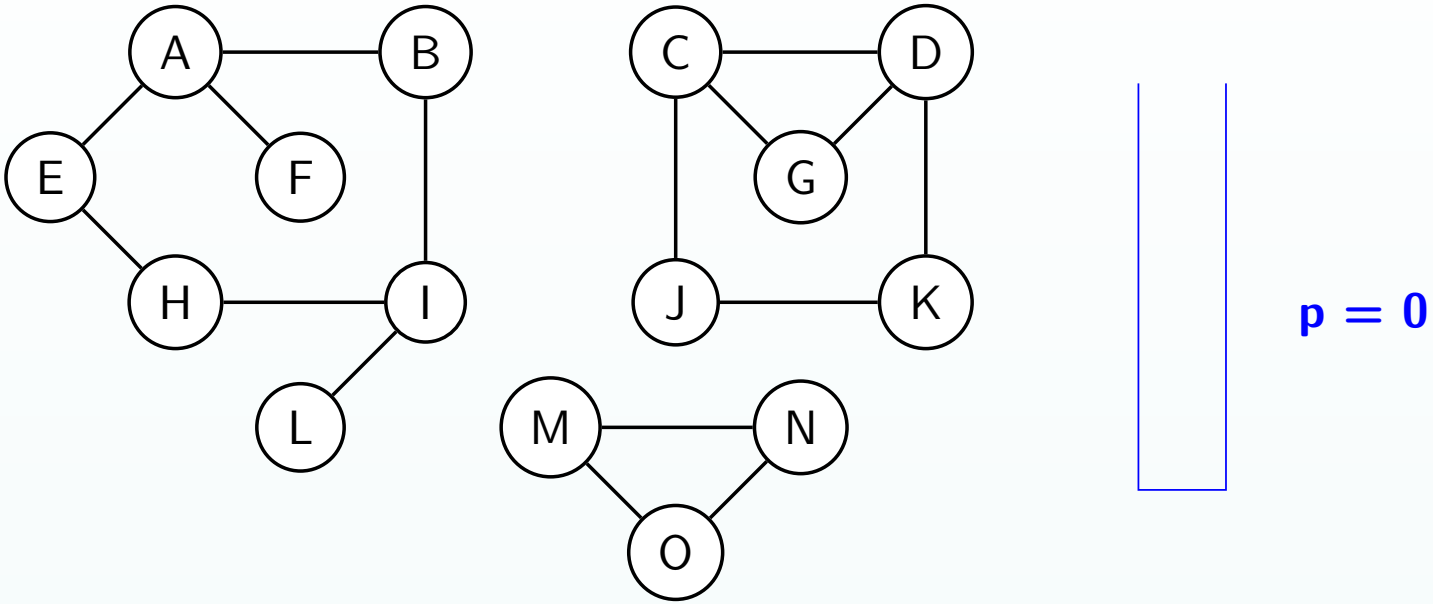
Parcours en profondeur

Un sommet non marqué est ouvert si et seulement s'il est adjacent au dernier sommet[3] précédemment ouvert. Si un tel sommet n'existe pas, le dernier sommet ouvert est alors fermé.

- Utilisation d'une pile (structure de données)
  - *Last In, First Out (LIFO)*

1. *successeur du dernier sommet* dans un graphe orienté

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

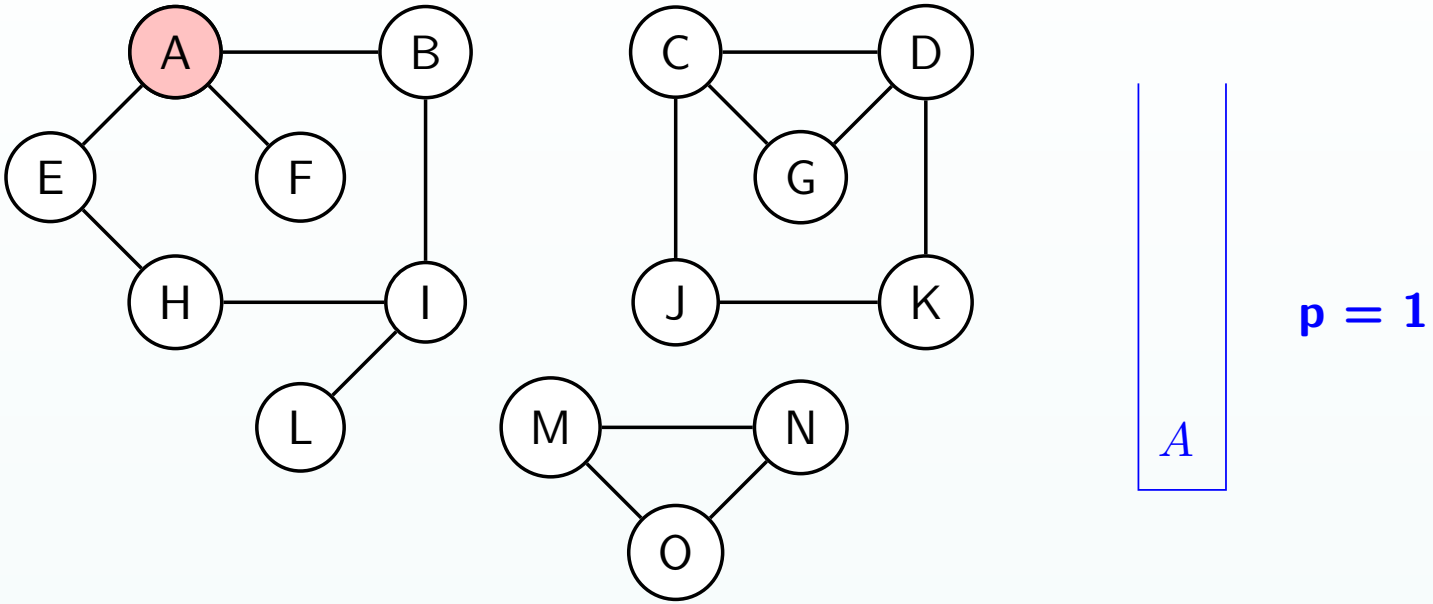
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

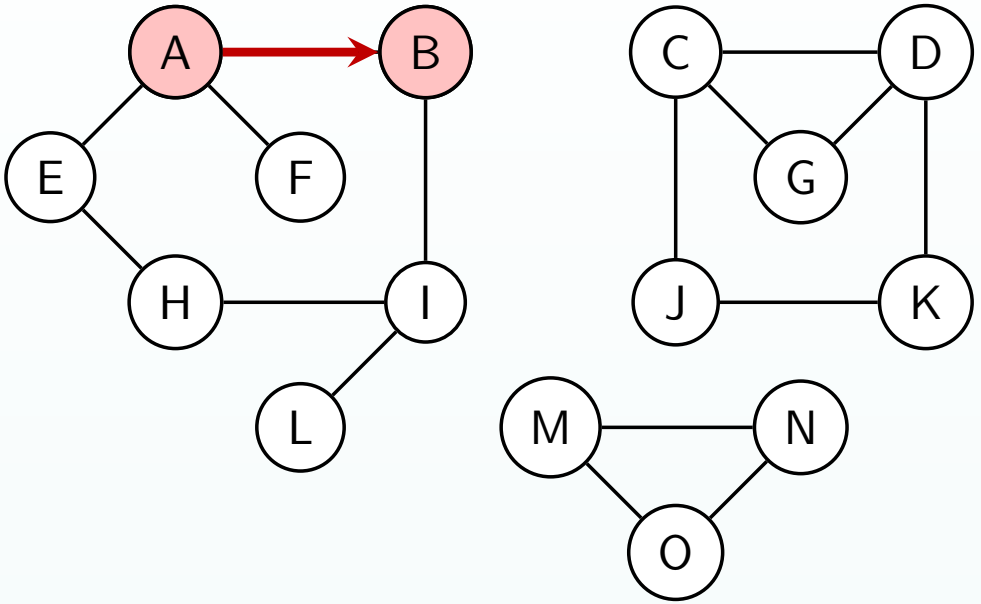
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



p = 1

1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

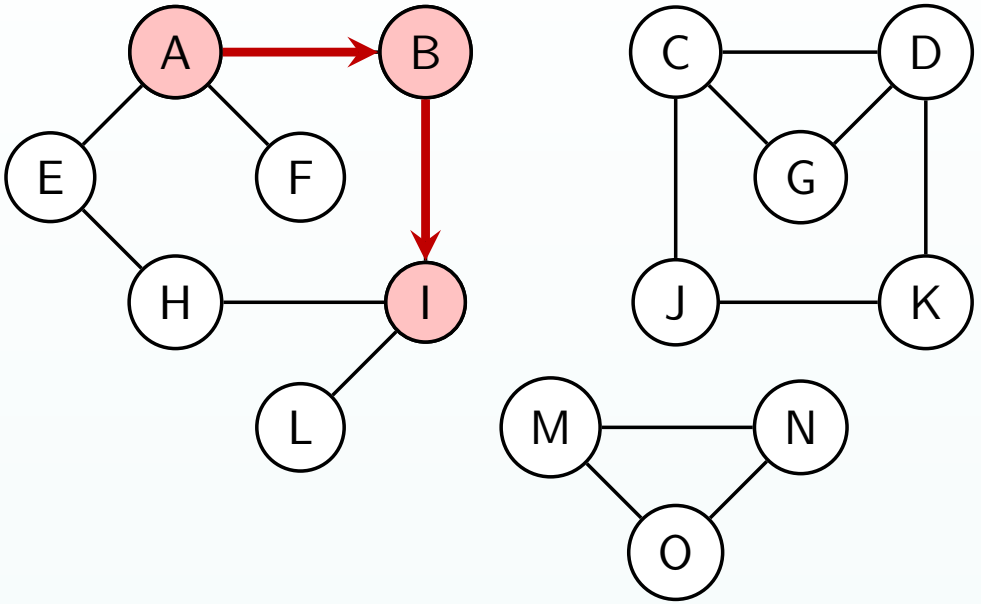
3: tant que $Pile$ n'est pas vide, faire ;

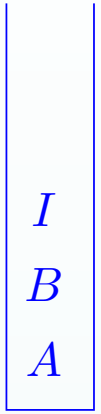
4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité





p = 1

1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

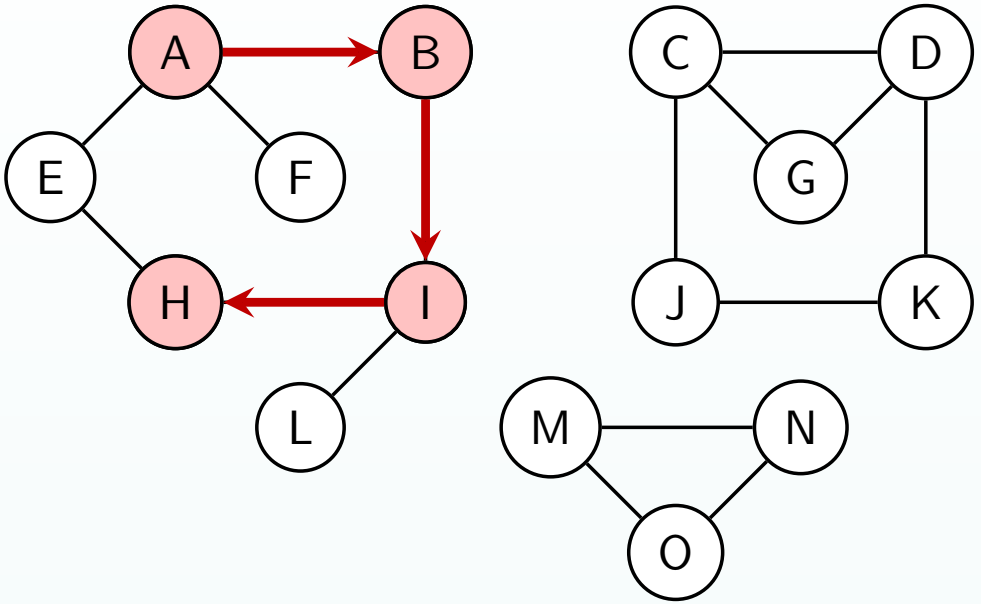
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



H
I
B
A

p = 1

1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

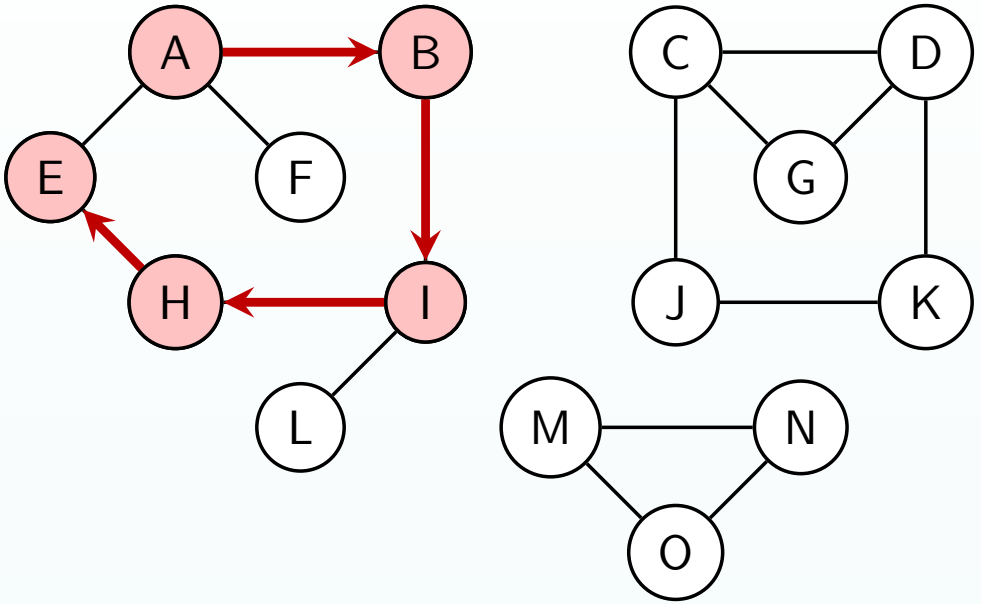
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



E
H
I
B
A

p = 1

1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

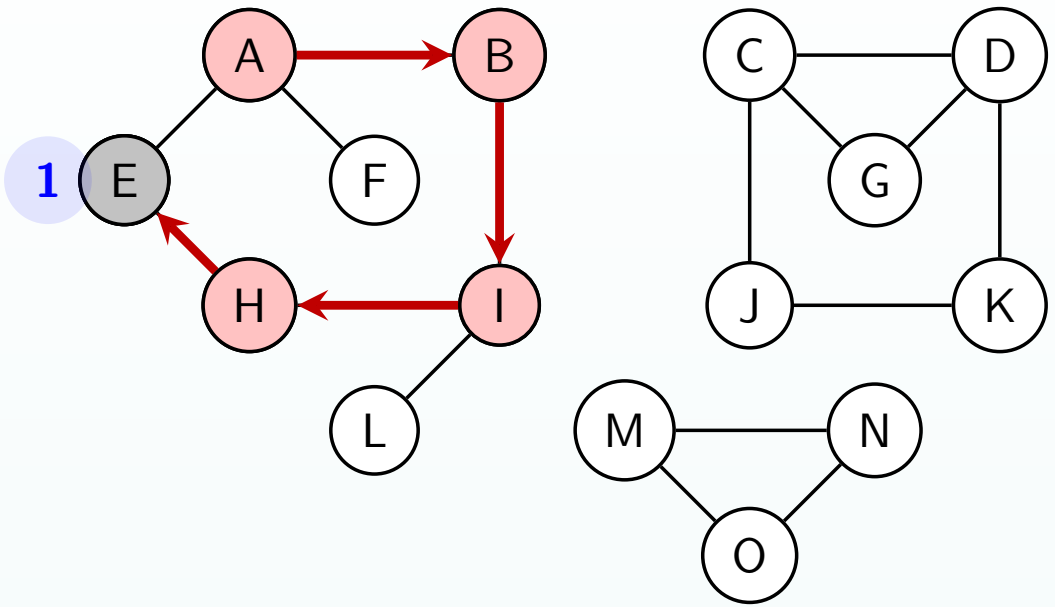
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



H
I
B
A

p = 1

1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

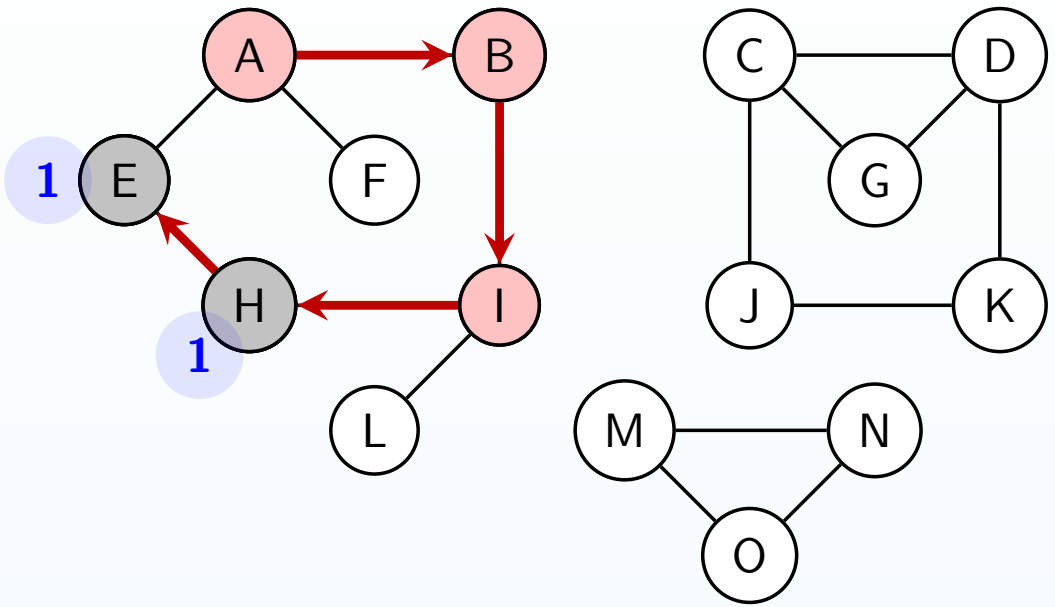
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



I
B
A

p = 1

1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

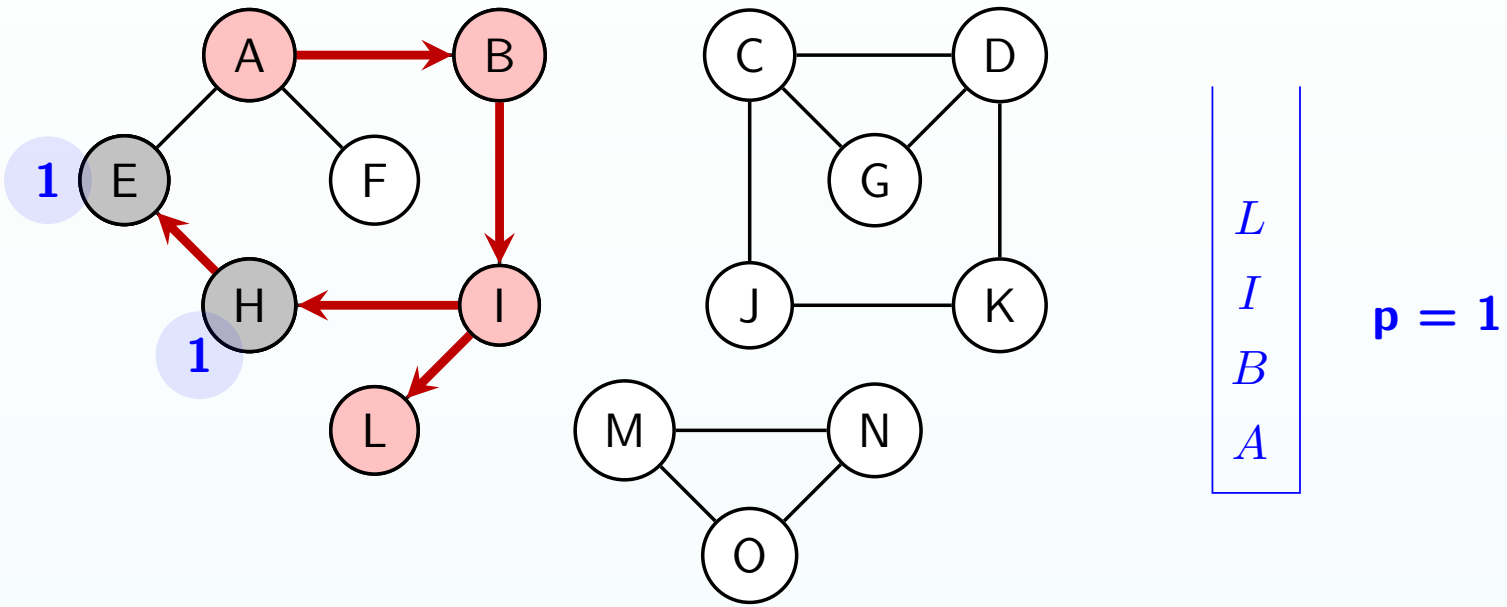
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

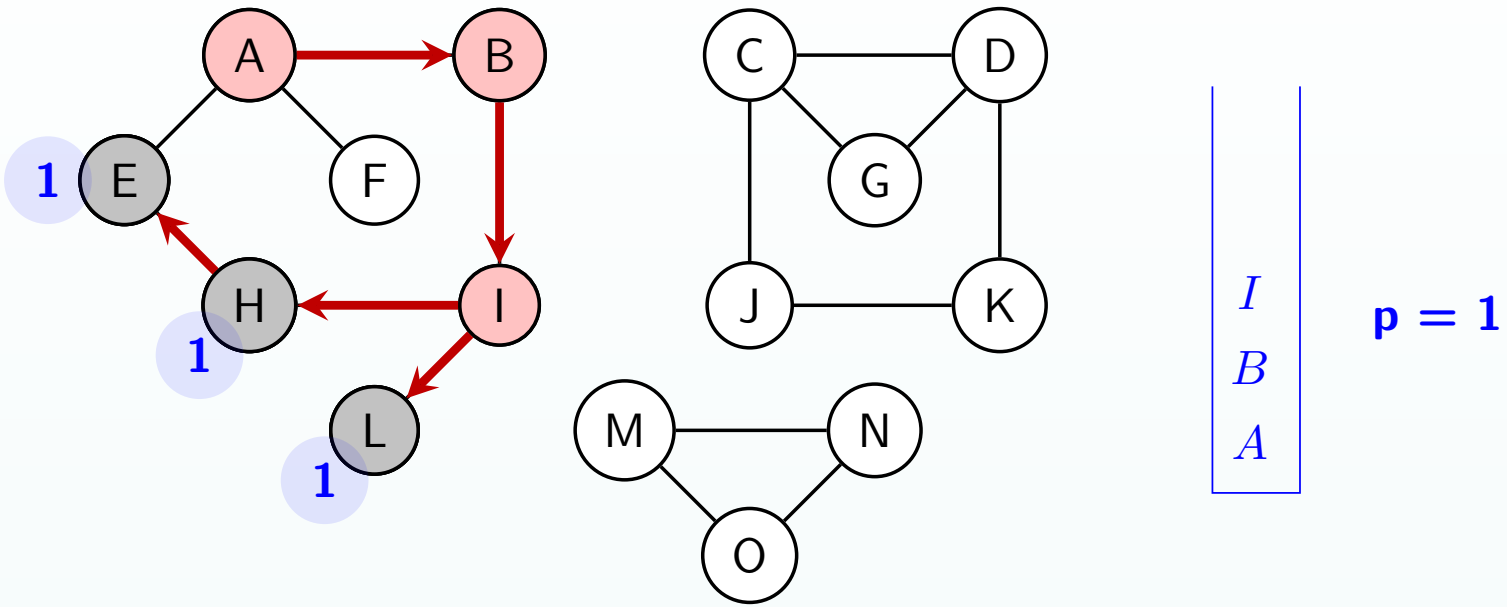
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

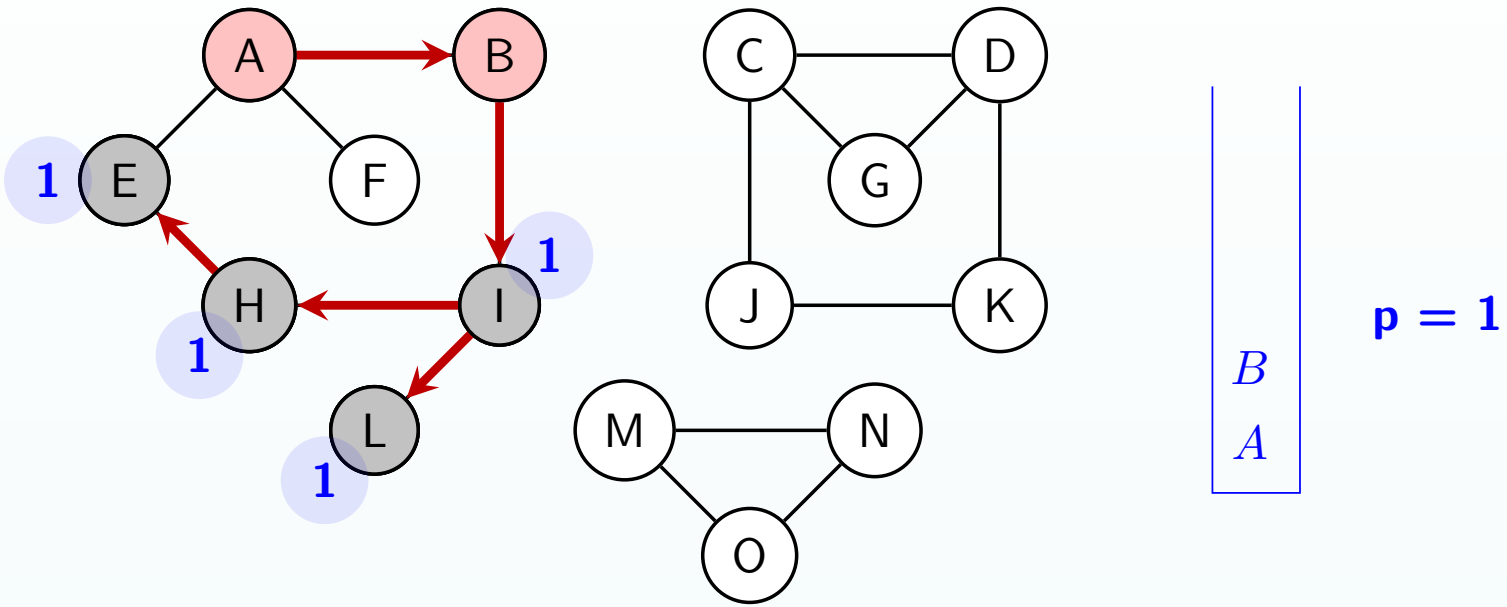
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

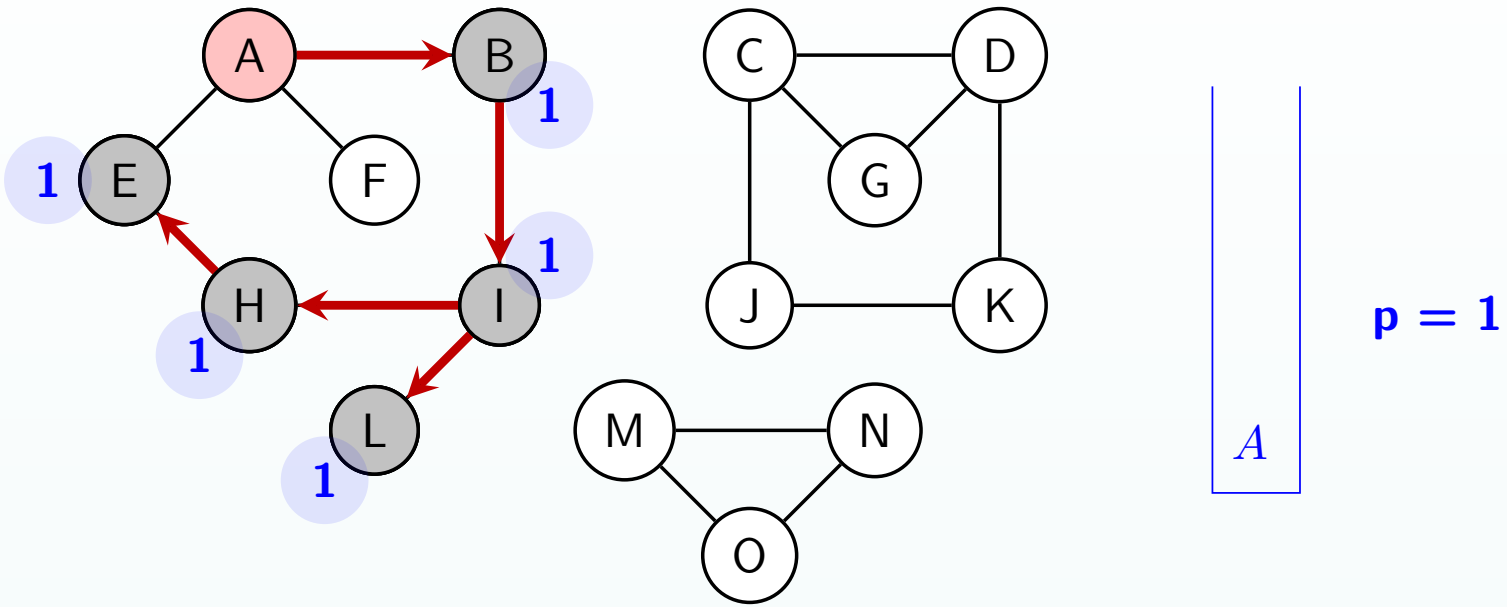
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

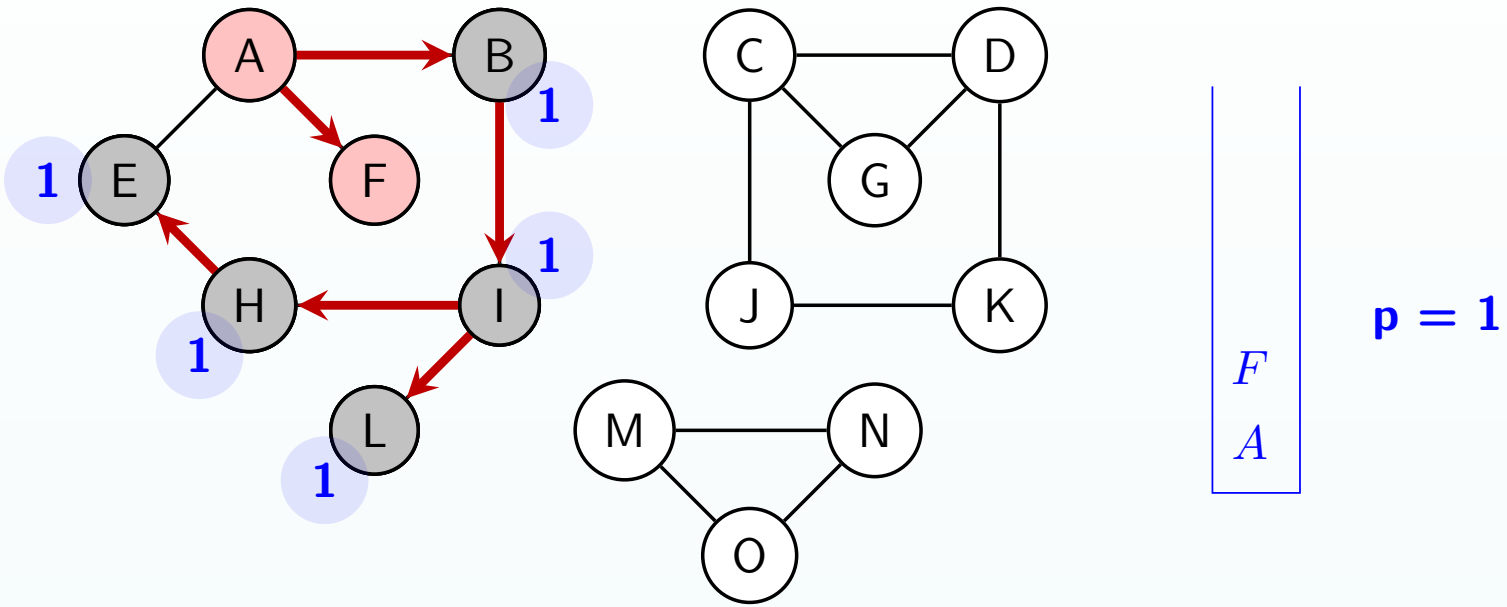
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

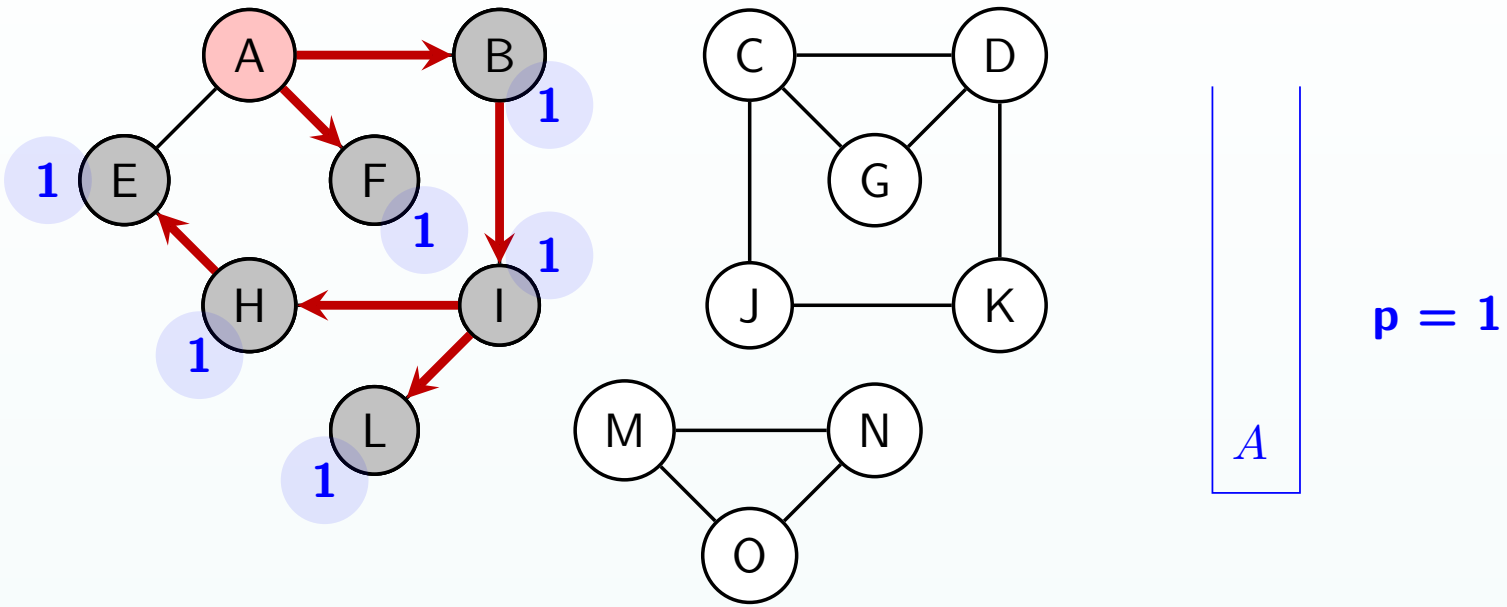
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

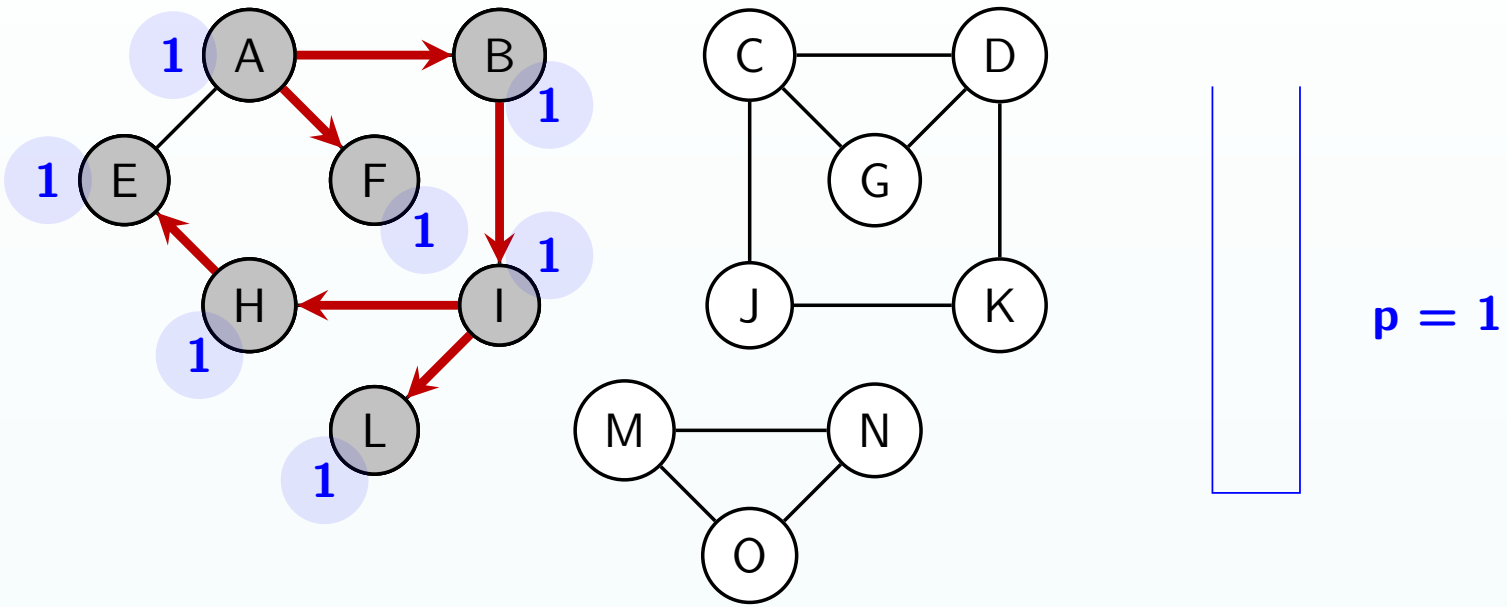
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

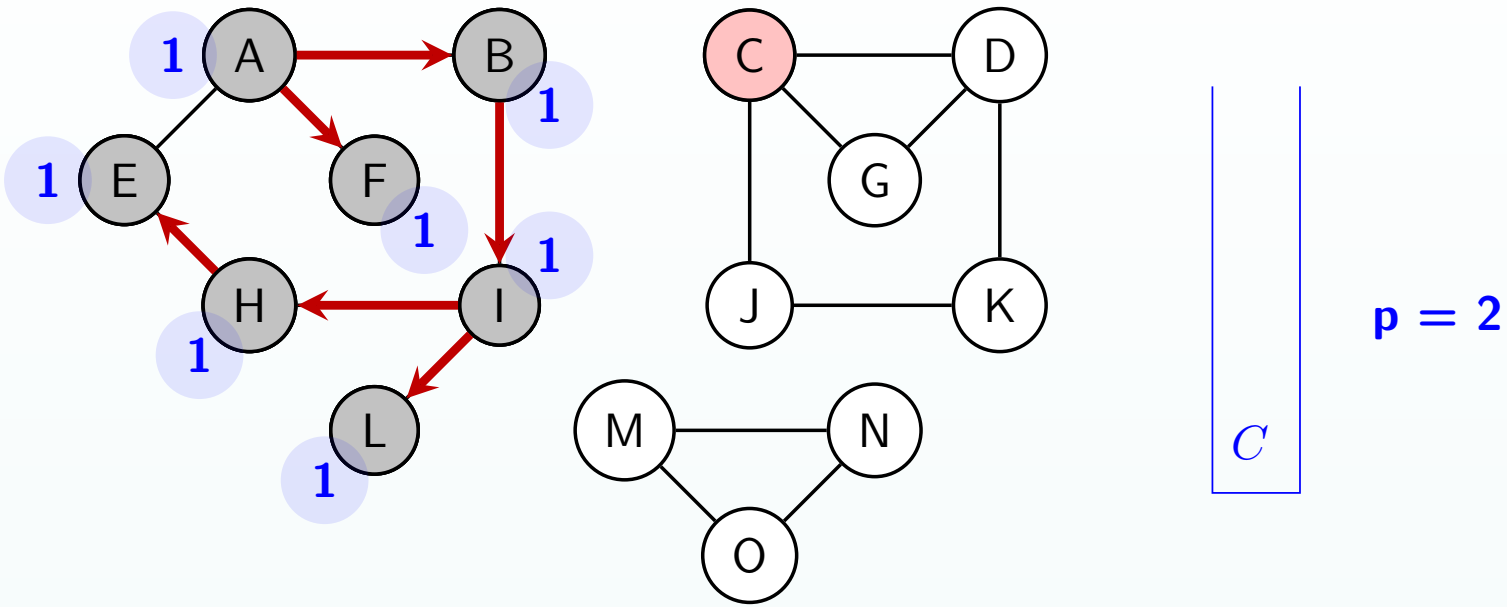
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

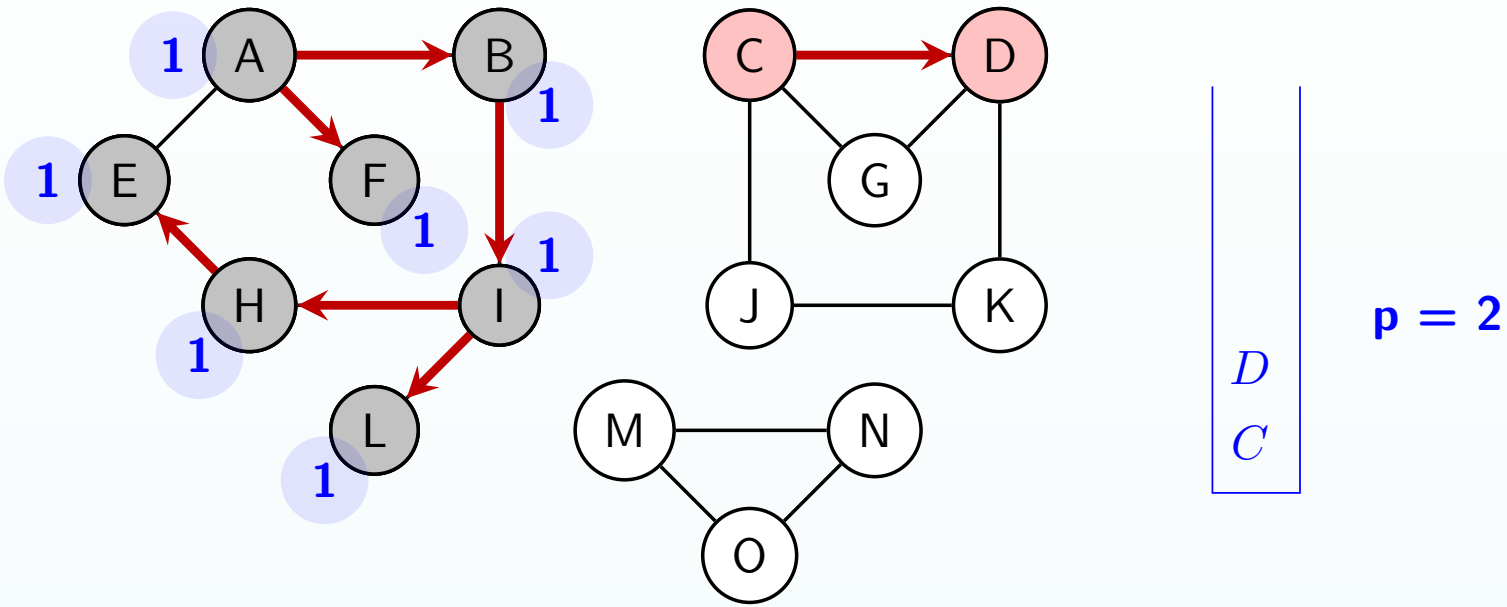
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

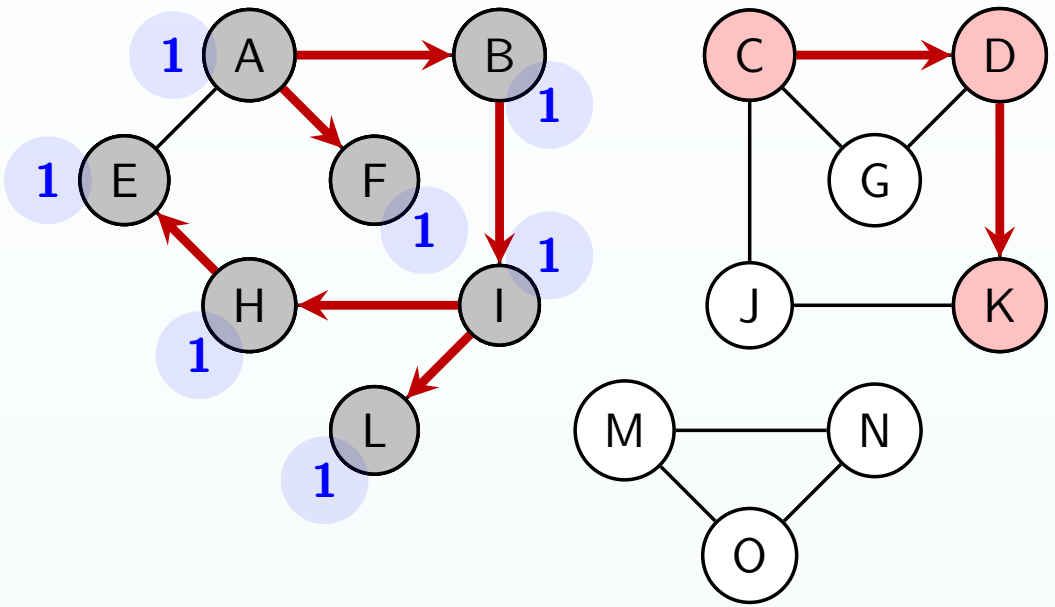
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



K
D
C

p = 2

1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

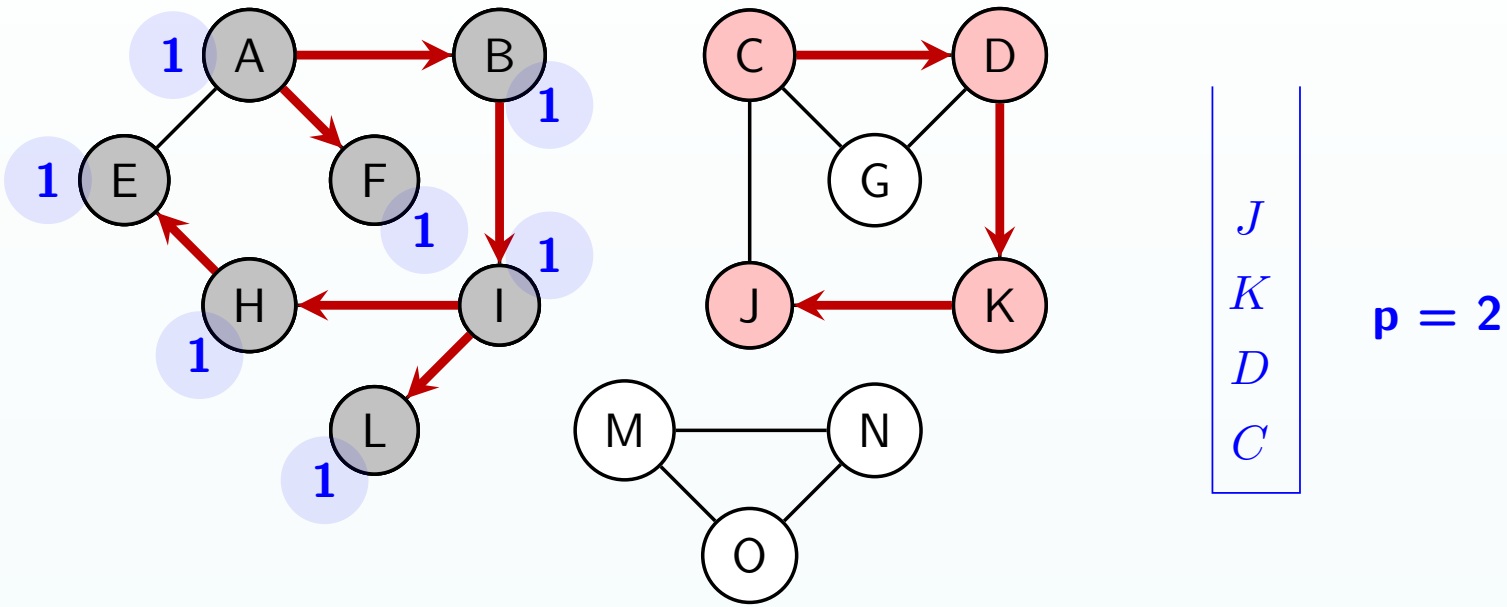
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

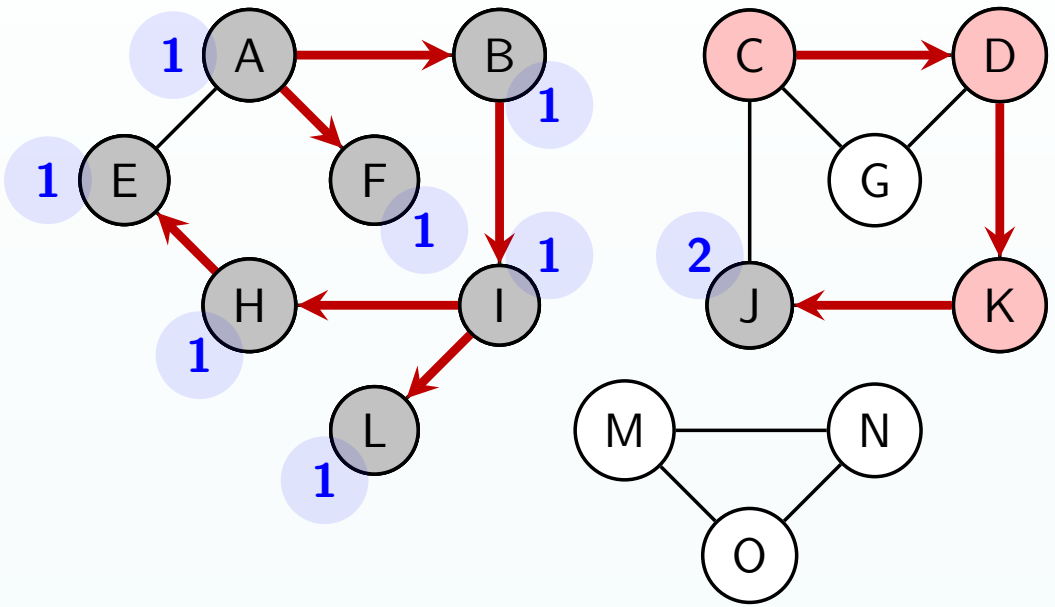
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



K
D
C

p = 2

1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

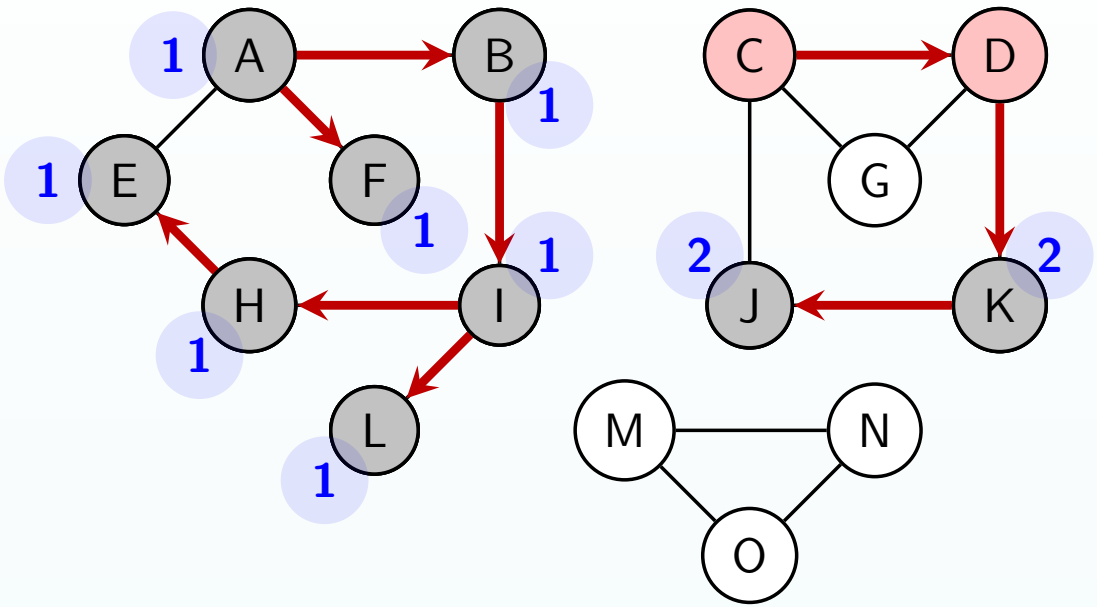
3: tant que $Pile$ n'est pas vide, faire ;

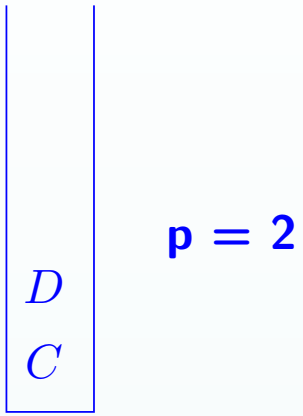
4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité





p = 2

1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

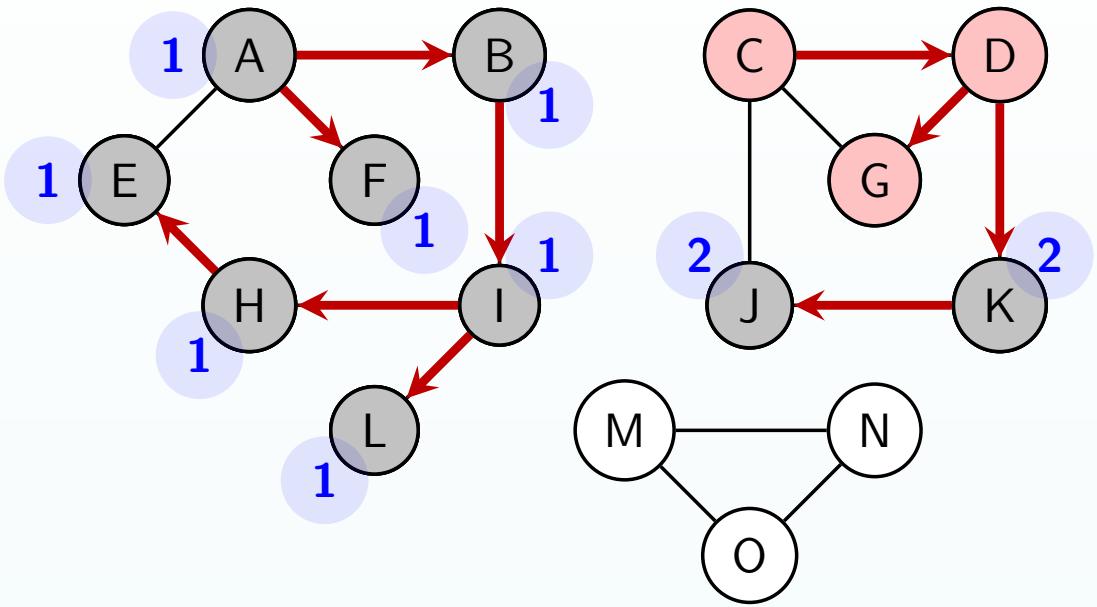
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



G
D
C

p = 2

1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

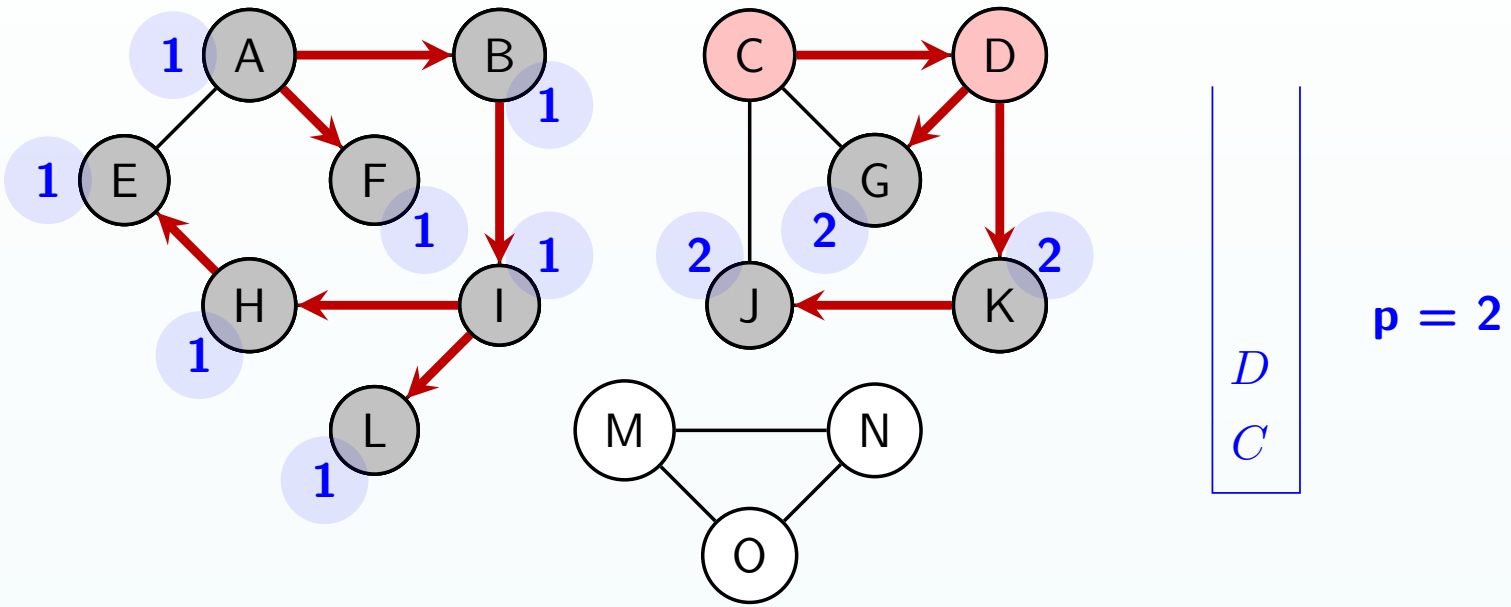
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

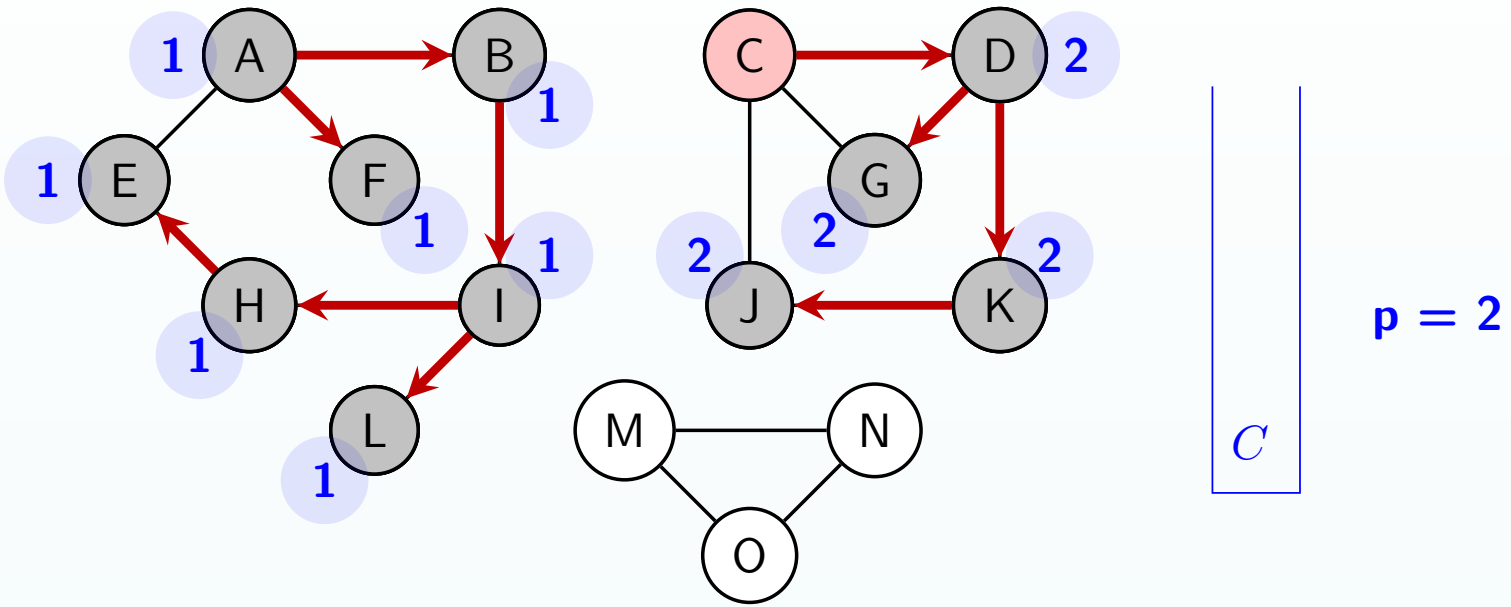
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

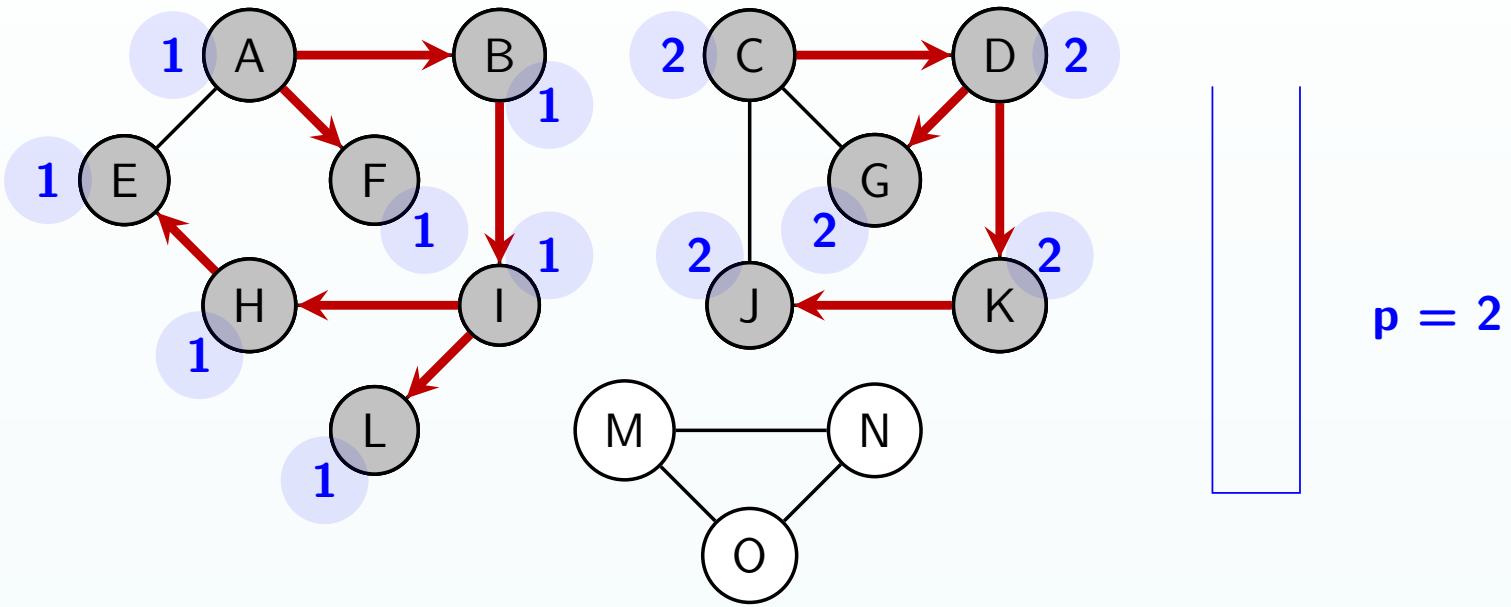
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

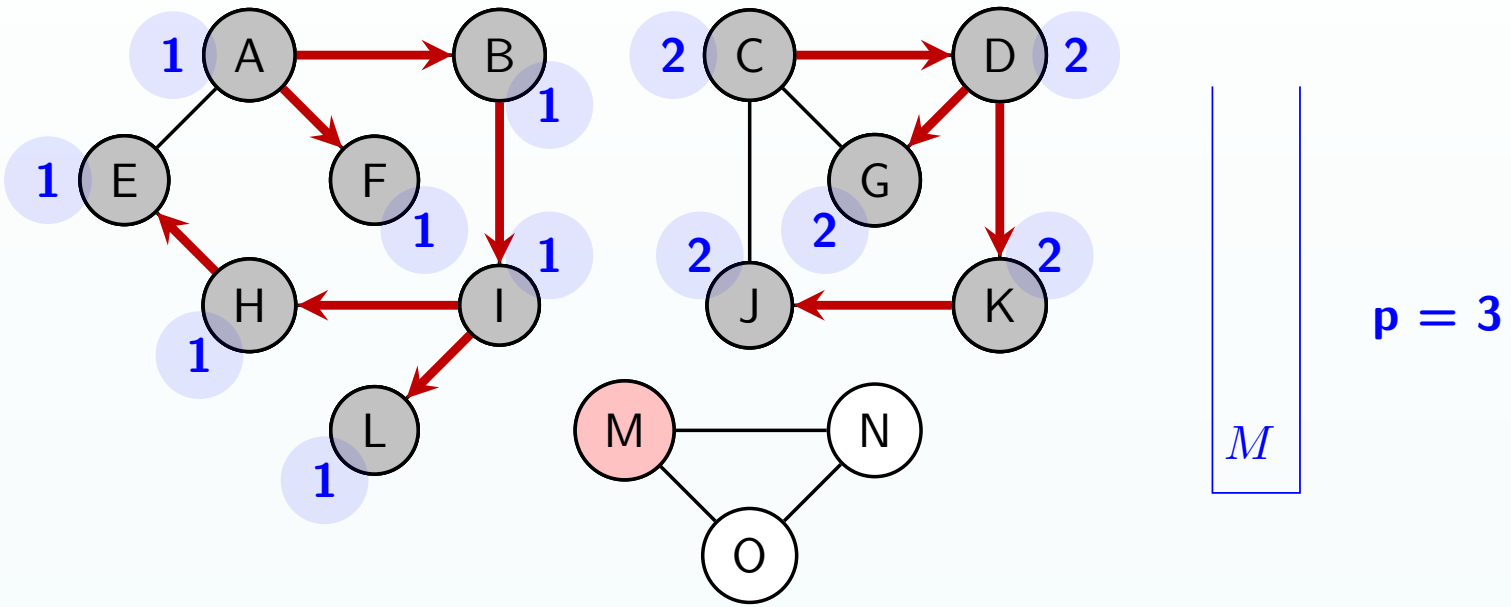
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

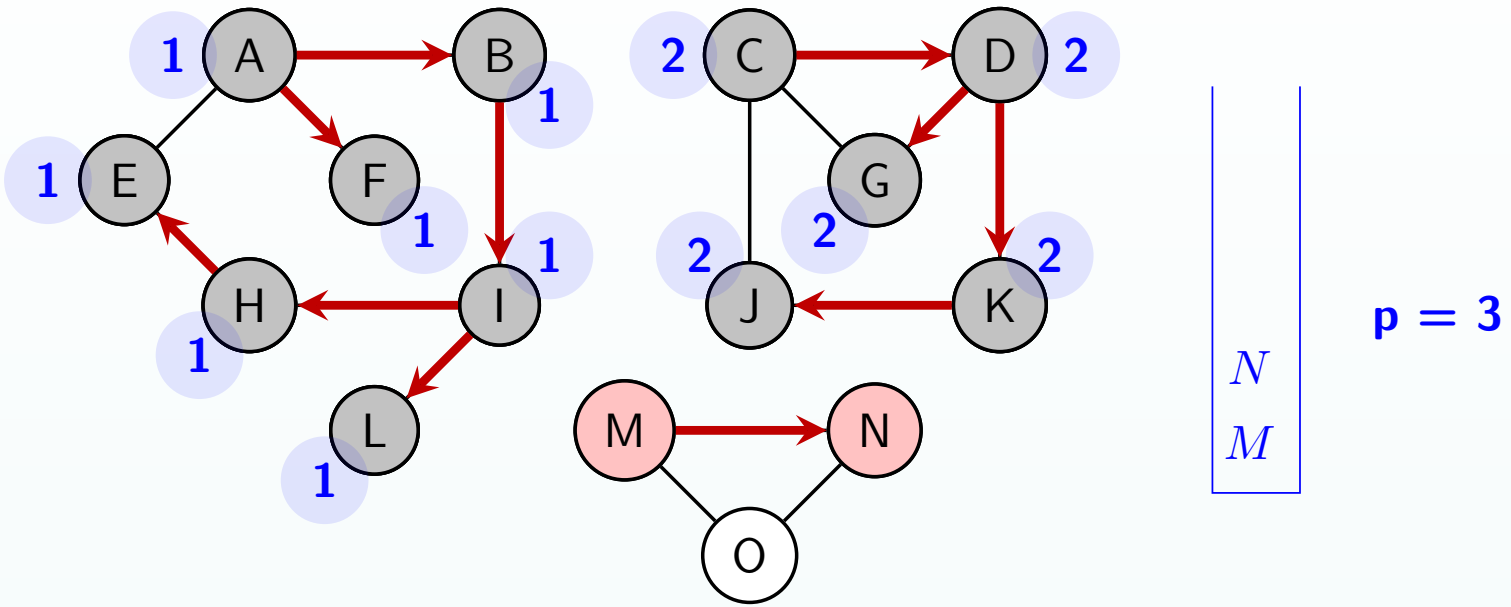
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

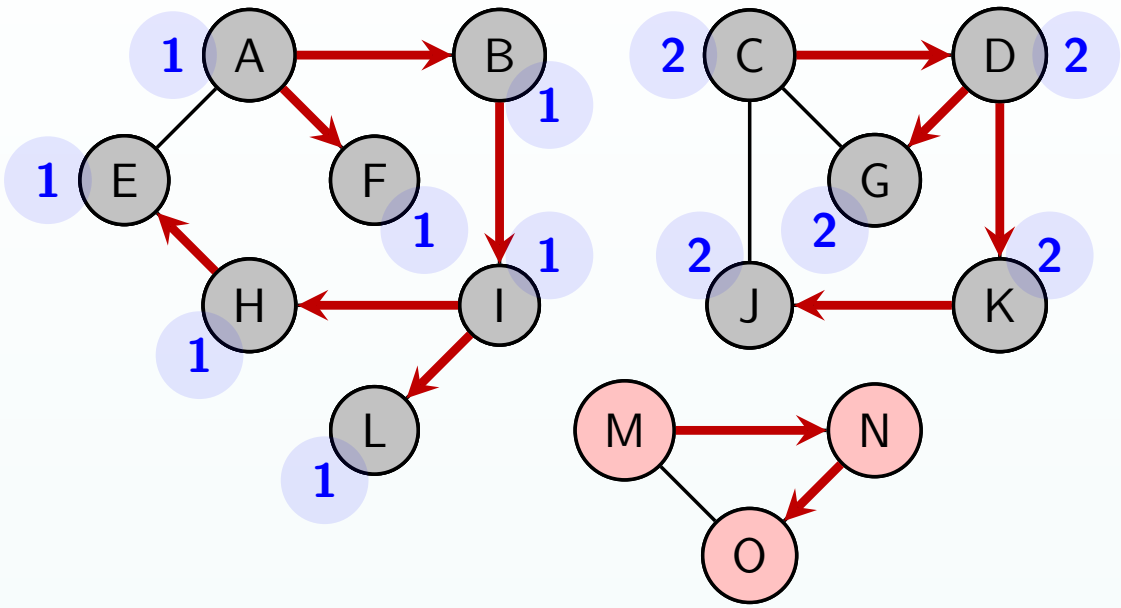
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



O
N
M

p = 3

1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

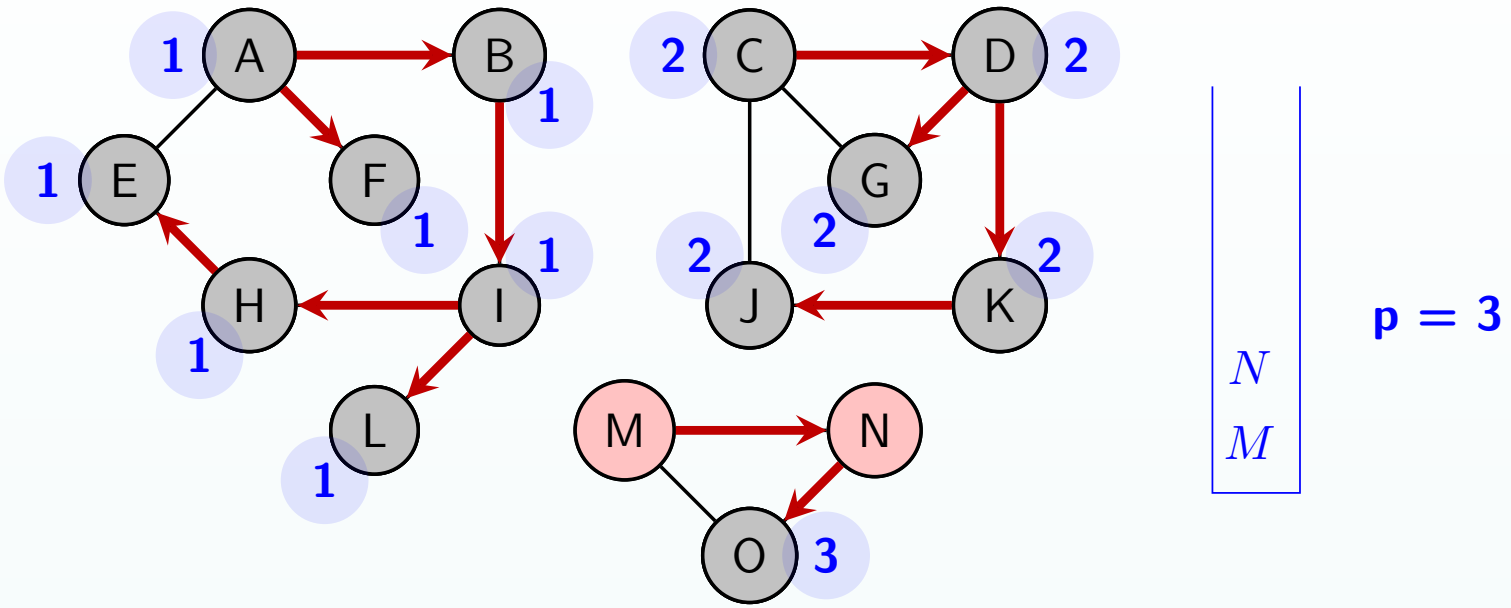
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

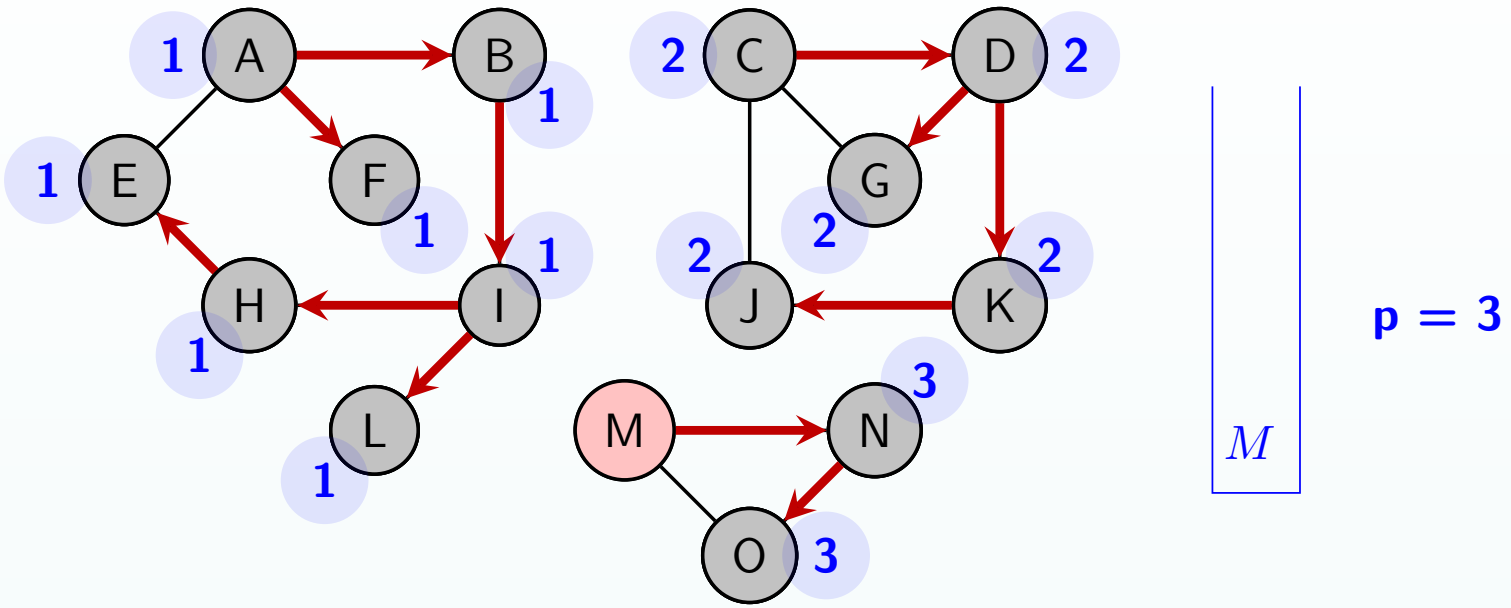
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

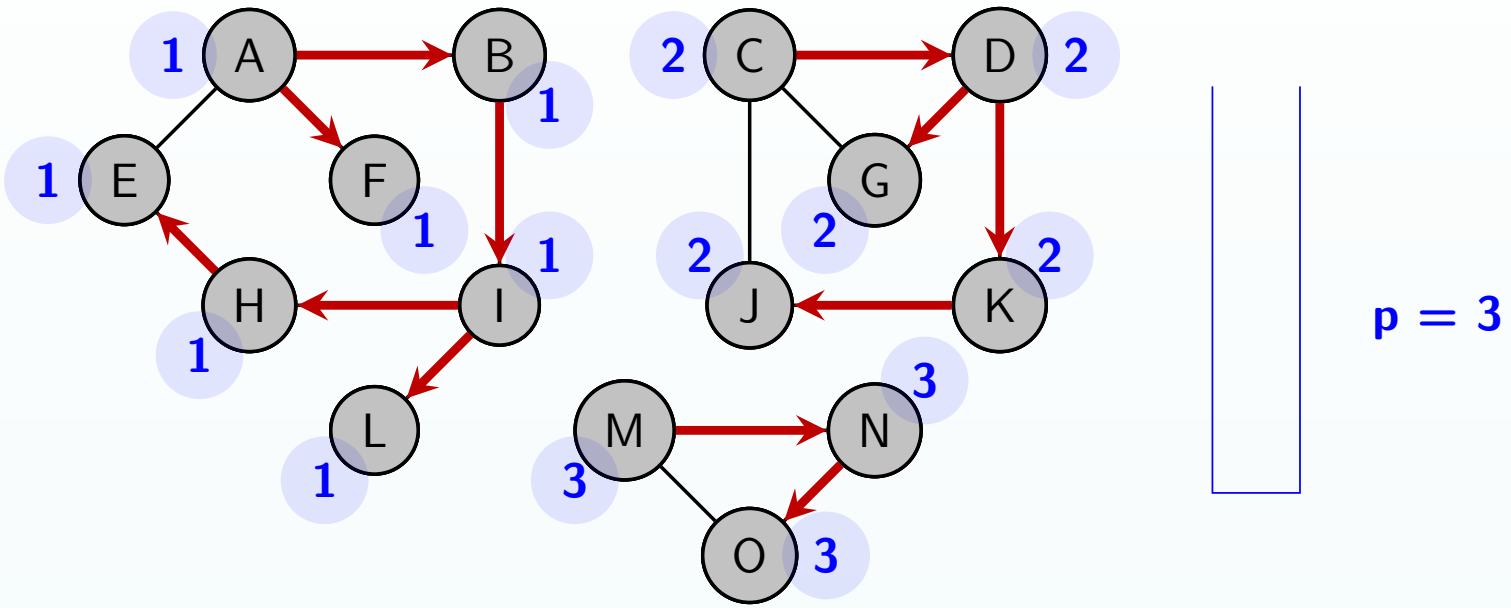
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



1: tous les sommets sont non-marqués ; $p \to 0$; $Pile \to \emptyset$

2: tant qu'il existe un sommet non marqué s, ouvrir et empiler s ; $p \to p + 1$

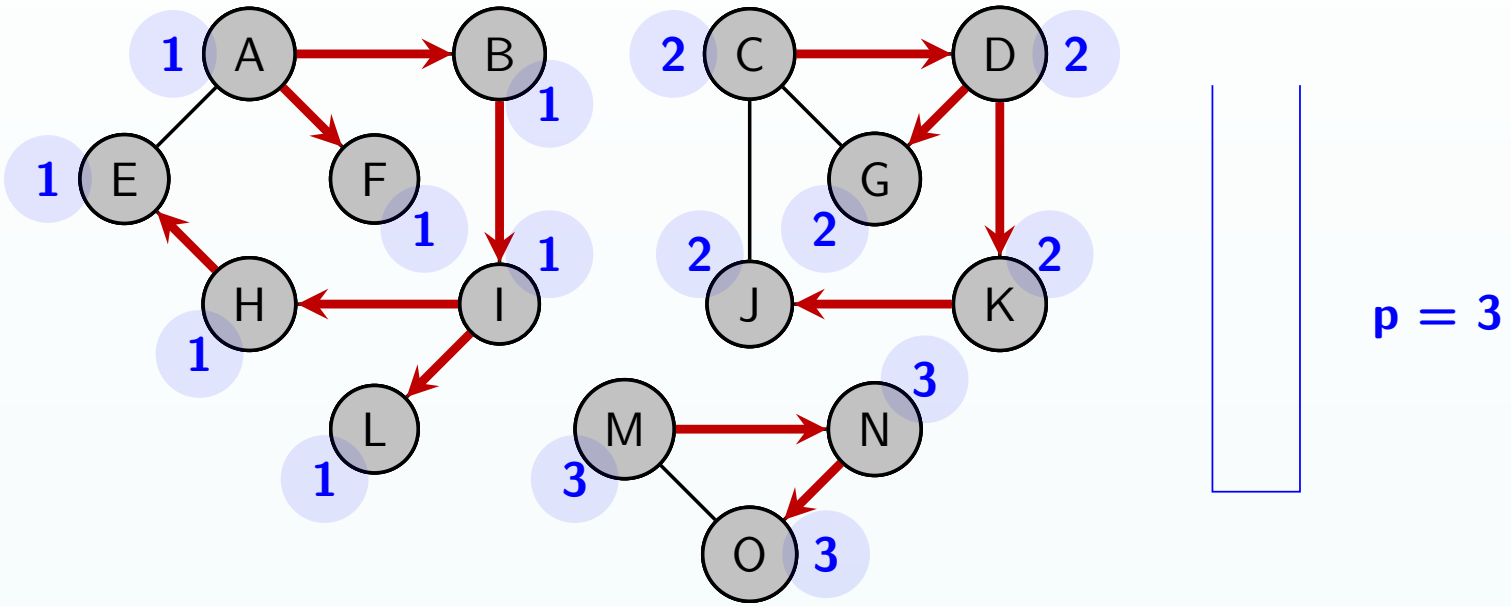
3: tant que $Pile$ n'est pas vide, faire ;

4: s'il existe un sommet y non marqué adjacent au sommet x de $Pile$, faire ;

5: ouvrir et empiler y ;

6: sinon fermer et dépiler x ; $c(x) = p$

Parcours de graphe en profondeur : calcul de connexité



- Connexité déterminée, chaque sommet est associé à une composante connexe
- Propriétés notables du parcours en profondeur :
  - Pile vide à chaque nouvelle composante connexe
  - Toute arrête non parcourue indique un circuit

Parcours de graphe en largeur : plus court chemin

- Objectif : Déterminer la longueur du chemin [4] le plus court d'un sommet s aux autres sommets du graphe
- Méthode : Le parcours de graphe en largeur

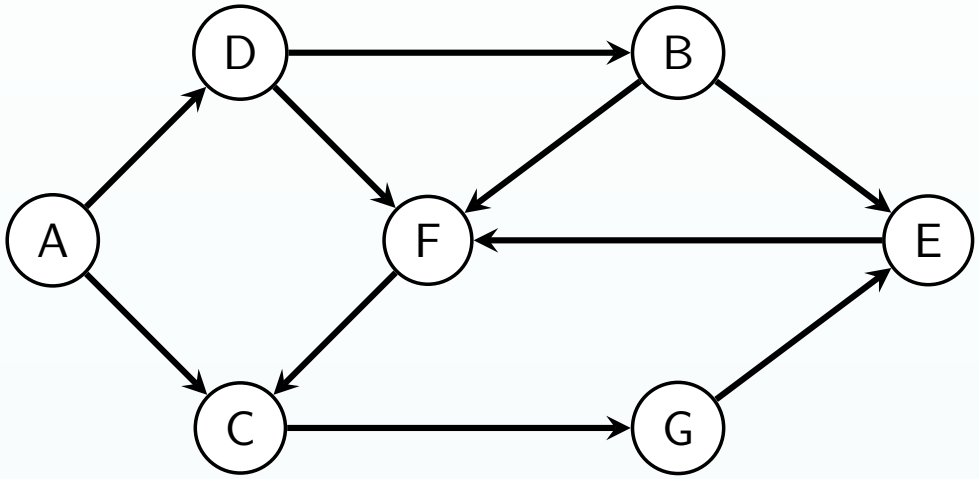
Parcours en largeur

Tous les successeurs non marqués du sommet en cours de visite sont ouverts successivement. Le sommet visité à chaque étape est, parmi les sommets ouverts, celui qui a été ouvert en premier.

- Utilisation d'une file (structure de données)
  - *First In, First Out (FIFO)*

4. ne pas confondre avec la valeur d'un chemin d'arcs valués

Parcours de graphe en largeur : plus court chemin



1: tous les sommets sont non-marqués ; $File \rightarrow \emptyset$

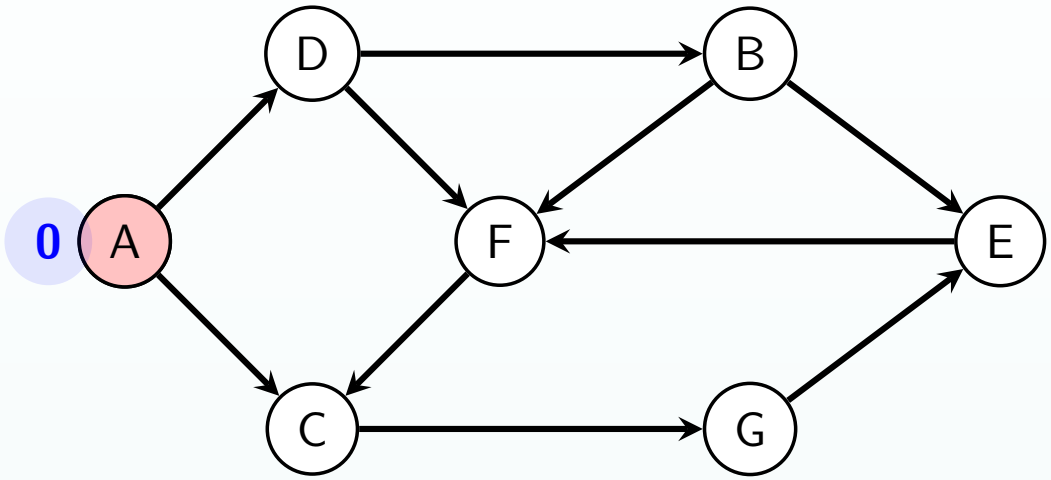
2: ouvrir et insérer s ; $d(s) \rightarrow 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



$A \leftarrow$

1: tous les sommets sont non-marqués ; $File \rightarrow \emptyset$

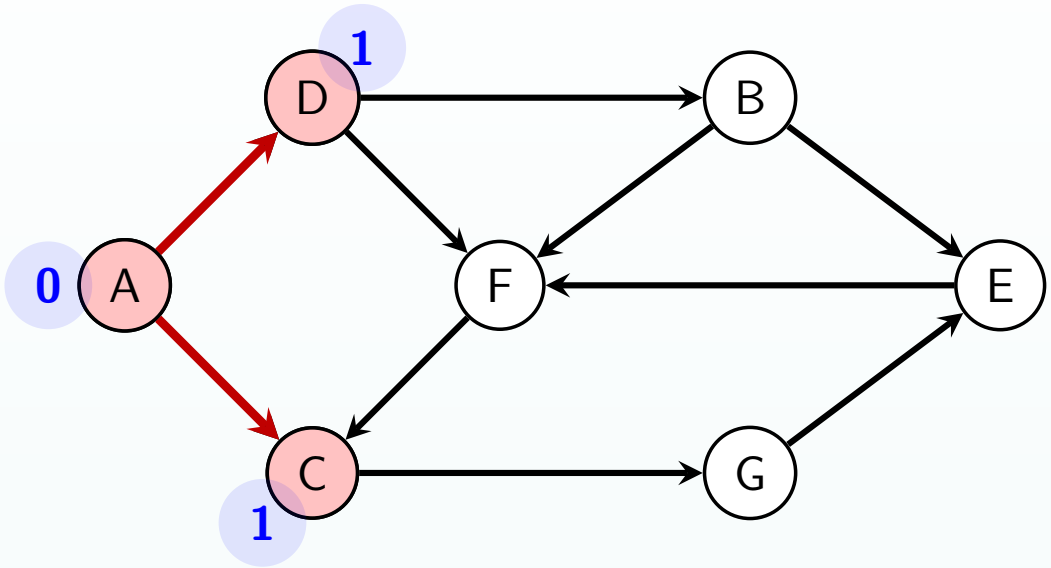
2: ouvrir et insérer s ; $d(s) \rightarrow 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



$A\ D\ C$

1: tous les sommets sont non-marqués ; $File \rightarrow \emptyset$

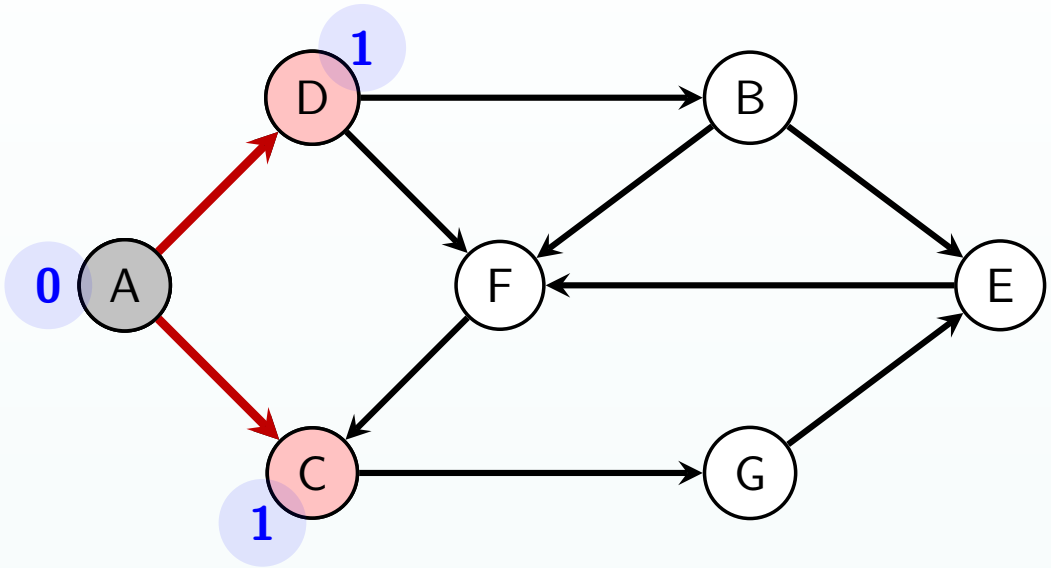
2: ouvrir et insérer s ; $d(s) \rightarrow 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



D C

1: tous les sommets sont non-marqués ; $File \rightarrow \emptyset$

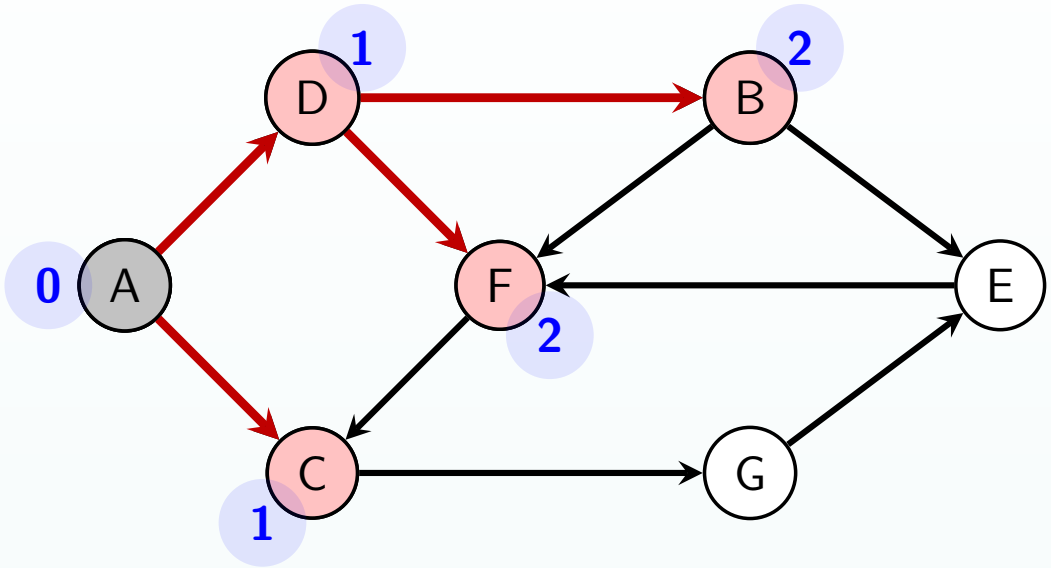
2: ouvrir et insérer s ; $d(s) \rightarrow 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



$D\ C\ B\ F$

1: tous les sommets sont non-marqués ; $File \rightarrow \emptyset$

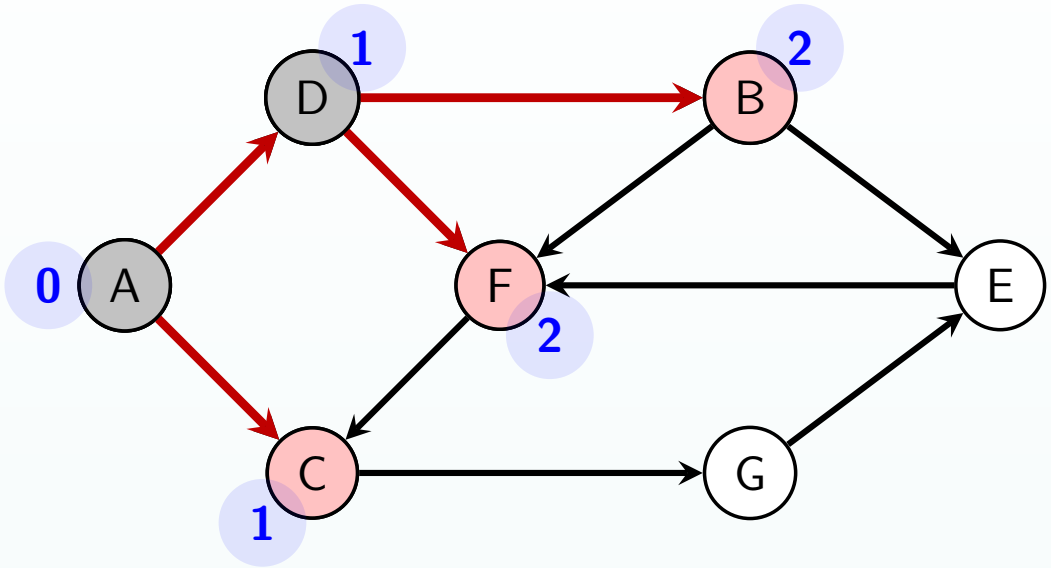
2: ouvrir et insérer s ; $d(s) \rightarrow 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



$C\ B\ F$

1: tous les sommets sont non-marqués ; $File \to \emptyset$

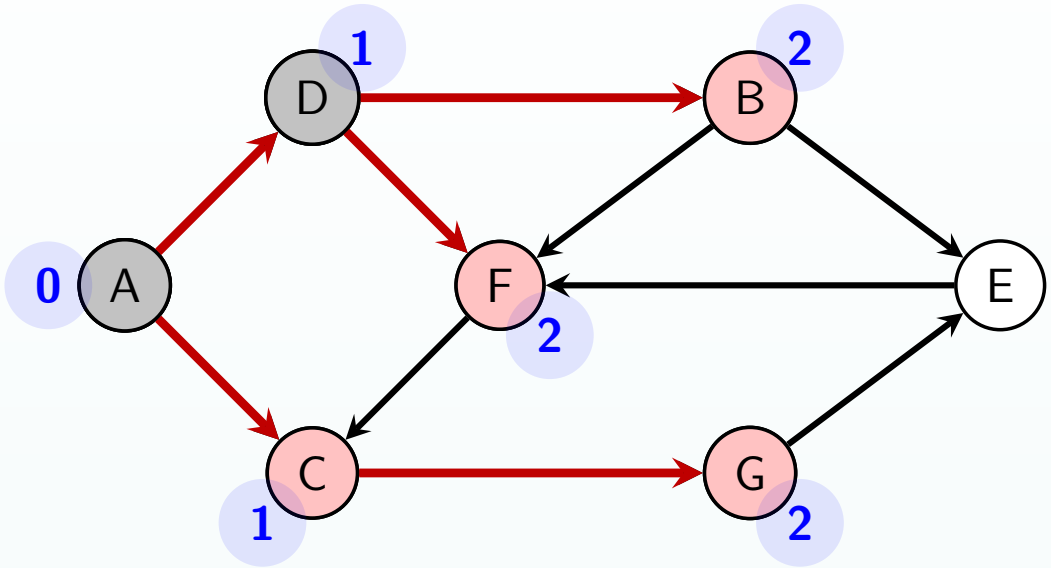
2: ouvrir et insérer s ; $d(s) \to 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



$C\ B\ F\ G$

1: tous les sommets sont non-marqués ; $File \to \emptyset$

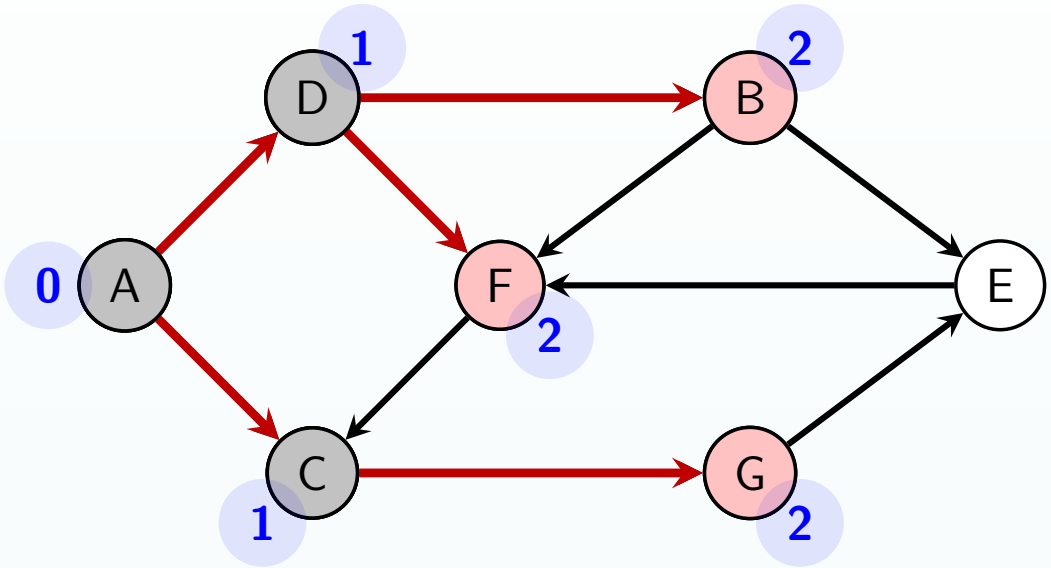
2: ouvrir et insérer s ; $d(s) \to 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



$B\ F\ G$

1: tous les sommets sont non-marqués ; $File \rightarrow \emptyset$

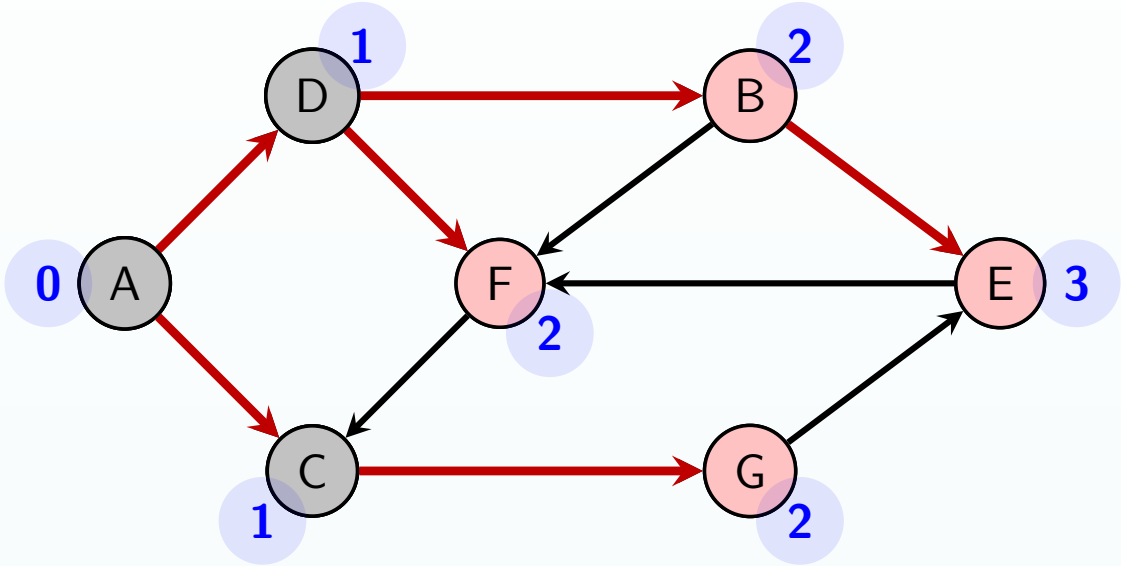
2: ouvrir et insérer s ; $d(s) \rightarrow 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



$B \; F \; G \; E$

1: tous les sommets sont non-marqués ; $File \rightarrow \emptyset$

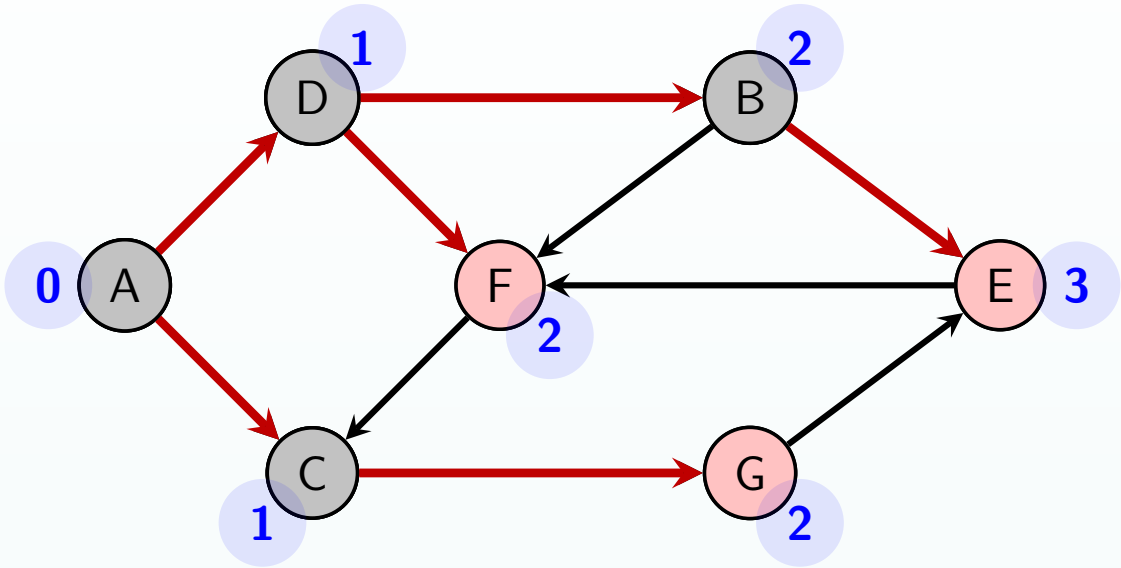
2: ouvrir et insérer s ; $d(s) \rightarrow 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



$F\ G\ E$

1: tous les sommets sont non-marqués ; $File \to \emptyset$

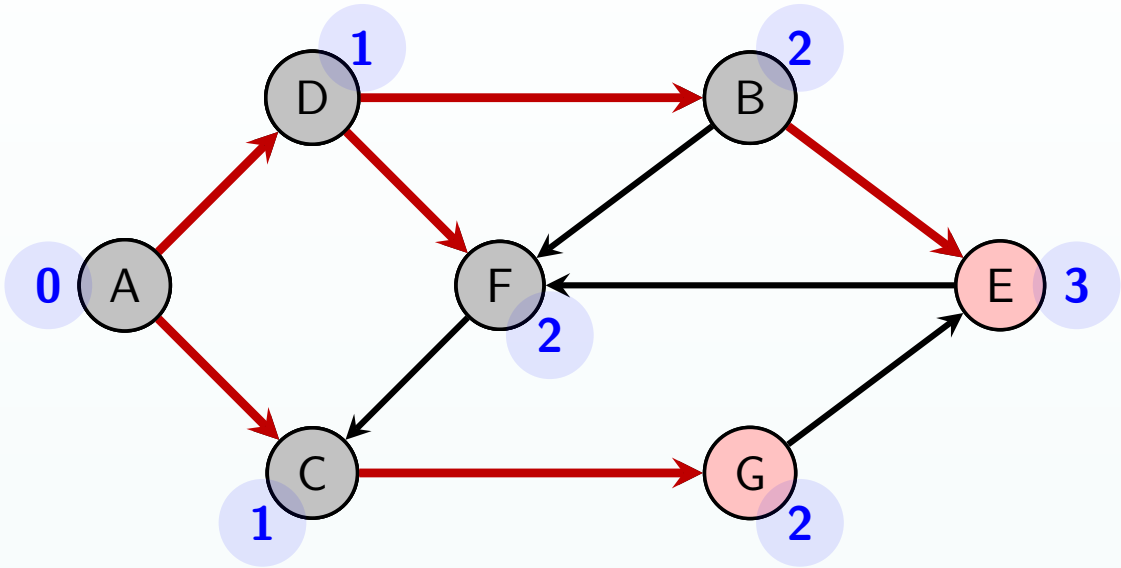
2: ouvrir et insérer s ; $d(s) \to 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



G E

1: tous les sommets sont non-marqués ; $File \to \emptyset$

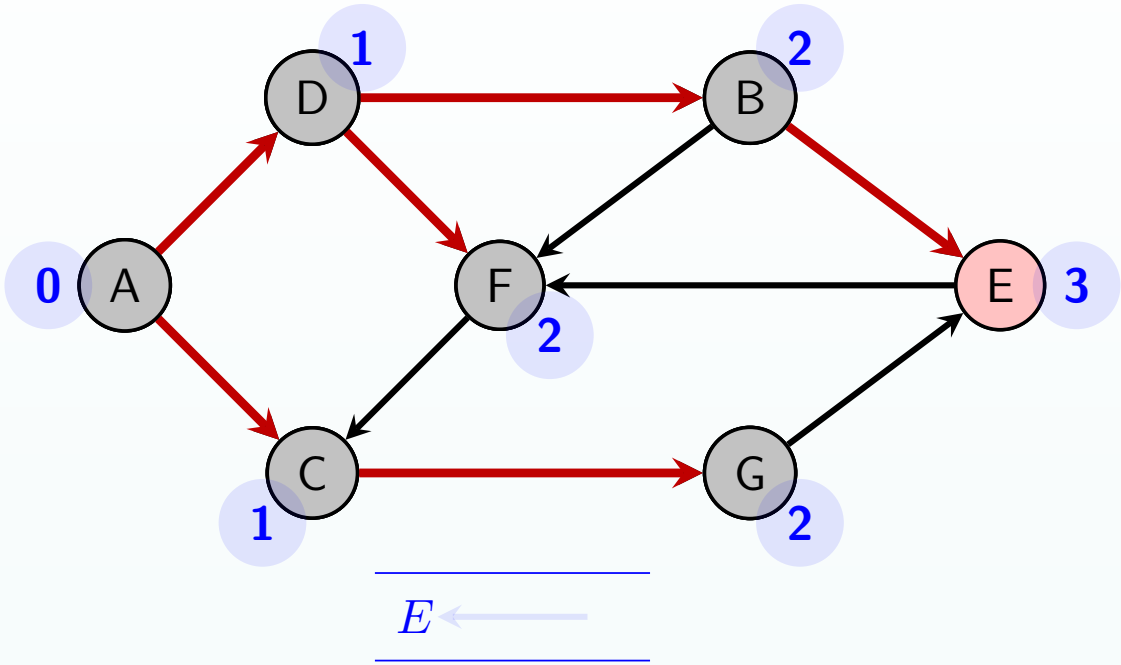
2: ouvrir et insérer s ; $d(s) \to 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



1: tous les sommets sont non-marqués ; $File \rightarrow \emptyset$

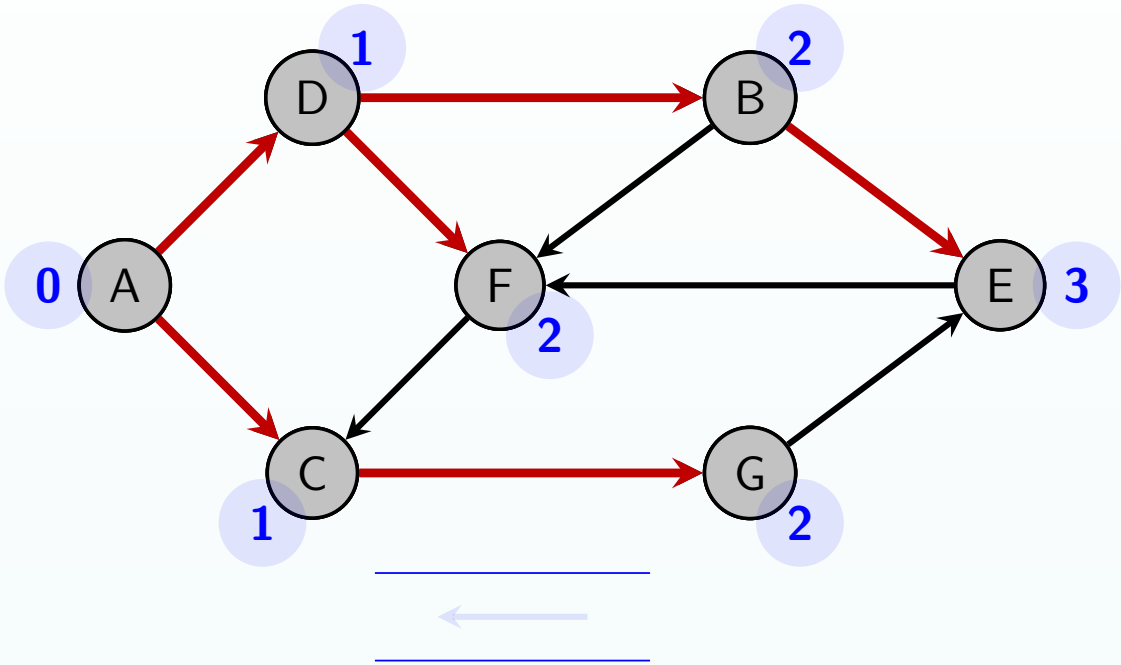
2: ouvrir et insérer s ; $d(s) \rightarrow 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en largeur : plus court chemin



1: tous les sommets sont non-marqués ; $File \rightarrow \emptyset$

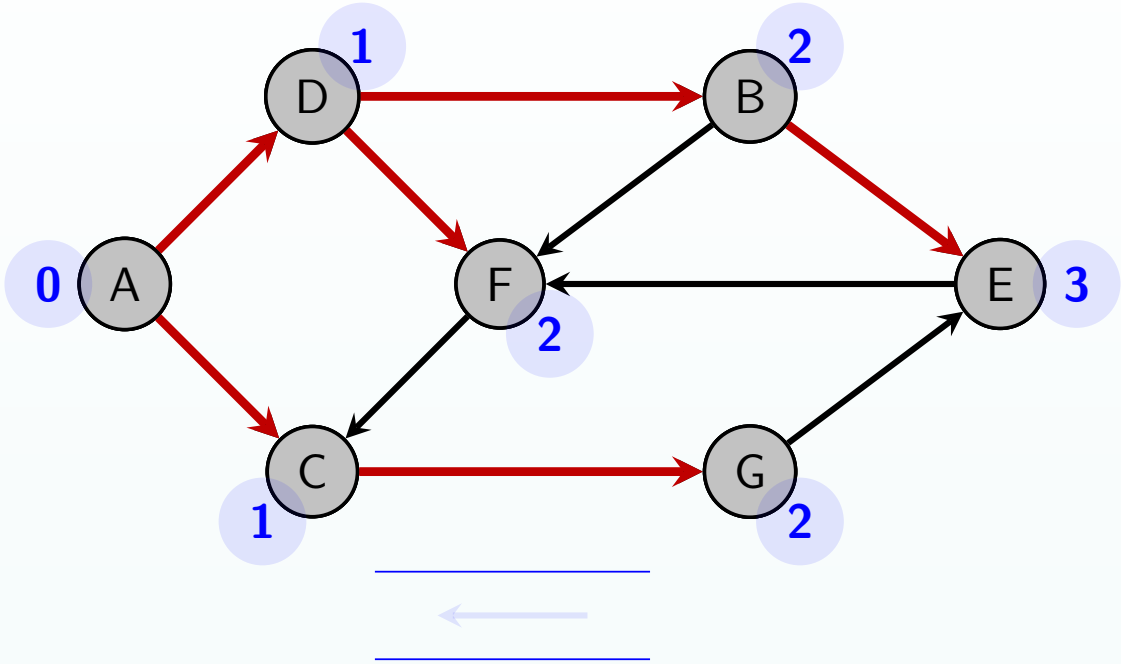
2: ouvrir et insérer s ; $d(s) \rightarrow 0$

3: tant que $File$ n'est pas vide, faire ;

4: ouvrir et insérer tous les sommets non marqués y successeurs de la tête de file x ; $d(y) = d(x) + 1$;

5: fermer et retirer x de la file ;

Parcours de graphe en profondeur : calcul de connexité



- Chaque sommet est associé sa distance du point de départ
- Point de départ déterminant
  - Une seule composante connexe parcourue
  - Sommet non marqué en fin d'algorithme : sommet inaccessible depuis s

Plan du cours

Origines

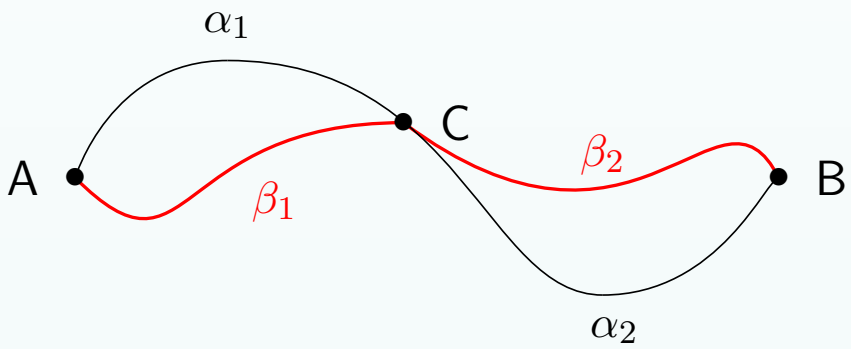
- Attribué à Richard Bellman ($\sim$ 1950)
  - Basée sur les travaux d'optique de Pierre de Fermat
- Résolution des problèmes chemins optimaux (min ou max)
- Énumération implicite : économie de calcul par *affaiblissement du caractère combinatoire* du problème
- S'appuie sur le principe d'optimalité

Principe d'optimalité

Définition

Toute partie (sous-chemin) d'un chemin optimal est, elle-même, optimale.

- Démonstration aisée par l'absurde



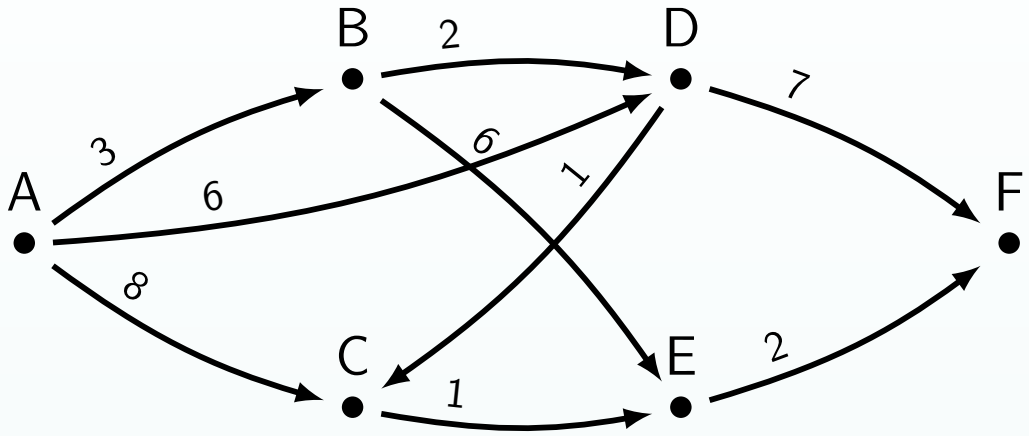
- Trouver une formulation récursive du problème

Domaines d'application

- Problèmes de distribution combinatoire
  - Problème de skis (éléments limités)
  - Problème du sac à dos / rendu de monnaie (éléments illimités)
- Problèmes d'algorithmique du texte
  - Calcul de la plus longue sous-suite commune
  - Calcul de la similarité (distance de Levenshtein)
  - Alignement de séquences (bio-informatique : algorithme de Smith-Waterman)

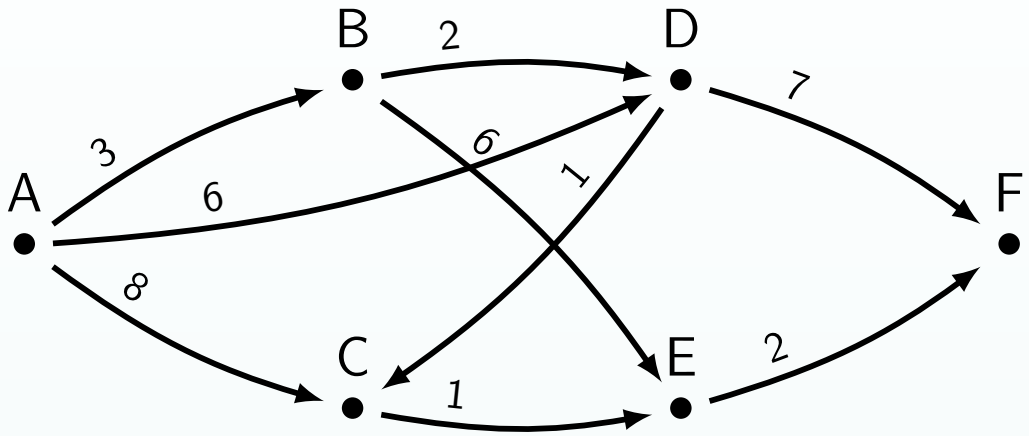
Tout algorithme de programmation dynamique peut se ramener à la recherche du plus court chemin dans un graphe (Martelli, 1976)

Exemple brut (Algorithme de Bellman)



- Une fois déterminé (A, B), plus court chemin entre A et B, et (A, B, D, C), plus court chemin entre A et C
- Le plus court chemin ente A et E se limite à la comparaison de (A, B, E) et (A, B, D, C, E)
  - (A, C, E) et (A, D, C, E) sont exclus avant calcul par le principe d'optimalité

Exemple brut (Algorithme de Bellman)



$$\mathcal{D}_A(F) = \min \begin{pmatrix} \mathcal{D}_A(D) + v(D,F) \\ \mathcal{D}_A(E) + v(E,F) \end{pmatrix}$$

$$\mathcal{D}_A(E) = \min \begin{pmatrix} \mathcal{D}_A(C) + v(C,E) \\ \mathcal{D}_A(B) + v(B,E) \end{pmatrix}$$

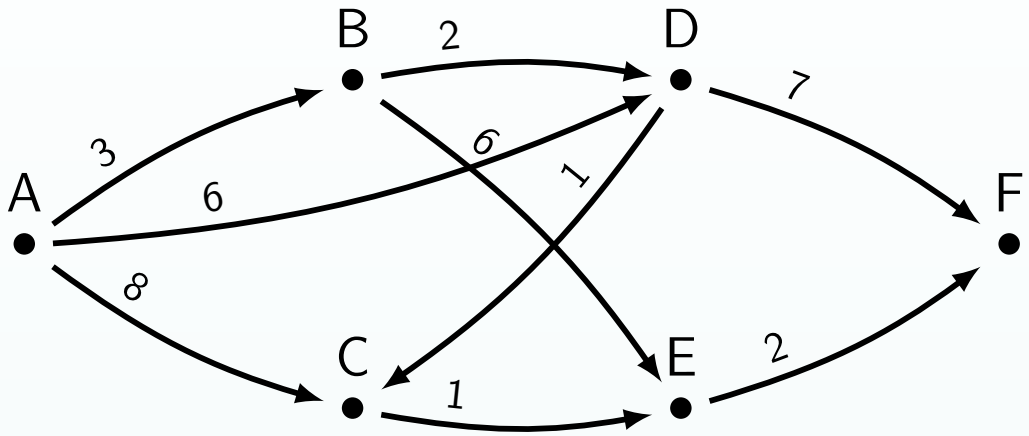
$$\mathcal{D}_A(D) = \min \begin{pmatrix} \mathcal{D}_A(B) + v(B,D) \\ \mathcal{D}_A(A) + v(A,D) \end{pmatrix}$$

$$\mathcal{D}_A(C) = \min \begin{pmatrix} \mathcal{D}_A(D) + v(D,C) \\ \mathcal{D}_A(A) + v(A,C) \end{pmatrix}$$

$$\mathcal{D}_A(B) = \mathcal{D}_A(A) + v(A,B)$$

$$\mathcal{D}_A(A) = 0$$

Exemple brut (Algorithme de Bellman)



$$\mathcal{D}_A(F) = \min \begin{pmatrix} \mathcal{D}_A(D) + 7 \\ \mathcal{D}_A(E) + 2 \end{pmatrix}$$

$$\mathcal{D}_A(E) = \min \begin{pmatrix} \mathcal{D}_A(C) + 1 \\ \mathcal{D}_A(B) + 6 \end{pmatrix}$$

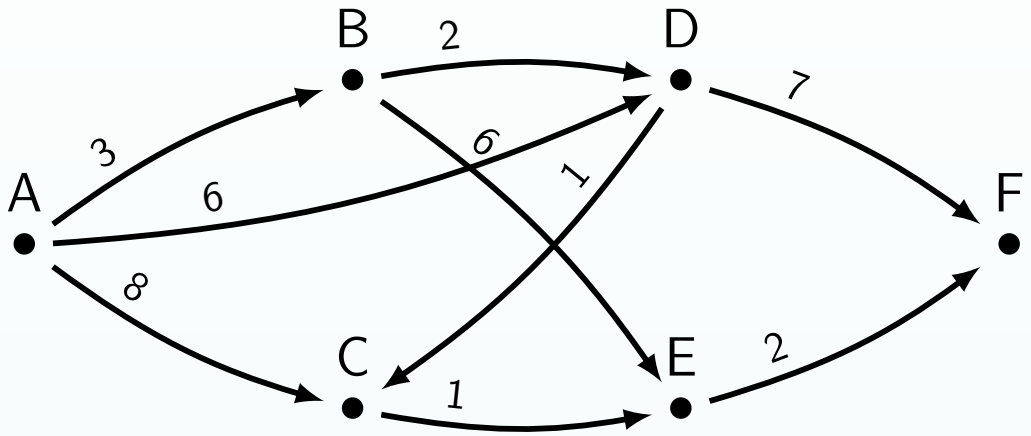
$$\mathcal{D}_A(D) = \min \begin{pmatrix} \mathcal{D}_A(B) + 2 \\ \mathcal{D}_A(A) + 6 \end{pmatrix}$$

$$\mathcal{D}_A(C) = \min \begin{pmatrix} \mathcal{D}_A(D) + 1 \\ \mathcal{D}_A(A) + 8 \end{pmatrix}$$

$$\mathcal{D}_A(B) = \mathcal{D}_A(A) + 3$$

$$\mathcal{D}_A(A) = 0$$

Exemple brut (Algorithme de Bellman)



$$\mathcal{D}_A(F) = \min \begin{pmatrix} \mathcal{D}_A(D) + 7 \\ \mathcal{D}_A(E) + 2 \end{pmatrix}$$

$$\mathcal{D}_A(E) = \min \begin{pmatrix} \mathcal{D}_A(C) + 1 \\ \mathcal{D}_A(B) + 6 \end{pmatrix}$$

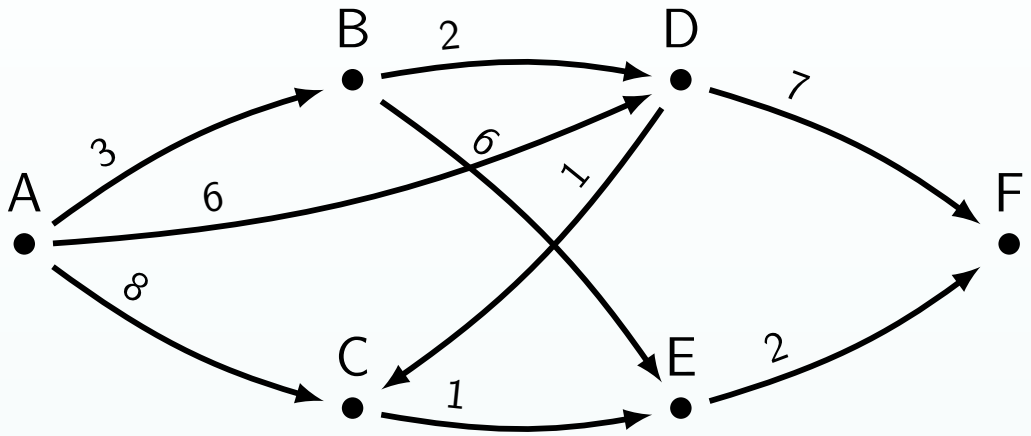
$$\mathcal{D}_A(D) = \min \begin{pmatrix} \mathcal{D}_A(B) + 2 \\ 0 + 6 \end{pmatrix}$$

$$\mathcal{D}_A(C) = \min \begin{pmatrix} \mathcal{D}_A(D) + 1 \\ 0 + 8 \end{pmatrix}$$

$$\mathcal{D}_A(B) = 0 + 3$$

$$\mathcal{D}_A(A) = 0$$

Exemple brut (Algorithme de Bellman)



$$\mathcal{D}_A(F) = \min \begin{pmatrix} \mathcal{D}_A(D) + 7 \\ \mathcal{D}_A(E) + 2 \end{pmatrix}$$

$$\mathcal{D}_A(E) = \min \begin{pmatrix} \mathcal{D}_A(C) + 1 \\ 3 + 6 \end{pmatrix}$$

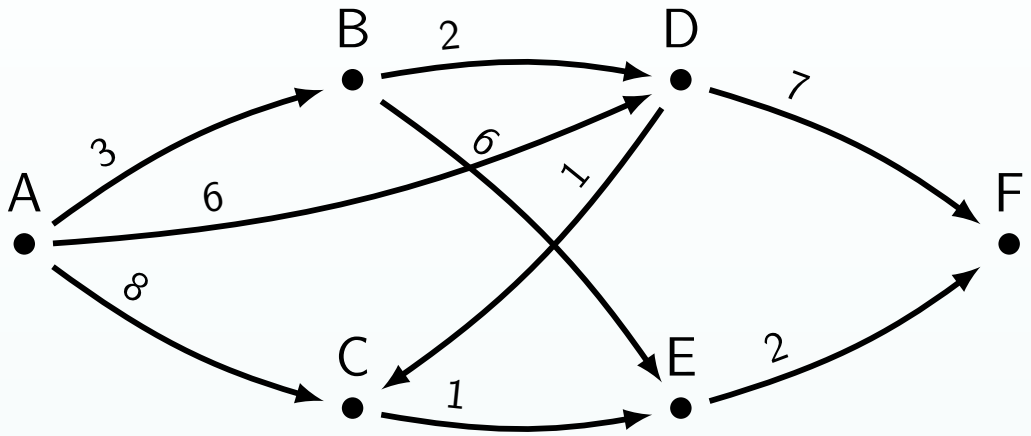
$$\mathcal{D}_A(D) = \min \begin{pmatrix} 3 + 2 \\ 0 + 6 \end{pmatrix}$$

$$\mathcal{D}_A(C) = \min \begin{pmatrix} \mathcal{D}_A(D) + 1 \\ 0 + 8 \end{pmatrix}$$

$$\mathcal{D}_A(B) = 3$$

$$\mathcal{D}_A(A) = 0$$

Exemple brut (Algorithme de Bellman)



$$\mathcal{D}_A(F) = \min \begin{pmatrix} 5+7 \\ \mathcal{D}_A(E)+2 \end{pmatrix}$$

$$\mathcal{D}_A(E) = \min \begin{pmatrix} \mathcal{D}_A(C)+1 \\ 3+6 \end{pmatrix}$$

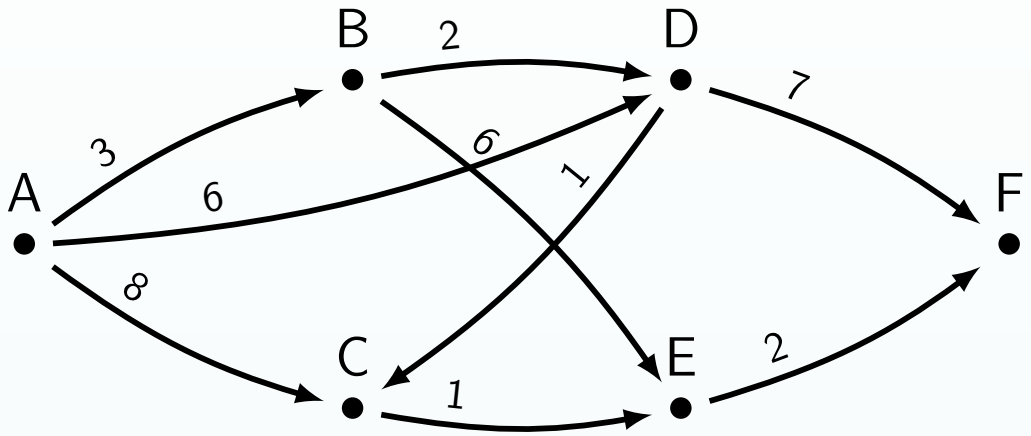
$$\mathcal{D}_A(D) = \min \begin{pmatrix} 3+2 \\ 0+6 \end{pmatrix} = 5$$

$$\mathcal{D}_A(C) = \min \begin{pmatrix} 5+1 \\ 0+8 \end{pmatrix}$$

$$\mathcal{D}_A(B) = 3$$

$$\mathcal{D}_A(A) = 0$$

Exemple brut (Algorithme de Bellman)



$$\mathcal{D}_A(F) = \min \begin{pmatrix} 5+7 \\ \mathcal{D}_A(E)+2 \end{pmatrix}$$

$$\mathcal{D}_A(E) = \min \begin{pmatrix} 6+1 \\ 3+6 \end{pmatrix}$$

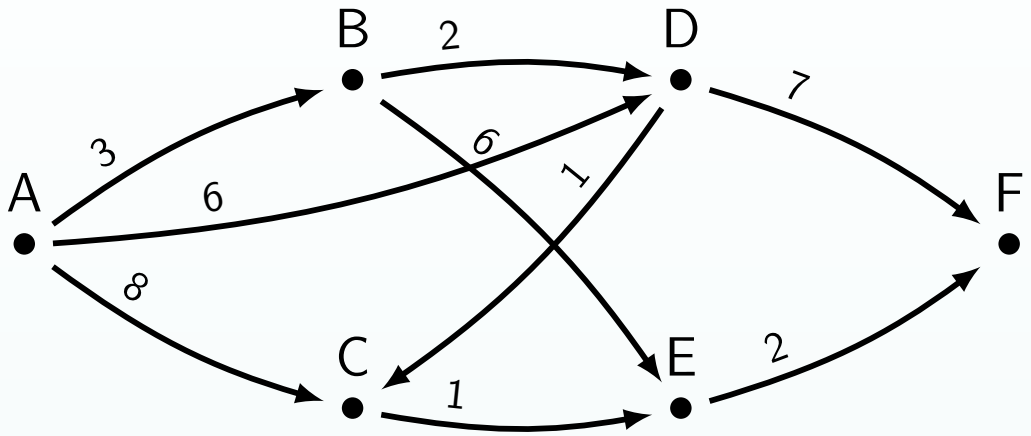
$$\mathcal{D}_A(D) = \min \begin{pmatrix} 3+2 \\ 0+6 \end{pmatrix} = 5$$

$$\mathcal{D}_A(C) = \min \begin{pmatrix} 5+1 \\ 0+8 \end{pmatrix} = 6$$

$$\mathcal{D}_A(B) = 3$$

$$\mathcal{D}_A(A) = 0$$

Exemple brut (Algorithme de Bellman)



$$\mathcal{D}_A(F) = \min \begin{pmatrix} 5+7 \\ 7+2 \end{pmatrix}$$

$$\mathcal{D}_A(E) = \min \begin{pmatrix} 6+1 \\ 3+6 \end{pmatrix} = 7$$

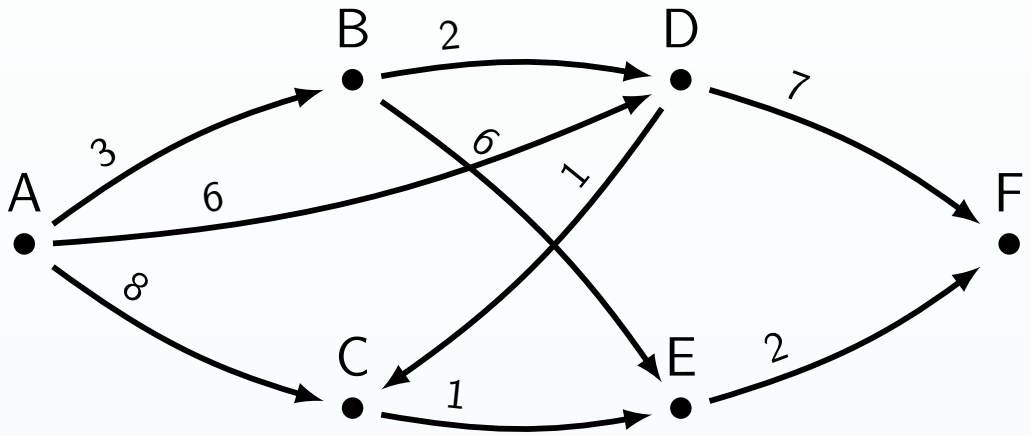
$$\mathcal{D}_A(D) = \min \begin{pmatrix} 3+2 \\ 0+6 \end{pmatrix} = 5$$

$$\mathcal{D}_A(C) = \min \begin{pmatrix} 5+1 \\ 0+8 \end{pmatrix} = 6$$

$$\mathcal{D}_A(B) = 3$$

$$\mathcal{D}_A(A) = 0$$

Exemple brut (Algorithme de Bellman)



$$\mathcal{D}_A(F) = \min \begin{pmatrix} 5+7 \\ 7+2 \end{pmatrix} = 9$$

$$\mathcal{D}_A(E) = \min \begin{pmatrix} 6+1 \\ 3+6 \end{pmatrix} = 7$$

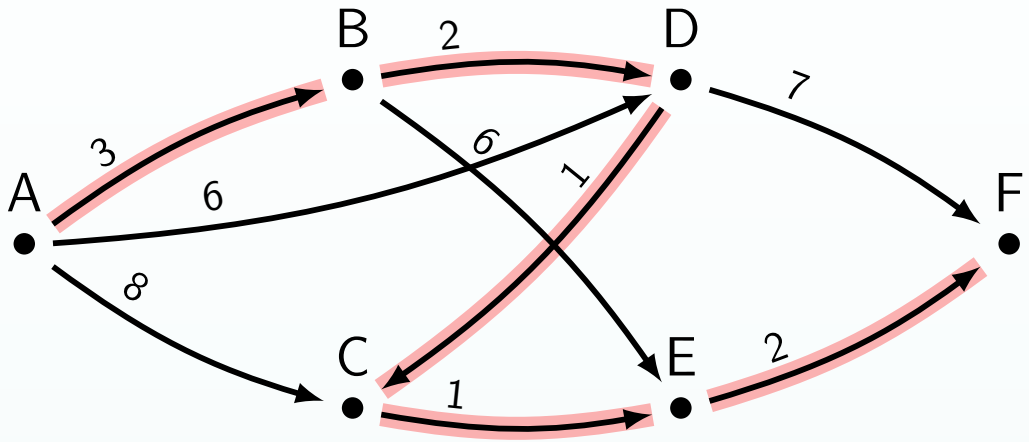
$$\mathcal{D}_A(D) = \min \begin{pmatrix} 3+2 \\ 0+6 \end{pmatrix} = 5$$

$$\mathcal{D}_A(C) = \min \begin{pmatrix} 5+1 \\ 0+8 \end{pmatrix} = 6$$

$$\mathcal{D}_A(B) = 3$$

$$\mathcal{D}_A(A) = 0$$

Exemple brut (Algorithme de Bellman)



$$\mathcal{D}_A(F) = \min \begin{pmatrix} \mathcal{D}_A(D) + v(D,F) \\ \mathcal{D}_A(E) + v(E,F) \end{pmatrix} =$$

9

$$\mathcal{D}_A(E) = \min \begin{pmatrix} \mathcal{D}_A(C) + v(C,E) \\ \mathcal{D}_A(B) + v(B,E) \end{pmatrix} =$$

7

$$\mathcal{D}_A(D) = \min \begin{pmatrix} \mathcal{D}_A(B) + v(B,D) \\ \mathcal{D}_A(A) + v(A,D) \end{pmatrix} =$$

5

$$\mathcal{D}_A(C) = \min \begin{pmatrix} \mathcal{D}_A(D) + v(D,C) \\ \mathcal{D}_A(A) + v(A,C) \end{pmatrix} = 6$$

$$\mathcal{D}_A(B) = v(A,B)$$

$$\mathcal{D}_A(A) = 0$$

Algorithme de Bellman

1: initialement $\mathcal{F}(x_0) = 0$; $\forall y \neq x_0, \mathcal{F}(y) = +\infty$
2: pour k de 1 à $N-1$
3: Pour tout sommet y
4: $\mathcal{F}'(y) = \min(\mathcal{F}(z) + l(z,y); z \in \mathcal{P}(y))$
5: $\mathcal{F} = \mathcal{F}'$

- Plus court chemin vers l'ensemble des sommets
- Possibilité de circuit non absorbant
- Inversable
- Complexité $O(n^2)$

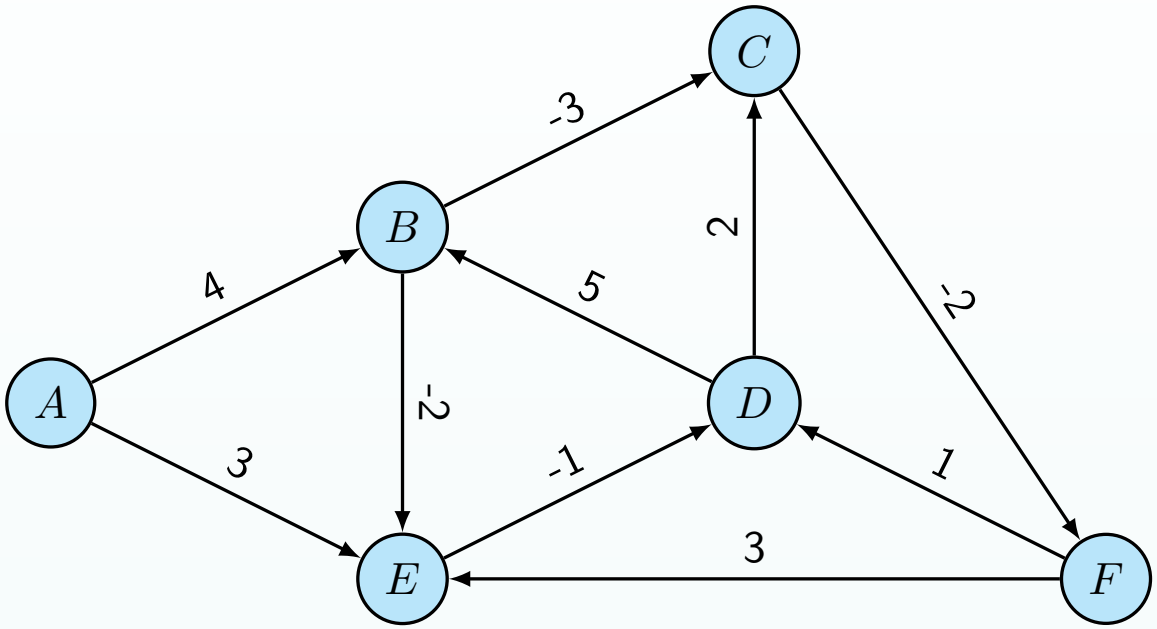
Plan du cours

Définition

- Problèmes de cheminement
  - Minimiser ou maximiser une fonction donnée
  - Cas classiques : longueur ou valeur des arcs
- De nombreux algorithmes
  - Propriétés variables (compatibilité avec les circuits, valeurs négatives, complexité, etc.)

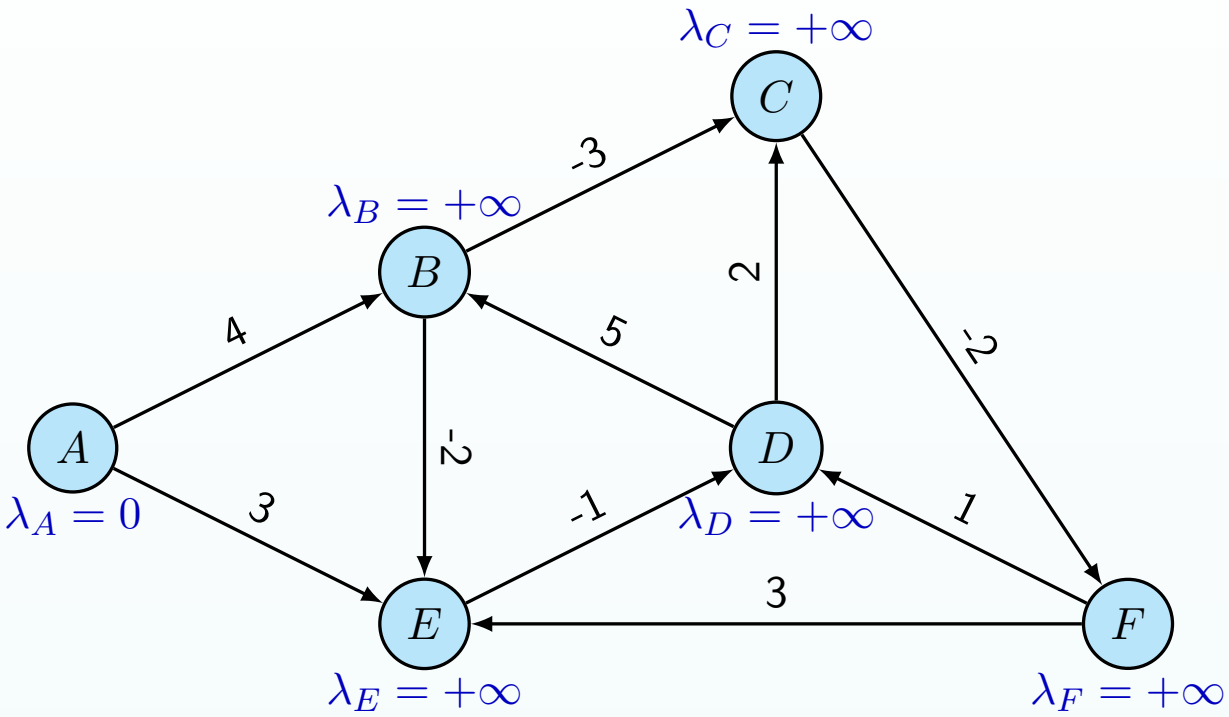
Plan du cours

Algorithme de Ford



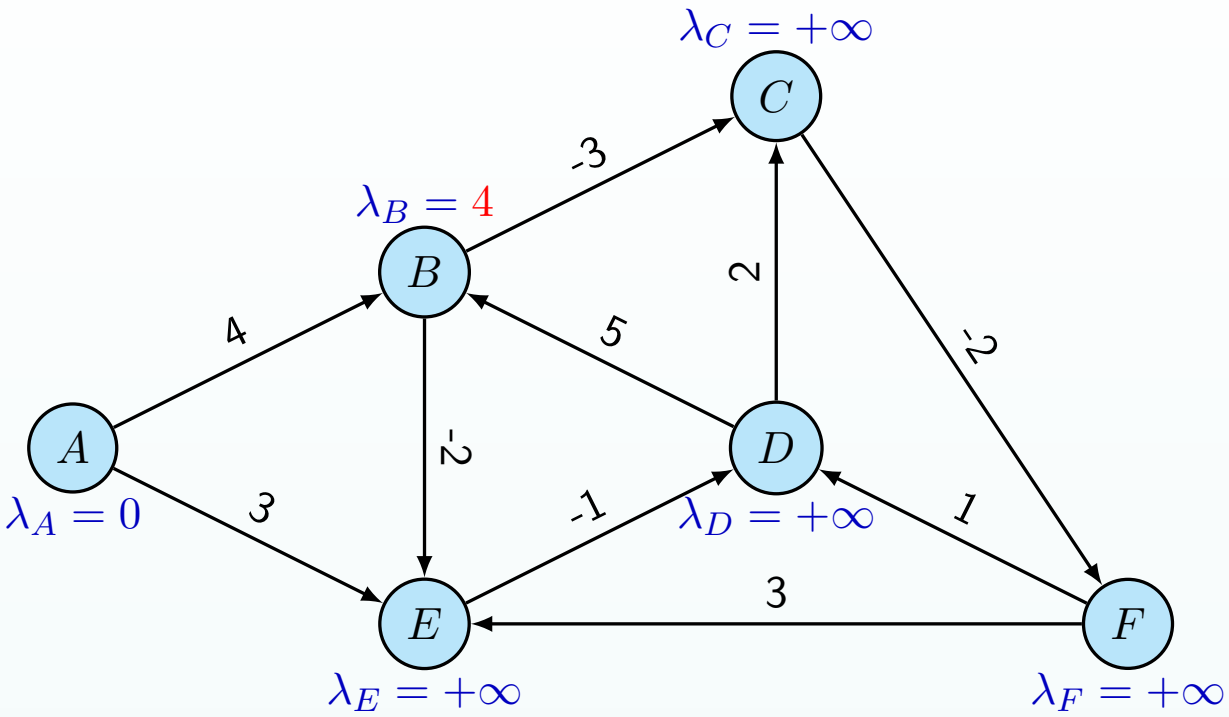
1: numéroter les sommets dans un ordre quelconque, avec X_0 sommet de départ et X_{n-1} sommet d'arrivée
2: $\lambda_0 \leftarrow 0$; $\forall i \neq 0, \lambda_i \leftarrow +\infty$
3: tant qu'il existe un arc (X_i, X_j) tel que $\lambda_j - \lambda_i > v(X_i, X_j)$
4: $\quad \lambda_j \leftarrow \lambda_i + v(X_i, X_j)$

Algorithme de Ford



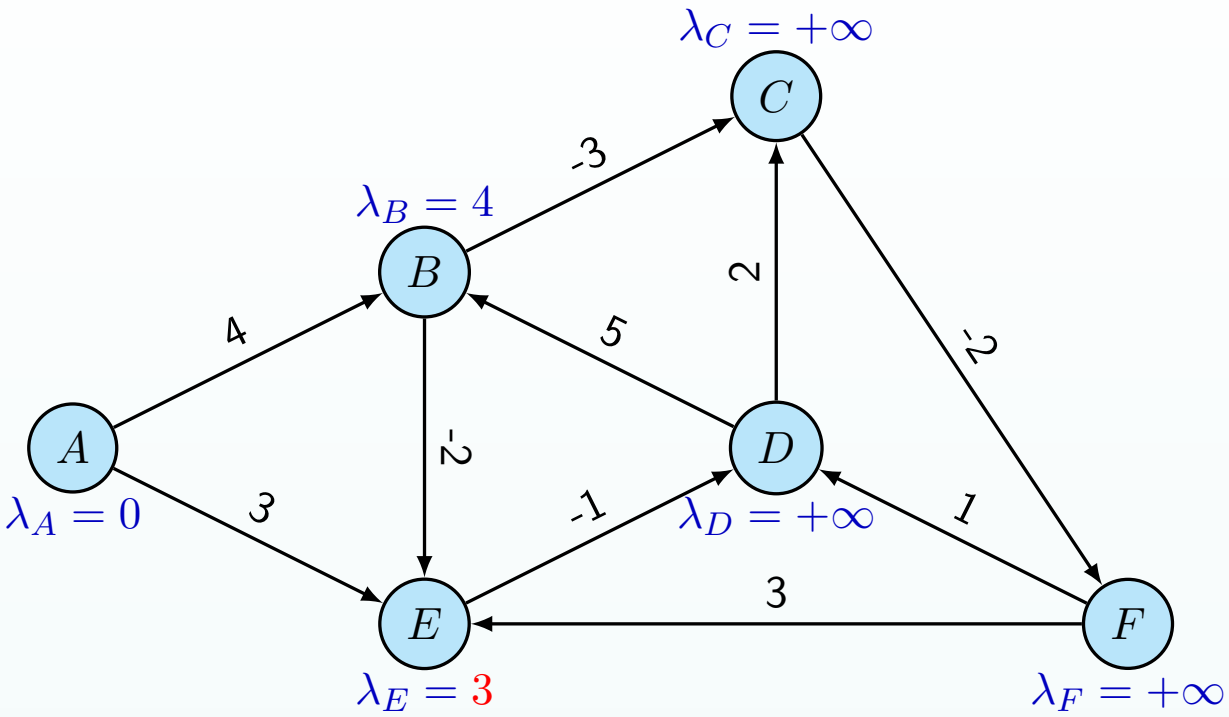
1: numéroter les sommets dans un ordre quelconque, avec X_0 sommet de départ et X_{n-1} sommet d'arrivée
2: $\lambda_0 \leftarrow 0$; $\forall i \neq 0, \lambda_i \leftarrow +\infty$
3: tant qu'il existe un arc (X_i, X_j) tel que $\lambda_j - \lambda_i > v(X_i, X_j)$
4: $\quad \lambda_j \leftarrow \lambda_i + v(X_i, X_j)$

Algorithme de Ford



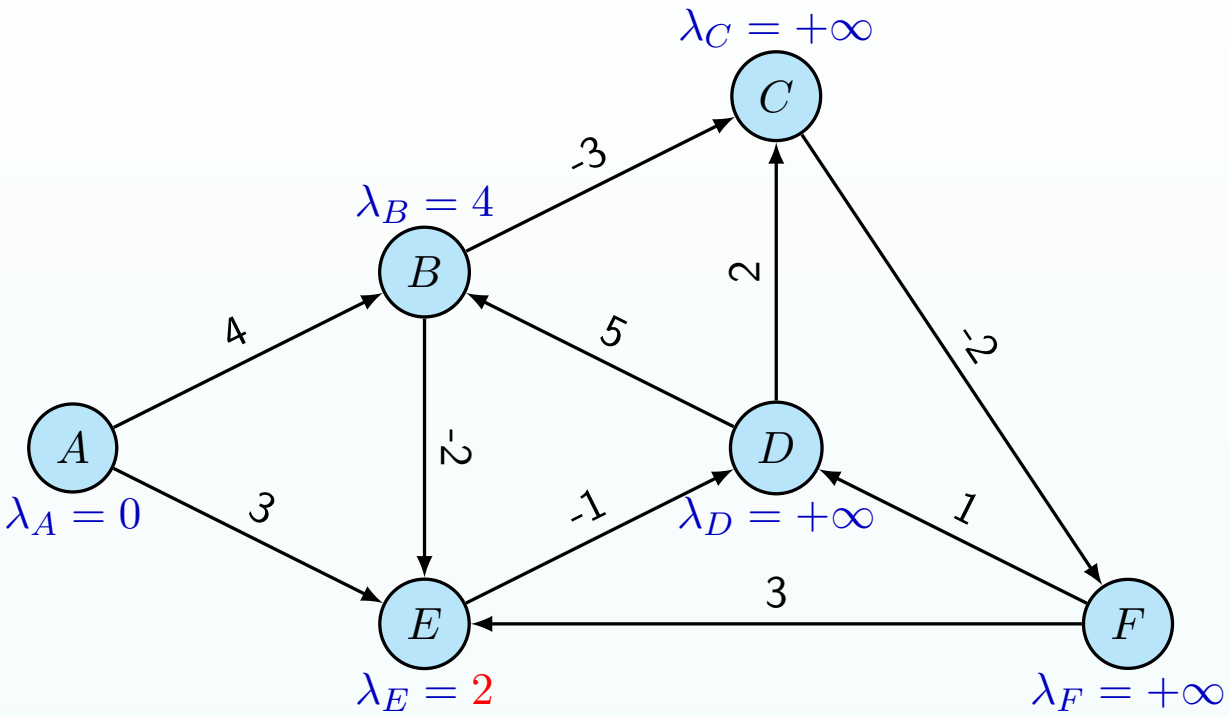
1: numéroter les sommets dans un ordre quelconque, avec X_0 sommet de départ et X_{n-1} sommet d'arrivée
2: $\lambda_0 \leftarrow 0$; $\forall i \neq 0, \lambda_i \leftarrow +\infty$
3: tant qu'il existe un arc (X_i, X_j) tel que $\lambda_j - \lambda_i > v(X_i, X_j)$
4: $\quad \lambda_j \leftarrow \lambda_i + v(X_i, X_j)$

Algorithme de Ford



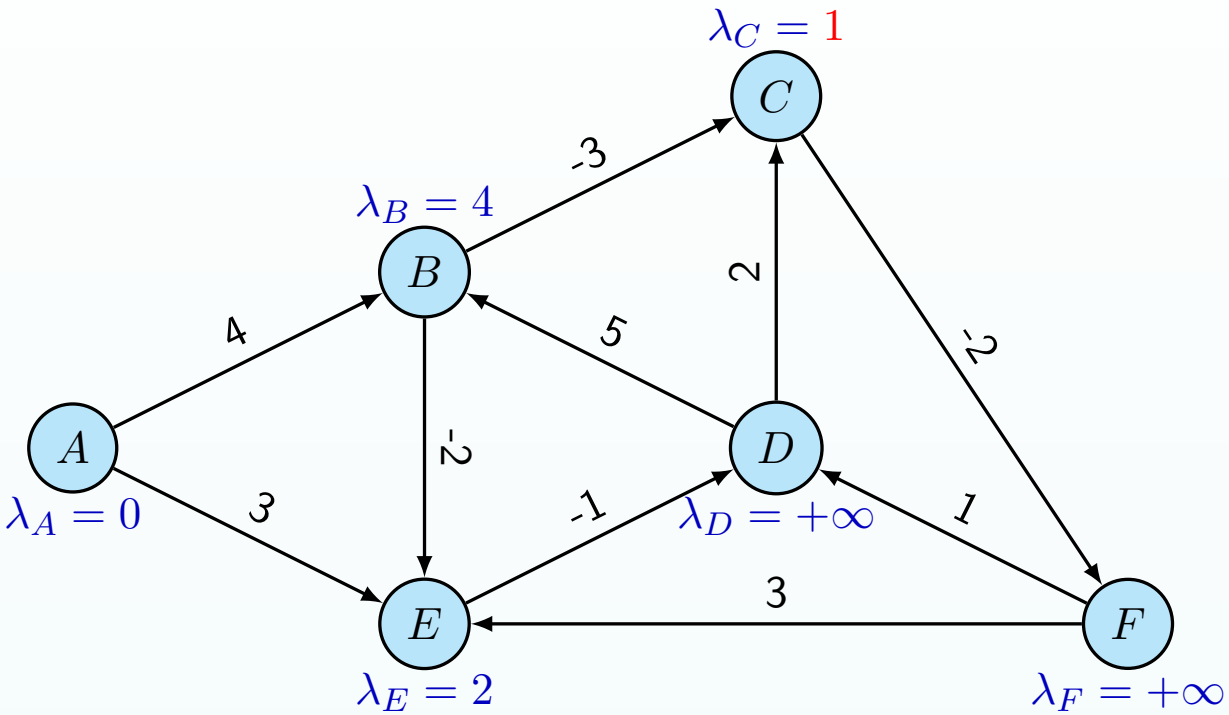
1: numéroter les sommets dans un ordre quelconque, avec X_0 sommet de départ et X_{n-1} sommet d'arrivée
2: $\lambda_0 \leftarrow 0$; $\forall i \neq 0, \lambda_i \leftarrow +\infty$
3: tant qu'il existe un arc (X_i, X_j) tel que $\lambda_j - \lambda_i > v(X_i, X_j)$
4: $\quad \lambda_j \leftarrow \lambda_i + v(X_i, X_j)$

Algorithme de Ford



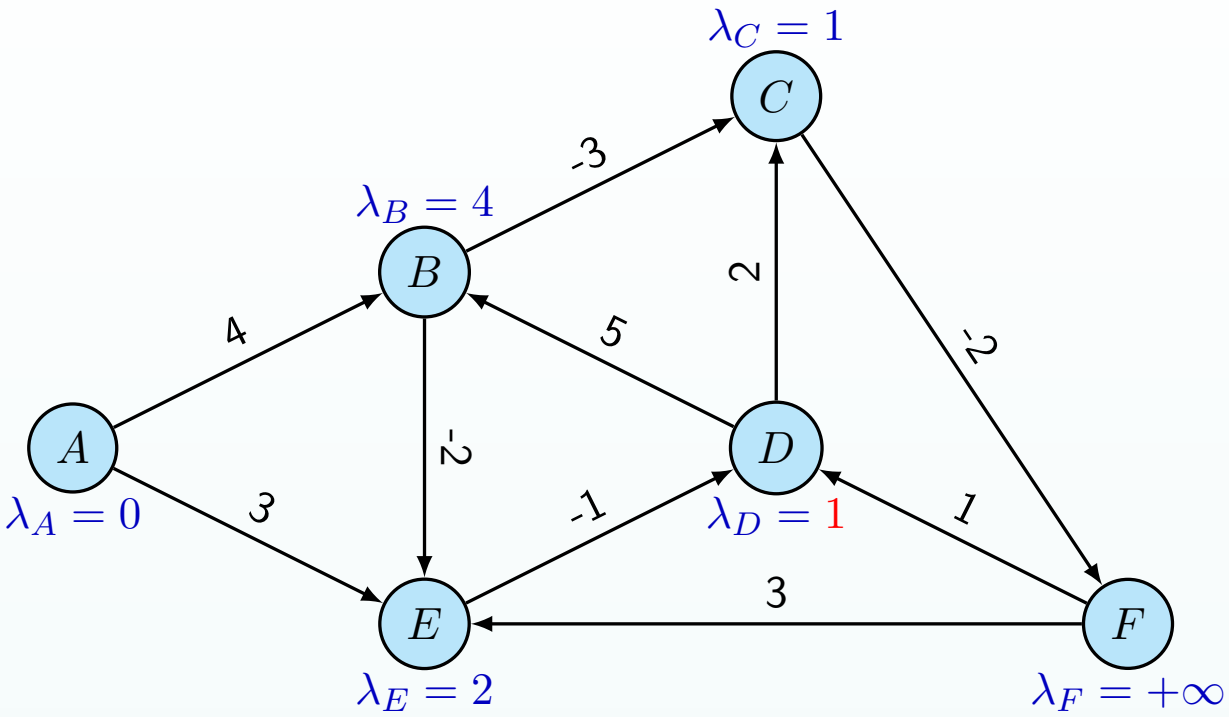
1: numéroter les sommets dans un ordre quelconque, avec X_0 sommet de départ et X_{n-1} sommet d'arrivée
2: $\lambda_0 \leftarrow 0$; $\forall i \neq 0, \lambda_i \leftarrow +\infty$
3: tant qu'il existe un arc (X_i, X_j) tel que $\lambda_j - \lambda_i > v(X_i, X_j)$
4: $\quad \lambda_j \leftarrow \lambda_i + v(X_i, X_j)$

Algorithme de Ford



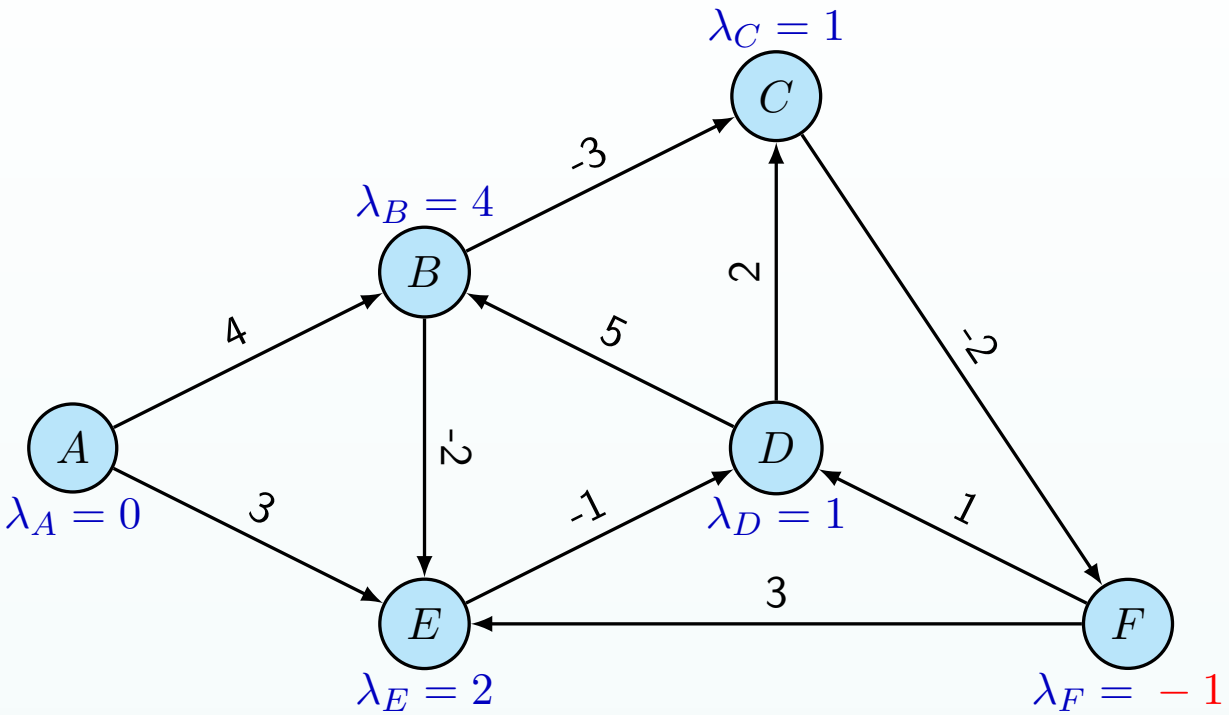
1: numéroter les sommets dans un ordre quelconque, avec X_0 sommet de départ et X_{n-1} sommet d'arrivée
2: $\lambda_0 \leftarrow 0$; $\forall i \neq 0, \lambda_i \leftarrow +\infty$
3: tant qu'il existe un arc (X_i, X_j) tel que $\lambda_j - \lambda_i > v(X_i, X_j)$
4: $\quad \lambda_j \leftarrow \lambda_i + v(X_i, X_j)$

Algorithme de Ford



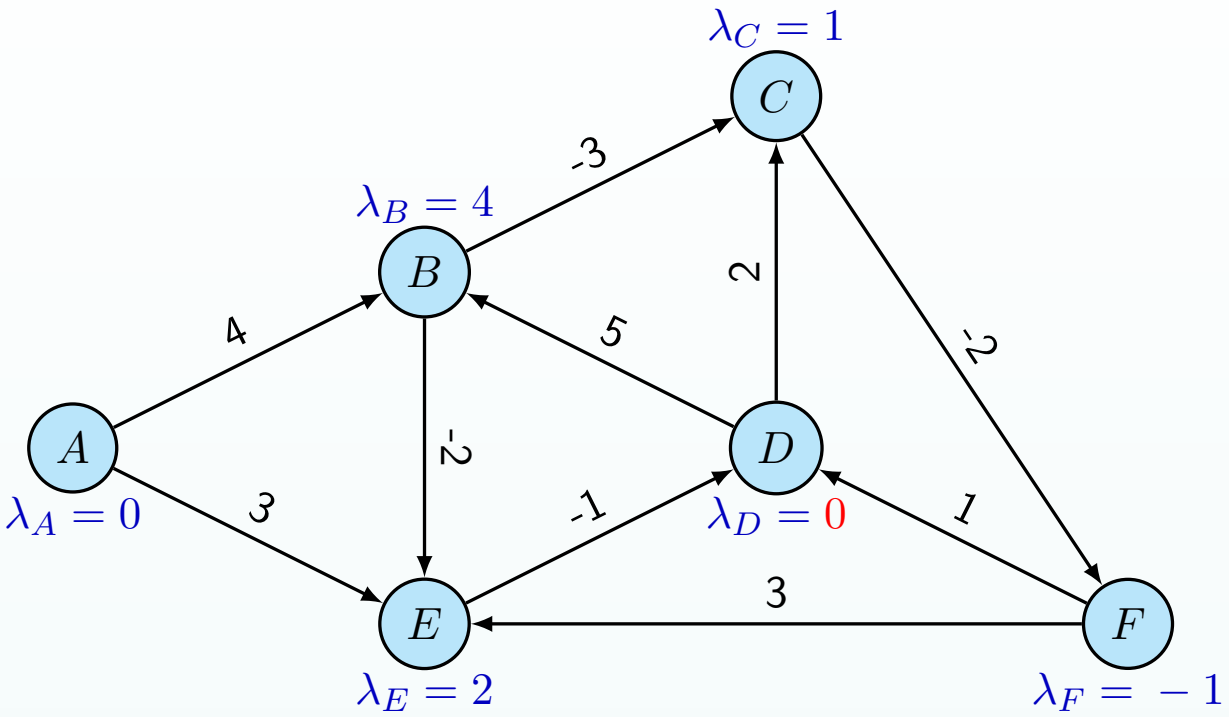
1: numéroter les sommets dans un ordre quelconque, avec X_0 sommet de départ et X_{n-1} sommet d'arrivée
2: $\lambda_0 \leftarrow 0$; $\forall i \neq 0, \lambda_i \leftarrow +\infty$
3: tant qu'il existe un arc (X_i, X_j) tel que $\lambda_j - \lambda_i > v(X_i, X_j)$
4: $\quad \lambda_j \leftarrow \lambda_i + v(X_i, X_j)$

Algorithme de Ford



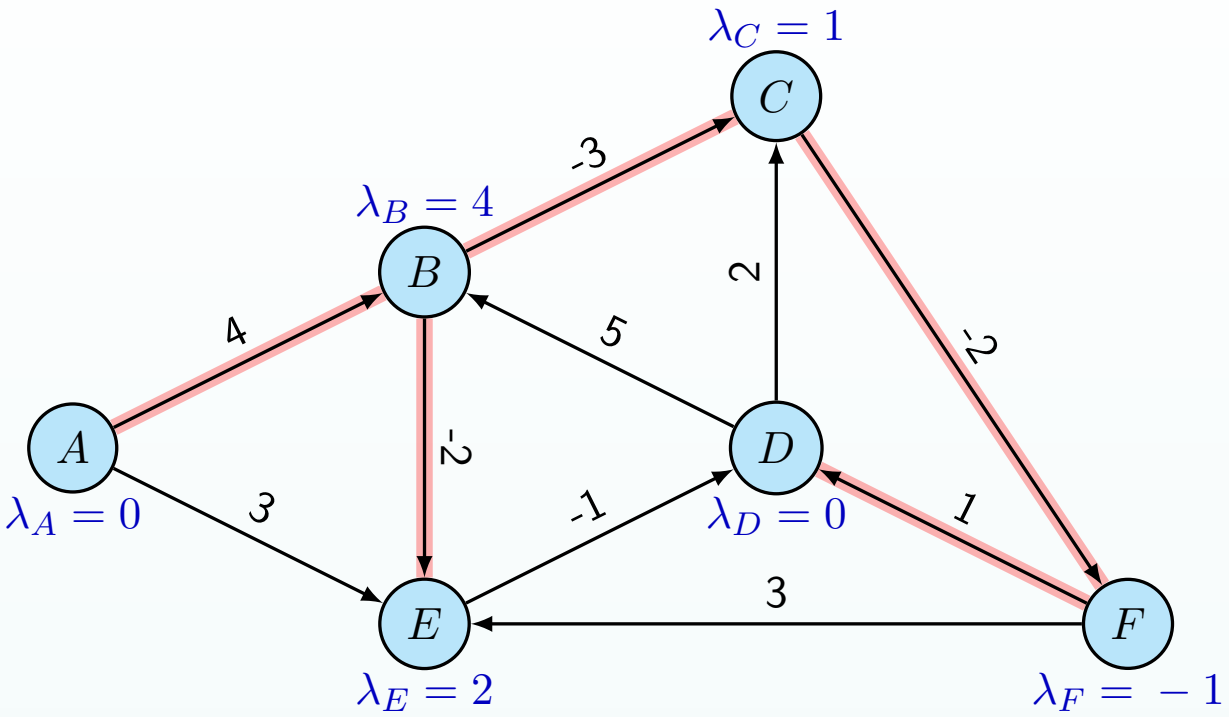
1: numéroter les sommets dans un ordre quelconque, avec X_0 sommet de départ et X_{n-1} sommet d'arrivée
2: $\lambda_0 \leftarrow 0$; $\forall i \neq 0, \lambda_i \leftarrow +\infty$
3: tant qu'il existe un arc (X_i, X_j) tel que $\lambda_j - \lambda_i > v(X_i, X_j)$
4: $\quad \lambda_j \leftarrow \lambda_i + v(X_i, X_j)$

Algorithme de Ford



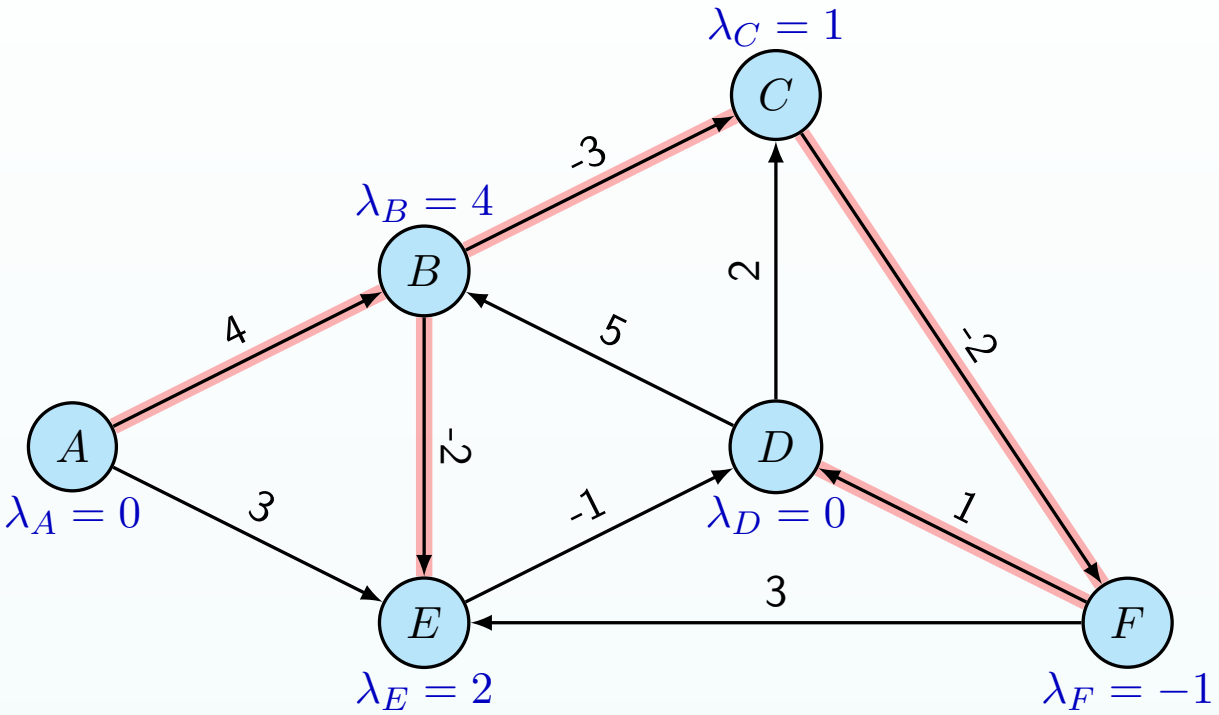
1: numéroter les sommets dans un ordre quelconque, avec X_0 sommet de départ et X_{n-1} sommet d'arrivée
2: $\lambda_0 \leftarrow 0$; $\forall i \neq 0, \lambda_i \leftarrow +\infty$
3: tant qu'il existe un arc (X_i, X_j) tel que $\lambda_j - \lambda_i > v(X_i, X_j)$
4: $\quad \lambda_j \leftarrow \lambda_i + v(X_i, X_j)$

Algorithme de Ford



1: numéroter les sommets dans un ordre quelconque, avec X_0 sommet de départ et X_{n-1} sommet d'arrivée
2: $\lambda_0 \leftarrow 0$; $\forall i \neq 0, \lambda_i \leftarrow +\infty$
3: tant qu'il existe un arc (X_i, X_j) tel que $\lambda_j - \lambda_i > v(X_i, X_j)$
4: $\quad \lambda_j \leftarrow \lambda_i + v(X_i, X_j)$

Algorithme de Ford



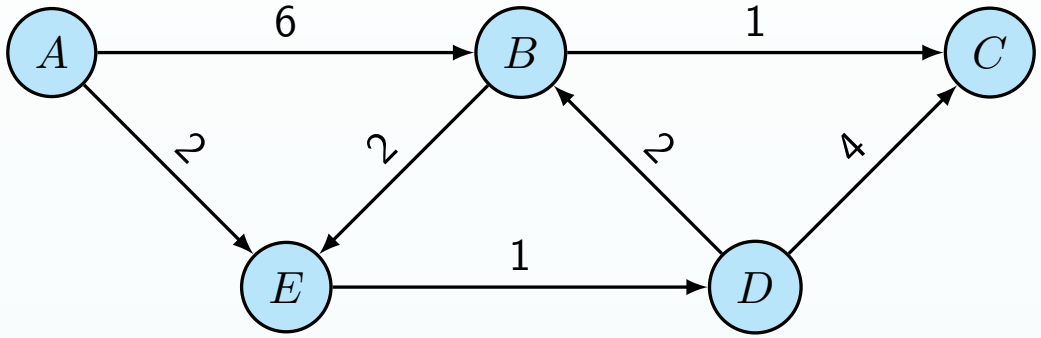
- Valeur des chemins optimaux vers chaque sommet
- *Arborescence des plus cours chemins*
- Complexité : $O(2^n)$
  - Algorithmes polynomiaux en ordonnant le traitement des arcs

Plan du cours

Algorithme de Dijkstra

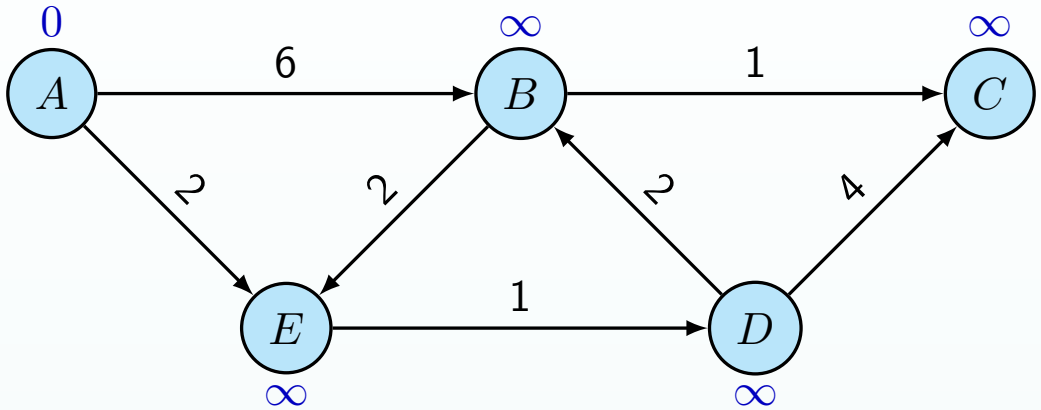
- Ordonner le parcours des arcs
- Restriction : Valuations strictement positives des arcs
  - Boucle absorbante impossible
- Complexité polynomiale
  - Utilisé pour le routage IP (*Open Shortest Path First*)

Algorithme de Dijkstra



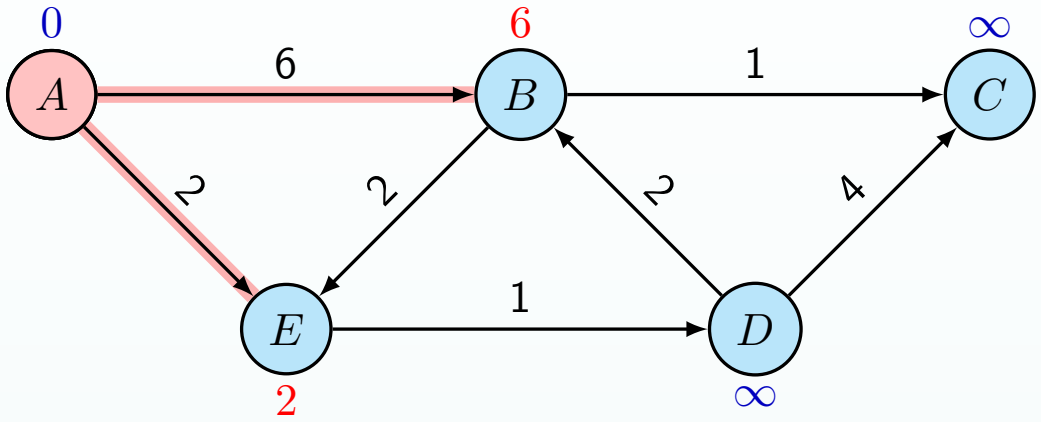
1: $\lambda_s \leftarrow 0$; $\forall i \neq s, \lambda_i \leftarrow +\infty$
2: tant qu'il existe un sommet i non traité de valeur λ_i minimale parmi les sommets non traités
3: pour tout arc (i, j)
4: si $\lambda_i + v(i, j) < \lambda_j$
5: $\lambda_j \leftarrow \lambda_i + v(i, j)$
6: i est traité

Algorithme de Dijkstra



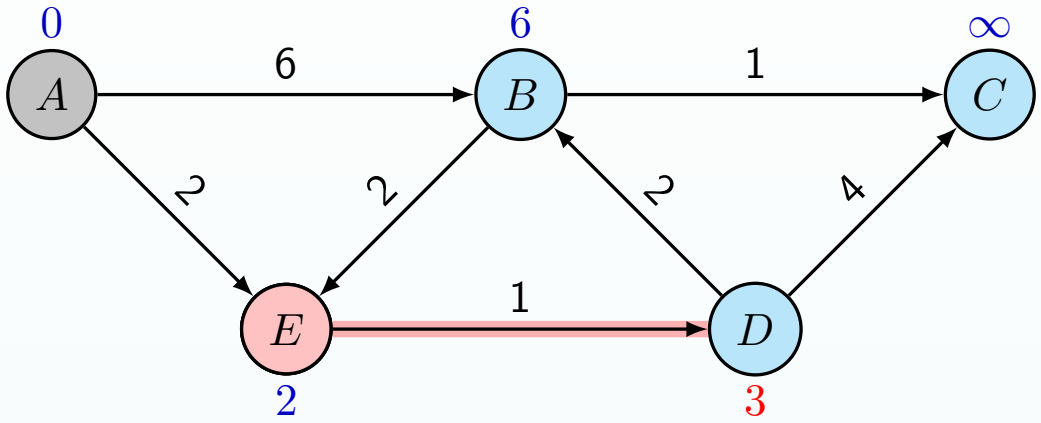
1: $\lambda_s \leftarrow 0$; $\forall i \neq s, \lambda_i \leftarrow +\infty$
2: tant qu'il existe un sommet i non traité de valeur λ_i minimale parmi les sommets non traités
3: pour tout arc (i, j)
4: si $\lambda_i + v(i, j) < \lambda_j$
5: $\lambda_j \leftarrow \lambda_i + v(i, j)$
6: i est traité

Algorithme de Dijkstra



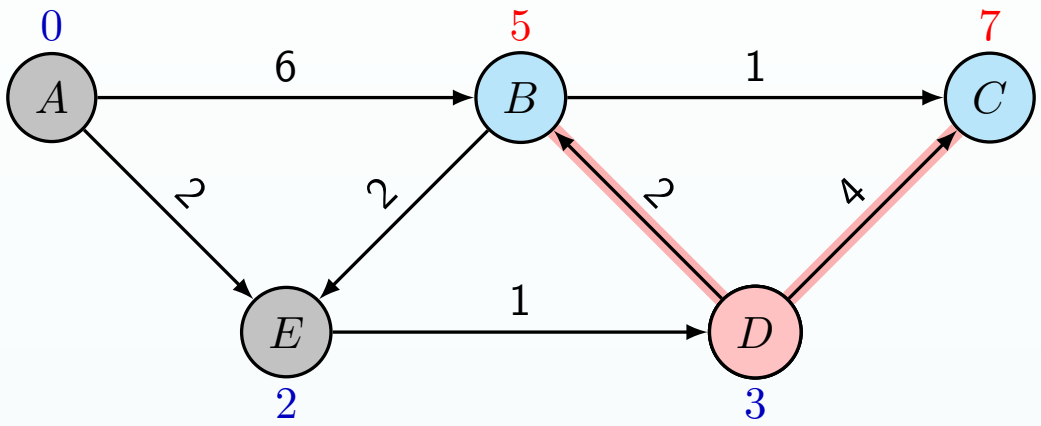
1: $\lambda_s \leftarrow 0$; $\forall i \neq s, \lambda_i \leftarrow +\infty$
2: tant qu'il existe un sommet i non traité de valeur λ_i minimale parmi les sommets non traités
3: pour tout arc (i, j)
4: si $\lambda_i + v(i, j) < \lambda_j$
5: $\lambda_j \leftarrow \lambda_i + v(i, j)$
6: i est traité

Algorithme de Dijkstra



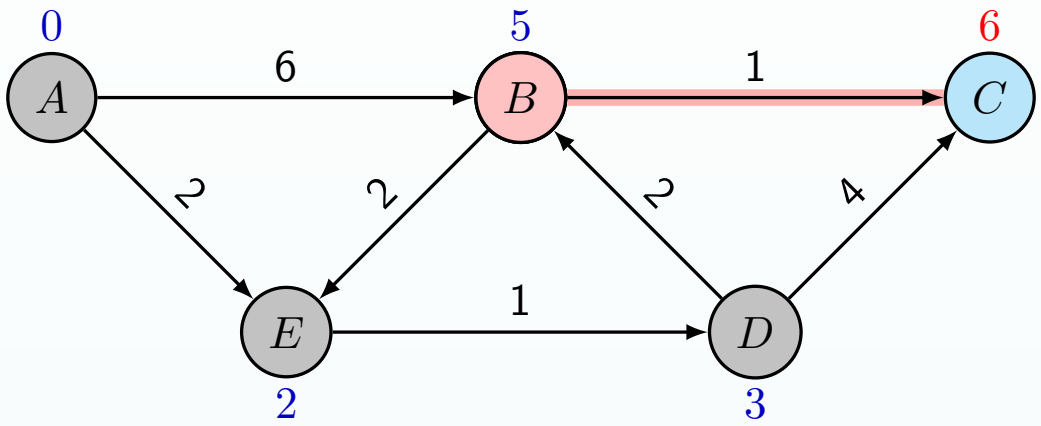
1: $\lambda_s \leftarrow 0$; $\forall i \neq s, \lambda_i \leftarrow +\infty$
2: tant qu'il existe un sommet i non traité de valeur λ_i minimale parmi les sommets non traités
3: pour tout arc (i, j)
4: si $\lambda_i + v(i, j) < \lambda_j$
5: $\lambda_j \leftarrow \lambda_i + v(i, j)$
6: i est traité

Algorithme de Dijkstra



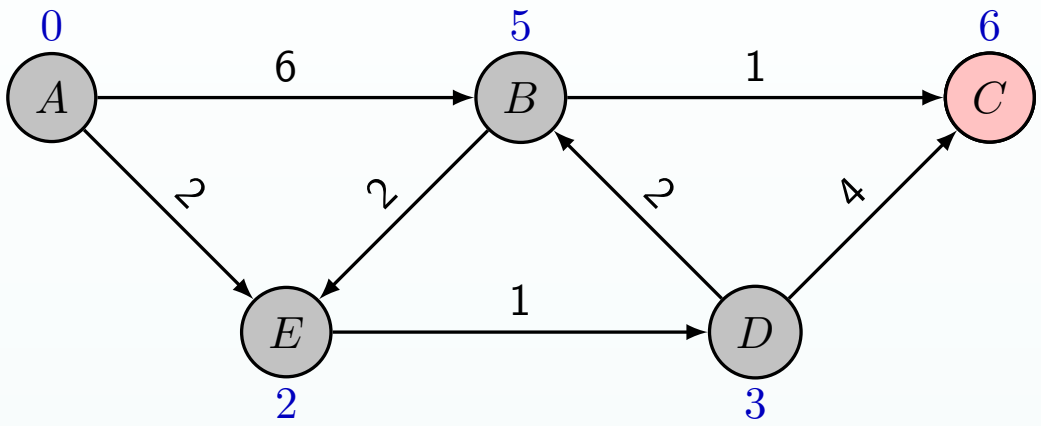
1: $\lambda_s \leftarrow 0$; $\forall i \neq s, \lambda_i \leftarrow +\infty$
2: tant qu'il existe un sommet i non traité de valeur λ_i minimale parmi les sommets non traités
3: pour tout arc (i, j)
4: si $\lambda_i + v(i, j) < \lambda_j$
5: $\lambda_j \leftarrow \lambda_i + v(i, j)$
6: i est traité

Algorithme de Dijkstra



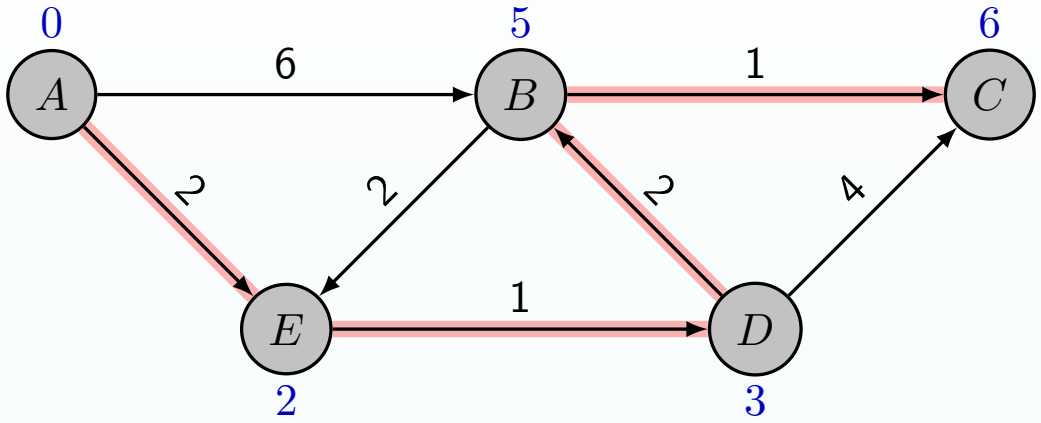
1: $\lambda_s \leftarrow 0$; $\forall i \neq s, \lambda_i \leftarrow +\infty$
2: tant qu'il existe un sommet i non traité de valeur λ_i minimale parmi les sommets non traités
3: pour tout arc (i, j)
4: si $\lambda_i + v(i, j) < \lambda_j$
5: $\lambda_j \leftarrow \lambda_i + v(i, j)$
6: i est traité

Algorithme de Dijkstra



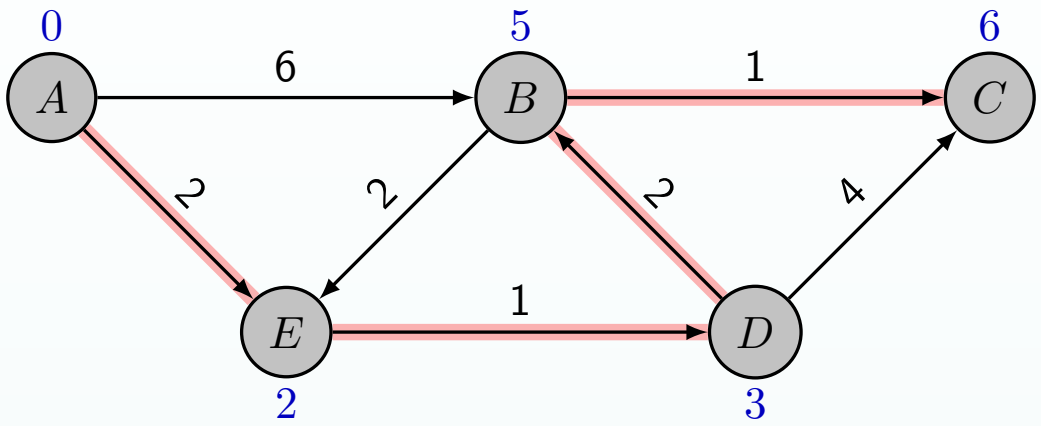
1: $\lambda_s \leftarrow 0$; $\forall i \neq s, \lambda_i \leftarrow +\infty$
2: tant qu'il existe un sommet i non traité de valeur λ_i minimale parmi les sommets non traités
3: pour tout arc (i, j)
4: si $\lambda_i + v(i, j) < \lambda_j$
5: $\lambda_j \leftarrow \lambda_i + v(i, j)$
6: i est traité

Algorithme de Dijkstra



1: $\lambda_s \leftarrow 0$; $\forall i \neq s, \lambda_i \leftarrow +\infty$
2: tant qu'il existe un sommet i non traité de valeur λ_i minimale parmi les sommets non traités
3: pour tout arc (i, j)
4: si $\lambda_i + v(i, j) < \lambda_j$
5: $\lambda_j \leftarrow \lambda_i + v(i, j)$
6: i est traité

Algorithme de Dijkstra



- Complexité polynomiale $O(m \log n)$
- Application impossible à la recherche de chemins de valeur maximale
- Représentation tabulaire : lecture immédiate de tous les plus cours chemins issus de s

Plan du cours

Méthode matricielle : Algorithme de Floyd-Warshall

- Rechercher tous les plus cours chemins entre tous les sommets du graphe
- Complexité $O(n^3)$

$$D_O = (v'_{i,j})_{1 \leq i \leq n, 1 \leq j \leq n} = \begin{cases} v(X_i, X_j) & \text{si } (X_i, X_j) \in U \\ +\infty & \text{si } (X_i, X_j) \notin U \end{cases}$$

À chaque étape $k \geq 1$:

$$v'^{(k-1)}_{i,j} = v^{(k-1)}_{i,k} + v^{(k-1)}_{k,j}$$

$$v^{(k)}_{i,j} = \min \{v'^{(k-1)}_{i,j}; v^{(k-1)}_{i,j}\}$$

On a donc : $D_k = \min \{D'_{k-1}; D_{k-1}\}$, et D_n matrice des plus courts chemins de G.

Méthode matricielle : Algorithme de Floyd-Warshall

- Représentation algorithmique :

1: pour $k = 1$ à n
2: pour $i = 1$ à n
3: pour $j = 1$ à n
4: $v_{ij} \leftarrow \min(v_{ik} + v_{kj}), v_{ij}$

- Balayage unique de la matrice (pas d'instances multiples)
- Extension optimale de sous-chemins optimaux (programmation dynamique)

Plan du cours

Définition

- Modéliser un réseau de transport et les contraintes qui y sont liées
  - Réseau routier
  - Réseau électrique
  - Réseau de distribution d'eau
  - ...
- Optimisation du flot
- Recherche d'éléments-clés

Réseau de transport

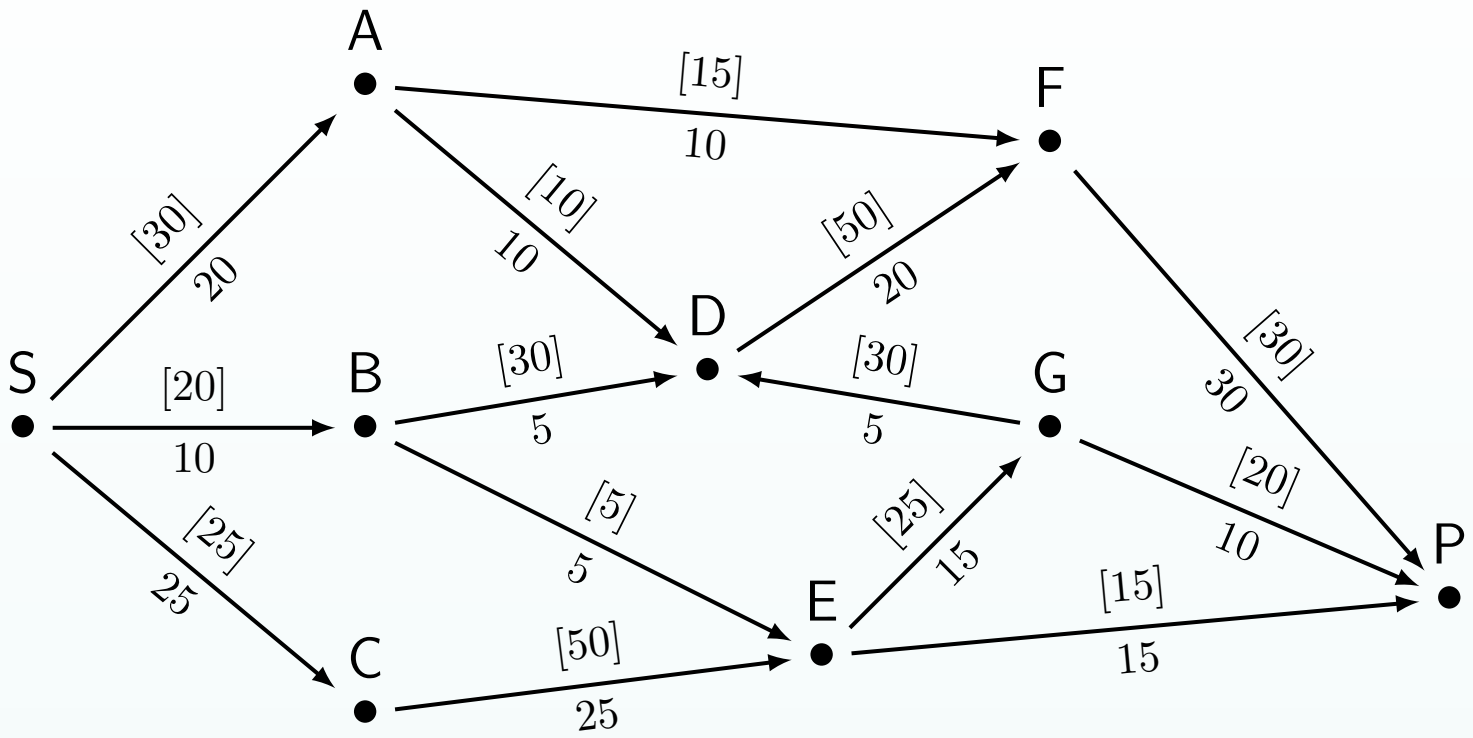
Définition

On appelle *réseau de transport* un graphe valué orienté G de n sommets sans boucle, comprenant deux sommets x_1 et x_n tel que pour tout sommet x_k de G, il existe au moins un chemin de x_1 vers x_n passant par x_k.

- x_1 est appelé *source* et x_n *puits* du graphe
- La valuation de l'arc u, noté $c(u)$ est la capacité de l'arc
- On associe à chaque arc un flot φ, tel que $0 \leq \varphi(u) \leq c(u)$
- Les flots doivent respecter la loi de Kirchhoff (loi des nœuds)
  - En conséquence :

$$\sum_{(x_1,x_i)\in U} \varphi(x_1, x_i) = \sum_{(x_j,x_n)\in U} \varphi(x_j, x_n)$$

Problématique de flots



$$\sum_{(x_1,x_i)\in U} c(x_1,x_i) = 75 \quad ; \quad \sum_{(x_j,x_n)\in U} c(x_j,x_n) = 65 \quad ; \quad \varphi = 55$$

- Ce flot est-il optimal ? Ou peut-il être amélioré, et comment ?

Plan du cours

Algorithme de Ford-Fulkerson

- Détection des chaînes améliorantes
- Optimisation du flux dans la chaîne améliorante

Définition

Une *chaîne améliorante* est une chaîne élémentaire d'origine x_1 et d'extrémité x_n dans laquelle aucun *arc direct* n'est saturé, et tous les *arcs indirects* ont un flux strictement positif.

- (x_a, x_b) est un arc direct de la chaîne si (x_a, x_b) est un arc du graphe
- (x_a, x_b) est un arc indirect de la chaîne si (x_b, x_a) est un arc du graphe
- On parle d'arc saturé si $c(u) = \varphi(u)$

Algorithme de Ford-Fulkerson

- **Détection d'une chaîne améliorante**

1: source marquée « $+$ », autres sommets « non marqués »

2: tant qu'un arc (x, y) vérifie l'une des deux définitions

3: x est déjà marqué, y est non marqué et (x, y) est non saturé : marquer « $+x$ » le sommet y

4: x est non marqué, y est déjà marqué et $\varphi(x, y)$ est non nul : marquer « -y » le sommet x

5: si le puits est marqué, le flot courant est améliorable par la chaîne trouvée en remontant les marques vers la source

6: si le puits n'est pas marqué, le flot courant est optimal

Algorithme de Ford-Fulkerson

- **Augmentation du flux dans chaîne améliorante**

1: $\delta^+ = \min\{c(u) - \varphi(u)\}$ où u est un arc direct de C

2: $\delta^- = \min\{\varphi(u)\}$ où u est un arc indirect de C

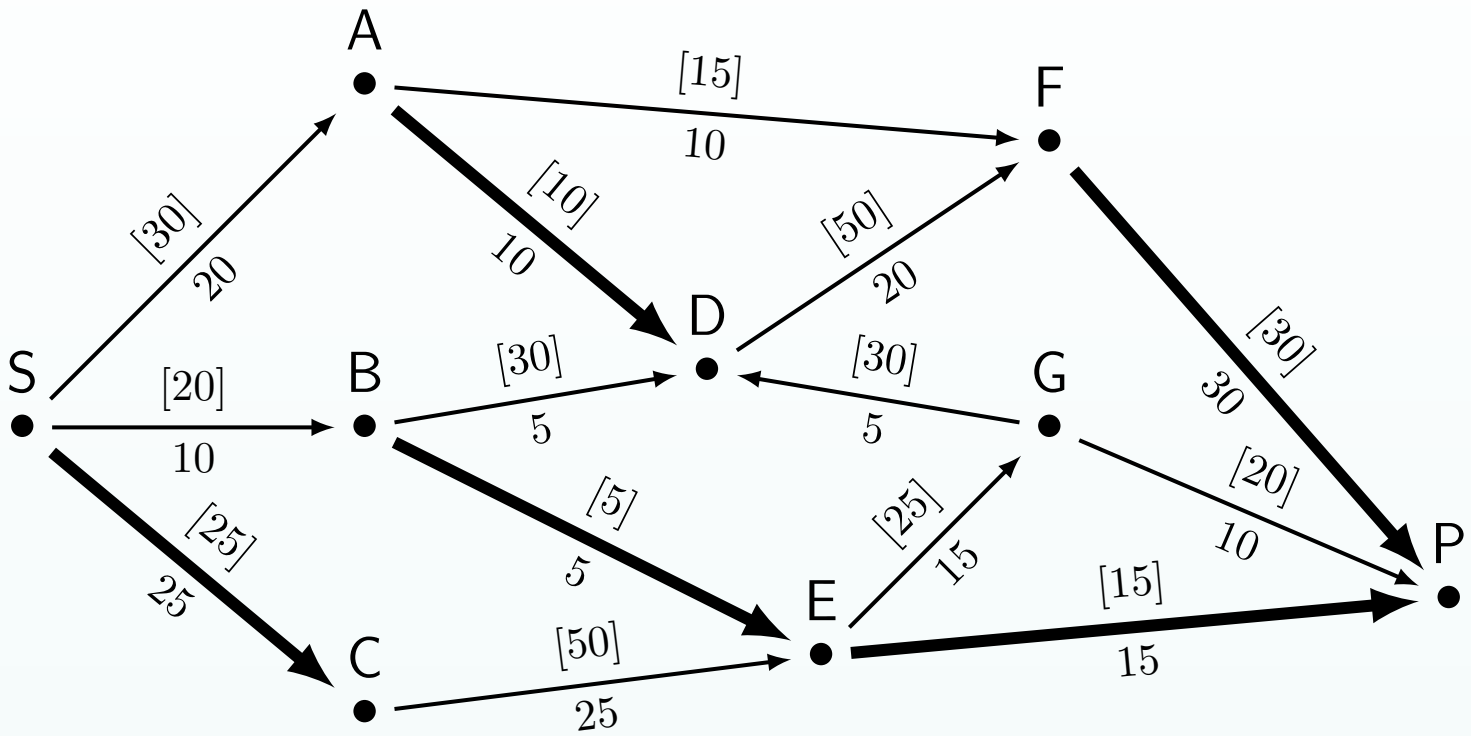
3: $\delta = \min\{\delta^+; \delta^-\}$

4: pour tout arc direct u de C : $\varphi_u \leftarrow \varphi_u + \delta$

5: pour tout arc indirect u de C : $\varphi_u \leftarrow \varphi_u - \delta$

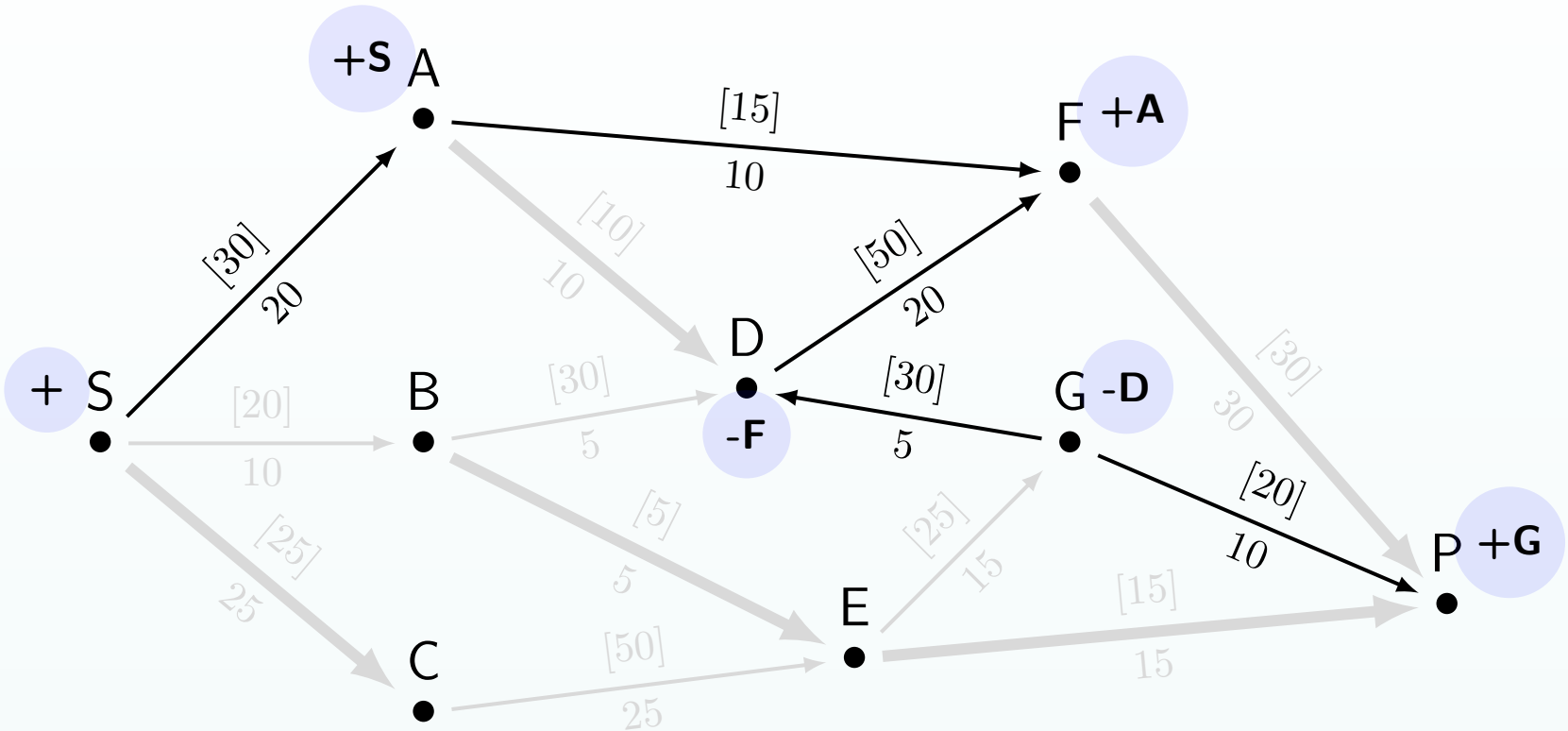
- $\delta > 0$, de par la définition de la chaîne améliorante

Algorithme de Ford-Fulkerson : Exemple



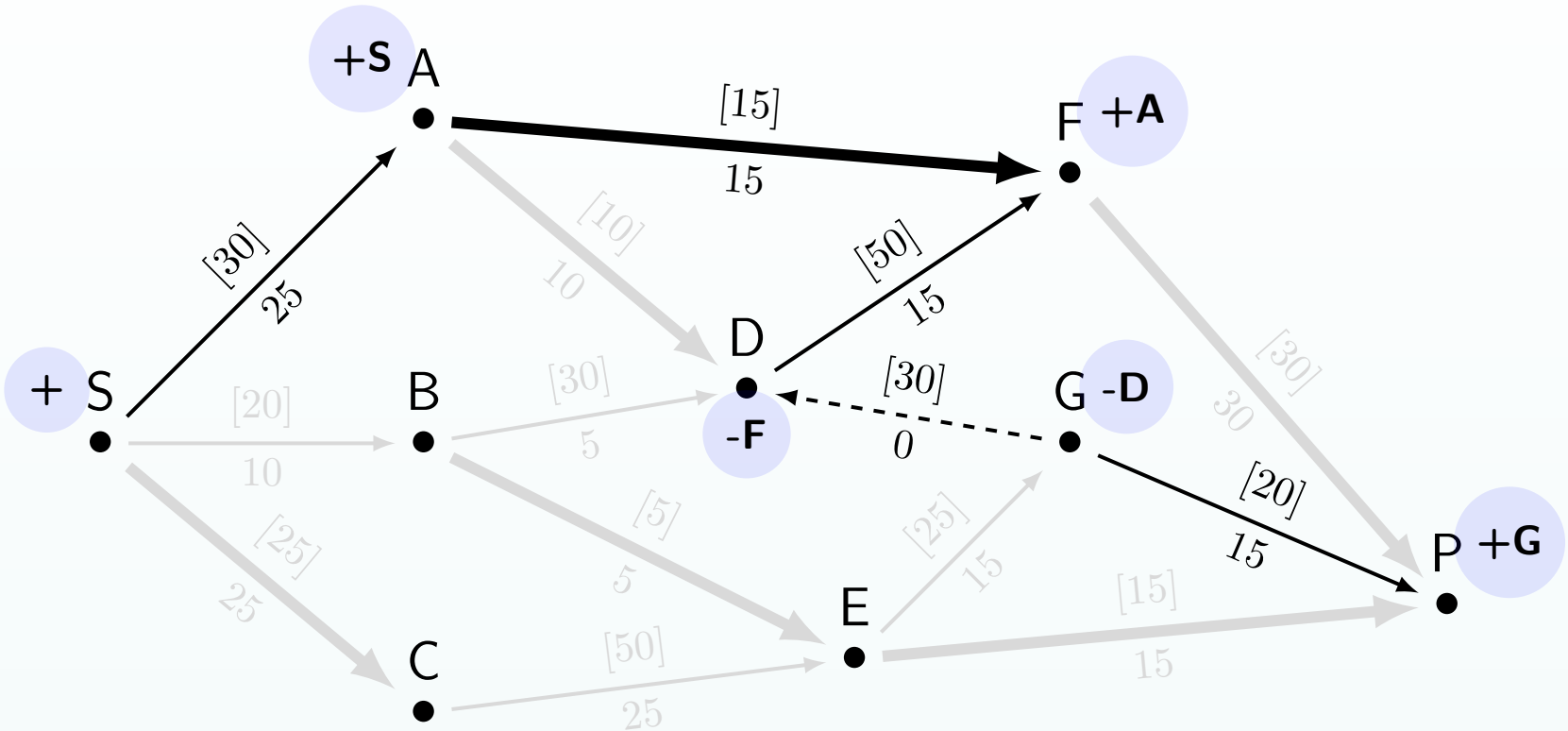
- Recherche de chaîne améliorante

Algorithme de Ford-Fulkerson : Exemple



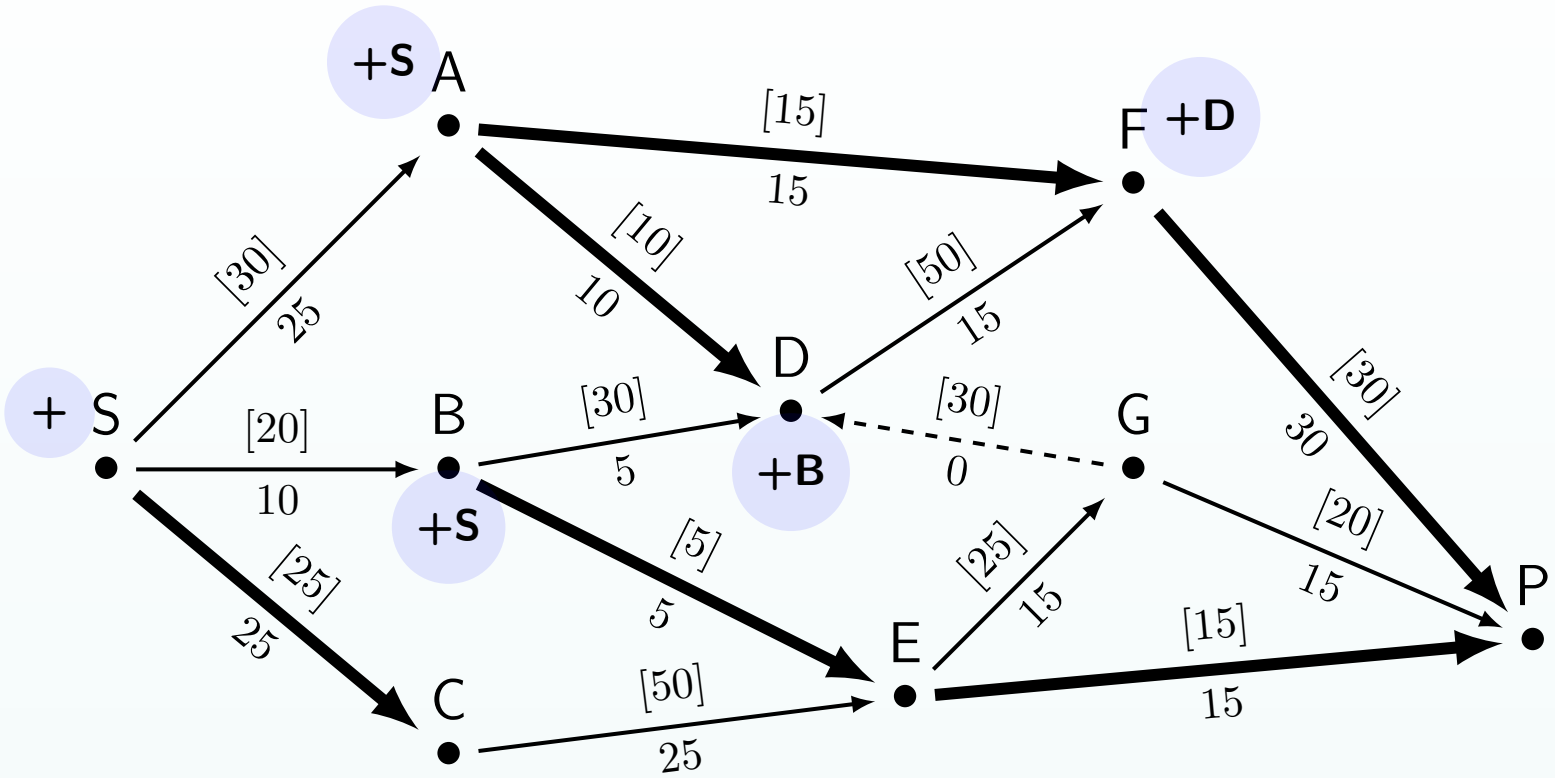
- Chaîne améliorante (S, A, F, D, G, P)

Algorithme de Ford-Fulkerson : Exemple



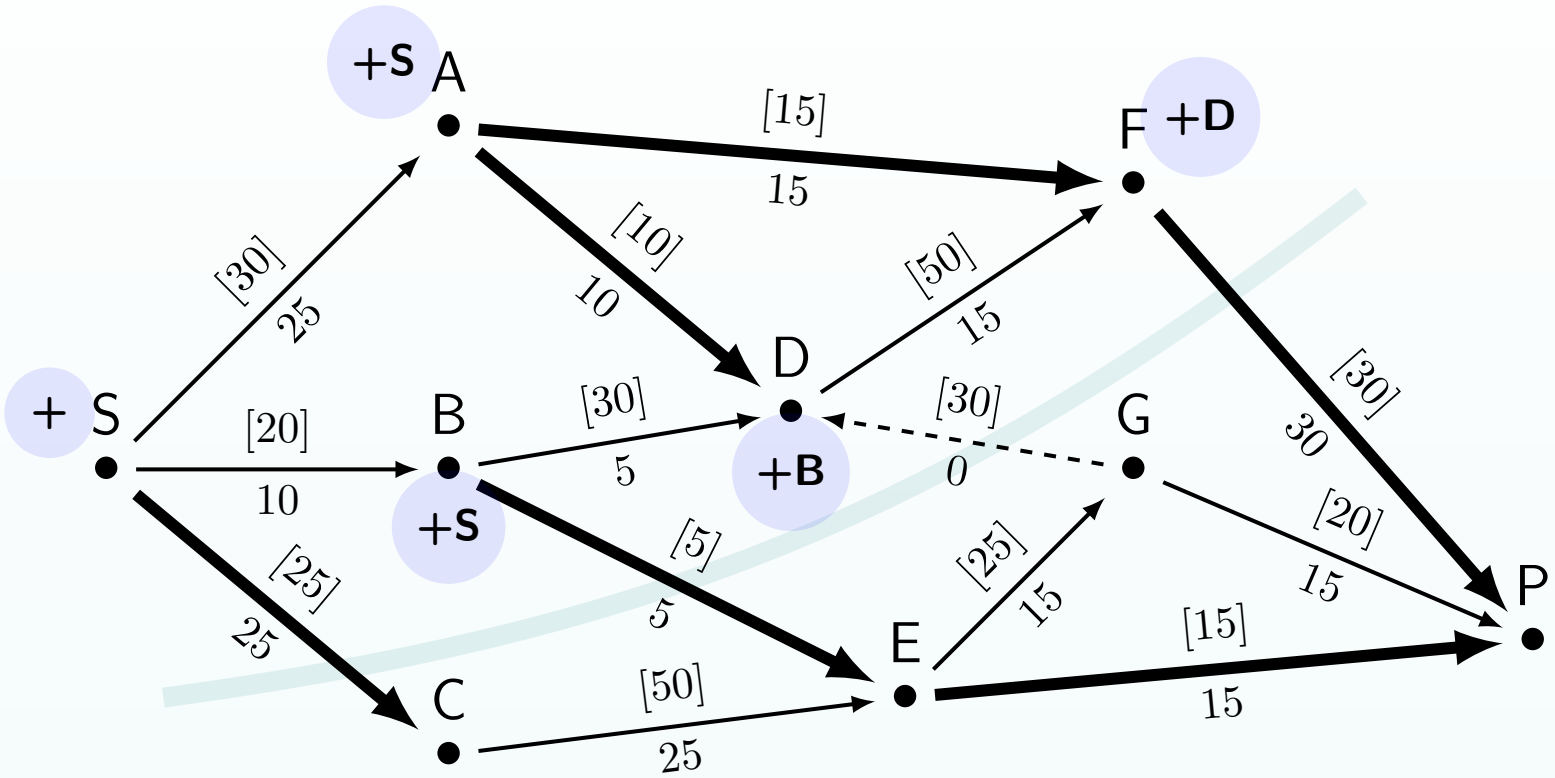
- Chaîne améliorante (S, A, F, D, G, P) : $\delta = 5$

Algorithme de Ford-Fulkerson : Exemple



- Aucune chaîne améliorante, le flot est maximal : $\delta = 60$

Algorithme de Ford-Fulkerson : Exemple



- Aucune chaîne améliorante, le flot est maximal : $\delta = 60$
- Détection de la coupe minimum

Théorème flot-max/coupe-min

Définition

Le flot maximum pouvant aller de la source au puits est égal à la *capacité minimale* devant être retirée du graphe afin d'empêcher qu'aucun flot ne puisse passer de la source au puits.

- La coupe minimum sépare le graphe en deux ensembles de sommets, dont :
  1. La suppression des arcs intermédiaires entraîne la nullité du flot
  2. La somme des capacités de ces arcs est minimale
- Cette somme des capacités est égale au flot maximal du graphe

Problématique de flots

- Algorithme de Ford-Fulkerson
  - Obtention d'un flot de valeur maximale
  - Coupe minimum
  - Preuve d'optimalité
- De nombreux autres modèles
  - Flot maximal à *coût* minimal
  - Coupes
  - Problèmes d'affectations
  - ...

Plan du cours

Définitions

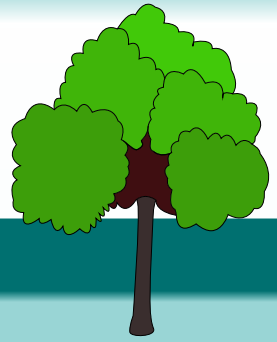
Définition

La complexité cyclomatique comptabilise le nombre de cycles au travers un graphe.

- $V(G) = m - n + p$
  - m arcs
  - n sommets
  - p composantes connexes

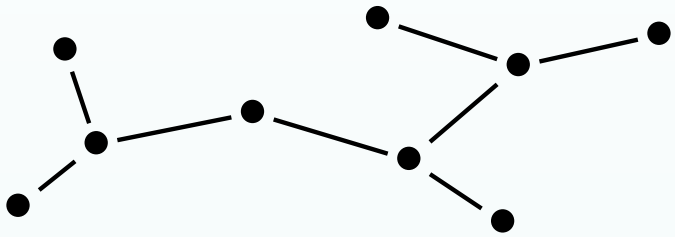
- $V(G) = 0 \Leftrightarrow G$ est connexe sans cycle

Définitions



Définition

Un arbre est un graphe connexe sans cycle.

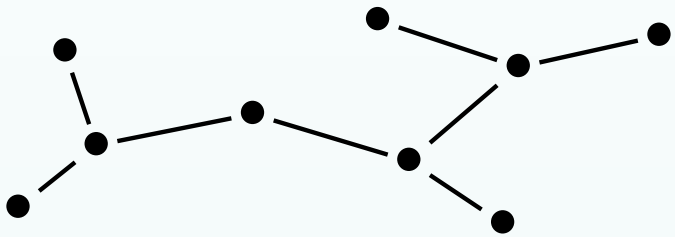


- Conséquence de la complexité cyclomatique, un arbre est un graphe connexe à $n-1$ arrêtes.

Définitions

Définition

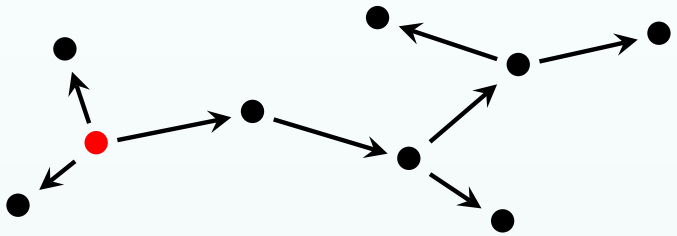
Une arborescence est, dans un graphe orienté, un arbre comportant un sommet *racine* pour lequel il existe un chemin unique vers tout autre sommet.



Définitions

Définition

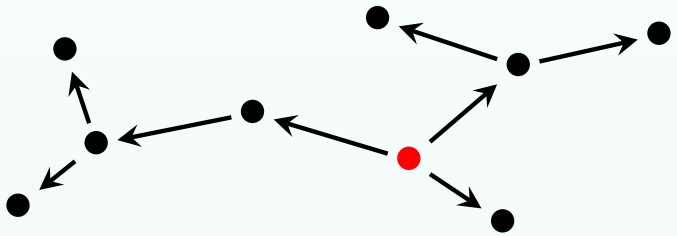
Une arborescence est, dans un graphe orienté, un arbre comportant un sommet *racine* pour lequel il existe un chemin unique vers tout autre sommet.



Définitions

Définition

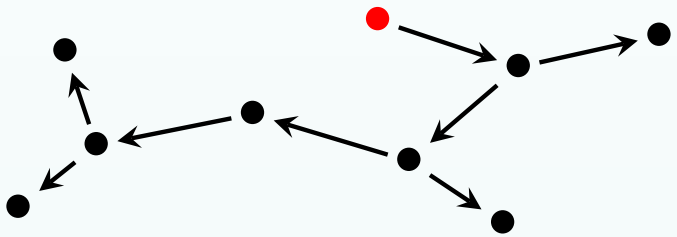
Une arborescence est, dans un graphe orienté, un arbre comportant un sommet *racine* pour lequel il existe un chemin unique vers tout autre sommet.



Définitions

Définition

Une arborescence est, dans un graphe orienté, un arbre comportant un sommet *racine* pour lequel il existe un chemin unique vers tout autre sommet.



Définitions

- Nécessairement, la racine ne peut admettre de prédécesseur et est unique
  - Toute racine double impliquerait un cycle

Attention !

En **Informatique**, le terme *arbre* est couramment utilisé pour désigner une *arborescence* !

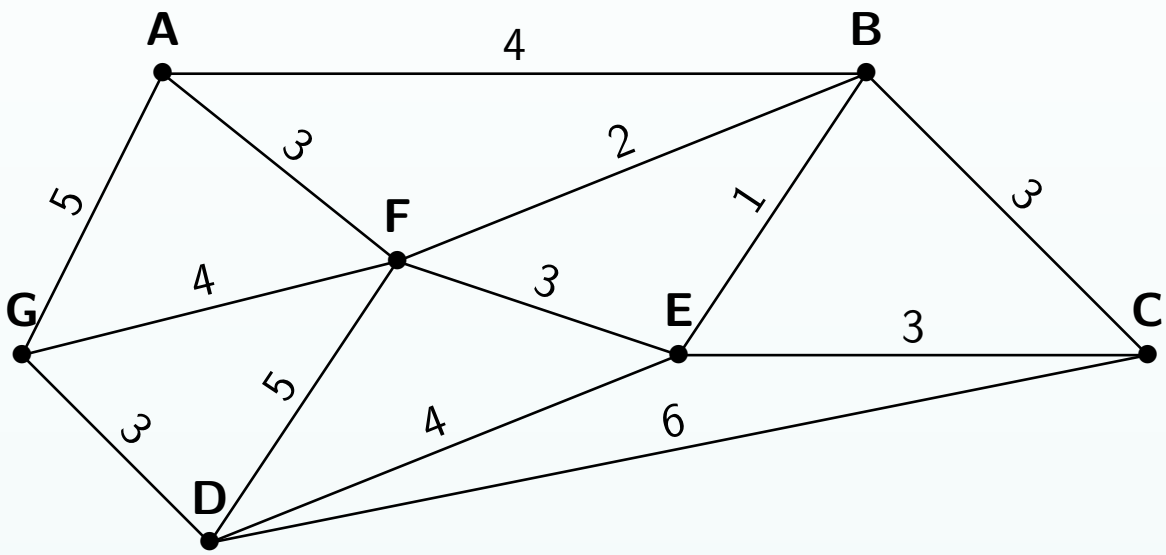
Exemples d'applications

- Arbres optimaux
  - Optimisation de parcours de réseaux
- Programmes de transport
  - Organisation entre sources et destinations multiples
  - Flot illimité mais coût limité
- Recherche arborescente
  - « Séparation et Évaluations Progressives »
  - Recherche optimale de solution d'ordonnancement

Plan du cours

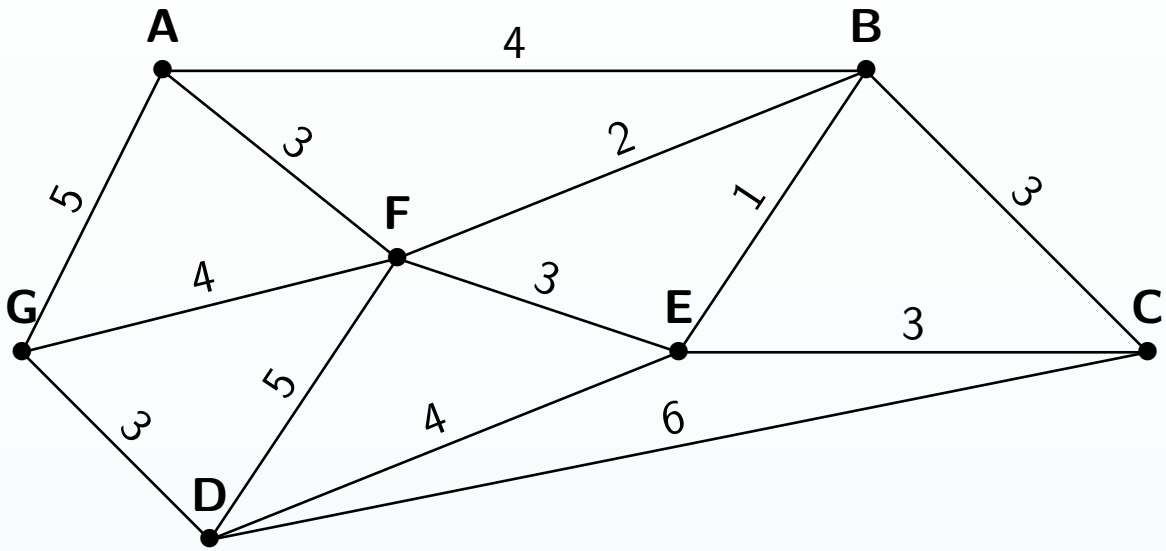
Arbres optimaux

- Quel arbre issu du graphe minimise la valeur des arrêtes ?



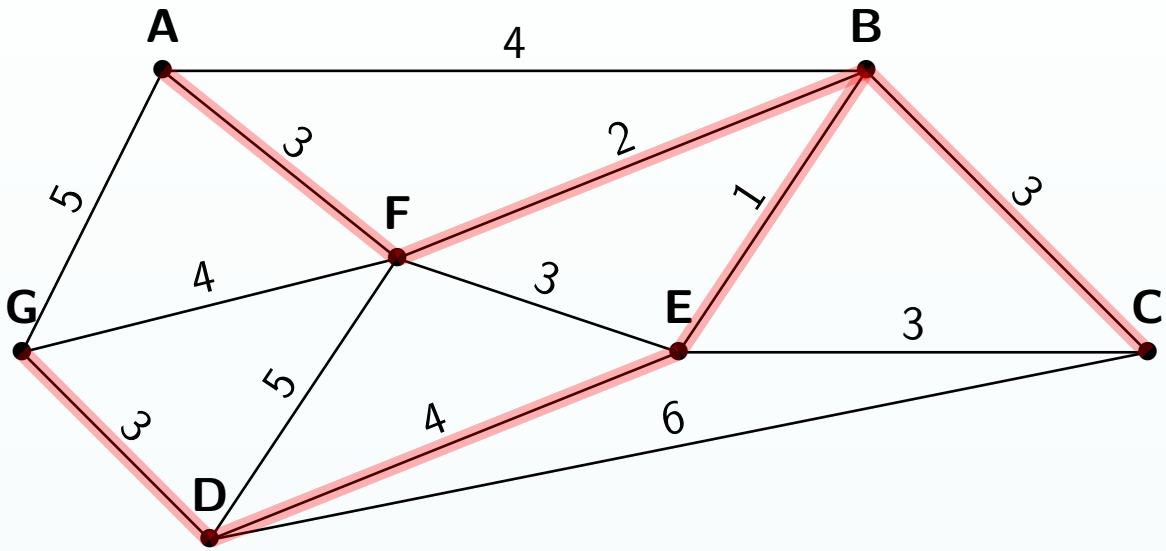
- Utilisation d'un algorithme glouton (Kruskal, 1956)

Arbres optimaux



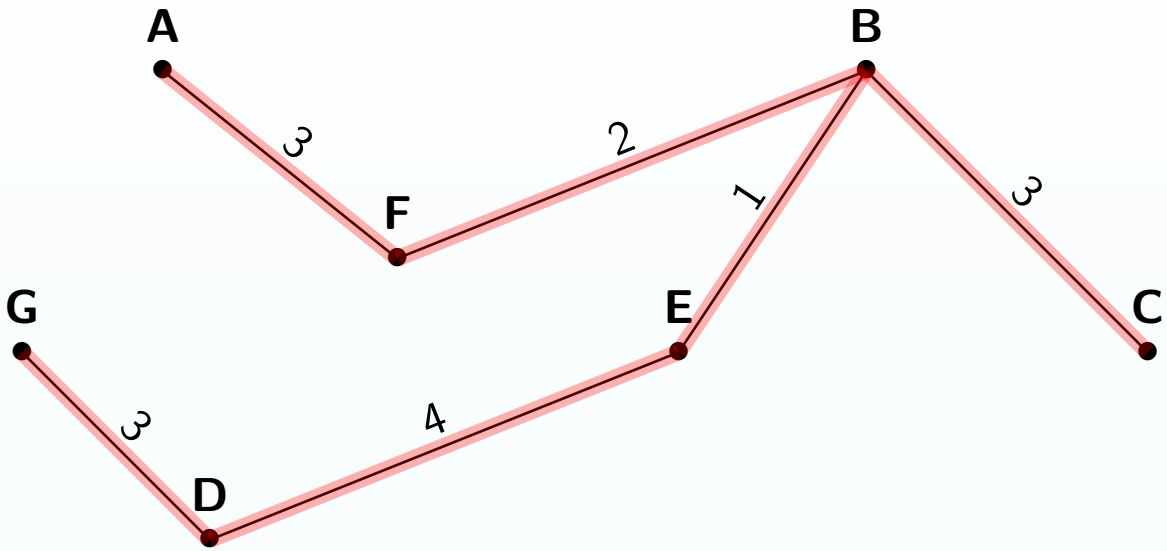
1: tant que cela est possible, faire
2: sélectionner l'arrête de plus petite valuation possible dont les deux extrémités se sont pas déjà toutes sélectionnées
3: sélectionner les extrémités de l'arrête
4: si plusieurs sous-graphes
5: recommencer à l'hypergraphe

Arbres optimaux



1: tant que cela est possible, faire
2: sélectionner l'arrête de plus petite valuation possible dont les deux extrémités se sont pas déjà toutes sélectionnées
3: sélectionner les extrémités de l'arrête
4: si plusieurs sous-graphes
5: recommencer à l'hypergraphe

Arbres optimaux



1: tant que cela est possible, faire
2: sélectionner l'arrête de plus petite valuation possible dont les deux extrémités se sont pas déjà toutes sélectionnées
3: sélectionner les extrémités de l'arrête
4: si plusieurs sous-graphes
5: recommencer à l'hypergraphe

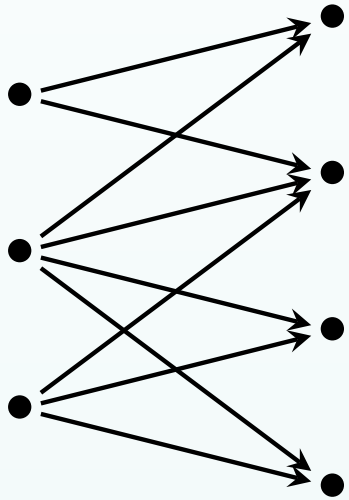
Plan du cours

Programmes de transport

Définition

Entre m origines et n destinations d'un *graphe biparti*, un programme de transport modélise $m \times n$ arcs comme autant de liaisons de transport.

- Capacité des arcs illimitée
- Coût unitaire par arc
- Recherche du coût minimal



Exemple de programme de transport

$i \setminus j$	1	2	3	4	5	6	*(clients)*
I	12	27	61	49	83	35	18
II	23	39	78	28	65	42	32
III	67	56	92	24	53	54	14
IV	71	43	91	67	40	49	9
(usines)	9	11	28	6	14	5	73

- Somme des demandes nécessairement égale à la somme des disponibilités
  - Utilisation de nœuds virtuels au besoin

Exemple de programme de transport

- Résoudre le programme de transport revient à trouver les valeurs numériques de $m \times n$ nombres non-négatifs x_{ij}, tels que :

1. L'offre égale la demande :
$$\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$$

2. Toute origine i livre toute sa quantité disponible a_i :
$$\sum_{j=1}^{n} x_{ij} = a_i$$

3. Toute destination j est livré de toute sa demande b_j :
$$\sum_{i=1}^{m} x_{ij} = b_j$$

4. z, fonction économique du programme, est minimale :
$$\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \cdot x_{ij} = z$$

Résolution d'un programme de transport

- Problème NP-difficile !
  - Non soluble en un temps polynomial
- Utilisation d'heuristiques
  - Recherche d'une solution valable
  - Réalisable (polynomiale)
  - Pas nécessairement optimale

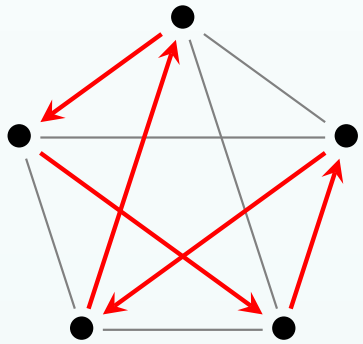
Résolution d'un programme de transport

- Exemple d'heuristique :
  - Recherche d'une solution admissible
    - Respectant les 3 premiers critères (non économique)
    - Non-dégénérée : $n \cdot m - (n + m - 1) = (n - 1) \cdot (m - 1)$ variables nulles
    - Procédure dite du *Nord-Ouest*
  - Recherche d'amélioration
    - Notion de potentiel
    - Chaîne améliorante du graphe biparti
    - Jusqu'à stabilité

Plan du cours

Recherche arborescente

- « Séparation et Évaluations Progressives »
  - Division en sous-problèmes à solutions réalisables
  - Utilisé pour résoudre des problèmes NP-complets
- Problèmes d'organisation
  - Optimisation combinatoire
- Exemple : *le voyageur de commerce*



Résolution d'une recherche arborescente

- Réalisation de la Matrice des coûts
- Parcours des blocs de résolution

Bloc A : Soustractions des plus petits éléments par rangée

Bloc B : Calcul de la borne inférieure, création de l'arborescence

Bloc C : Calcul des coûts de substitution

Bloc D : Croissance de l'arborescence : (X, Y), non–(X, Y)

Bloc E : Simplification de la matrice par l'arc choisi

Bloc F : L'arborescence mène à (X, Y) : revenir au Bloc C.
Sinon, Bloc G.

Bloc G : Interdire (X, Y) par coûts infini. Revenir au Bloc C.

Plan du cours

Ressources bibliographiques

Parce qu'il est impossible de tout traiter ici, pour revoir ou approfondir le cours comme les exemples d'application :

- **Précis de recherche opérationnelle – Méthodes et exercices d'application**. *Robert Faure, Bernard Lemaire, Christophe Picouleau*. Dunod, 6e édition, 2014.
- **Aide à la décision – Une approche par les cas**. *Philippe Vallin, Daniel Vanderpooten*. Ellipses, 2e édition, 2002.
- **Graphes et algorithmes**. *Michel Gondran, Michel Minoux*. Tec et Doc, 4e édition, 2009.

Les graphes en Recherche Opérationnelle
Damien Leprovost
version initiale, 21 octobre 2014 – v3, 31 octobre 2016

permalien : https://www.damien-leprovost.fr/enseignements/graphes.2016.pdf