1 Graphes Orientés

1.1 p-graphes

- Un p-**graphe** est un couple $\boldsymbol{G = (X,U)}$ où $X = \{x_1,x_2,\ldots,x_{\mathbf{N}}\}$ est un ensemble de **sommets** et $U = (u_1,u_2,\ldots,u_{\mathbf{M}})$ une famille (*i.e.* ensemble avec répétition autorisée) de couples de sommets appelés **arcs** ; chaque couple ne pouvant apparaître plus de p fois dans U.
- Un graphe est **valué** (ou **pondéré**) si chaque arc possède un **coût** (ou un **poids**, ou une **valeur**) représenté par une application $C : U \to \mathbb{R}$. On le note alors $\boldsymbol{G = (X,U,C)}$.
- Soit u un arc de la forme $(x,y) \in X^2$:
  - x est l'**origine** (ou **extrémité initiale**) de u.
    u **part** de x, u est **incident extérieurement** à x.
  - y est l'**extrémité** (ou **extrémité terminale**) de u.
    u **arrive** de y, u est **incident intérieurement** à y.
  - x et y sont des sommets **voisins**.
  - x est **prédécesseur** de y, y est **successeur** de x.
  - Si $x = y$, u est appelé une **boucle**.
  - Deux arcs sont dits **adjacents** s'ils possèdent une extrémité commune.
- Soit x un sommet :
  - $\boldsymbol{\omega^+(x)}$ est l'ensemble des arcs incidents extérieurement à x,
    $\boldsymbol{\omega^-(x)}$ est l'ensemble des arcs incidents intérieurement à x et
    $\boldsymbol{\omega(x)} = \omega^+(x) \cup \omega^-(x)$.
  - $\boldsymbol{d^+(x)} = |\omega^+(x)|$ est le **demi-degré extérieur** de x,
    $\boldsymbol{d^-(x)} = |\omega^-(x)|$ est le **demi-degré intérieur** de x et
    $\boldsymbol{d(x)} = d^+(x) + d^-(x)$ (*rem :* $d(x) \neq |\omega(x)|$, les boucles étant comptées deux fois.).
  - x est un **sommet isolé** si $d(x) = 0$.
  - $\boldsymbol{\Gamma(x)}$ est l'ensemble des successeurs de x,
    $\boldsymbol{\Gamma^{-1}(x)}$ est l'ensemble des prédécesseurs de x.

1.2 1-graphes

Dans un 1-graphe, il ne peut exister au plus qu'un seul arc entre deux sommets donnés. U est donc un sous-ensemble de X^2, *i.e.*, par définition, U est une *relation binaire* sur X.

- La **densité** est le rapport du nombre d'arcs sur le nombre maximal d'arcs possibles, *i.e.* $\frac{M}{N^2}$. C'est donc un réel compris entre 0 et 1.
- Pour tout sommet x, il y a une bijection entre l'ensemble des successeurs (resp. prédécesseurs) de x et l'ensemble des arcs incidents extérieurement (resp. intérieurement) à x : $y \in \Gamma(x) \leftrightarrow (x,y) \in U$ (resp. $y \in \Gamma^{-1}(x) \leftrightarrow (y,x) \in U$). On peut donc représenter G par (X,Γ).

2 Graphes non-orientés

- Un **graphe non-orienté** est un couple $G = (X,E)$ où X est l'ensemble des sommets et où E est une famille de paires de sommets $e = [x,y]$ appelées **arêtes**. (*rem :* la notation ensembliste $\{x,y\}$ n'est pas adaptée pour distinguer les boucles ($[x,x]$) des sommets ($x = \{x,x\}$).)
- Un **multigraphe** est l'équivalent non-orienté d'un p-graphe,
  un **graphe simple** est l'équivalent non-orienté d'un 1-graphe et *sans boucle*. (*rem :* dans un graphe simple, $E \subset \mathcal{P}^2(X)$.)
- La **densité** d'un graphe simple est alors égale à $\frac{M}{N(N-1)/2}$.

3 Partie de graphes

Soit $G = (X,U)$ un graphe orienté.

- Soit $A \subset X$, le **sous-graphe** de G engendré par A est $G_A = (A,A^2 \cap U)$ *i.e.* les sommets de A et les arcs ayant leurs deux extrémités dans A.
- Soit $V \subset U$, le **graphe partiel** de G engendré par V est $G' = (X,V)$.

4 Parcours

- Un **chemin** μ de **longueur** q est une séquence de q arcs $(u_1,u_2,\ldots,u_q)$ telle que, pout tout $i \in \{2,\ldots,q\}$, l'origine de u_i est l'extrémité de $ui-1$.
  Un **circuit** est un chemin fermé (i.e. origine de u_1 = extrémité de u_q).
- Un sommet y est **descendant** de x (ou x est **ascendant** de y) s'il existe un chemin de x vers y.
- Une **chaîne** de **longueur** q est une séquence de q arêtes
  $([x_1,x_2],[x_2,x_3],\ldots,[x_q,x_{q+1}])$.
  Un **cycle** est une chaîne fermée (*i.e.* $x_1 = x_{q+1}$).
- Un **parcours** est soit un chemin, soit un circuit, soit une chaîne, soit un cycle.
- Un parcours est dit **simple** s'il ne possède pas deux fois le même arc ; et dit **élémentaire** s'il n'emprunte pas deux fois le même sommet.
- Dans un graphe valué, la **valeur** d'un parcours est la somme des valeurs des arcs (ou arêtes) de ce parcours.

5 Connexité

- Un graphe est **connexe** s'il existe une chaîne entre chaque paire de sommets.
  Dans tout graphe, les sous-graphes engendrés par les classes d'équivalence de la relation $x\mathcal{R}y \Leftrightarrow$ « il existe une chaîne entre x et y », sont les **composantes connexes** du graphe et en forment une partition.
- Un graphe orienté est **fortement connexe** si pour toute couple de sommets (x,y), il existe un chemin de x vers y et un chemin de vers x (*i.e.* x et y sont sur un circuit).
  Les **composantes fortement connexes** correspondent à la relation d'équivalence $x\mathcal{R}y \Leftrightarrow$ « il existe un circuit passant par x et y ».

1 Représentation des graphes

1.1 Matrice d'adjacence

Soit un 1-graphe orienté $G = (X,U)$. On définit une matrice d'adjacence $A,N \times N$ représentant le graphe G.

- $A_{i,j} = 1$ si et seulement si $(i,j) \in U$
- $A_{i,j} = 0$ si et seulement si $(i,j) \notin U$

Soit un 1-graphe orienté et valué $G = (X,U,C)$. La matrice d'ajacence devient:

- $A_{i,j} = C_{i,j}$ si et seulement si $(i,j) \in U$
- $A_{i,j} = 0$ si et seulement si $(i,j) \notin U$

Remarque 1: pour un graphe non orienté $G = (X,E)$, on définit la matrice d'ajacence de G par la matrice d'ajacence du graphe orienté obtenu en remplaçant chaque arête $[i,j]$ par deux arcs (i,j) et (j,i).

Remarque 2: La représentation informatique a une taille minimale théorique en mémoire de N^2 bits

1.2 Liste d'ajacence ou de successeurs

Cette représentation matérialise l'application multivoque Γ.

- Soit un graphe $G = (X,\Gamma)$. On associe à chaque sommet $i \in X$ la "liste" $\Gamma(i)$ de ses successeurs.
- Pour un graphe orienté et valué $G = (X,\Gamma,C)$, on associé à chaque sommet i une liste de couples (successeur,coût).
- Pour un graphe non orienté on peut utiliser la représentation par liste du graphe orienté symétrique correspondant
- Cette représentation est valable pour les p-graphes tels que $p > 1$

Les listes de successeurs utilisent $O(N + M)$ emplacements mémoire.

2 Représentation des graphes et complexité des algorithmes

cf cours.

3 Algorithmes d'exploration de graphes

3.1 Utilité et Principe général

L'objectif est de répondre aux questions : quels sommets peut-on atteindre à partir d'un sommet donné s. Y-a-t-il un chemin d'un sommet vers un autre ? Quel est ce chemin ? Explorer un graphe à partir de s, c'est déterminer l'ensemble des descendants de s, c'est-à-dire les sommets situés sur un chemin à partir de s. Le principe est de marquer les sommets dès qu'ils sont visités pour éviter de cycler. Au fur et à mesure

de l'exploration le chemin de l'origine vers chaque sommet visité est mémorisé dans un tableau `predChemin` tel que `predChemin`(i) désigne le prédécesseur de i dans le chemin issu de s. On dit que `predChemin` définit une arborescence de visite. Les deux algorithmes explorent un graphe orienté à partir d'un sommet en $\mathcal{O}(M)$ (sous réserve d'utiliser les bonnes représentations!).

3.2 Exploration en largeur

L'algorithme d'exploration en largeur vise à explorer d'abord tous les successeurs du sommet origine, (chemins composés d'un arc) puis tous les successeurs des successeurs (chemins composés de 2 arcs), etc. Il utilise une **file** pour stocker les sommets dont les successeurs ne sont pas encore visités. L'algorithme est le suivant :

`marquer` s
`Ajouter` s à la file
`Répéter`
    `Enlever` un sommet i de la file
    `Pour` chaque successeur non marqué j de i
        marquer j
        `predChemin`(j)=i
        `Ajouter` j à la file
    `finPour`
`jusqu'à` file vide

3.3 Exploration en profondeur

L'algorithme d'exploration en profondeur consiste à explorer un successeur du sommet origine, puis un successeur de ce successeur, etc. .. afin d'aller le plus loin possible par un chemin. Il utilise une **pile** pour stocker les sommets dont les successeurs ne sont pas encore visités. L'algorithme est le suivant :

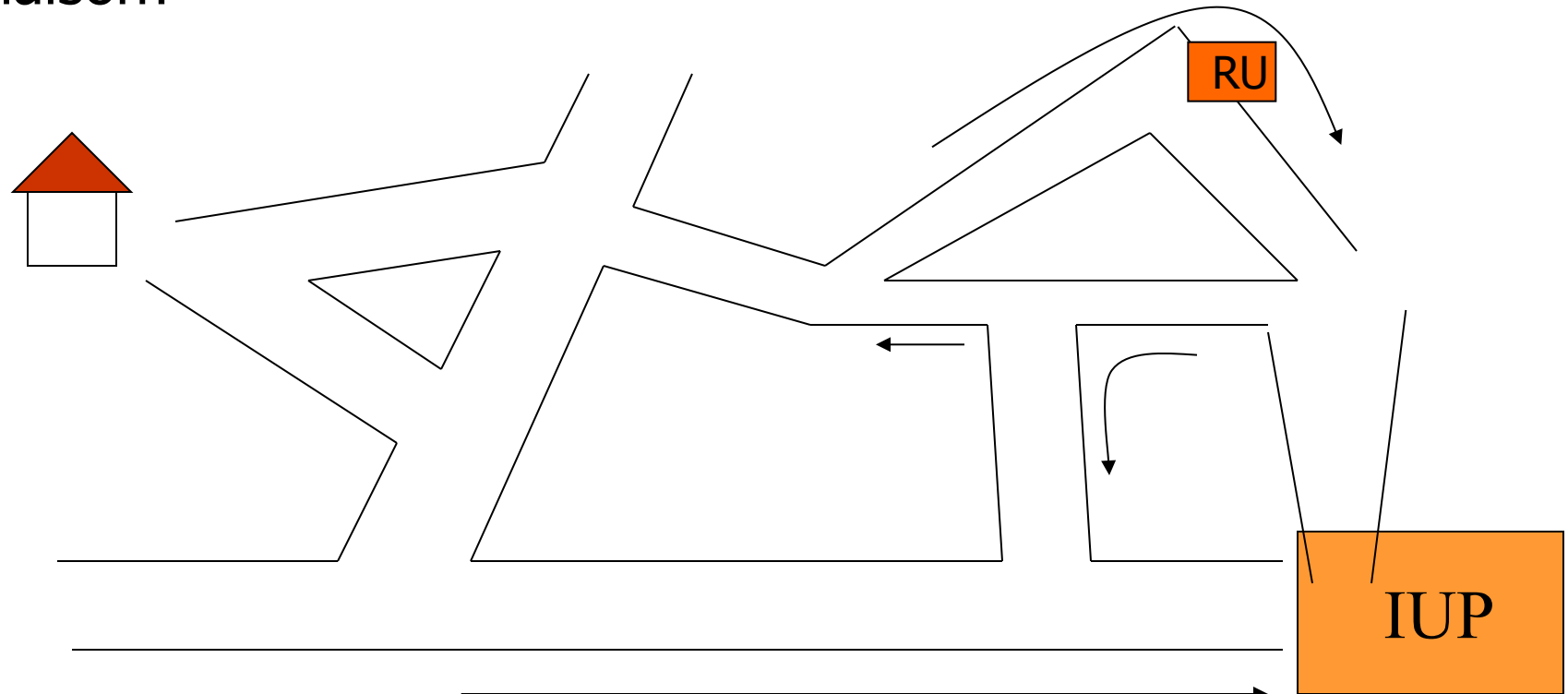
`marquer` s
`Empiler` s
`Répéter`
    $i \leftarrow$ sommet en tète de pile
    `Si` tous les successeurs de i sont marqués
        `Dépiler` i
    `Sinon`
        $j \leftarrow$ un successeur non marqué de i
        marquer j
        `predChemin`(j)=i
        `Empiler` j
    `finSi`
`jusqu'à` pile vide

4 Applications de l'exploration

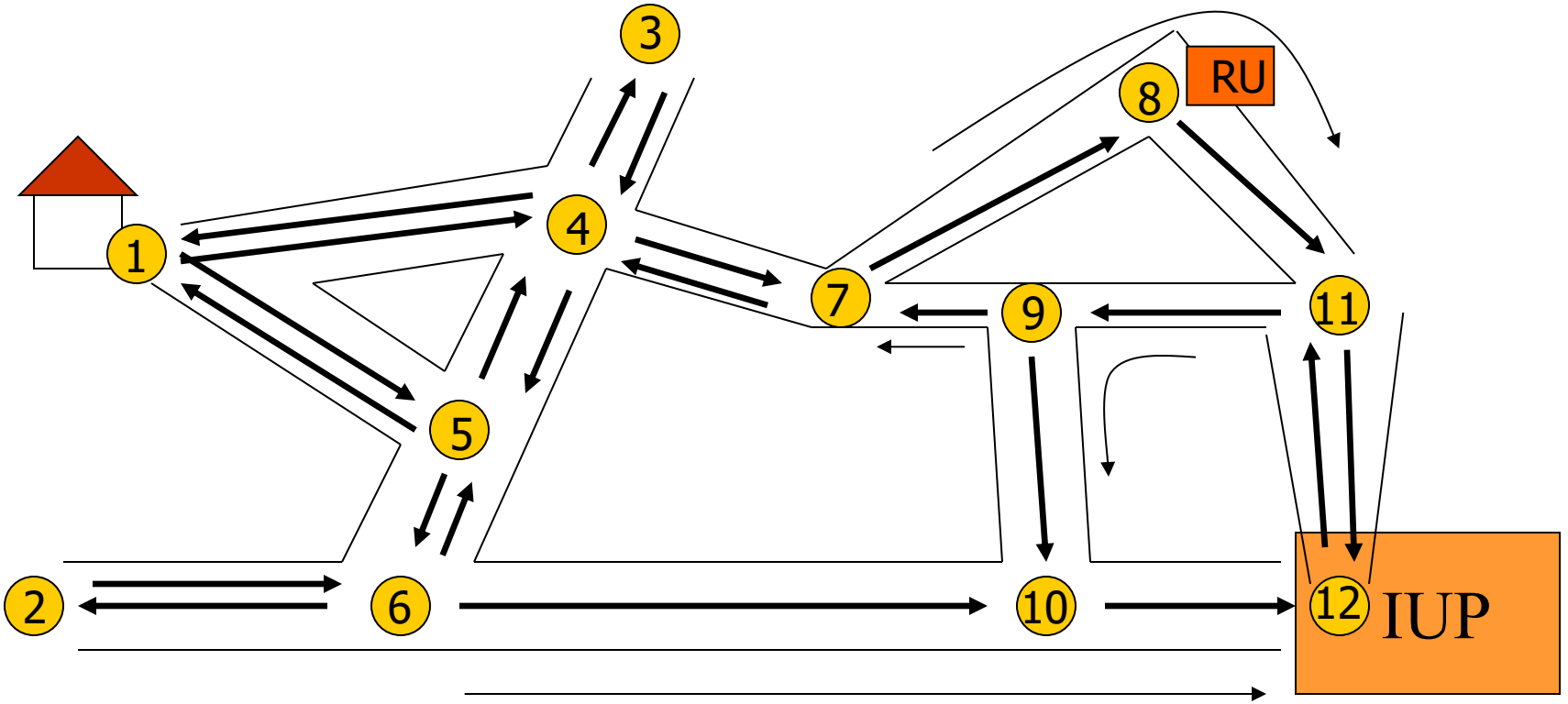
- Plus court chemin en nombre d'arcs (largeur)
- Détermination de composantes connexes et fortement connexes

Problèmes d'exploration

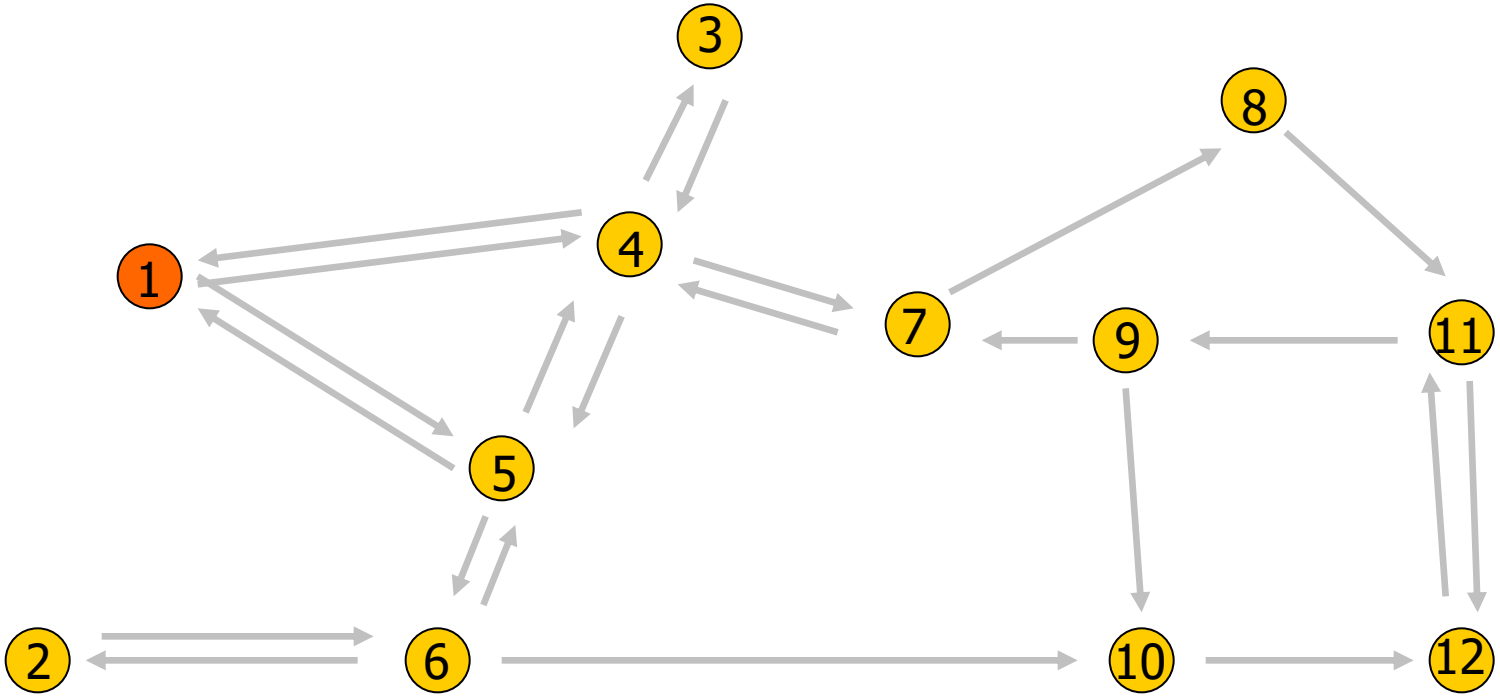
- Y-a-t-il un chemin pour aller à l'iup ?
- Quel est le chemin qui va à l'IUP en moins d'étapes intermédiaires possible ?
- Quels sont les points qu'on peut atteindre à partir de la maison?



Modélisation

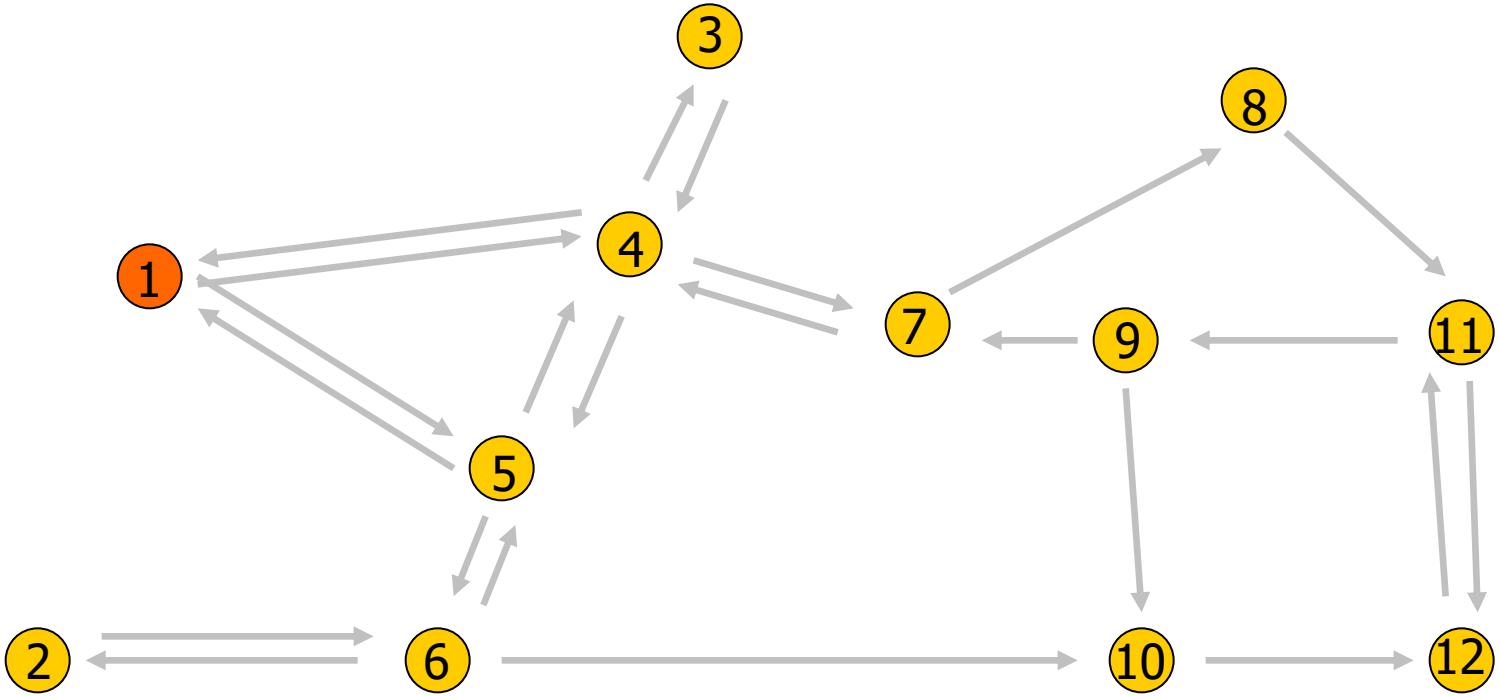


Exploration en largeur



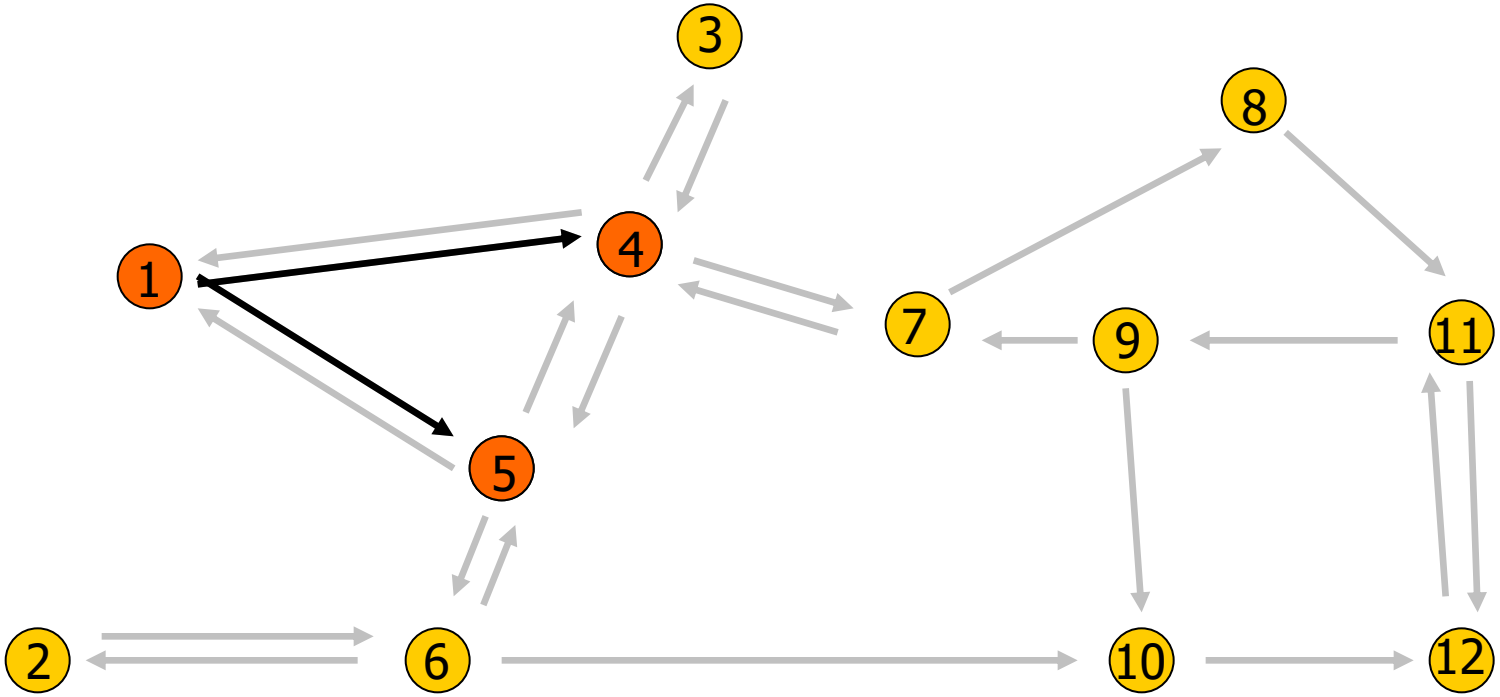
File 1

Exploration en Profondeur

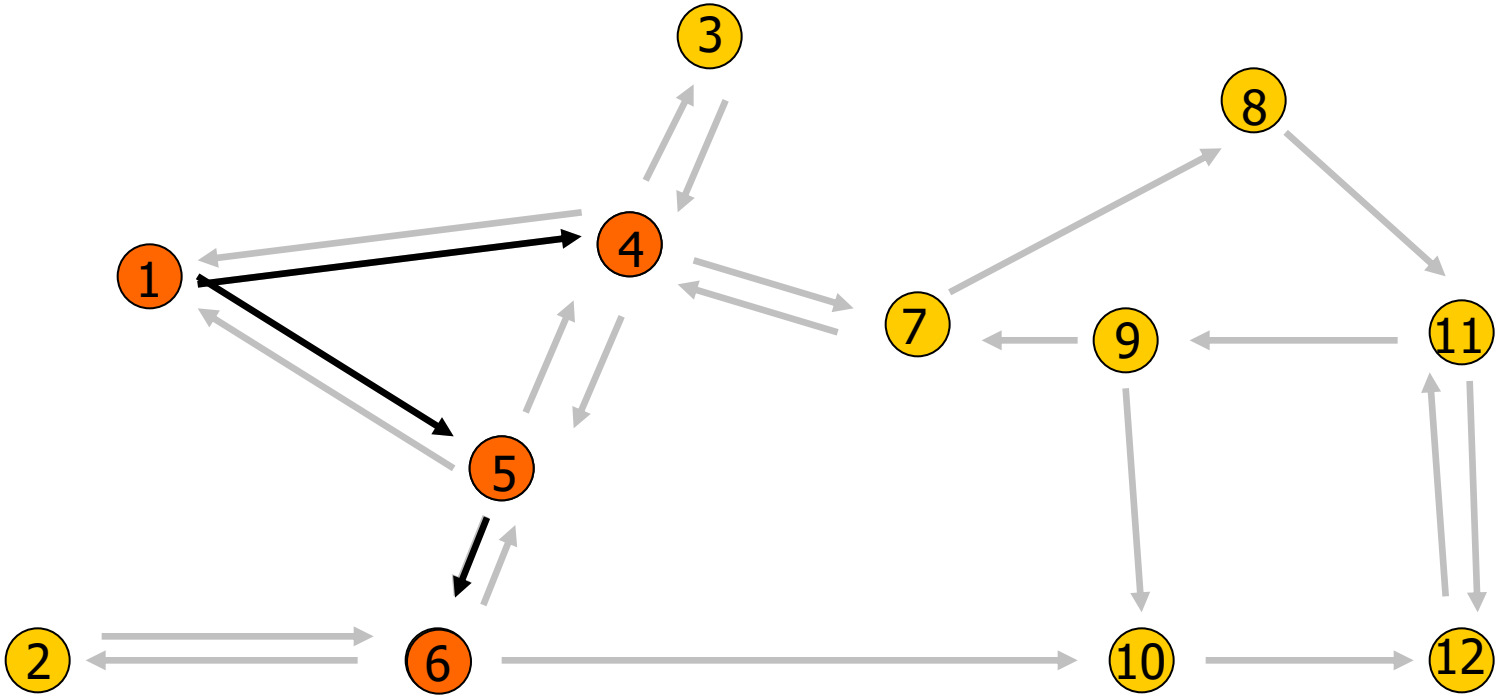


Pile

Exploration en largeur

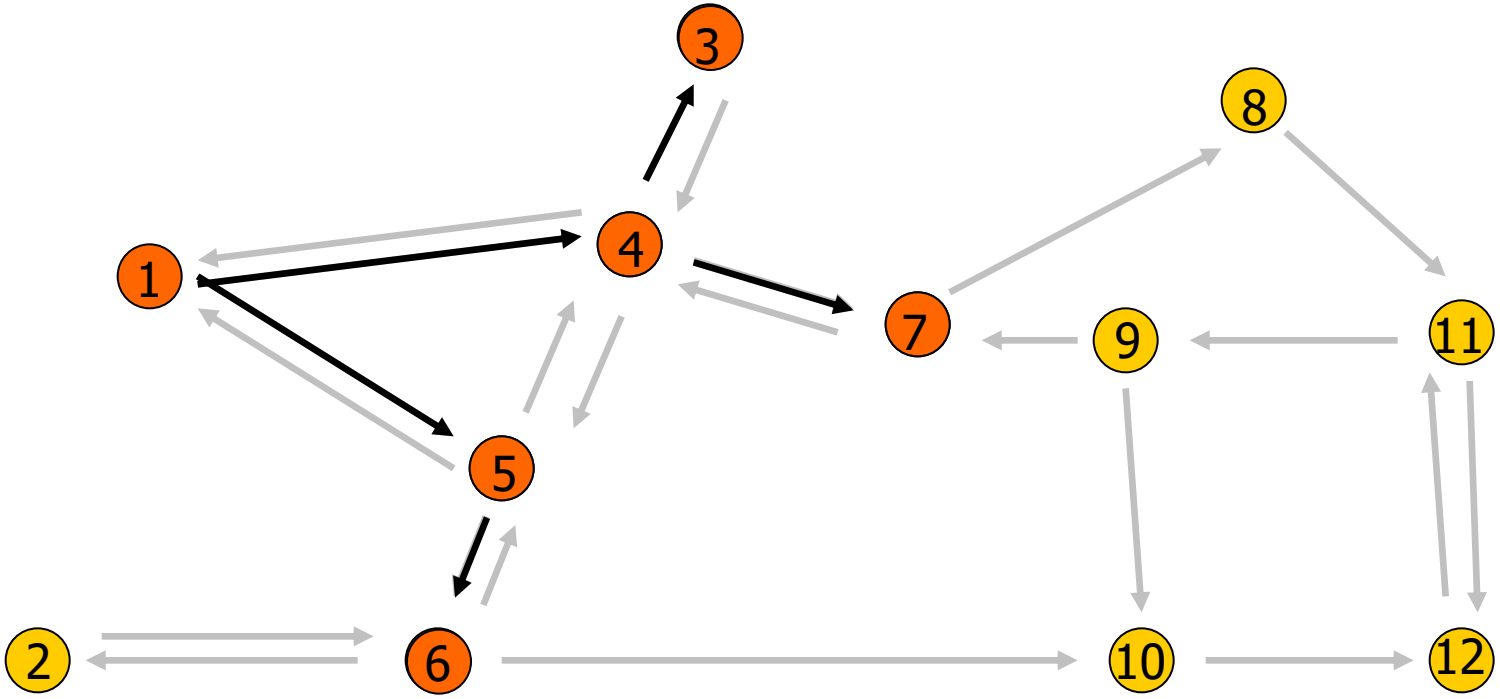


Exploration en largeur

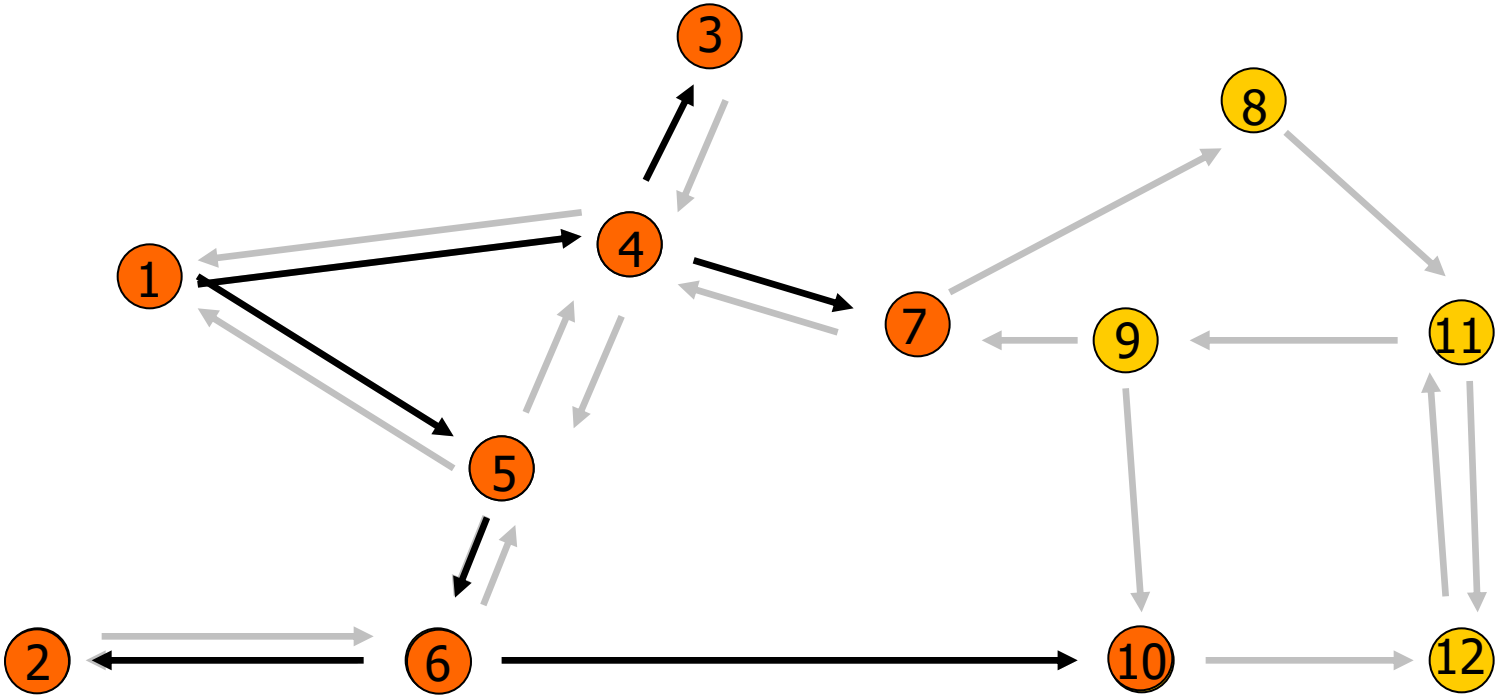


File 5 ← 4 6 ←

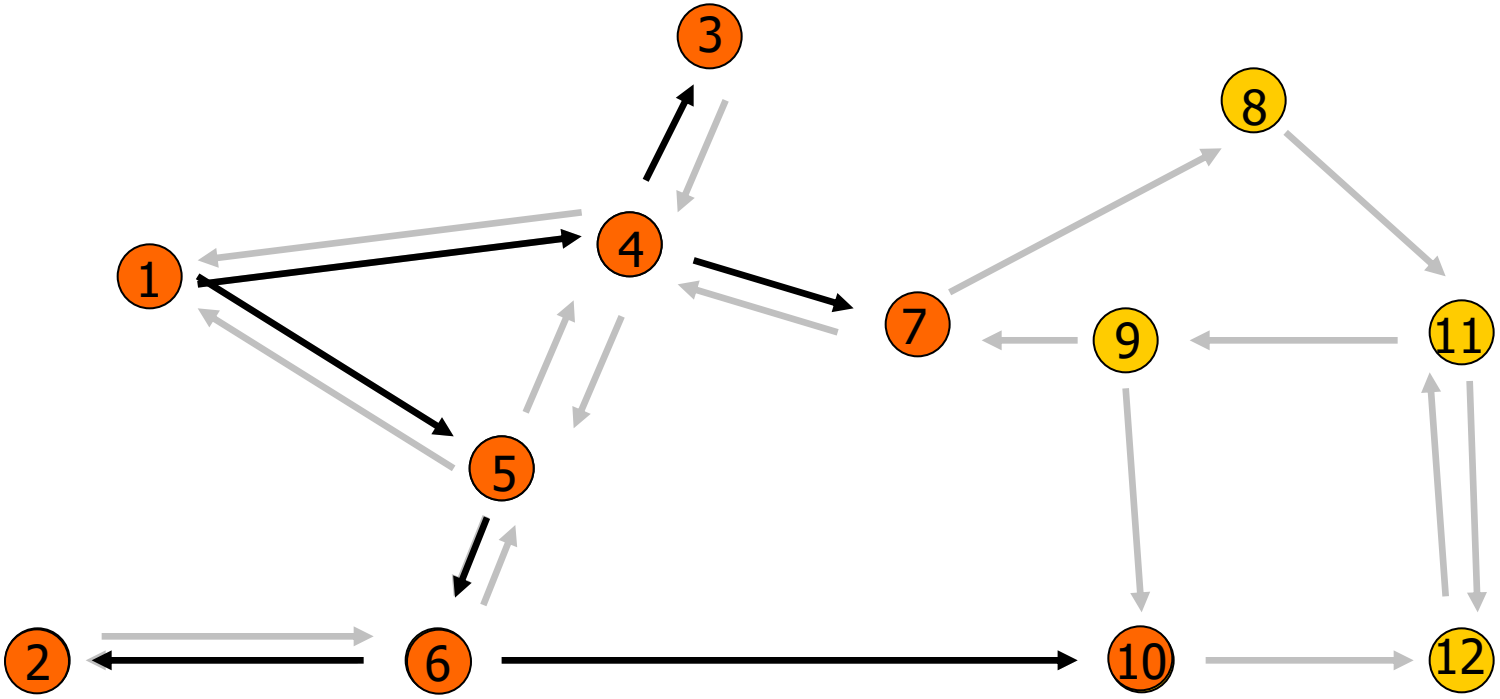
Exploration en largeur



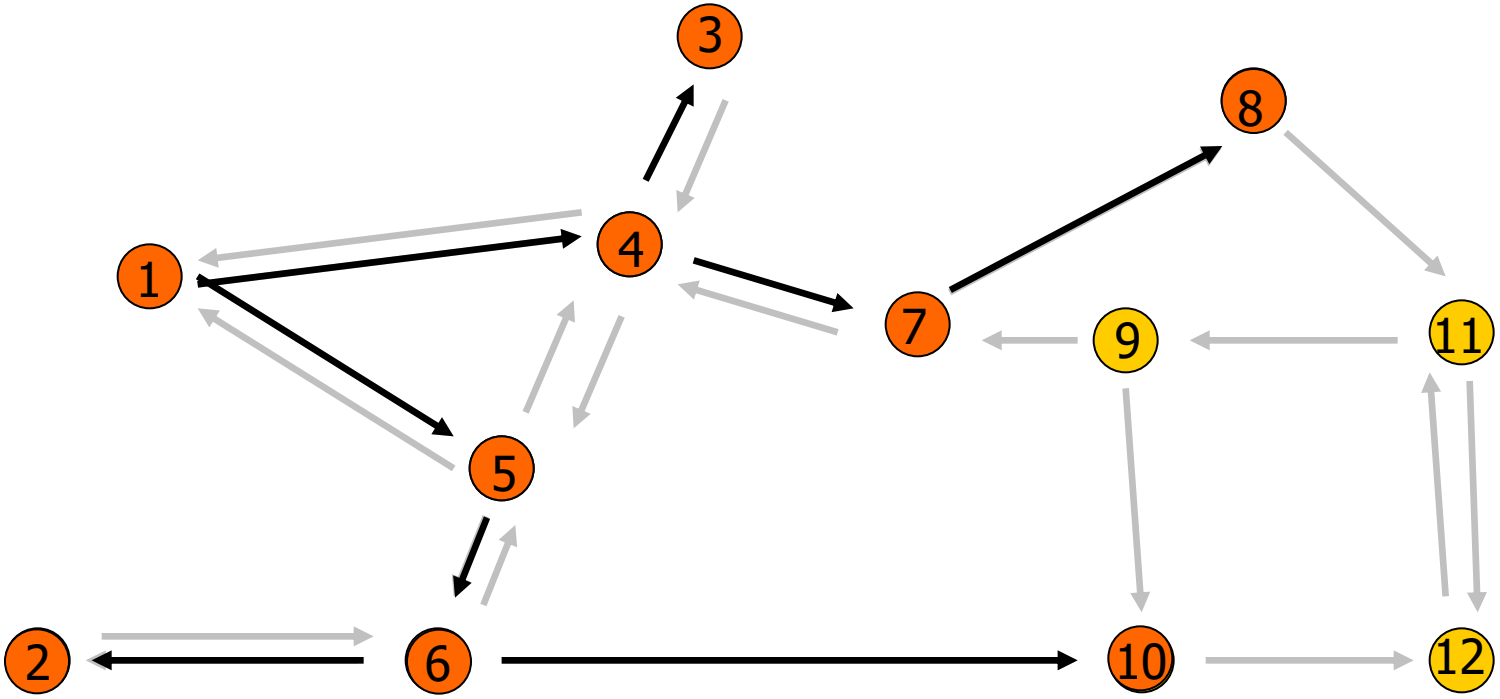
Exploration en largeur



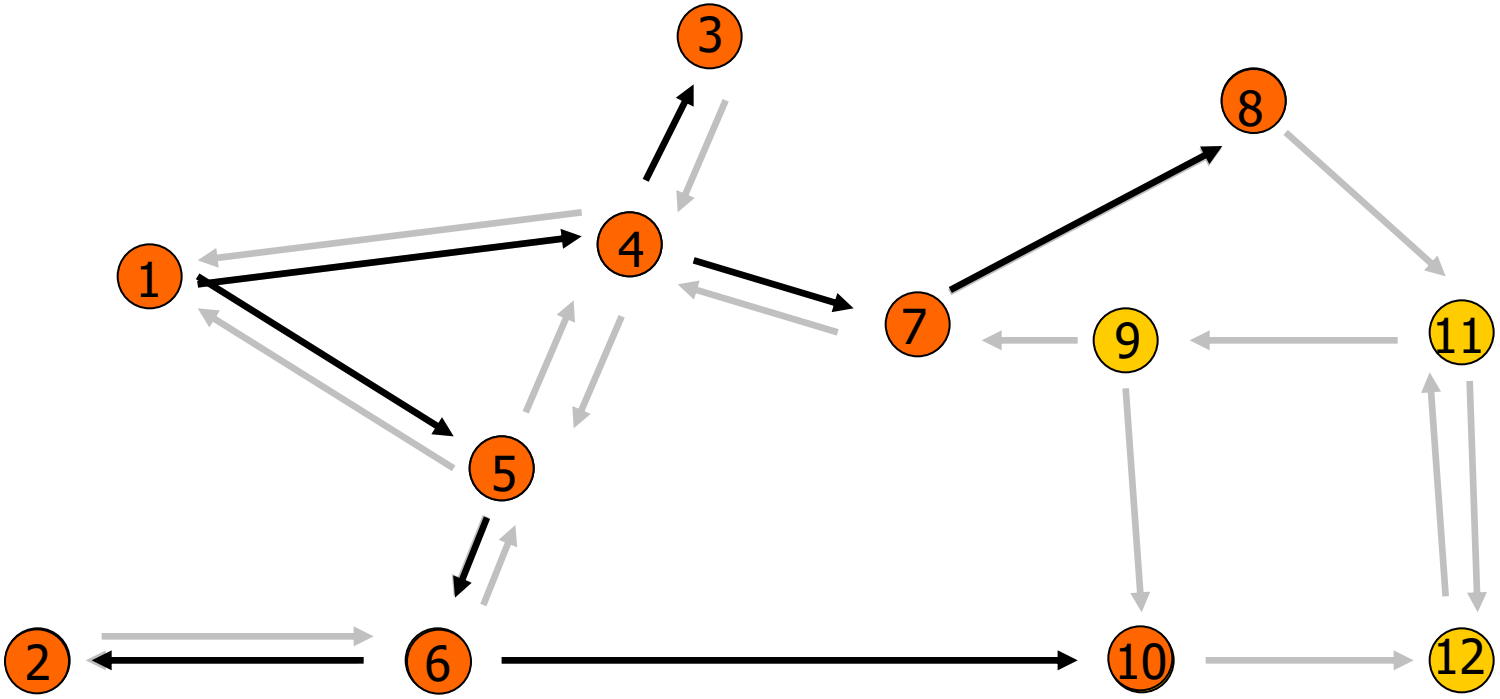
Exploration en largeur



Exploration en largeur

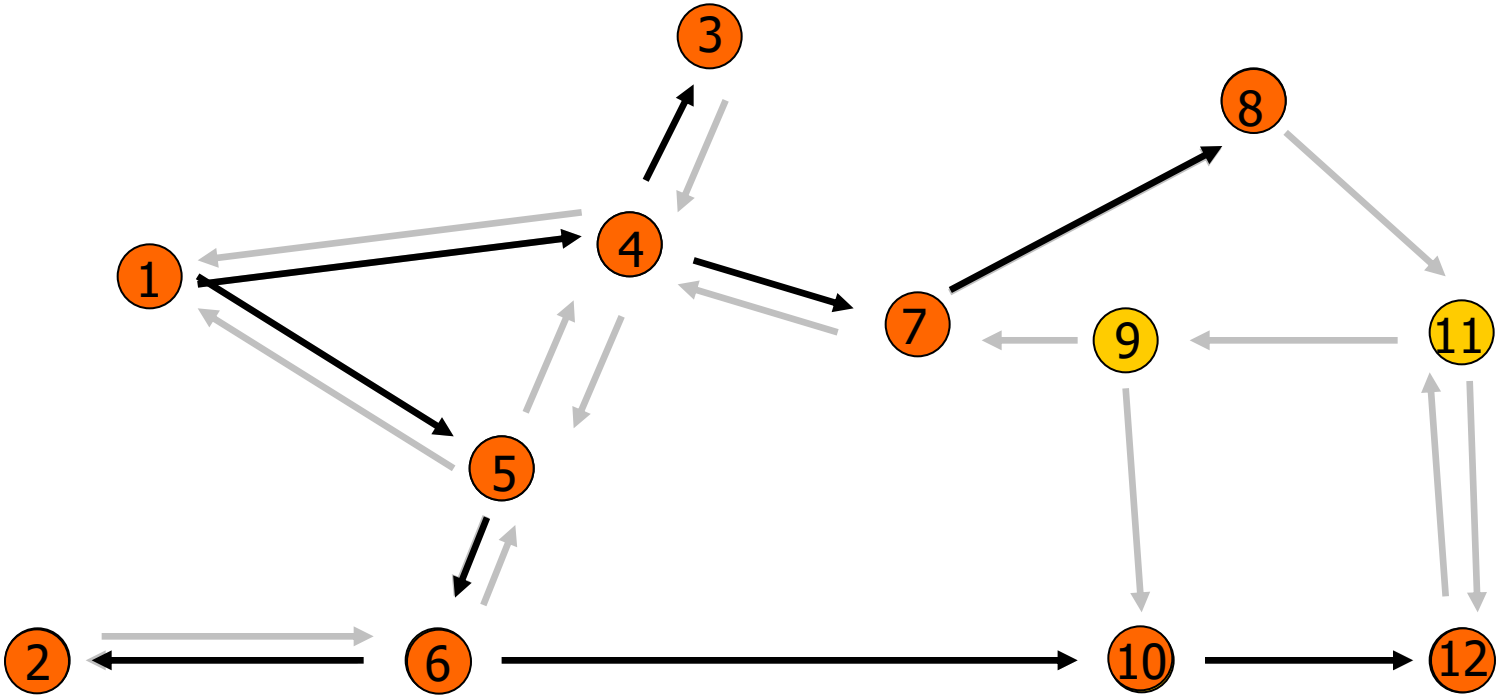


Exploration en largeur



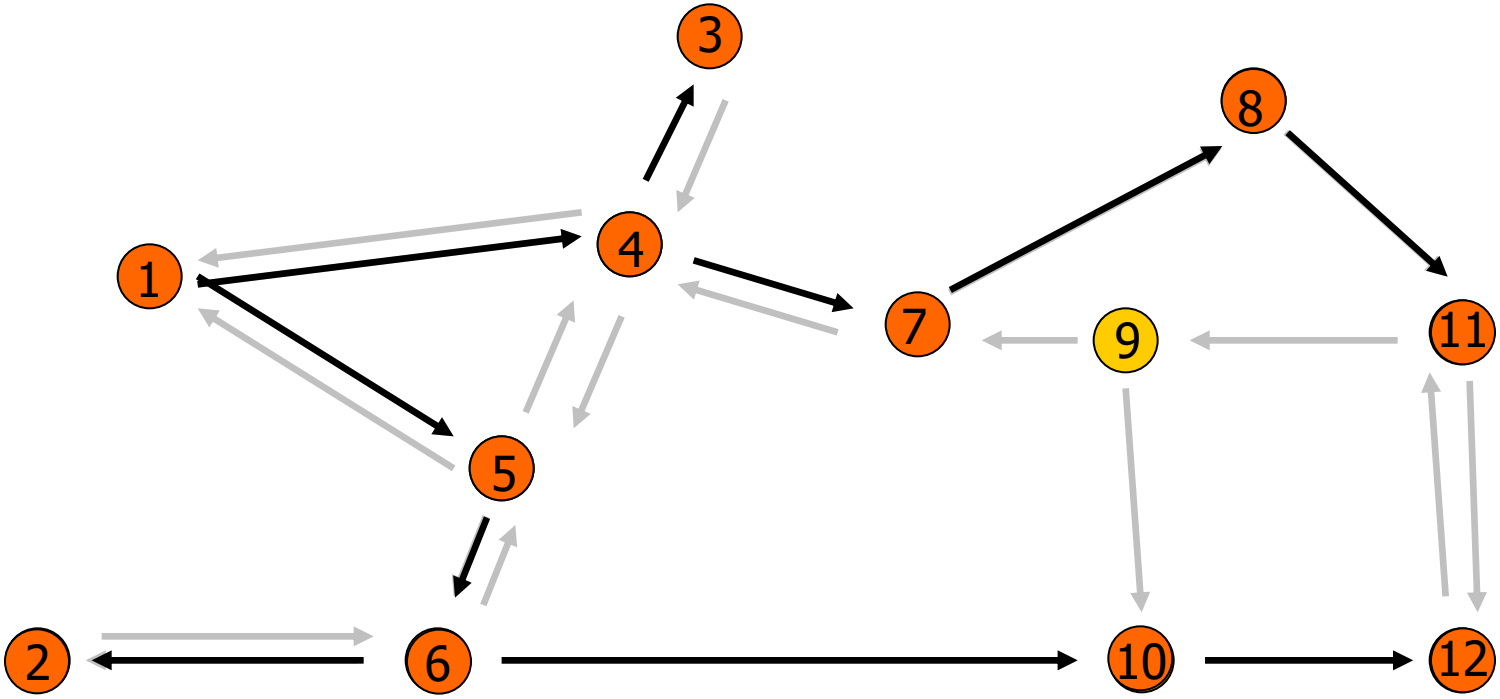
File 2 ← 10 8 ←

Exploration en largeur



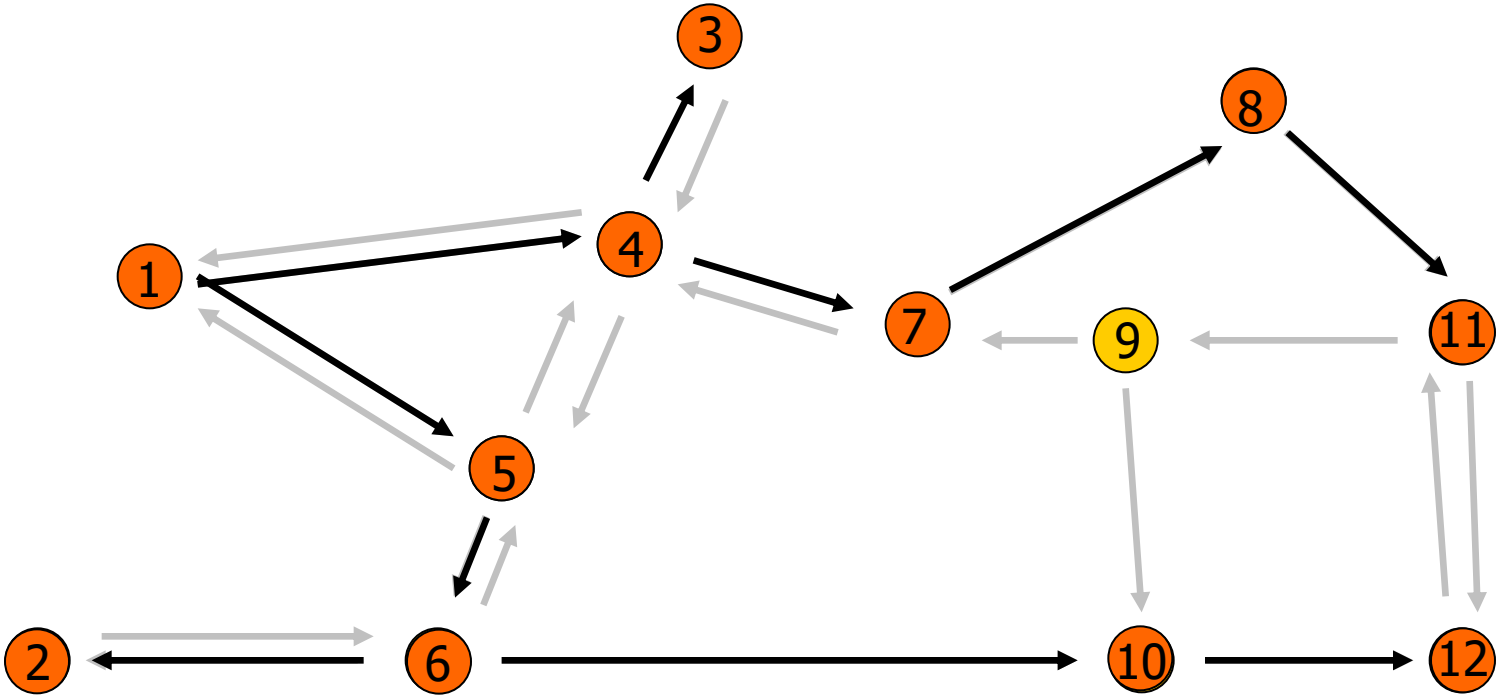
File 10 ← 8 12 ←

Exploration en largeur



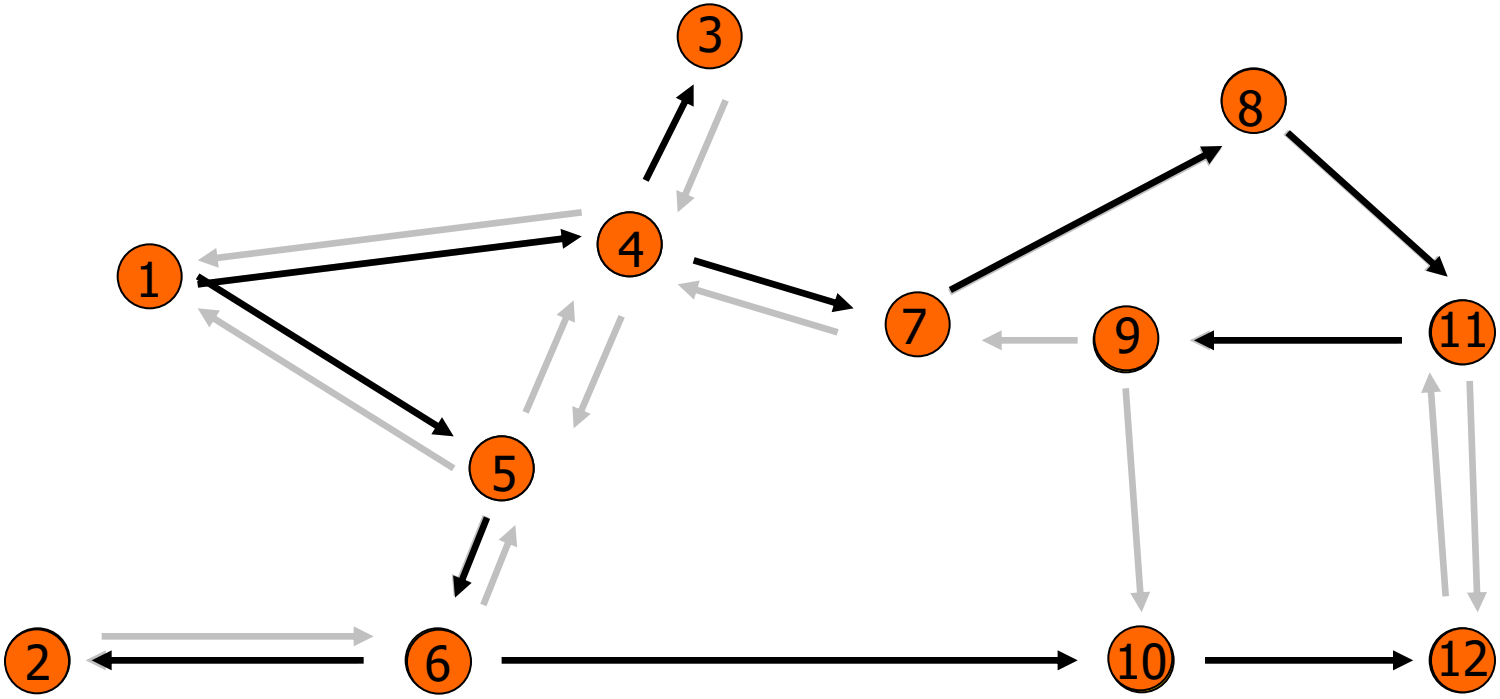
File 8 ← 12 11 ←

Exploration en largeur

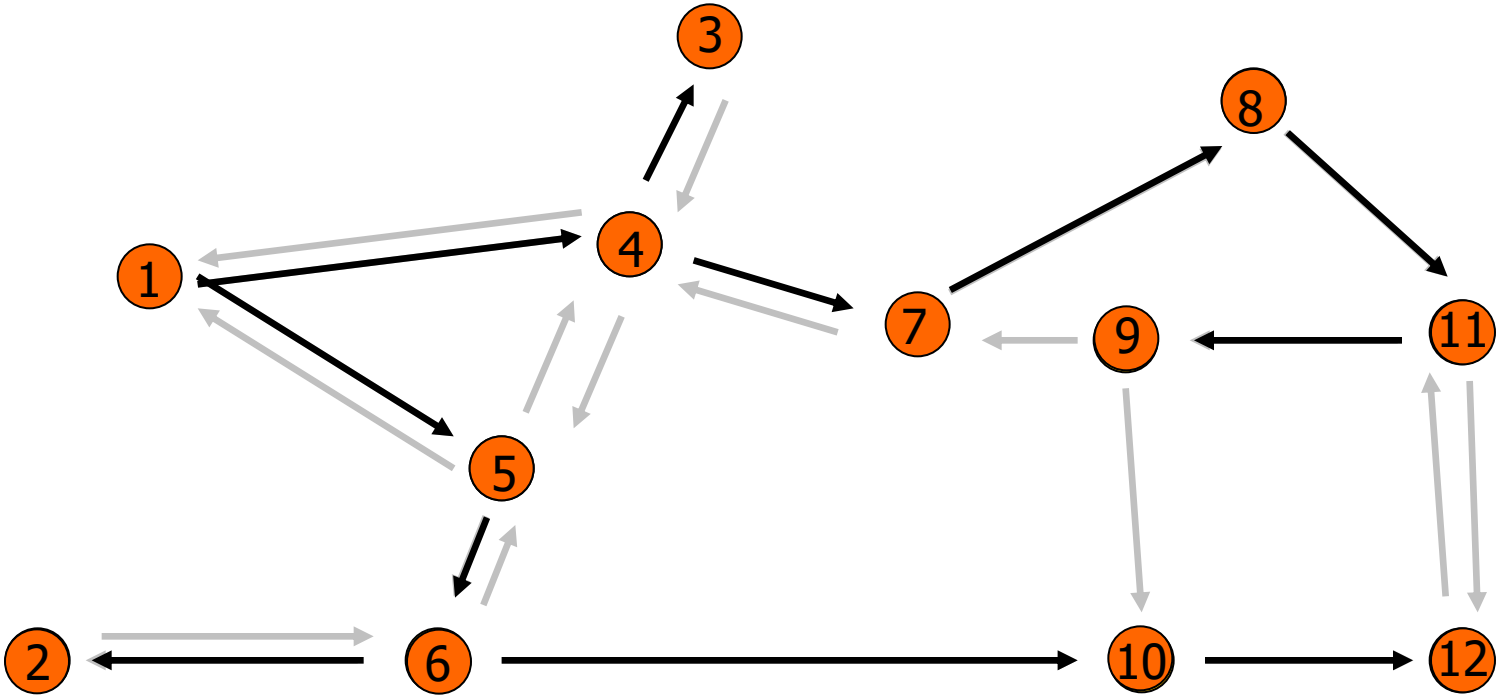


File 12 11

Exploration en largeur

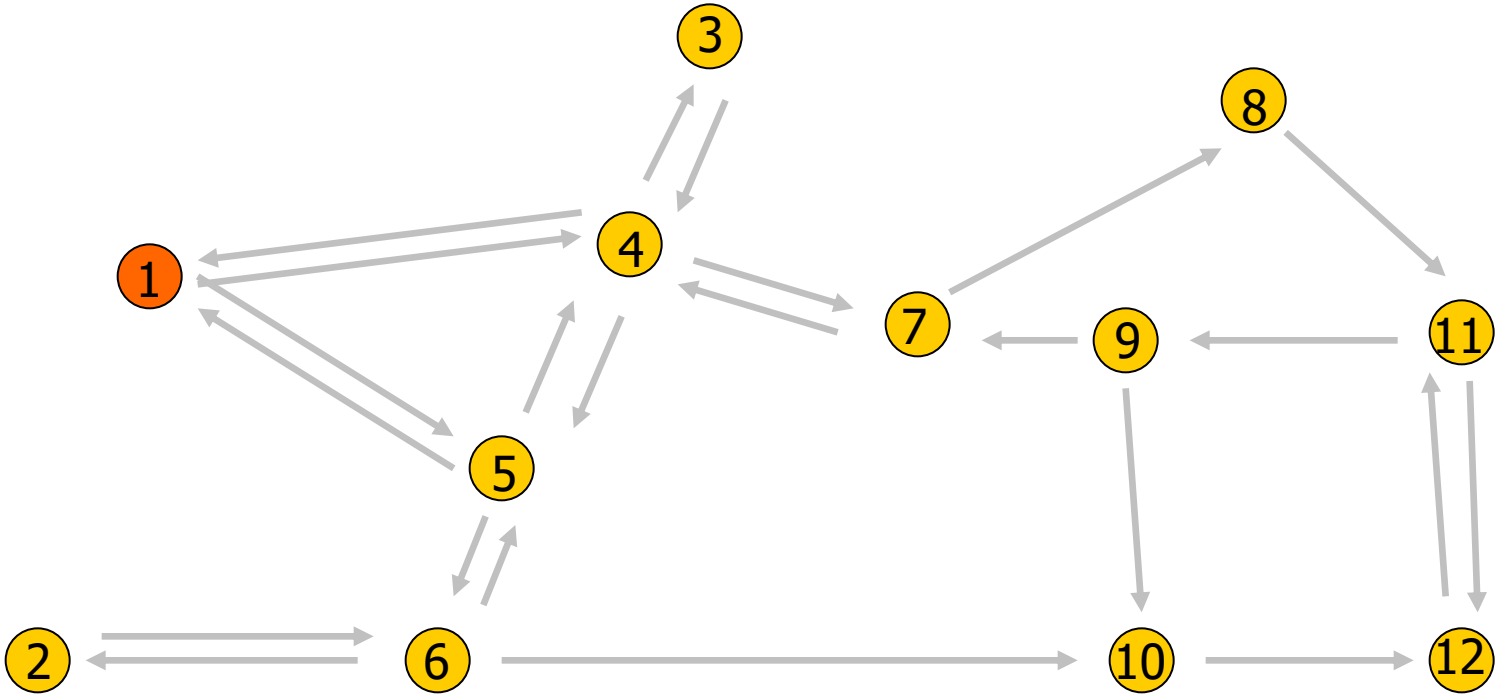


Exploration en largeur



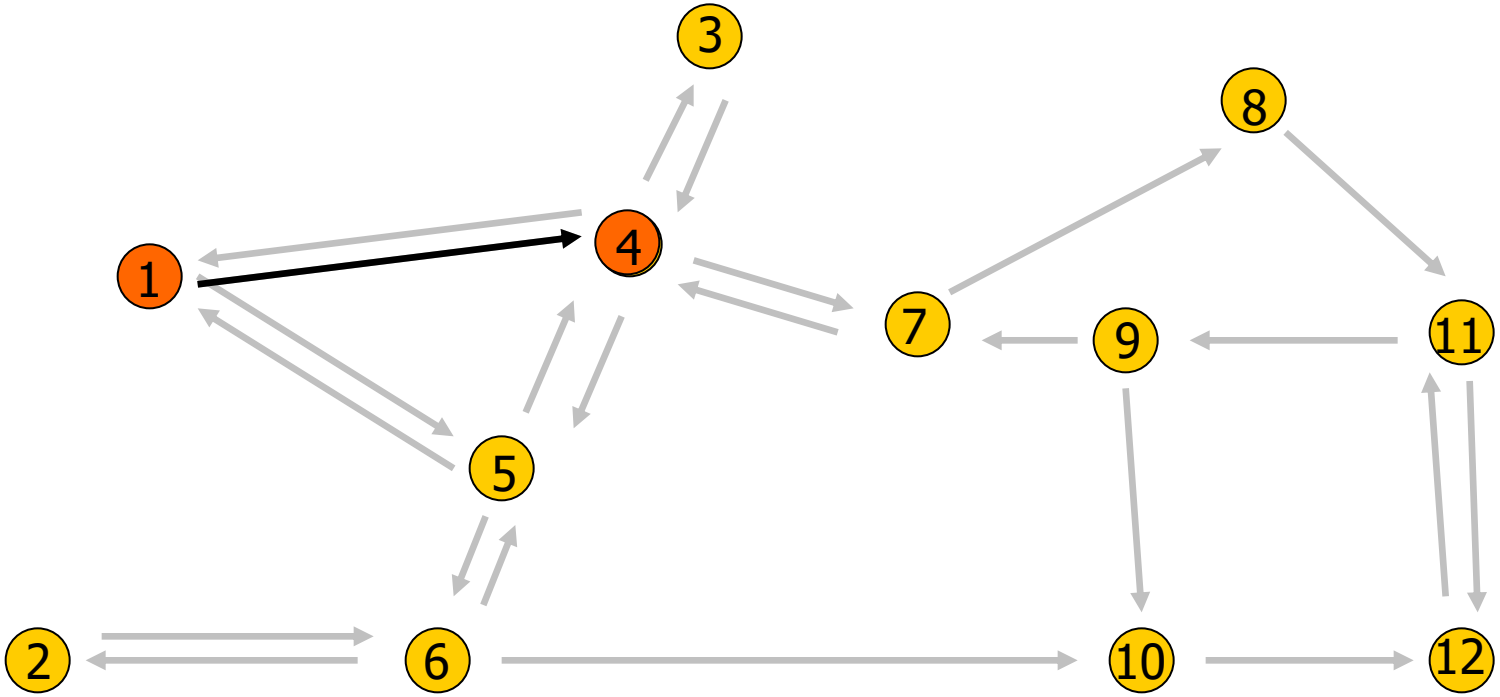
File 9

Exploration en Profondeur



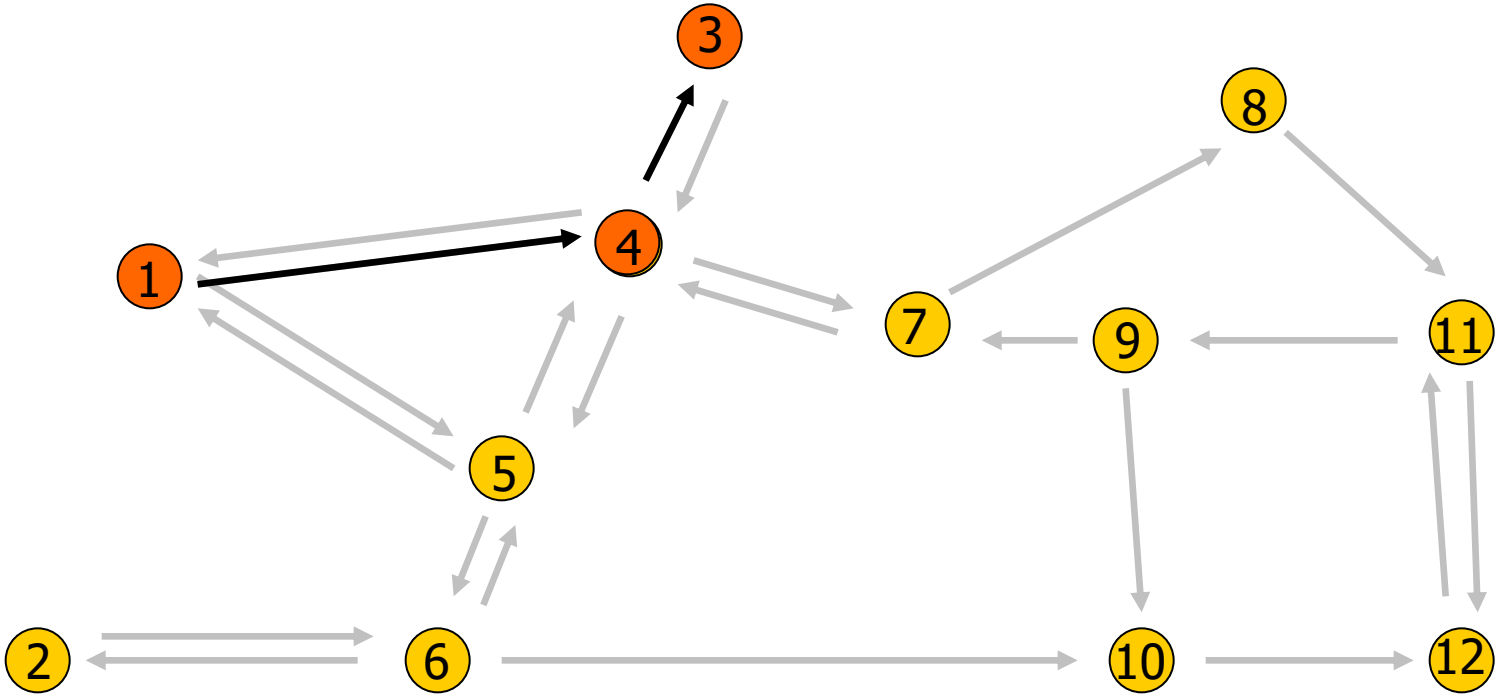
Pile

Exploration en Profondeur



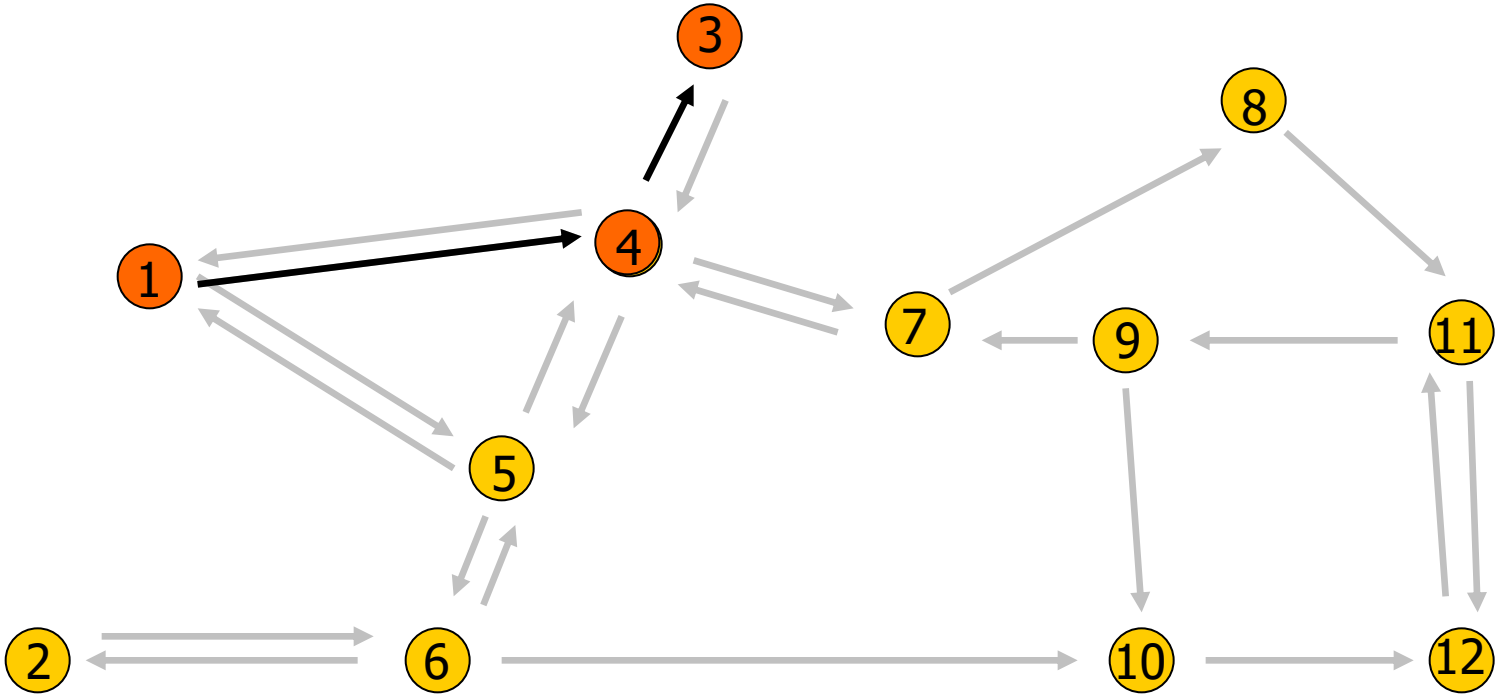
Pile

Exploration en Profondeur



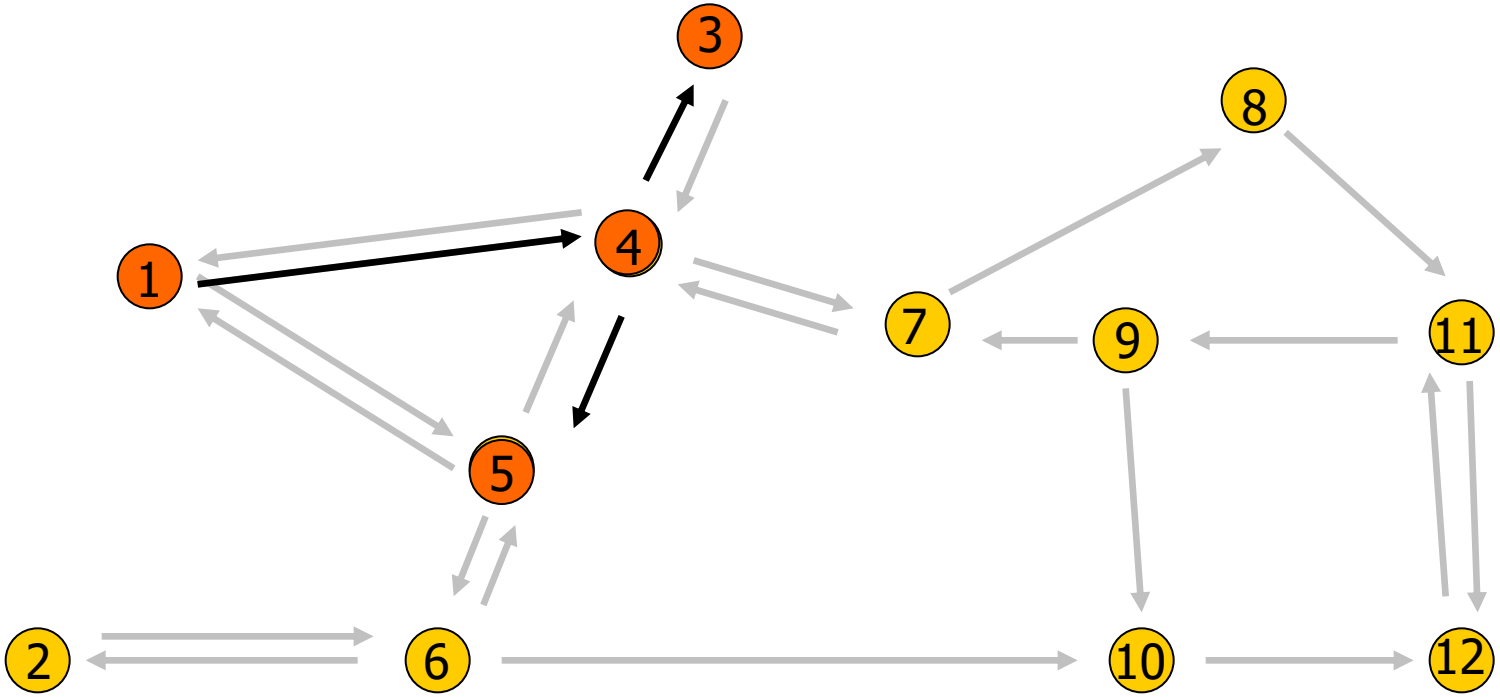
Pile

Exploration en Profondeur



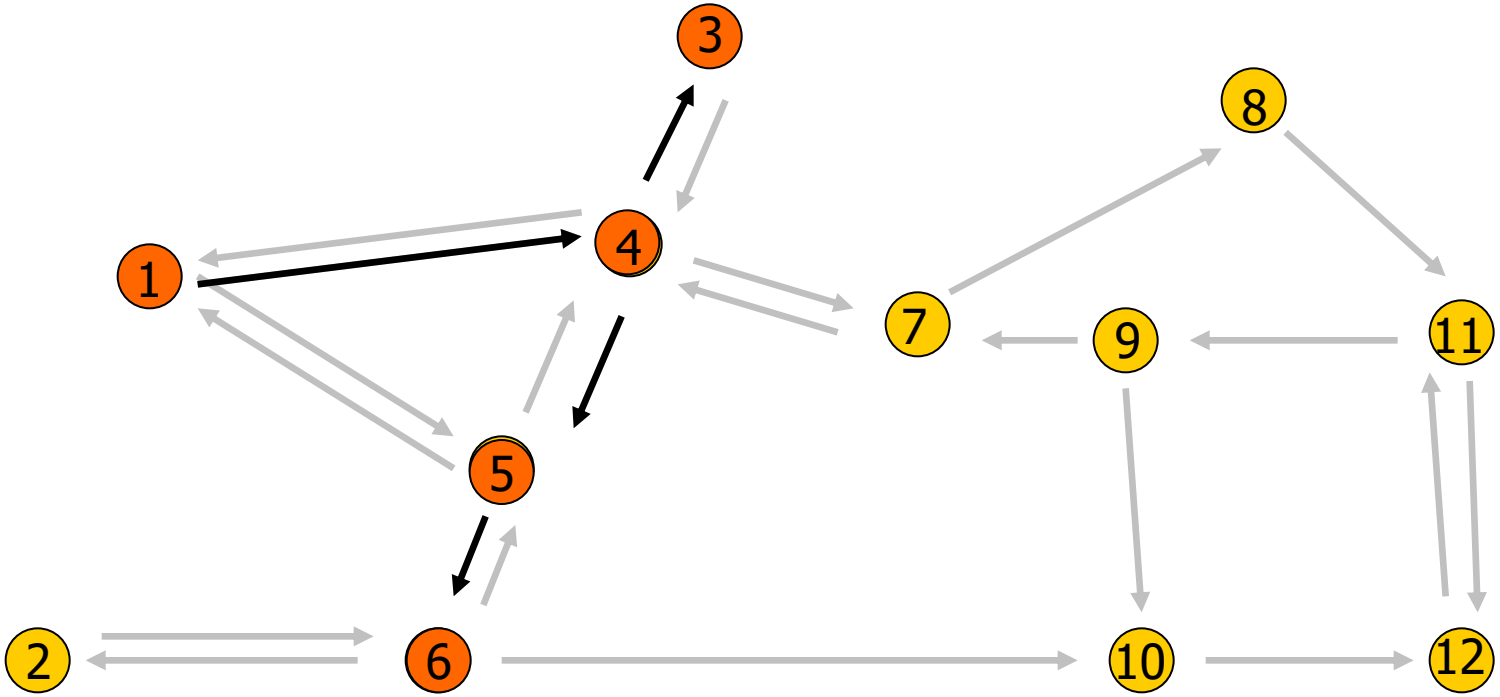
Pile

Exploration en Profondeur



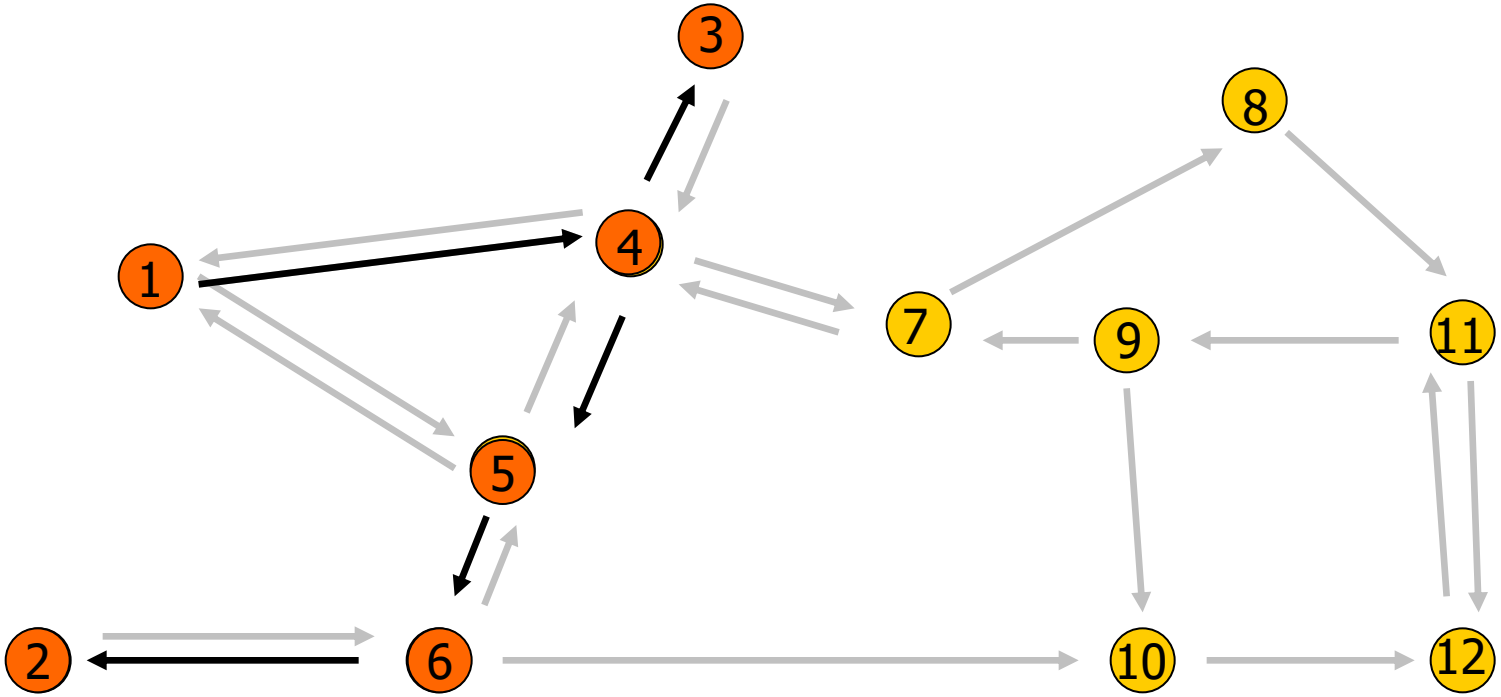
Pile

Exploration en Profondeur



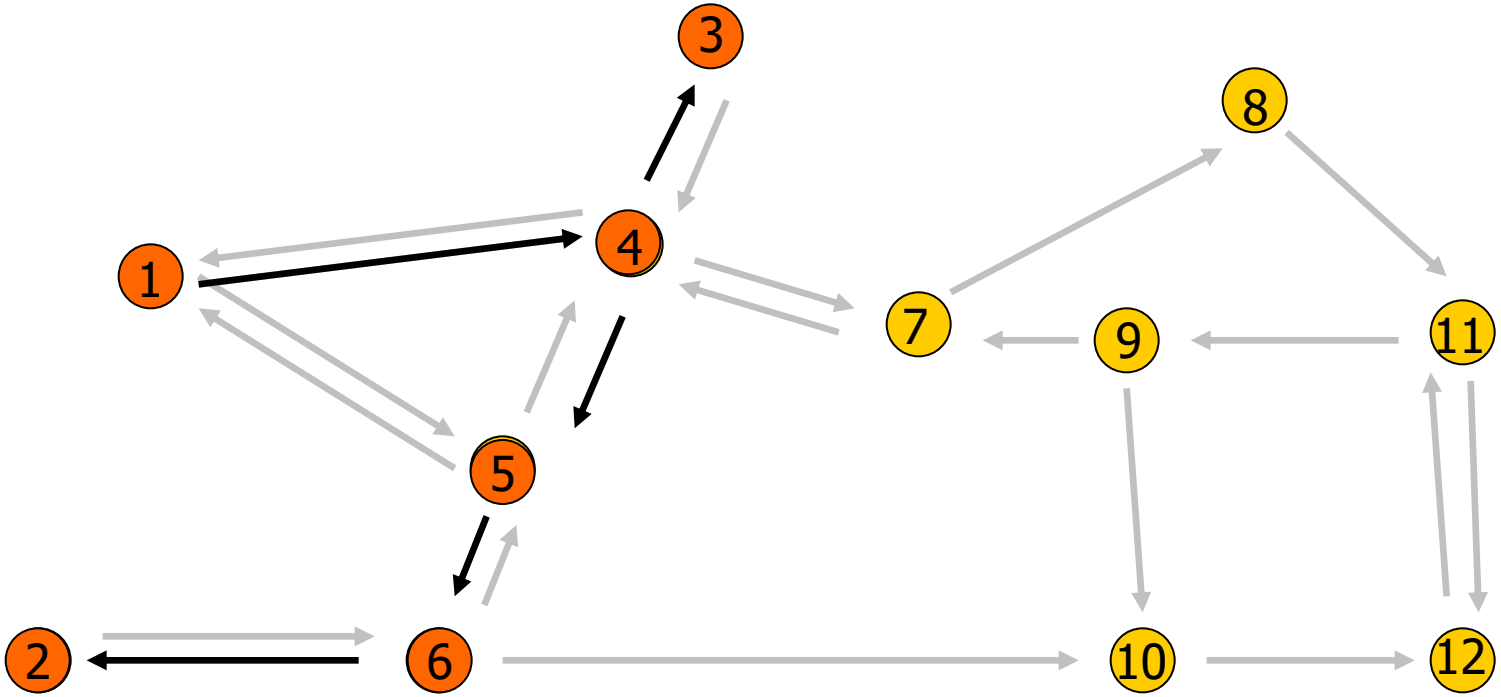
Pile

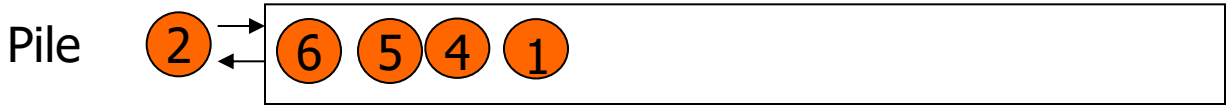
Exploration en Profondeur



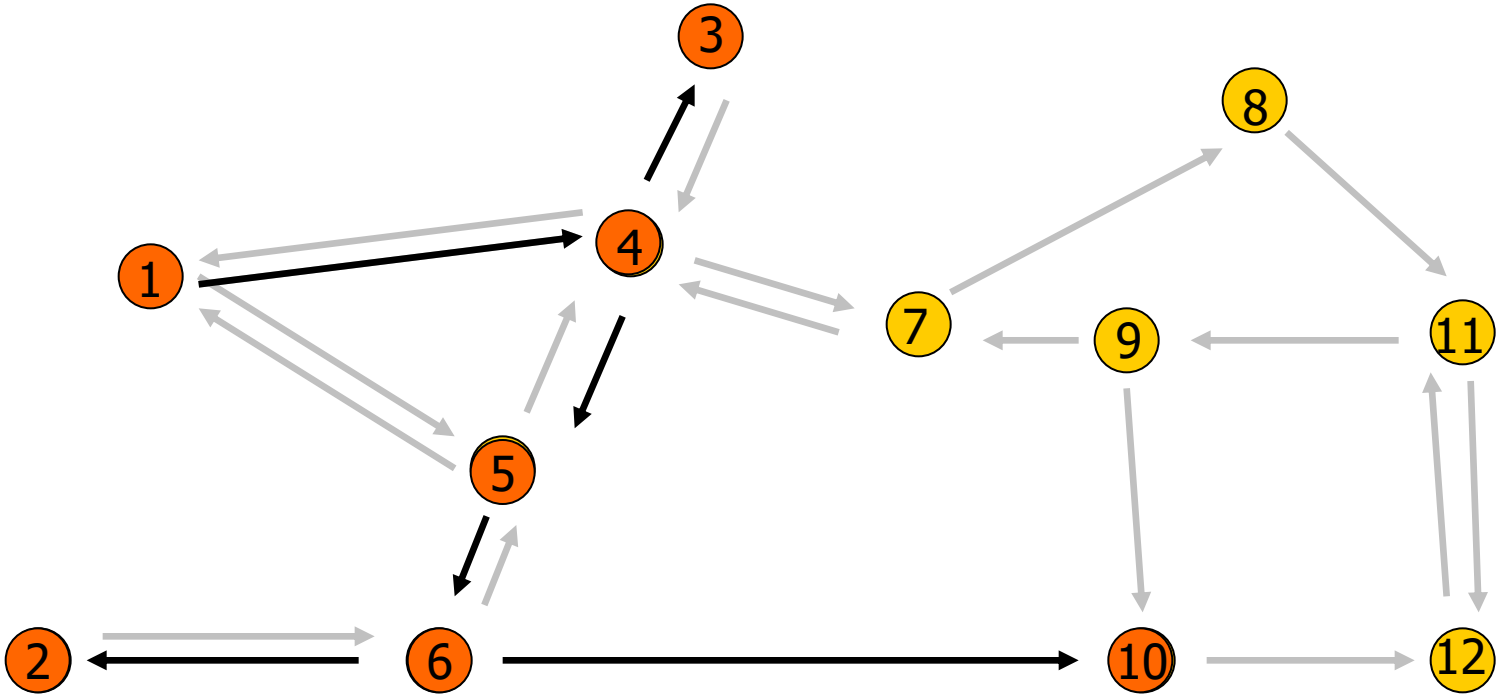
Pile

Exploration en Profondeur



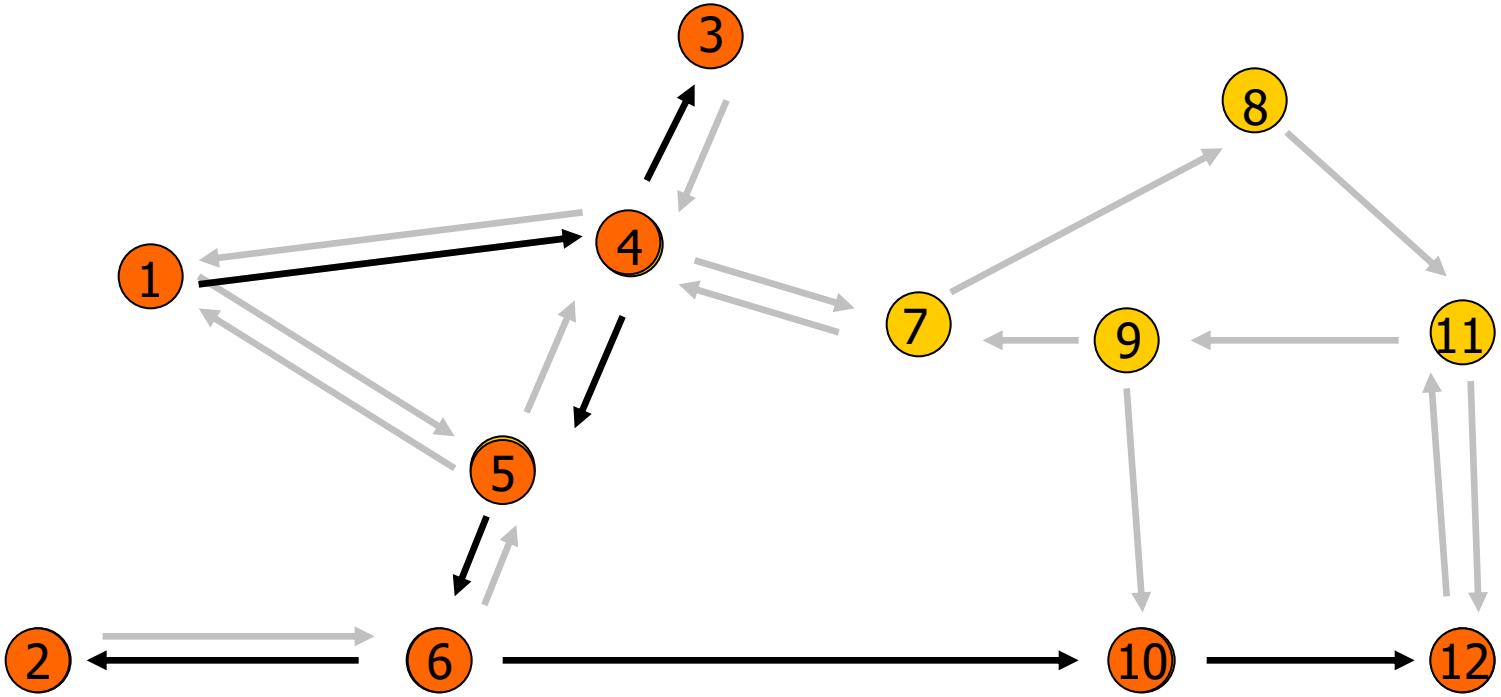


Exploration en Profondeur

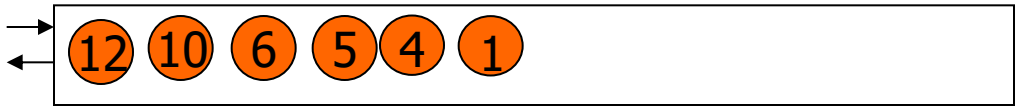


Pile

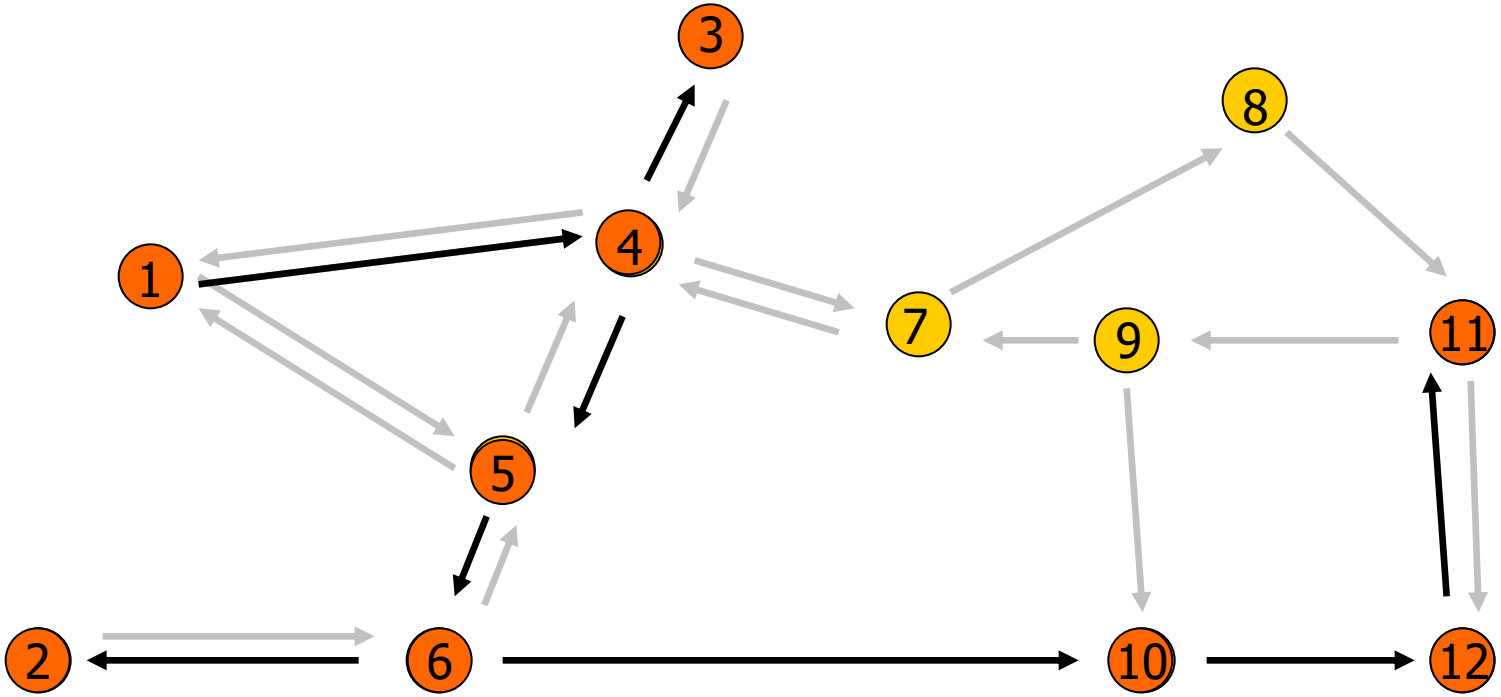
Exploration en Profondeur



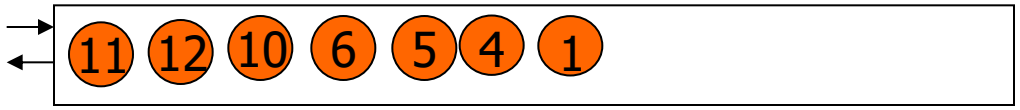
Pile



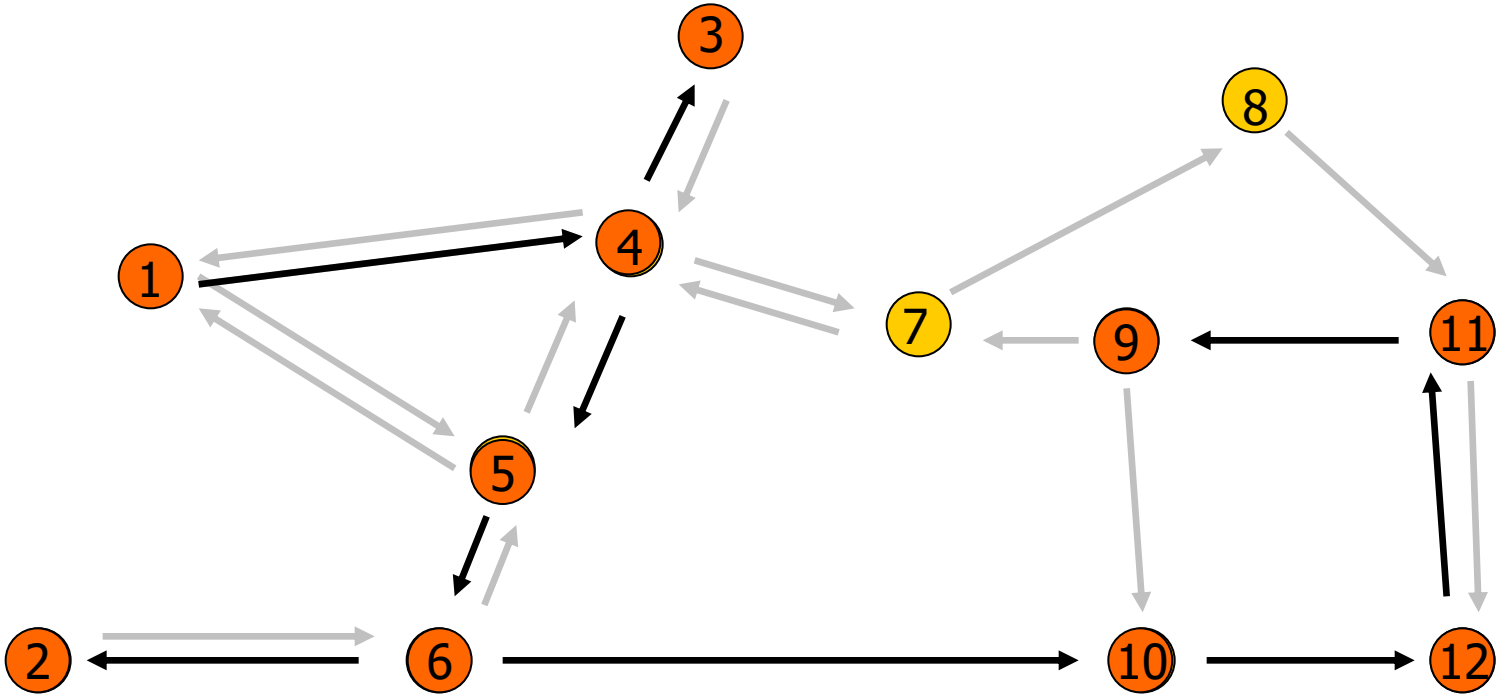
Exploration en Profondeur



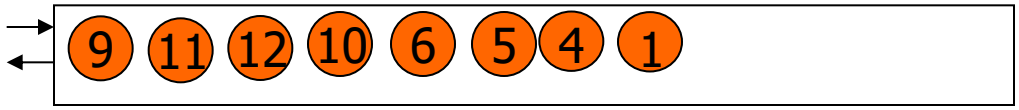
Pile



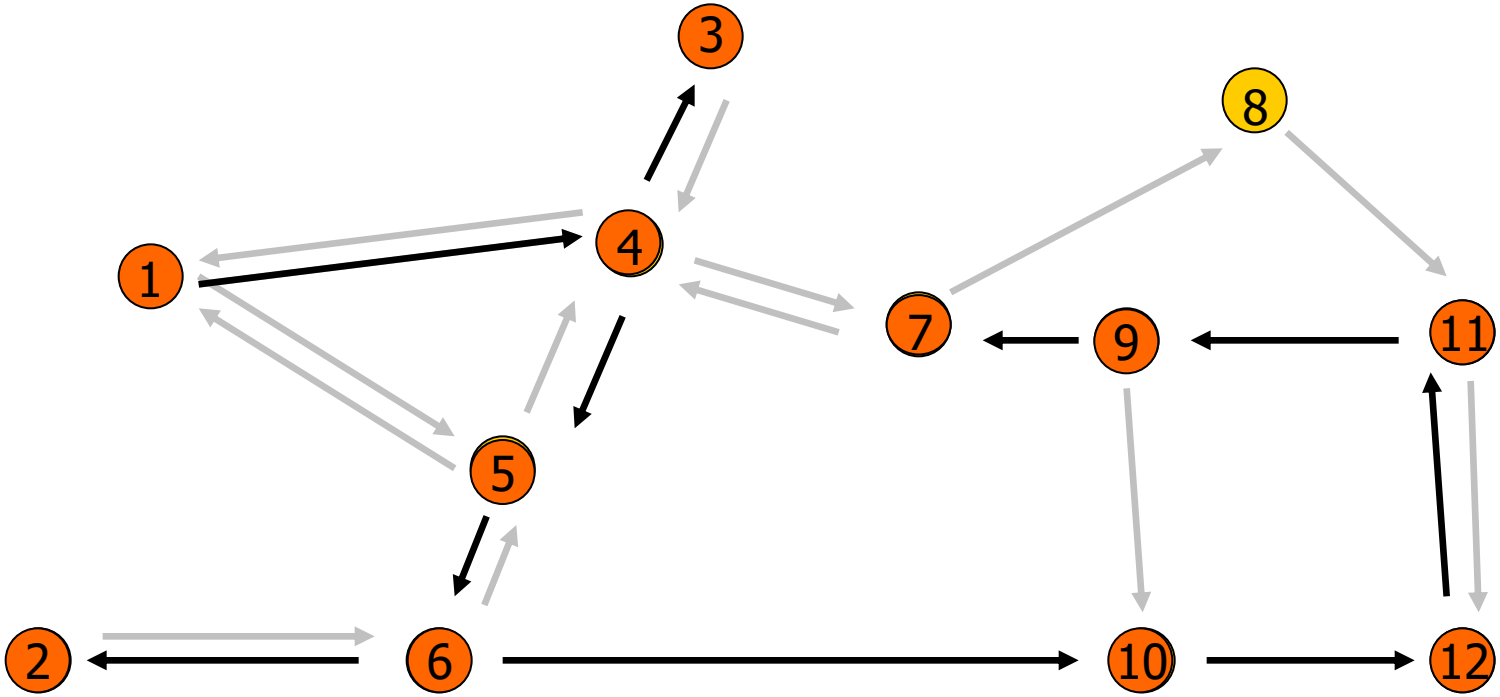
Exploration en Profondeur



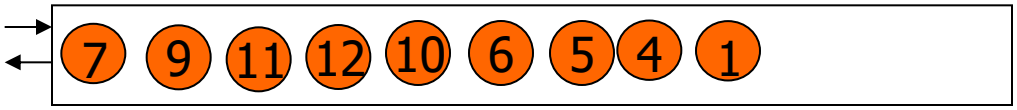
Pile



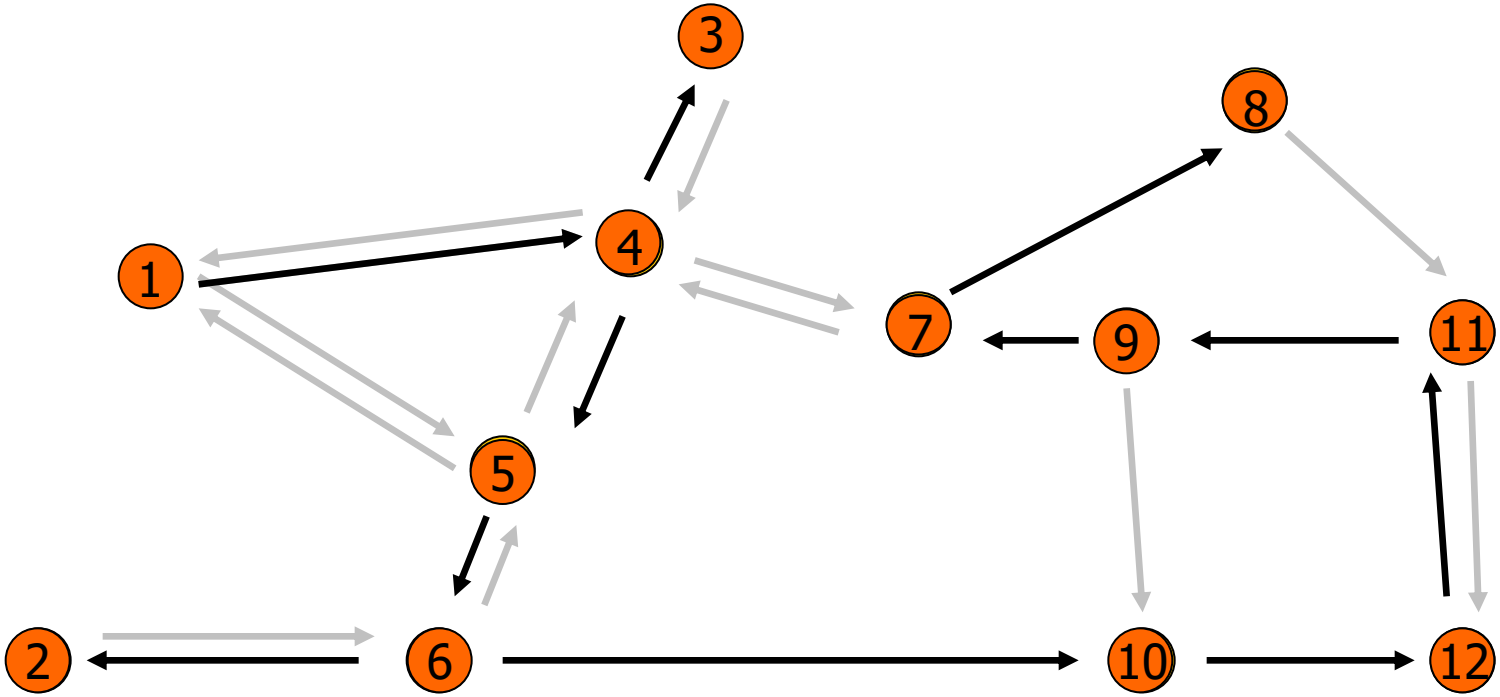
Exploration en Profondeur



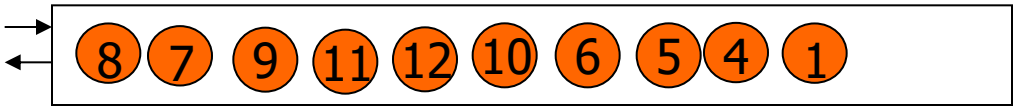
Pile



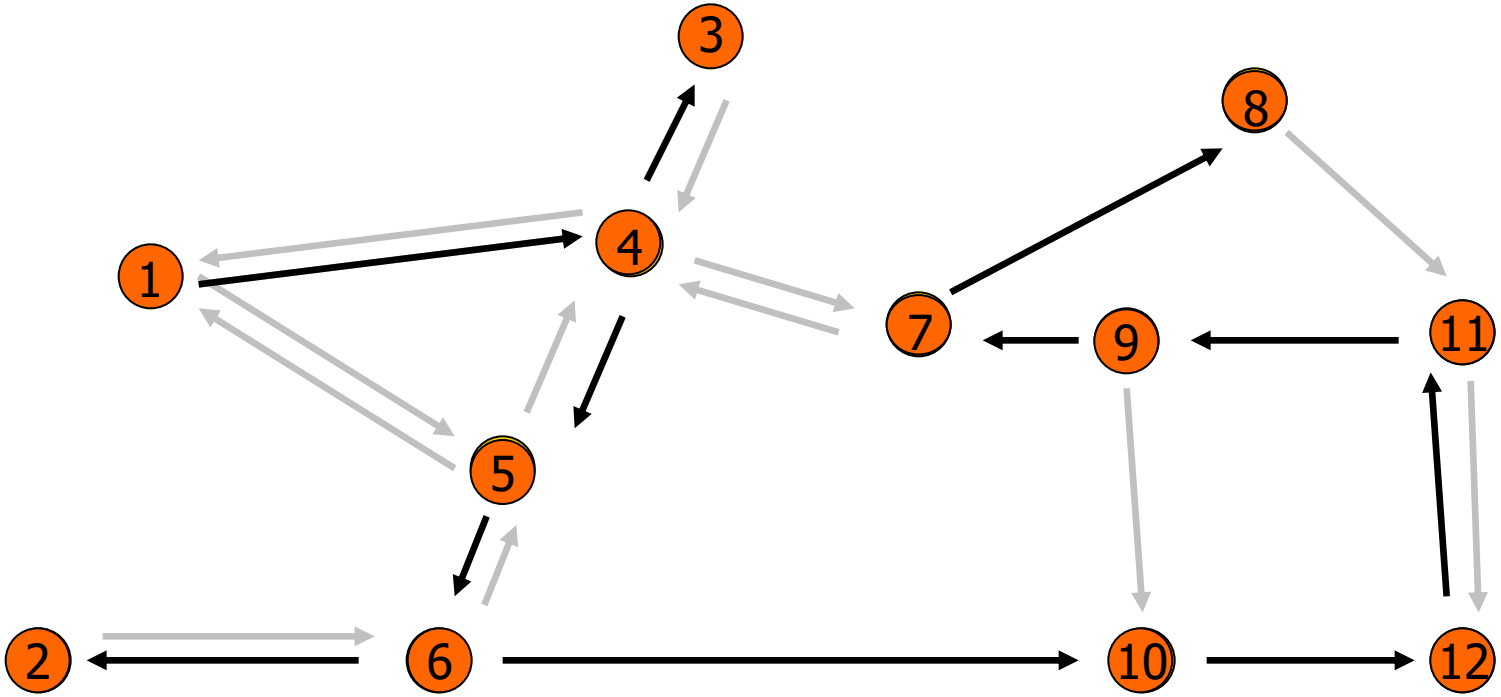
Exploration en Profondeur

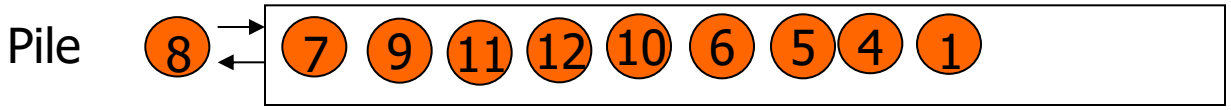


Pile

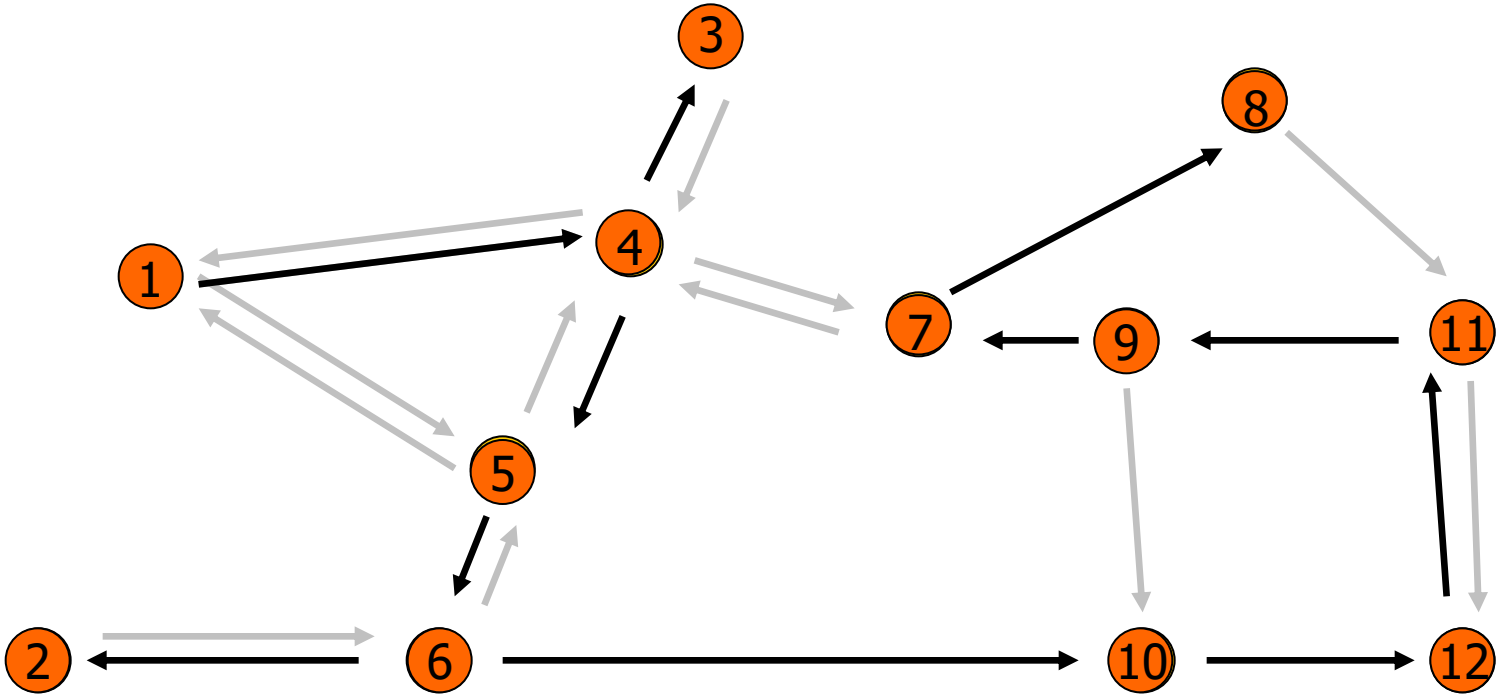


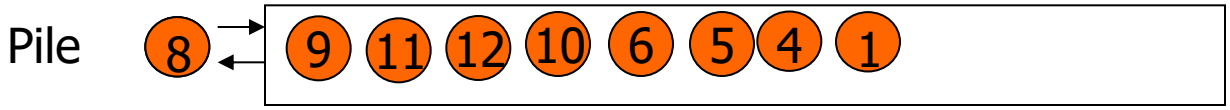
Exploration en Profondeur



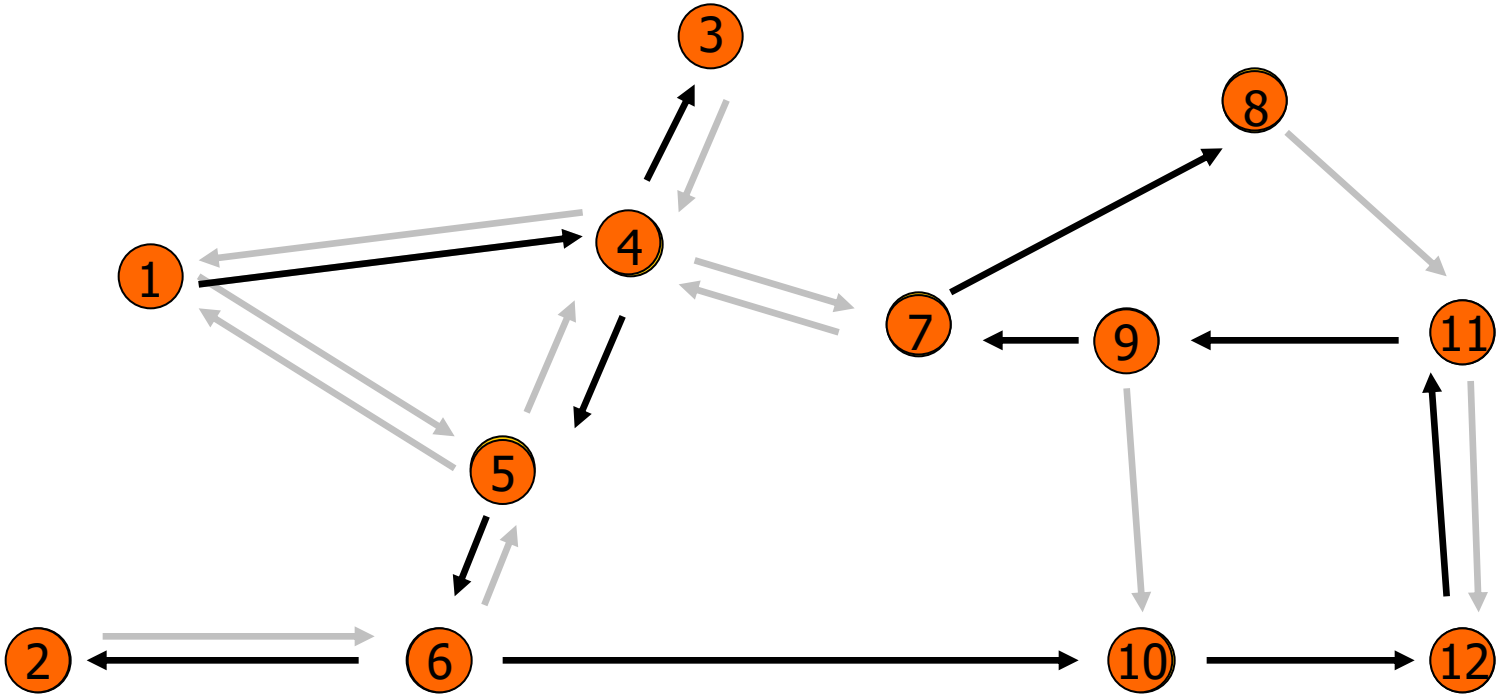


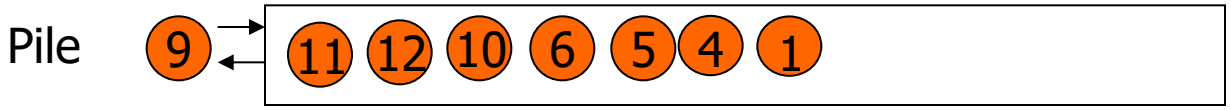
Exploration en Profondeur



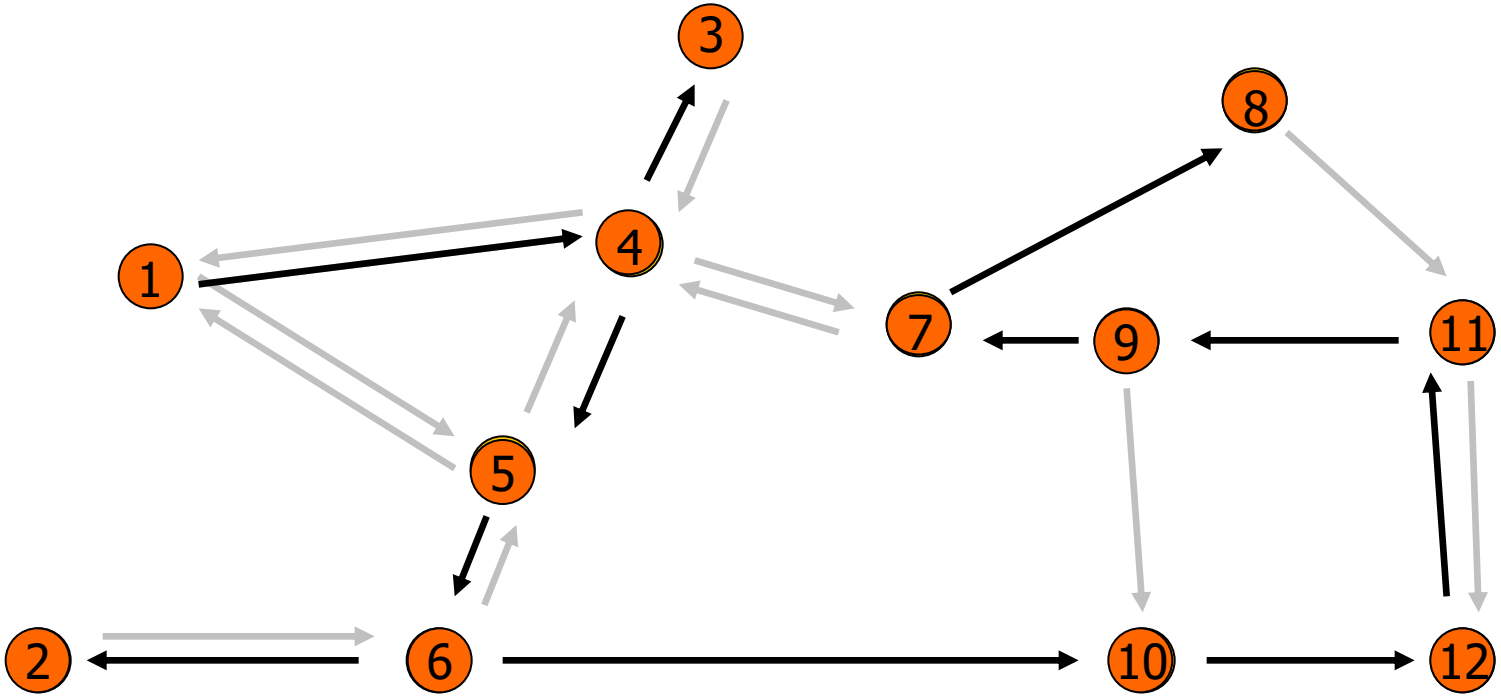


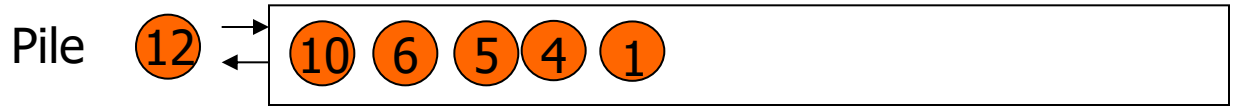
Exploration en Profondeur



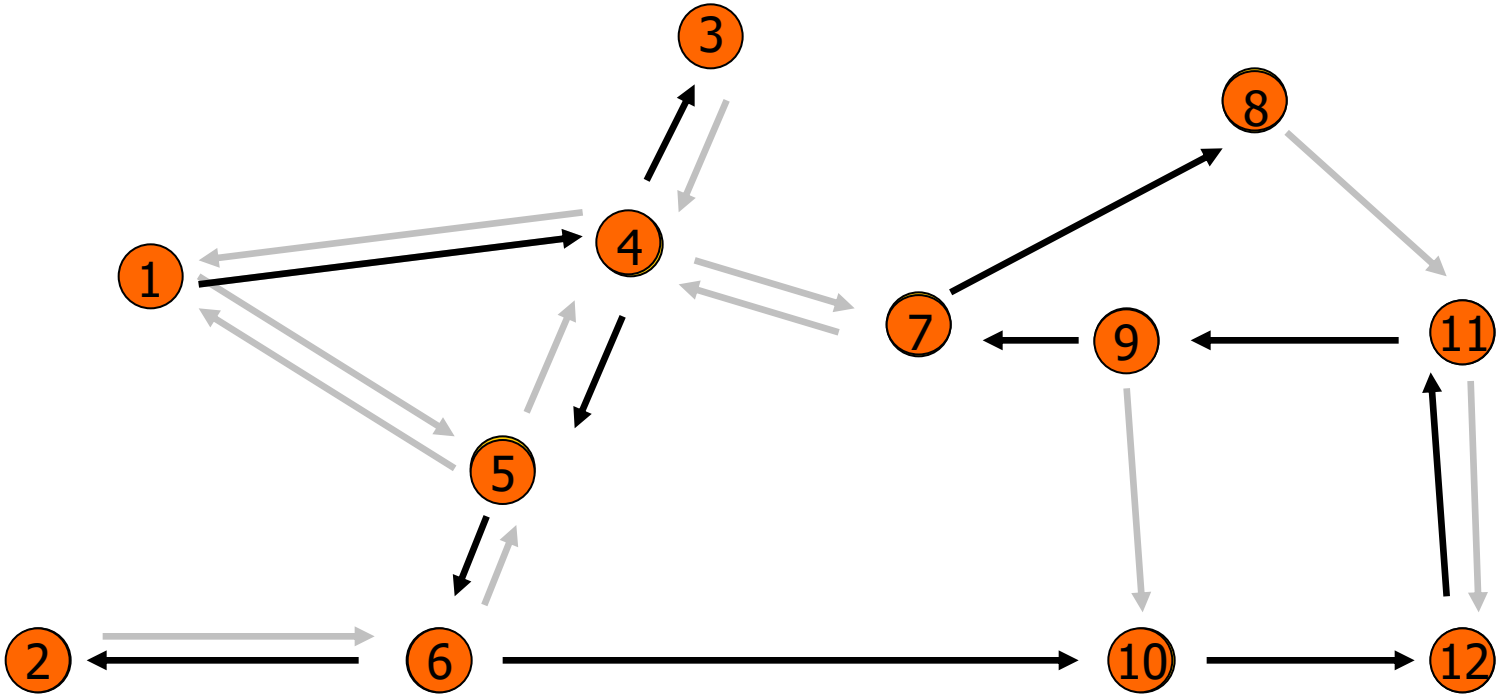


Exploration en Profondeur



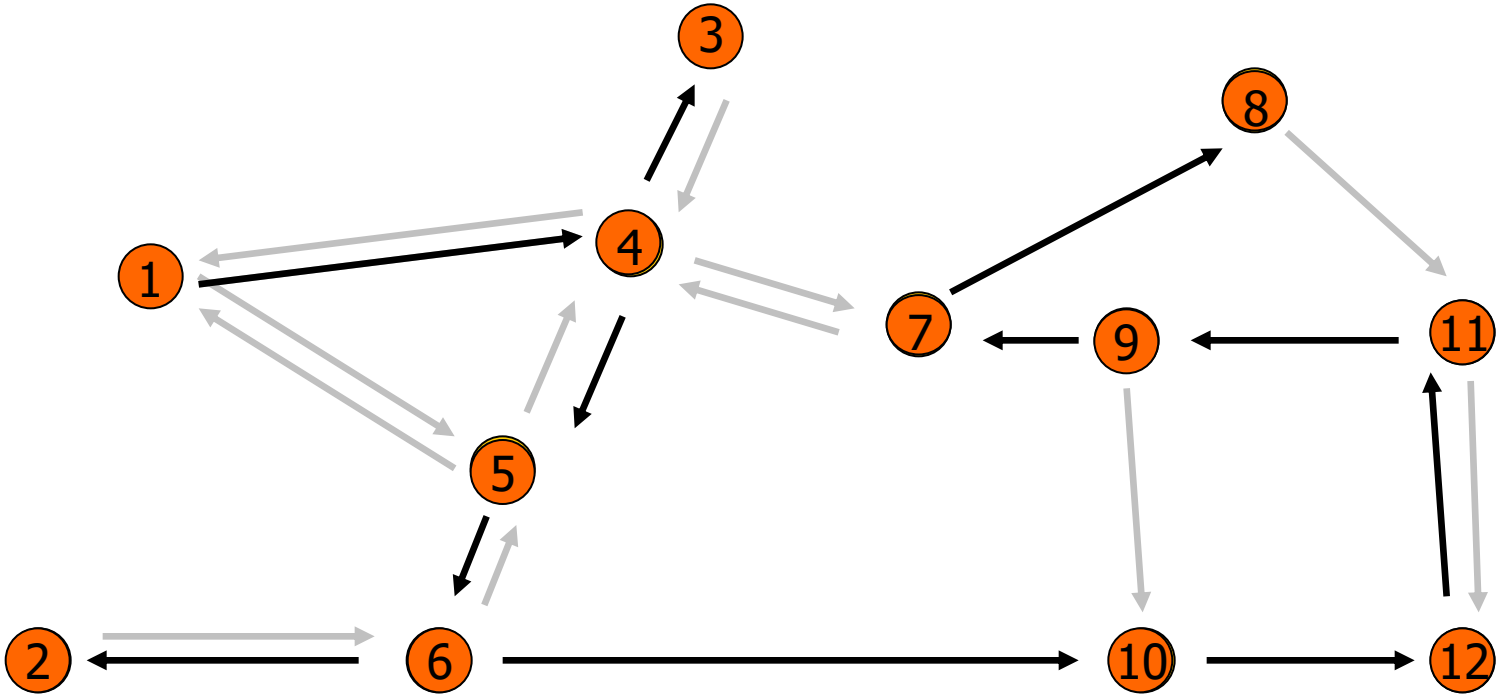


Exploration en Profondeur

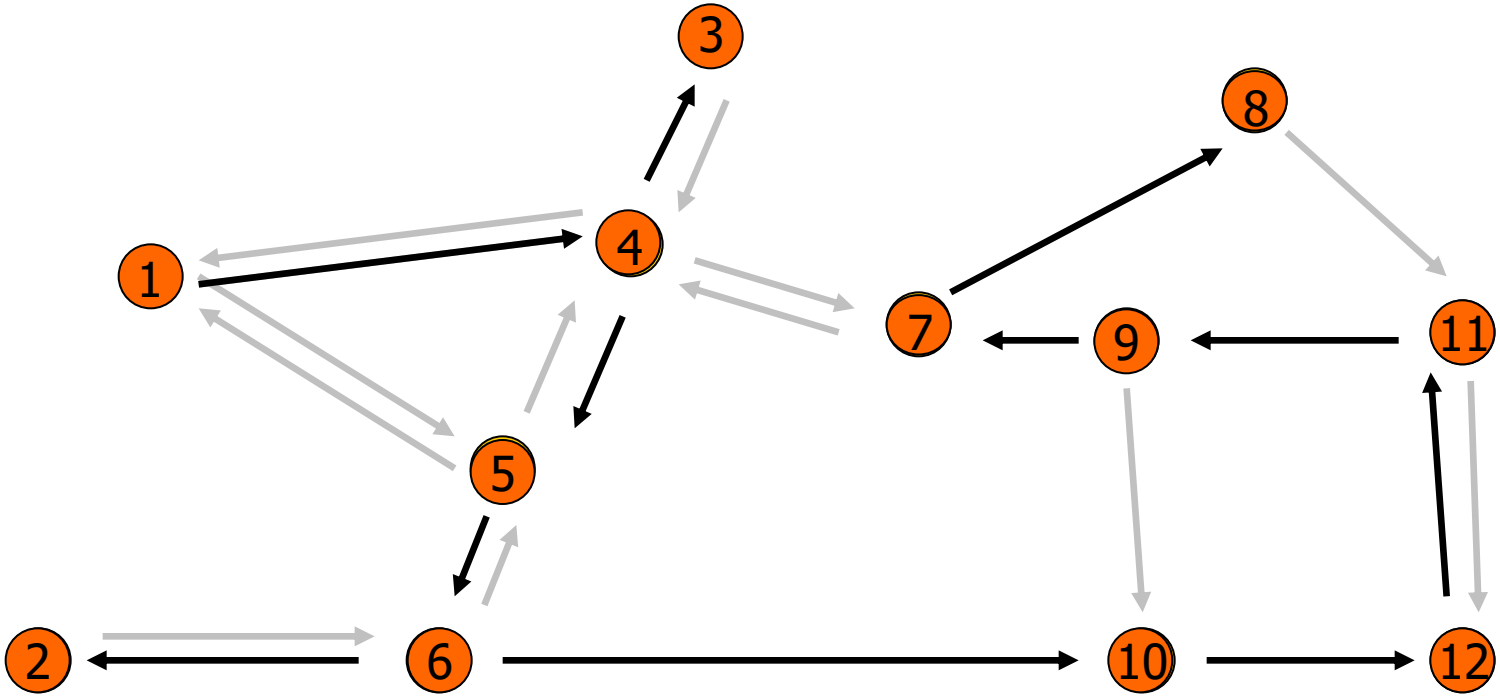


Pile

Exploration en Profondeur

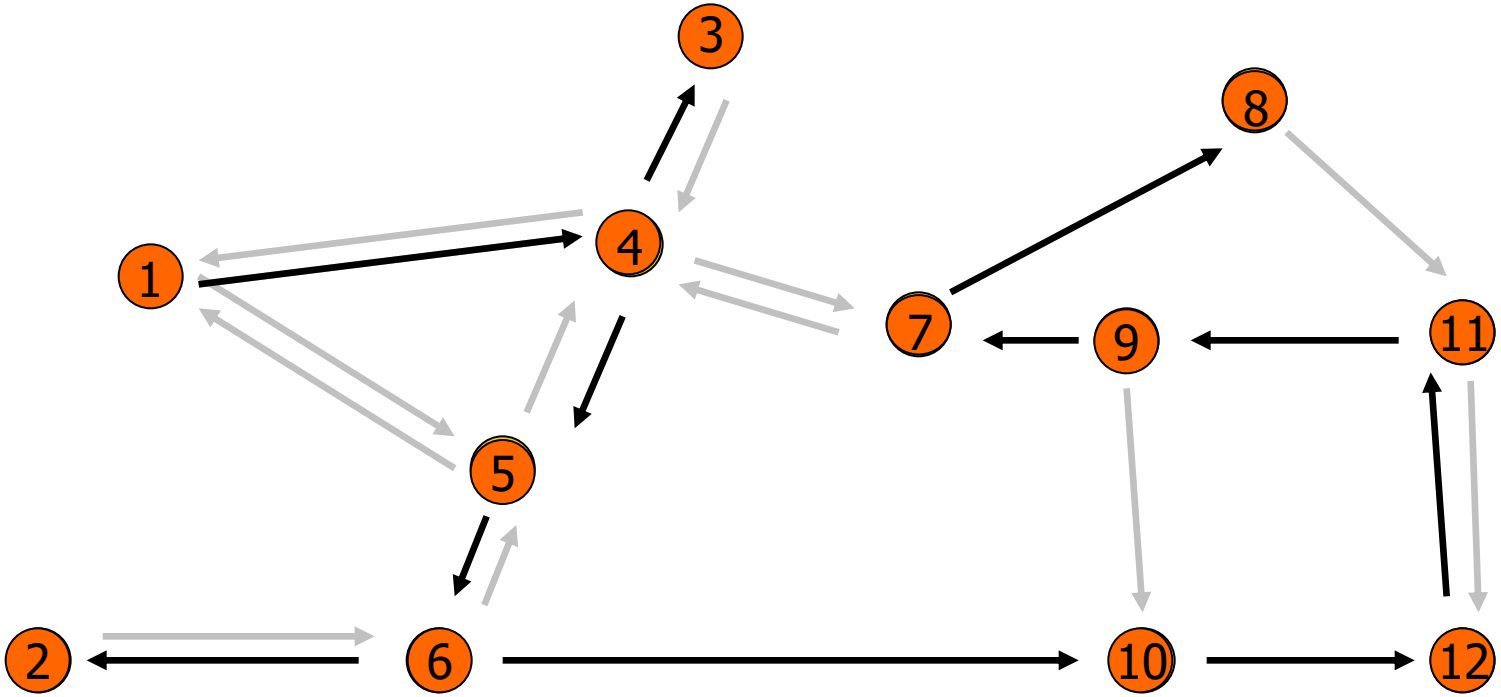


Exploration en Profondeur



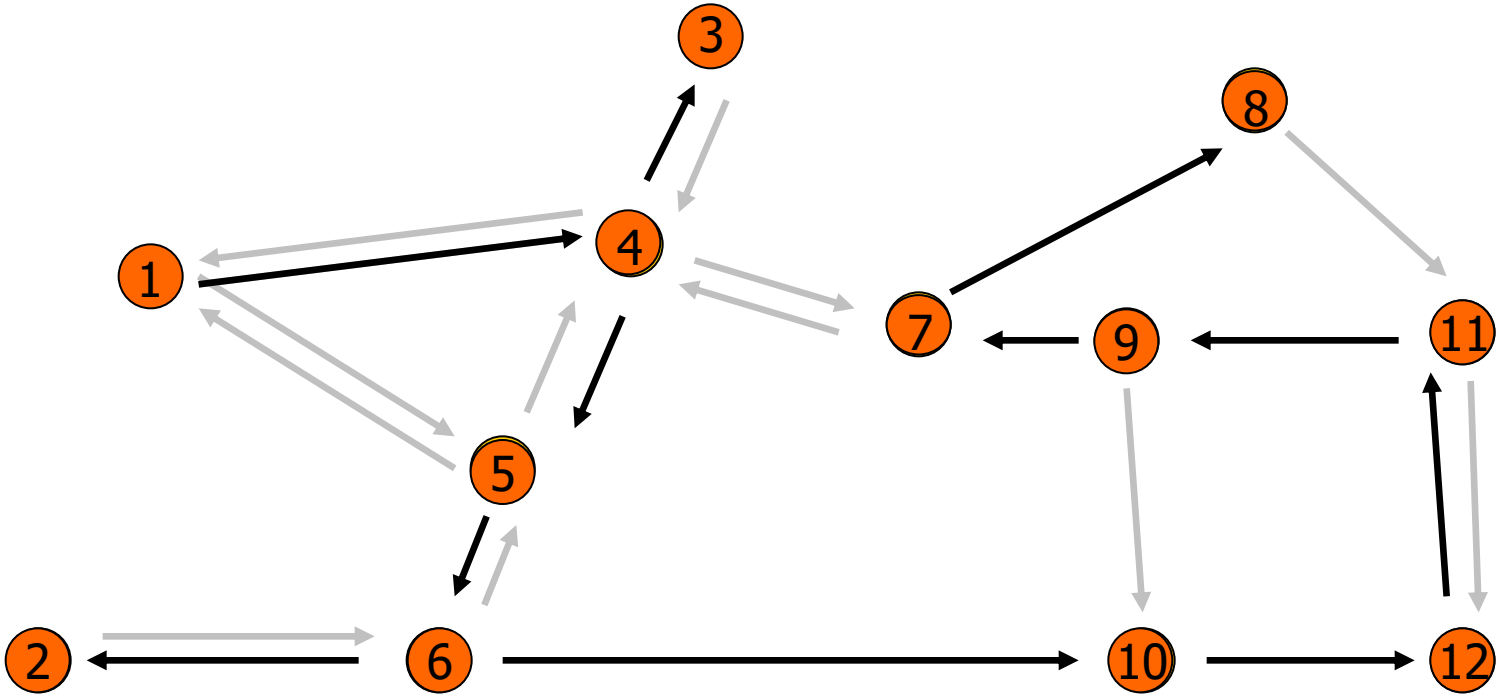
Pile

Exploration en Profondeur



Pile 4 1

Exploration en Profondeur



Pile

1 Définitions et propriétés élémentaires

Soit un 1-graphe orienté et valué $G = (X, U, C)$. $C_{i,j}$ est la valuation de l'arc (i, j) (ou son poids, ou son coût). Le coût (ou la *longueur*) d'un chemin est la somme des coûts des arcs qui consituent le chemin. On note $l(c)$ la longueur d'un chemin c. Les problèmes associés consistent à calculer des chemins de coût minimal (ou chemins minimaux, ou plus courts chemins). On distingue trois classes de problèmes :

1. Etant donné deux sommets $s \in X$ et $t \in X$, trouver un chemin de coût minimal entre s et t.
2. Etant donné un sommet $s \in X$, trouver un chemin de coût minimal de s vers tout autre sommet du graphe.
3. Trouver un plus court chemin entre tout couple de sommets. La matrice $N \times N$ des coûts minimaux est appelée un *distancier*.

Considérons un chemin μ entre $i \in X$ et $j \in X$. Supposons qu'il existe un circuit ω sur ce chemin. Soit μ' le chemin de i à j n'empruntant pas ω.
Propriété 1 si $l(\omega) < 0$) il n'existe pas de plus court chemin de i à j.
Propriété 2 si $l(\omega) \geq 0$ alors $l(\mu') \leq l(\mu)$.
Des propriétés 1 et 2, on déduit qu'on peut restreindre la recherche des plus courts chemins aux *chemins élémentaires* entre i et j.

Un raisonnement similaire peut être appliqué à la recherche de chemins de longueur maximale (ou plus longs chemins).

2 Applications

Les problèmes de chemins optimaux sont rencontrés dans les problèmes de tournées, d'investissement, de gestion des stocks, d'optimisation de réseaux (routiers ou de télécommunications), d'intelligence artificielle et de reconnaissances de formes.

3 Algorithme de Dijkstra (1959)

L'algorithme s'applique aux problèmes de classe 2 lorsque toutes les valeurs des arcs sont positives. Il peut être implémenté en $\mathcal{O}(n^2)$. En sortie, V_i est le plus court chemin de s à i. pC représente l'arborescence de visite.

```
marquer s
V_s = 0, pC_s = s.
V_i = C_si, pC_i = s, ∀i ∈ Γ(s)
V_i = +∞, pC_i = −1, ∀i ∉ Γ(s)∪{s}
pC_s=s
Répéter
  Sélectionner i tel que
  V_i = min{V_j | j ∈ X, j non
  marqué}.
  Si i a été trouvé,
    marquer i
    Pour chaque successeur non
    marqué j de i
      Si V_j > V_i + C_ij alors
        pC_j=i
        V_j = V_i + C_ij
      FinSi
    FinPour
  FinSi
tant qu' on trouve un sommet i
```

4 Algorithme de Bellman (1958)

L'algorithme s'applique aux problèmes de classe 2 lorsque les valeurs des arcs sont quelconques. Il détecte également l'existence d'un circuit négatif. Sa complexité est $\mathcal{O}(MN)$.

$\mathtt{pC}_s{=}s$
$V_s^0 = 0.$
$V_i^0 = +\infty$, $\mathtt{pC}_i{=}-1$, $\forall i \in X - \{s\}$
$k = 0$
`Répéter`
  $k = k + 1$; $V_i^k = V_i^{k-1}$, $\forall i \in X$
  `Pour` $i \in X$ `faire`
    `Pour` $j \in \Gamma^{-1}(i)$ `faire`
      `Si` $V_i^k > V_j^{k-1} + C_{ji}$ `alors`
        $\mathtt{pC}_i{=}j$
        $V_i^k = V_j^{k-1} + C_{ji}$
      `FinSi`
    `FinPour`
  `FinPour`
`jusqu'à` $V_i^k = V_i^{k-1}$, $\forall i \in X$ `ou` $k = N$

Le cas où $k = N$ détecte un circuit de longueur négative. Sinon V_i^k est la longueur du plus court chemin de s à i et $\mathtt{pC}$ représente l'arborescence de visite.

Cas des graphes sans circuit

On décompose le graphe en niveaux en associant un *rang* à chaque sommet. Le rang d'un sommet est la longueur du plus long chemin en *nombre d'arcs* depuis le sommet de départ. En parcourant les sommets dans l'ordre des rangs croissants, l'algorithme de Bellman converge en une itération, donc $\mathcal{O}(M)$.

Application à l'ordonnancement cf cours

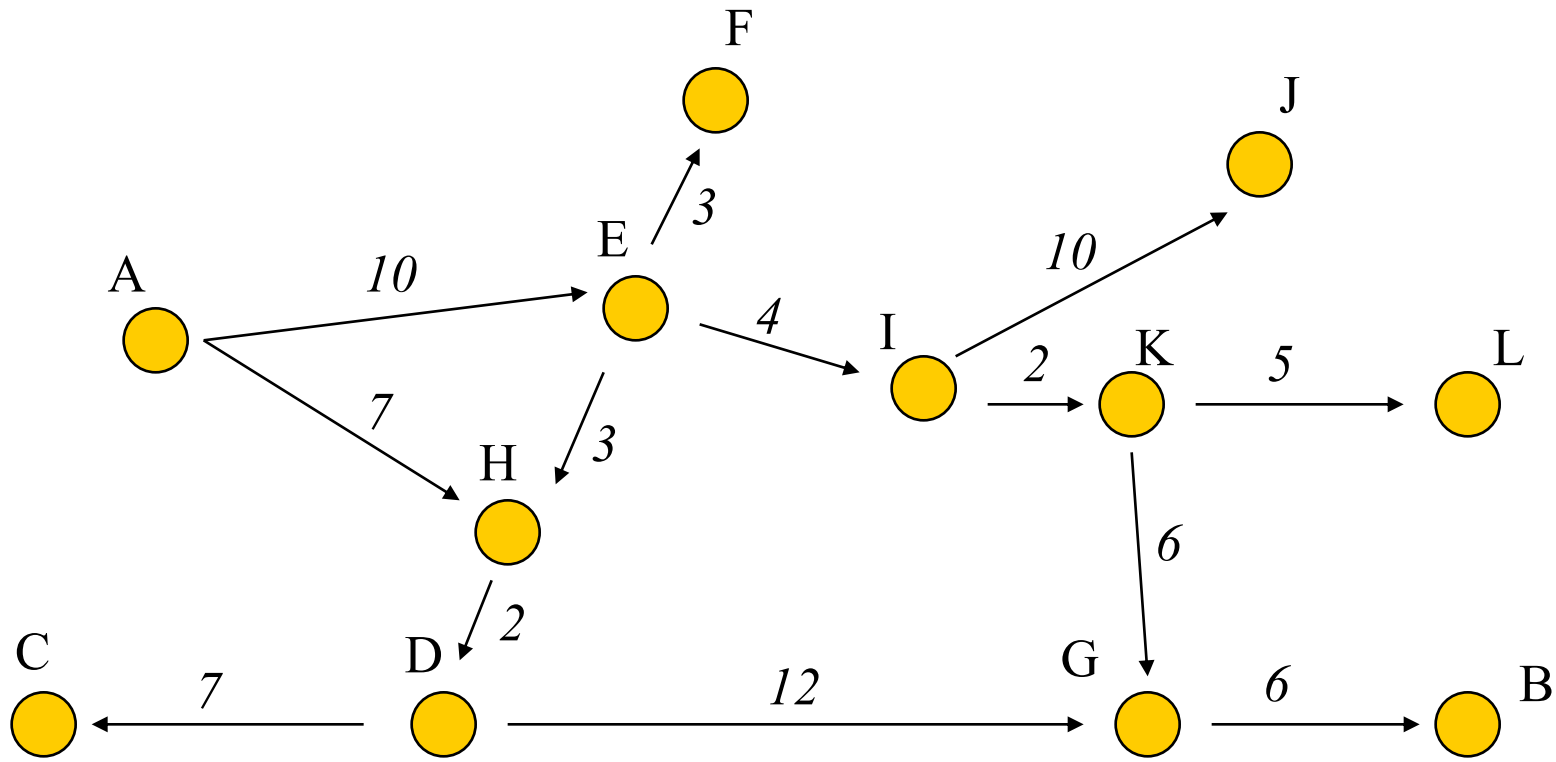
5 Algorithme de Floyd (1962)

L'algorithme s'applique aux problèmes de classe 3 (distancier) lorsque les valeurs des arcs sont quelconques. Il détecte également l'existence d'un circuit négatif. Sa complexité est $\mathcal{O}(N^3)$ en utilisant la représentation matricielle. Les distances sont stockées dans une matrice V et les chemins sont stockés dans une matrice $\mathtt{pC}$ ou $\mathtt{pC}_{ij}$ est le précedesseur du sommet j dans le plus court chemin de i à j. Un circuit de longueur négative est détecté dès qu'on obtient un $V_{ii} < 0$.

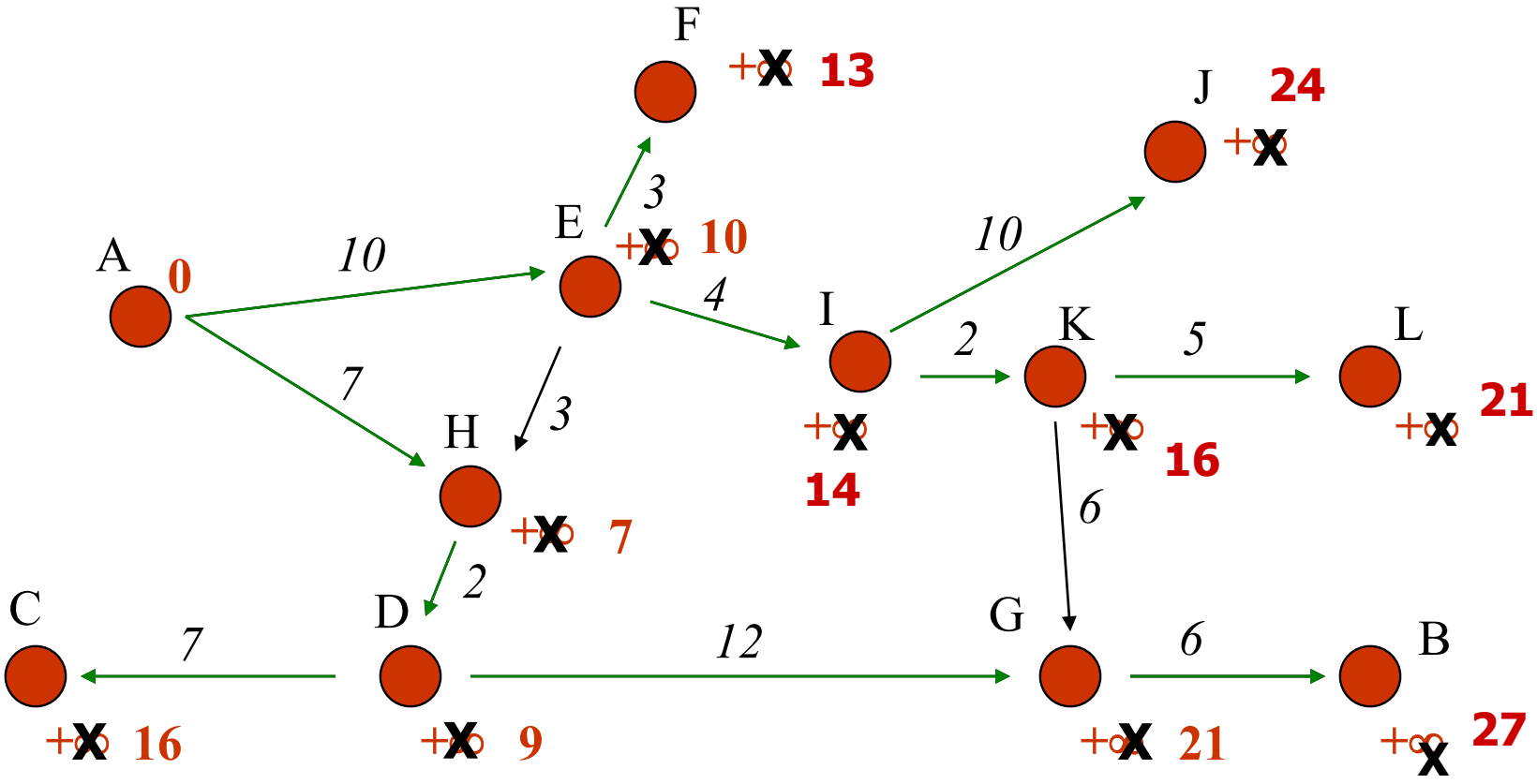
$V_{ij} = C_{ij}$, $\mathtt{Pc}_{ij} = i$, $\forall (i,j) \in U$
$V_{ii} = 0$, $\mathtt{Pc}_{ii} = i$, $\forall i \in X$
$V_{ij} = +\infty$, $\mathtt{Pc}_{ii} = -1$, $\forall (i,j) \notin U, i \neq j$
`Pour` $k = 1, \ldots, N$ `Faire`
  `Pour` $i = 1, \ldots, N$ `Faire`
    `Pour` $j = 1, \ldots, N$ `Faire`
      `Si` $V_{ik} + V_{kj} < V_{ij}$ `alors`
        $V_{ij} = V_{ik} + V_{kj}$
        $\mathtt{pC}_{ij}{=}\mathtt{pC}_{kj}$
      `FinSi`
    `FinPour`
  `FinPour`
`FinPour`

Problème de Plus Court chemin

- De A à tous les autres

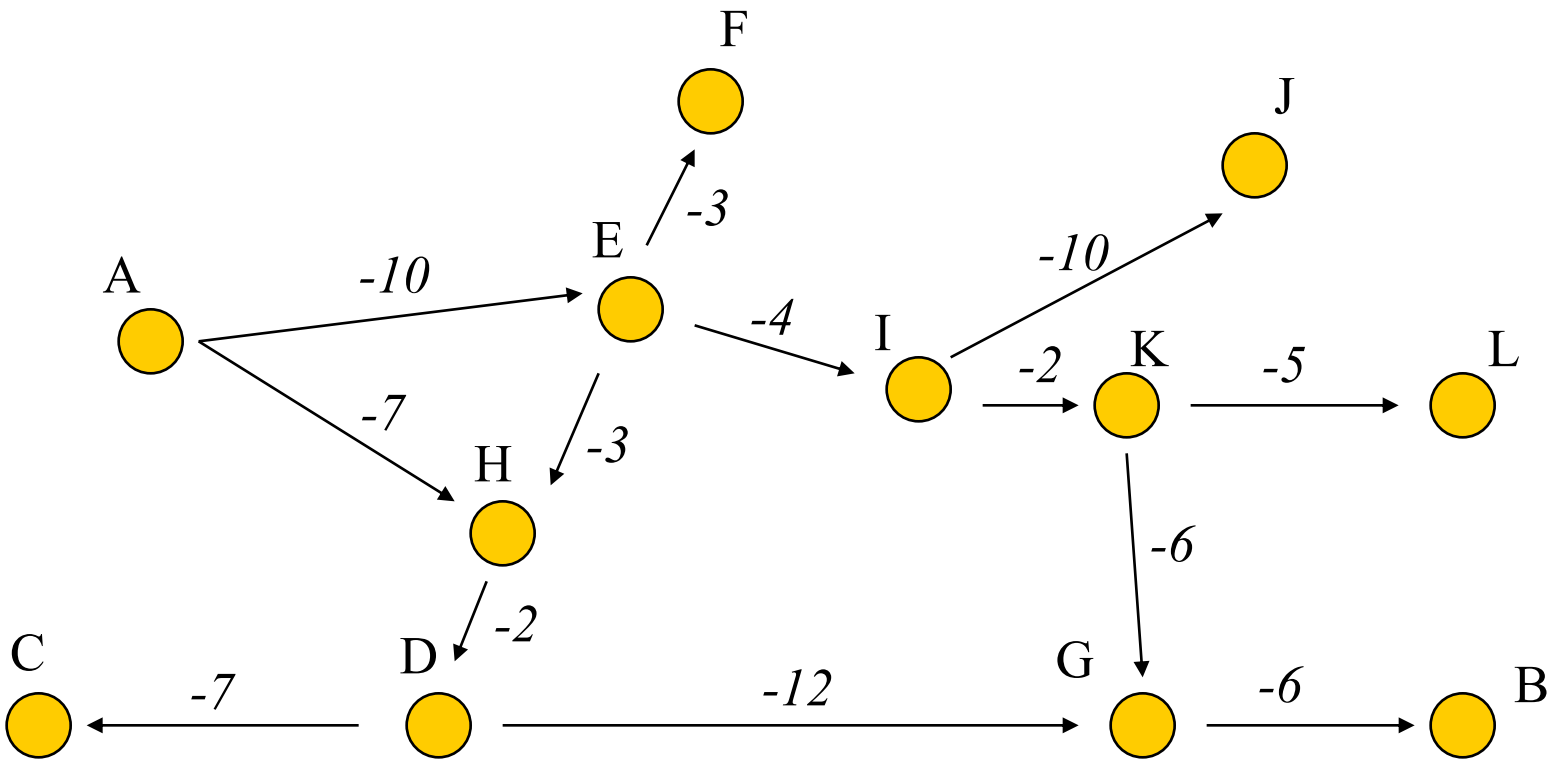


Execution de l'algorithme de Dijkstra



Arcs à valeurs négatives

Appliquer l'algorithme de Dijkstra à ce graphe

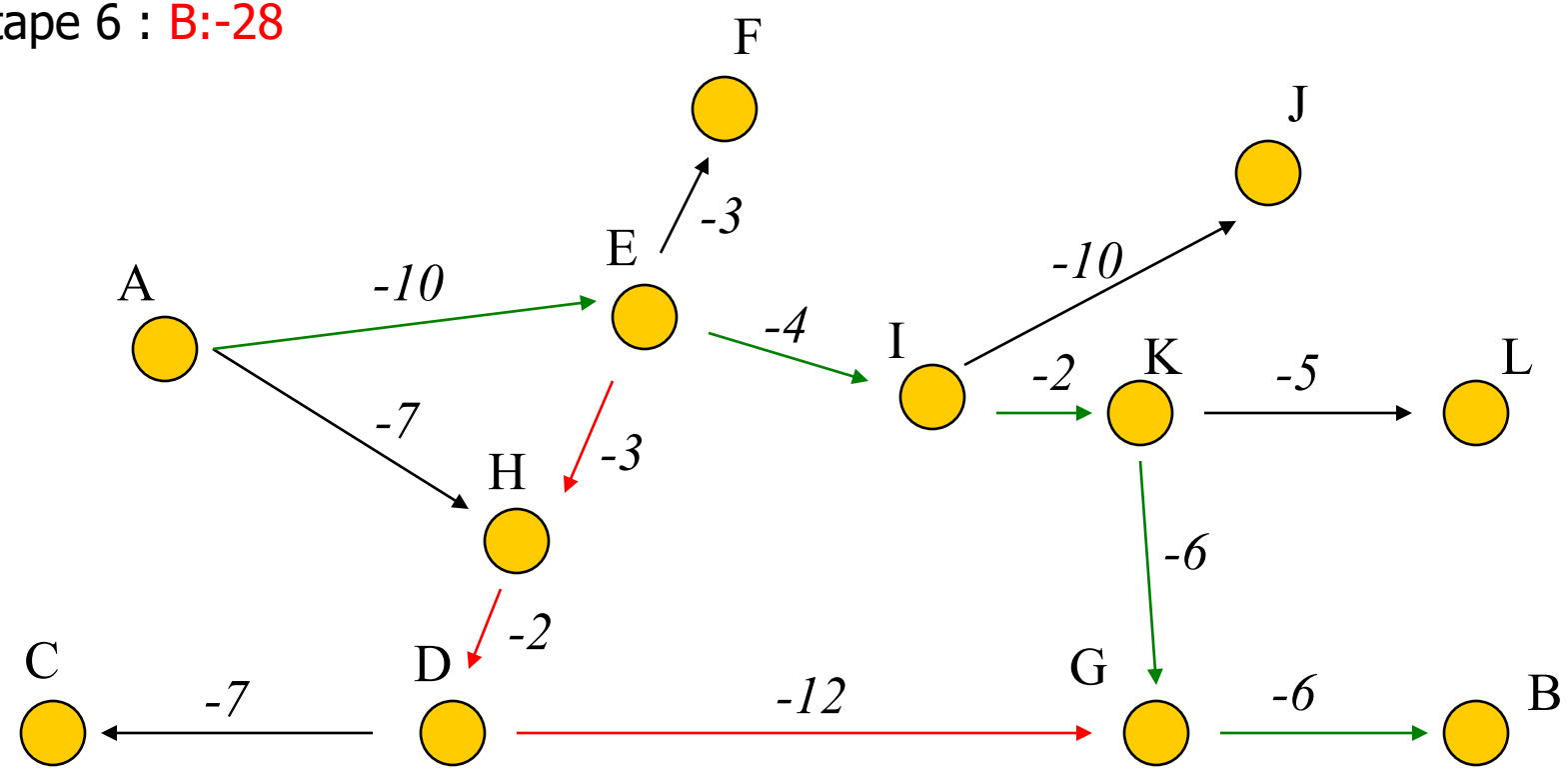


Contrexemple

Etape 1 : A:0
Etape 2 : H: –7, E:-10
Etape 3 : F:-13, I:-14
Etape 4 : J:-24, K:-16
Etape 5 : L:-21, G:-22
Etape 6 : B:-28

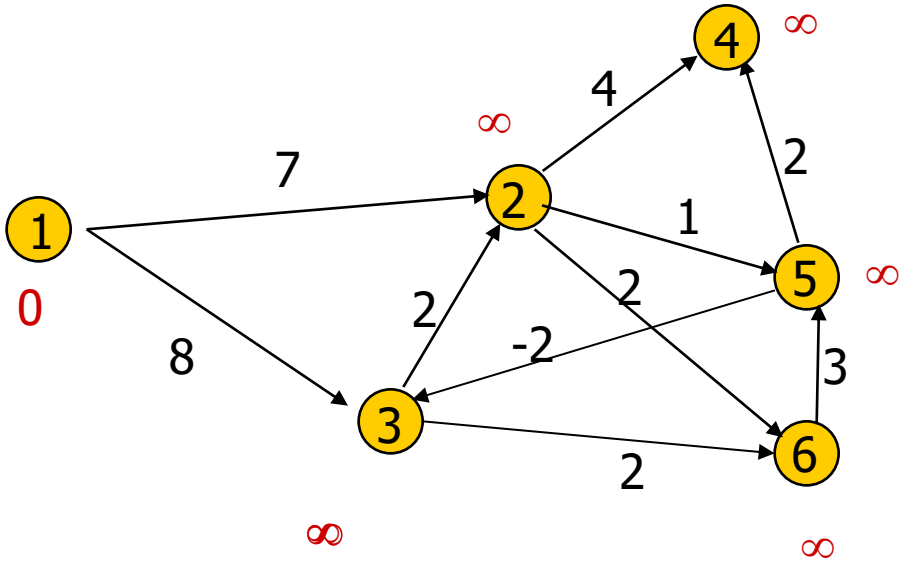
Dijkstra trouve A,E,I,K,G,N de longueur 28

Le chemin A,E,H,D,G,B a une longueur de 33 !!



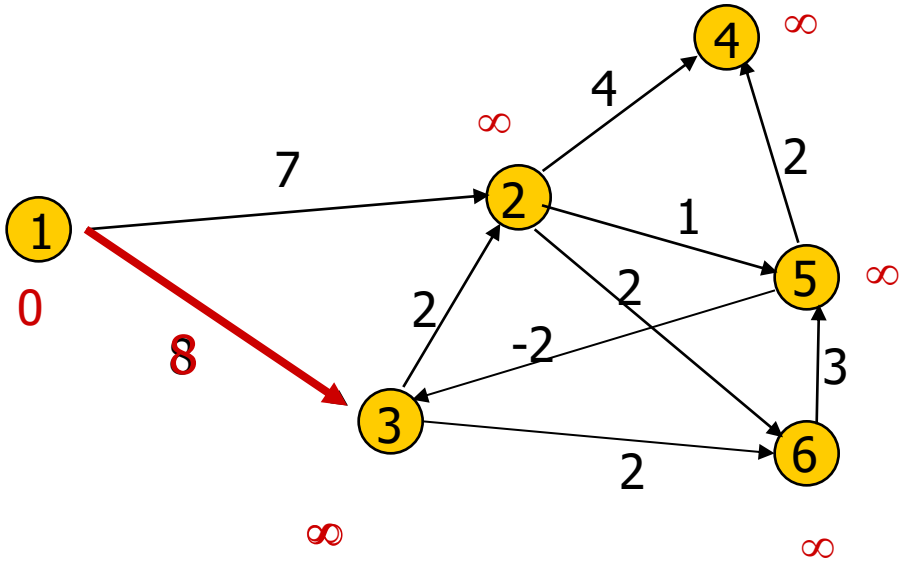
Méthode des potentiels (Berge 1958)

$V_s=0$
$V_i=+\infty$, $\forall i \neq s$
Tant qu'il existe un arc (i,j) tel que $V_i+C_{ij}<V_j$ Alors
$predChemin_j = i$
$V_j = V_i+C_i$
Fin Tant Que



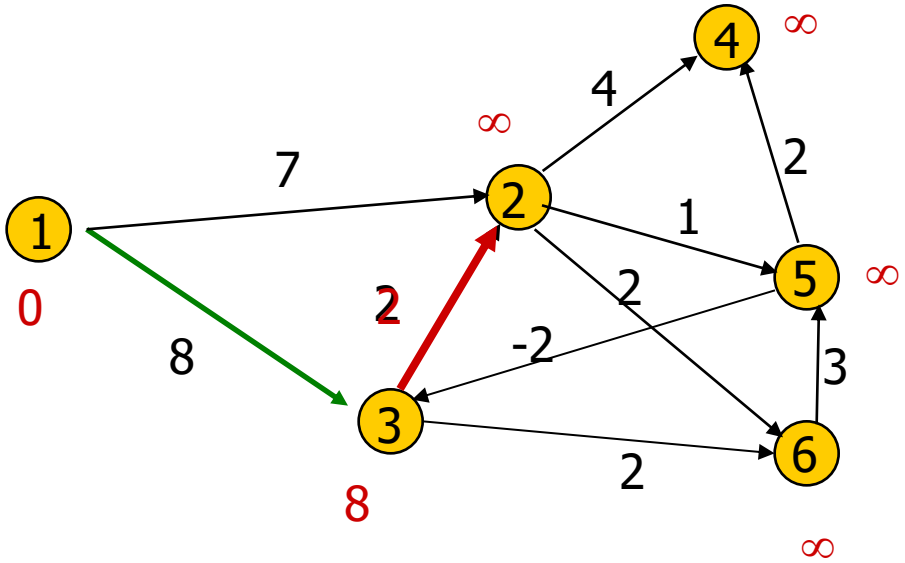
Méthode des potentiels (Berge 1958)

$V_s=0$
$V_i=+\infty$, $\forall i \neq s$
Tant qu'il existe un arc (i,j) tel que $V_i+C_{ij}<V_j$ Alors
$predChemin_j = i$
$V_j = V_i+C_i$
Fin Tant Que

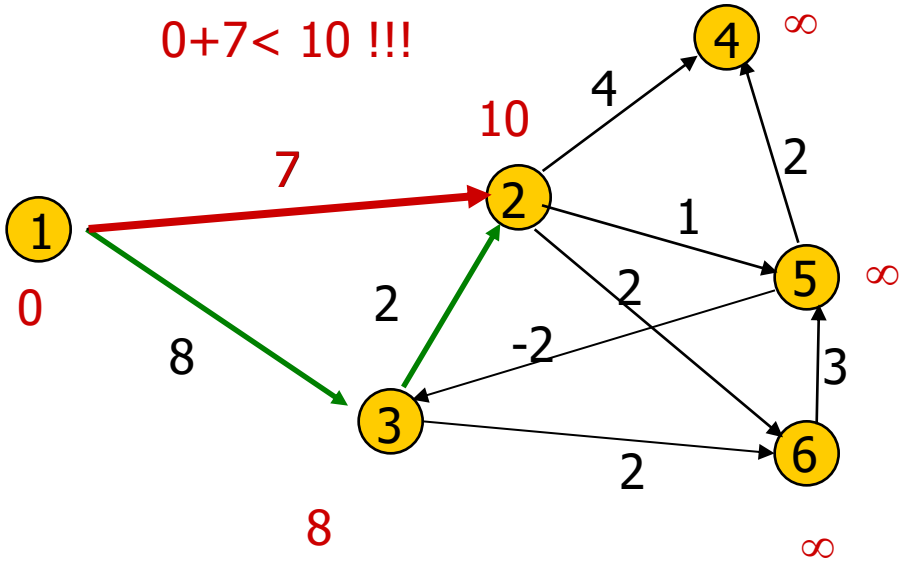


0+8<∞ !!!

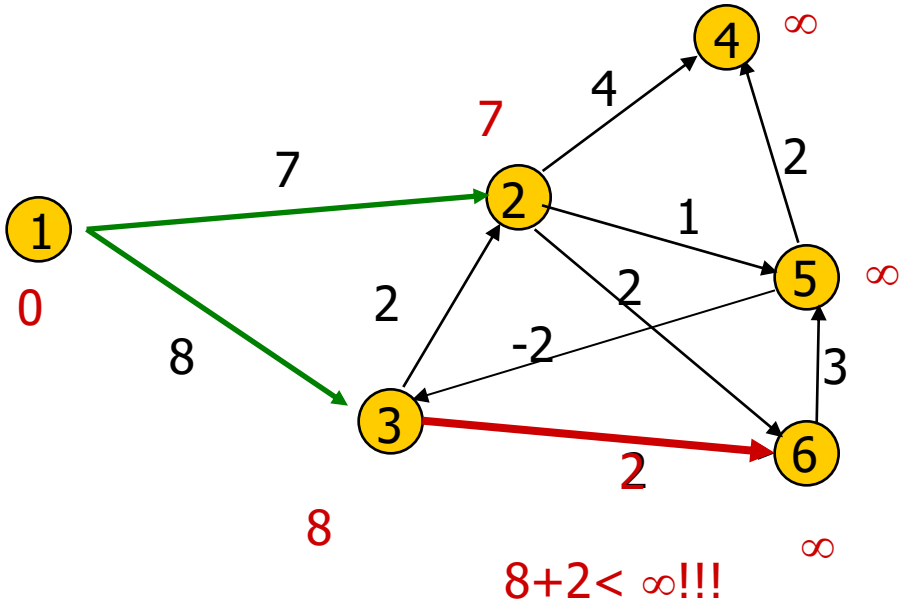
Méthode des potentiels



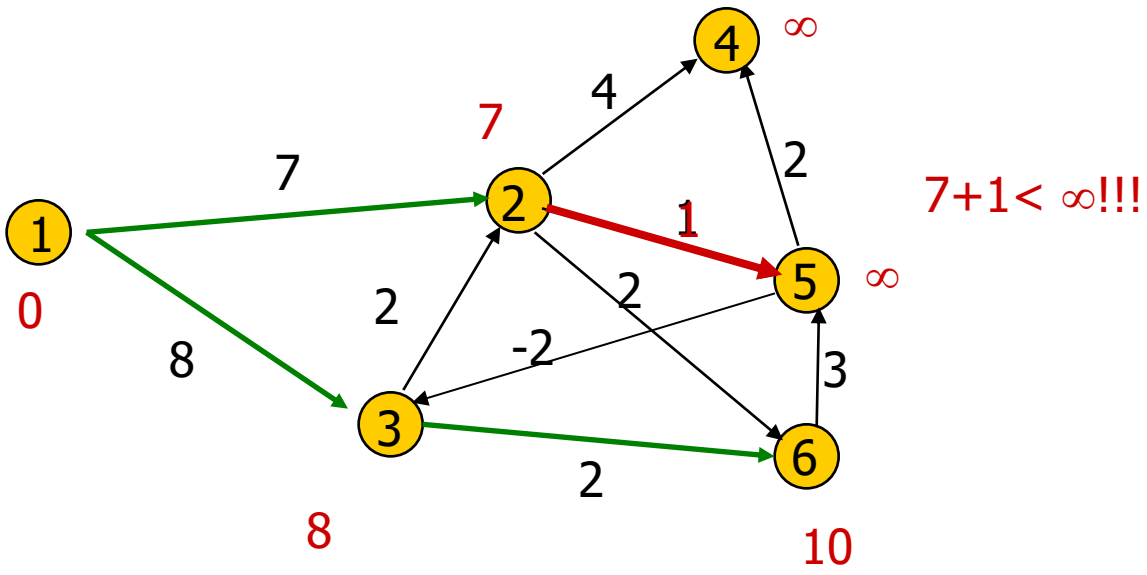
Méthode des potentiels



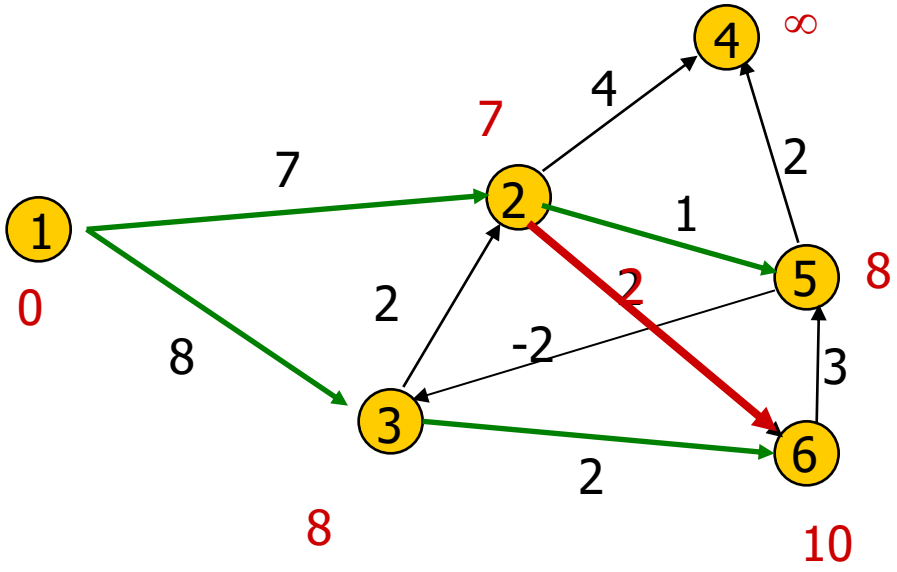
Méthode des potentiels



Méthode des potentiels

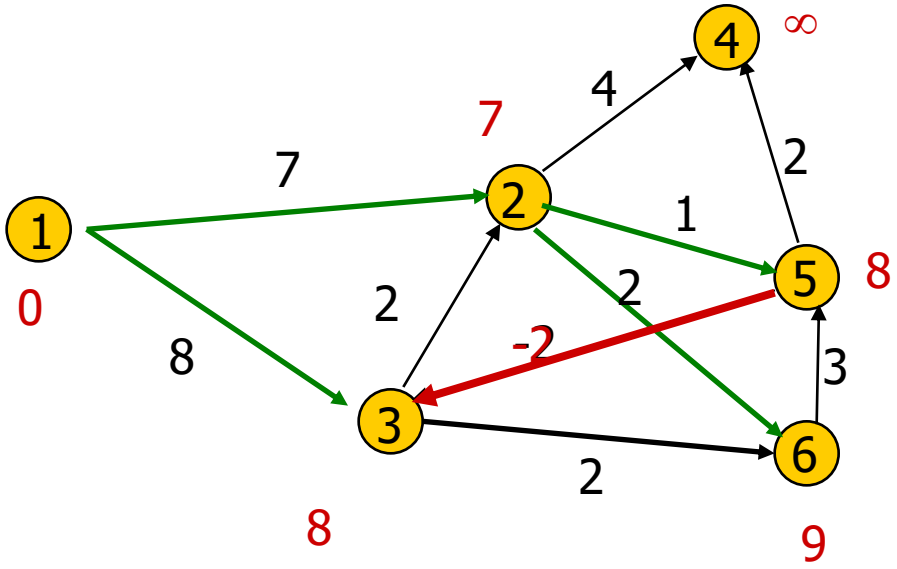


Méthode des potentiels



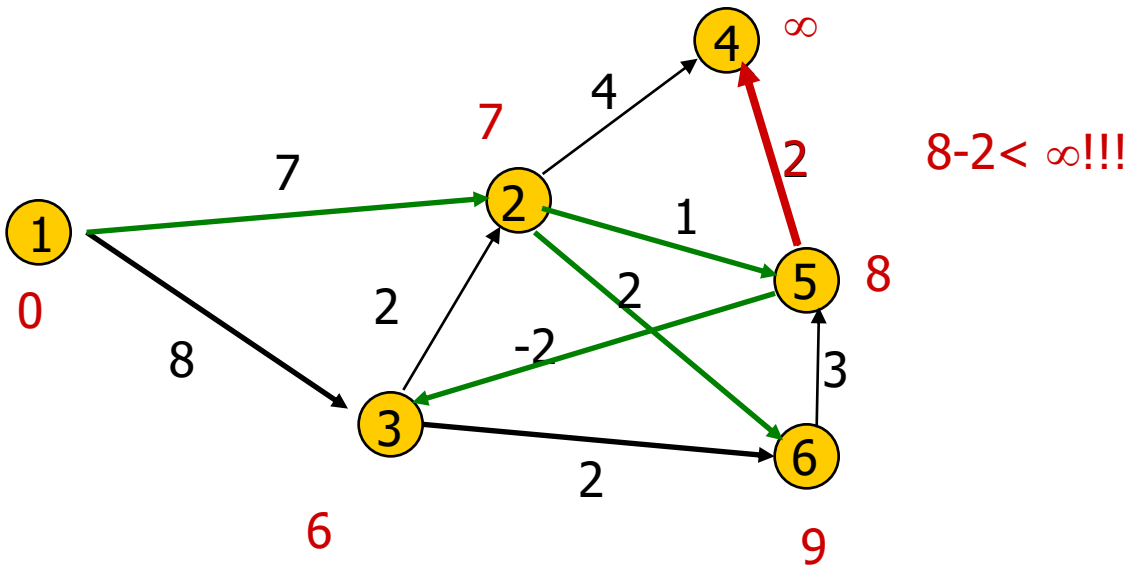
7+2< 10!!!

Méthode des potentiels

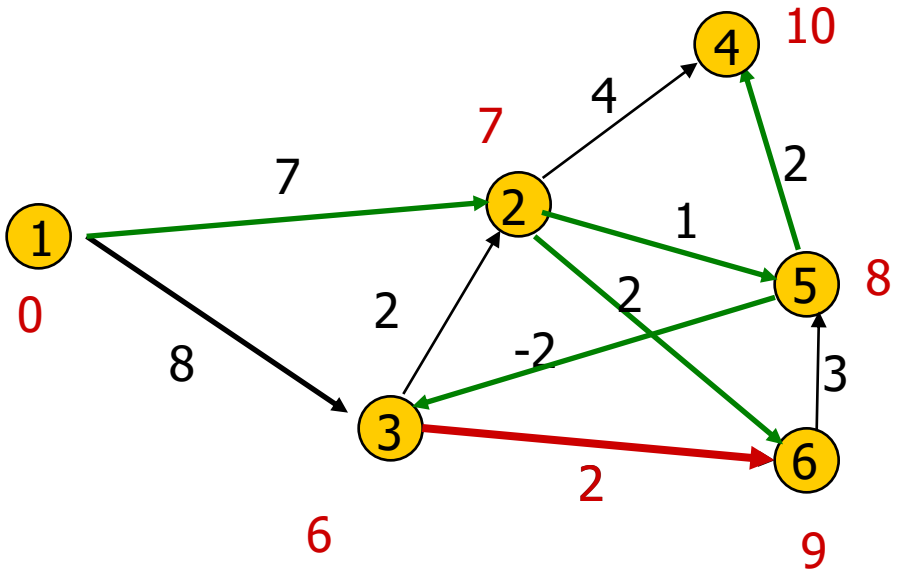


8-2< 8!!!

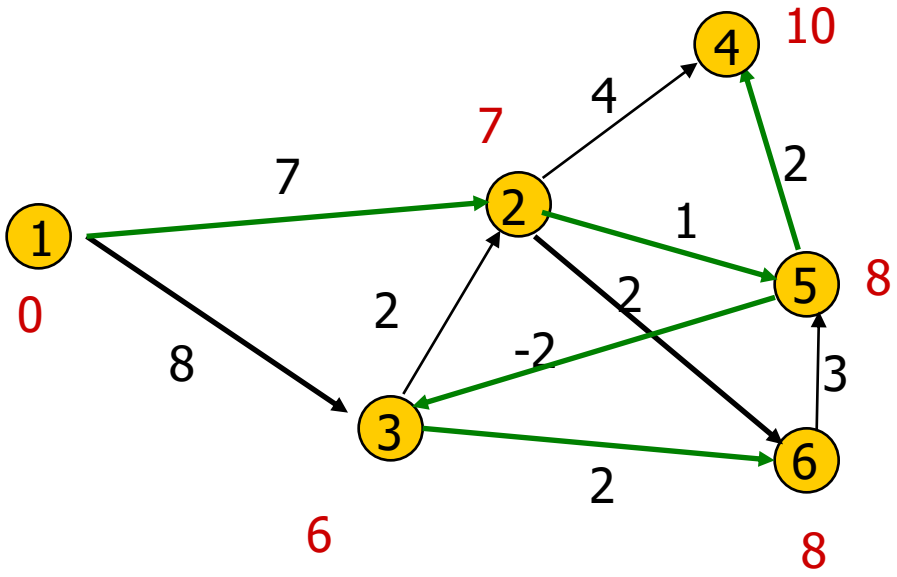
Méthode des potentiels



Méthode des potentiels

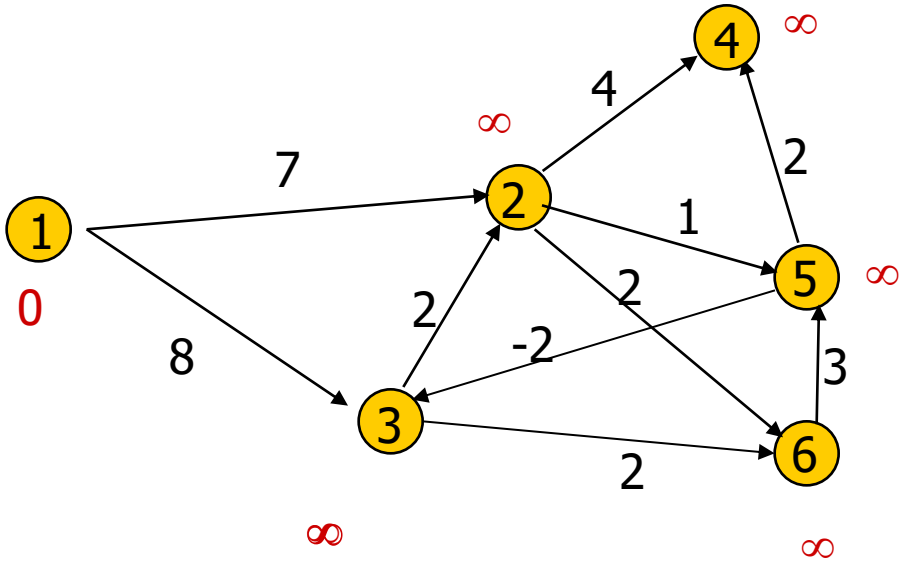


Méthode des potentiels



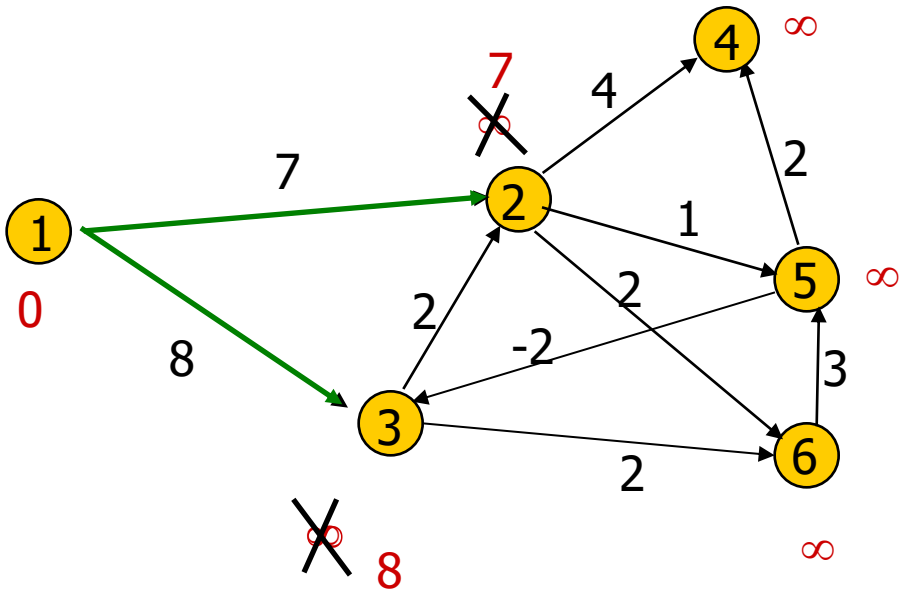
Algorithme de Bellman

k= 0



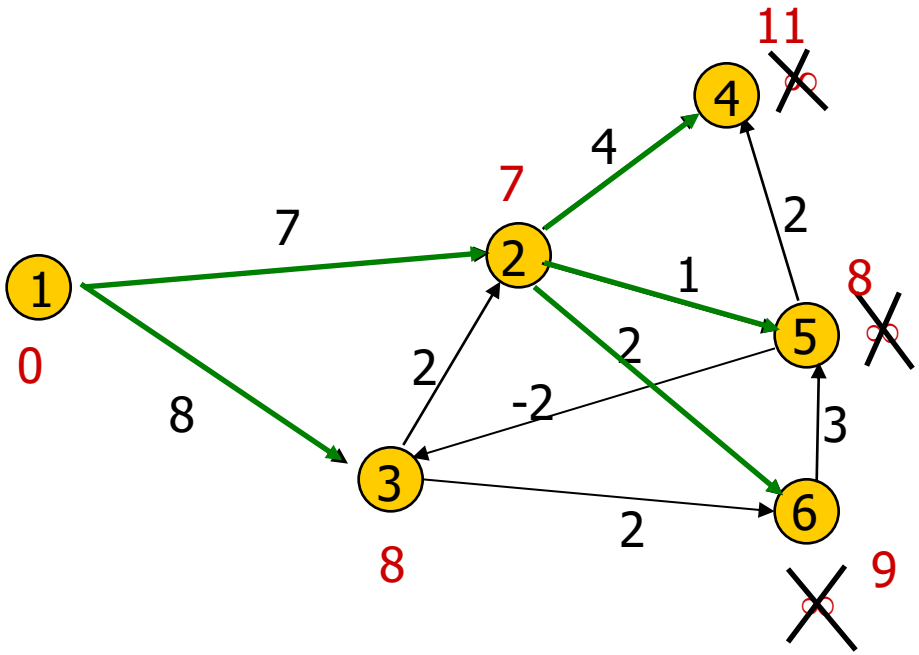
Algorithme de Bellman

k= 1



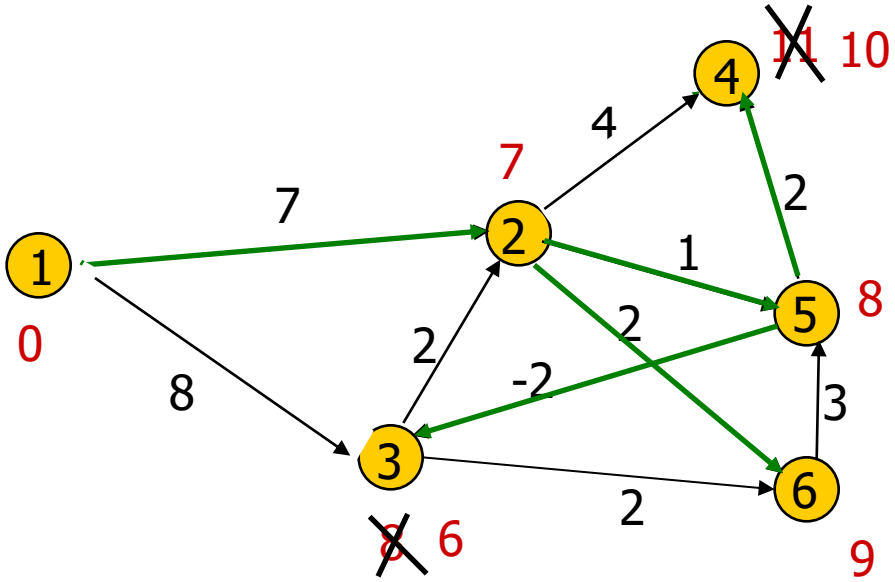
Algorithme de Bellman

k= 2



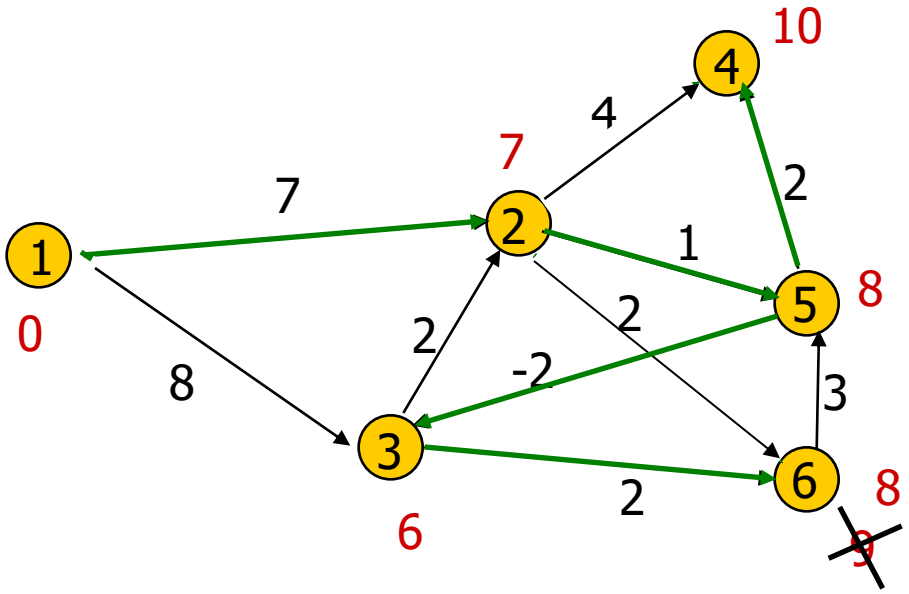
Algorithme de Bellman

k= 3



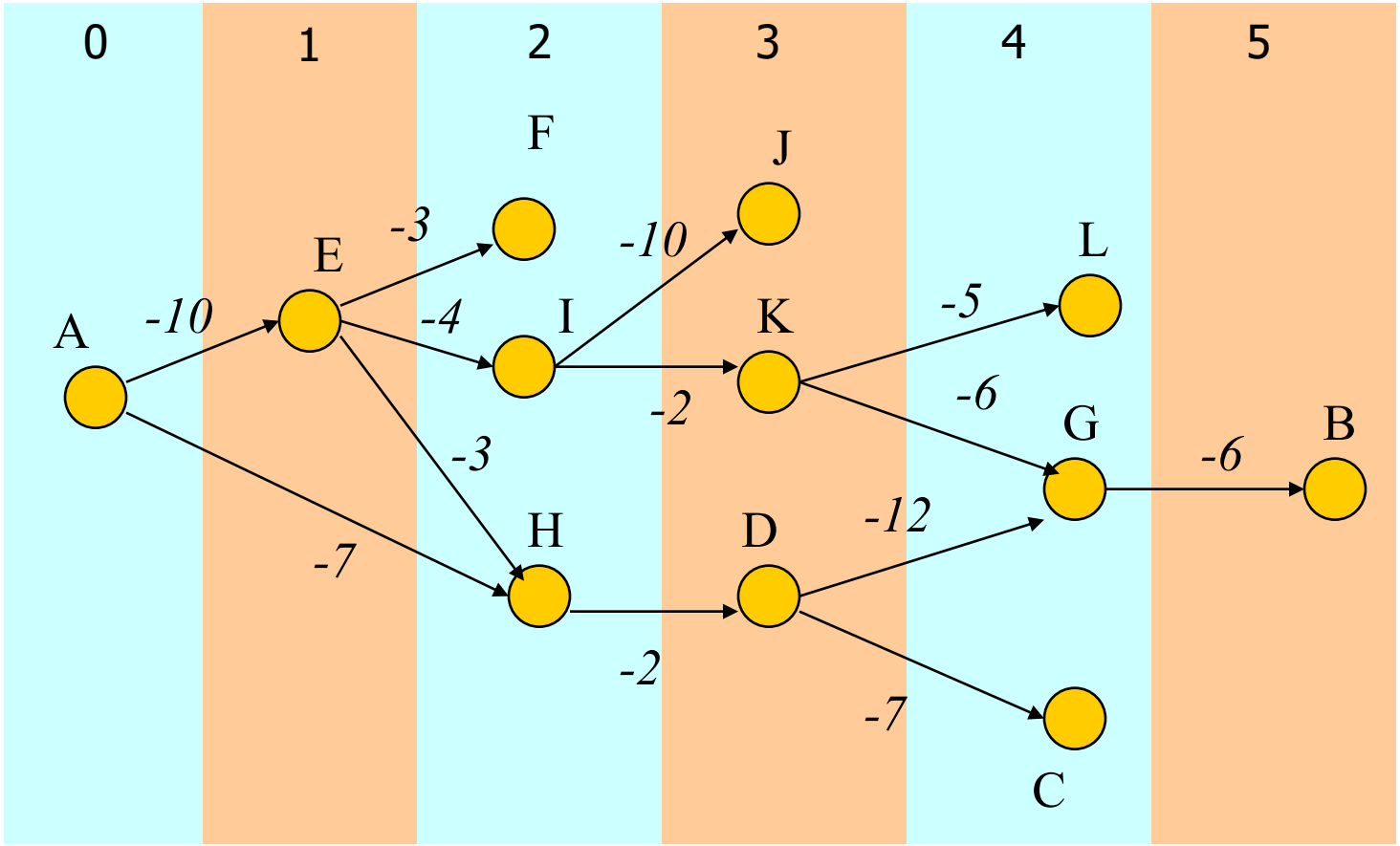
Algorithme de Bellman

k= 4



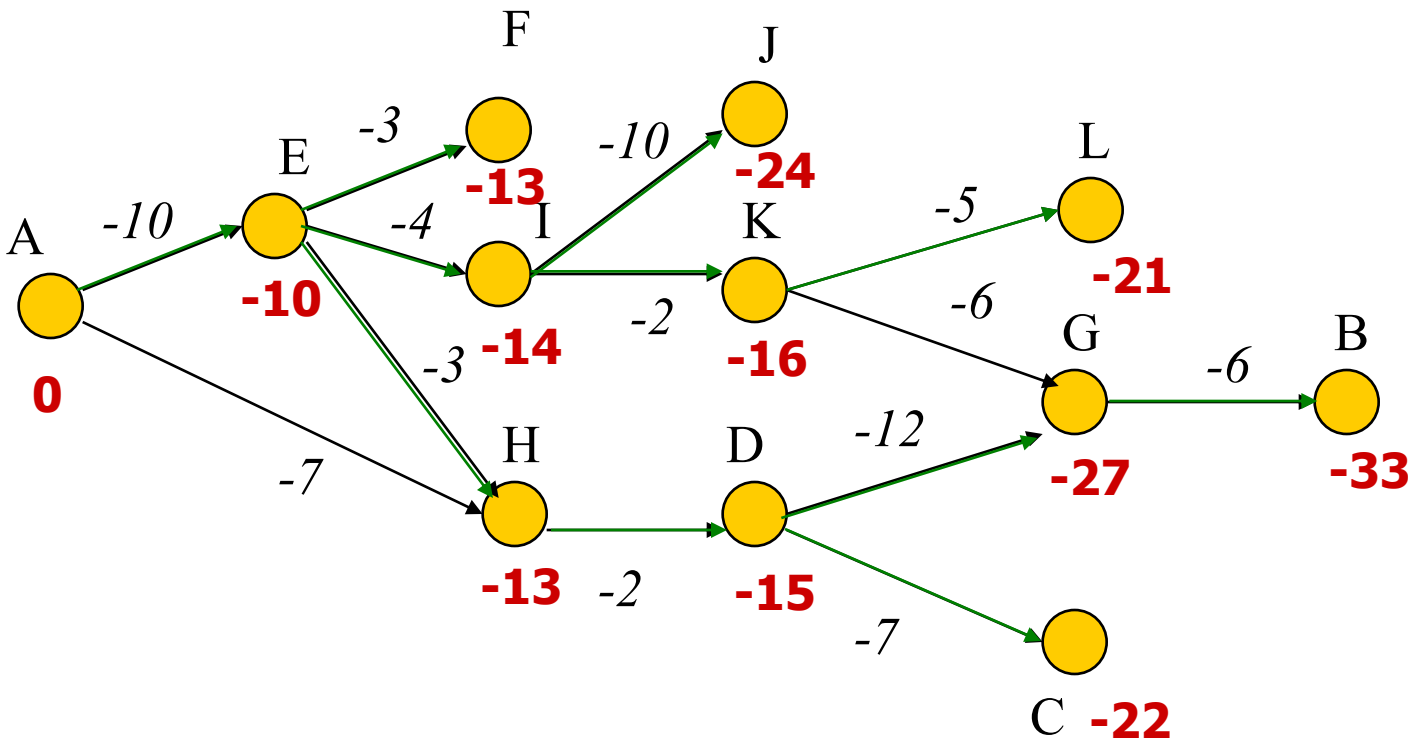
Cas des graphes sans circuit

a) Décomposer le graphe en niveaux



Cas des graphes sans circuits

Ordre de parcours des sommets A, E, F, I, H, J, K , D, L, G, C, B

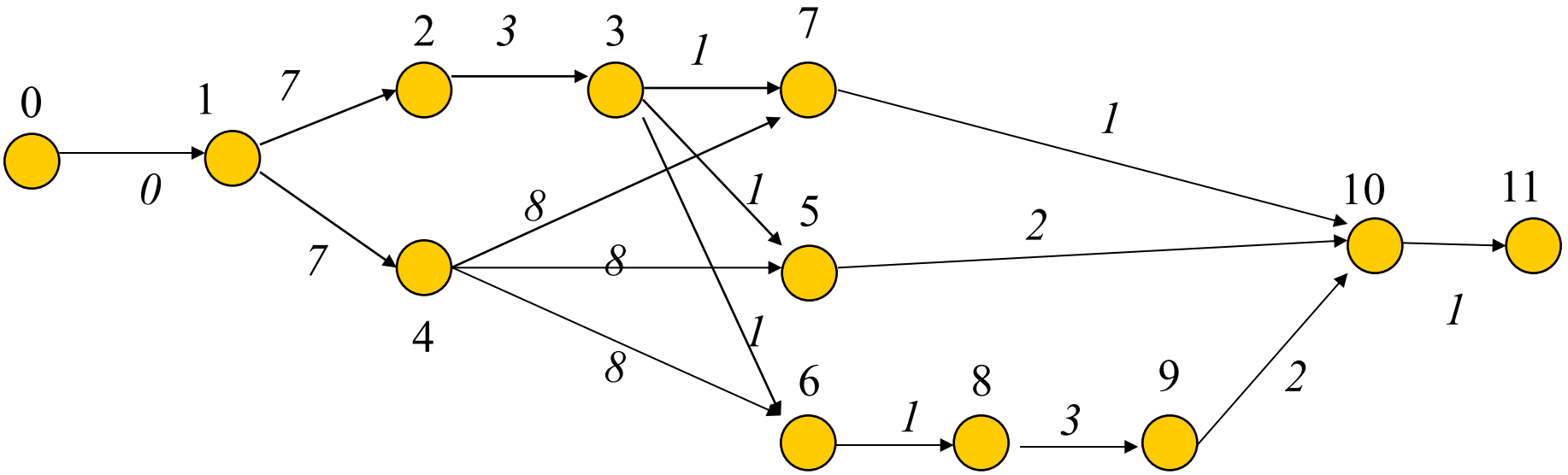


Problèmes d'ordonnancements

Task		Duration	predecessors
1	Murs	7	-
2	Charpente	3	1
3	Toiture	1	2
4	Plomberie	8	1
5	Electricité	2	3,4
6	Fenêtres	1	3,4
7	Jardins	1	3,4
8	Peinture	3	6
9	Cuisine	2	8
10	installation	1	5,7,9

Modelling as a longest path problem

Minimal start time of a task i = longest path from 0 to i in the following graph



Exercice : solve the scheduling problem

1 Définitions

Soit un 1-graphe orienté et doublement valué $G = (X, U, C, W)$. C_{ij} est une valuation de l'arc (i, j) appelée *capacité*. W_{ij} est une valuation de l'arc qui définit le *coût* de passage d'une unité de flot sur l'arc (i, j). On considère deux sommets particuliers s et t appelés respectivement la *source* et le *puit*. Ce graphe est appelé un *réseau de transport*. On appelle un flot une application Φ de U dans IN vérifiant :

- $0 \leq \Phi_{ij} \leq C_{ij}, \forall (i,j) \in U$ (repect des capacités),
- $\sum_{j \in \Gamma(i)} \Phi_{ij} = \sum_{j \in \Gamma^{-1}(i)} \Phi_{ji}, \forall i \in X, i \neq s, t$ (conservation des flots),
- $\sum_{j \in \Gamma(s)} \Phi_{sj} = \sum_{j \in \Gamma^{-1}(t)} \Phi_{jt} = F$ où F est le débit du flot.

Le débit du flot est égal au flot total sortant de la source s et aussi au flot total entrant par le puit. On considère les problèmes

- du **flot maximal** qui consiste à trouver un flot Φ qui maximise F.
- du **flot à coût minimal** qui consite à trouver un flot Φ d'un débit donné $\tilde{F}$ et de coût total W minimal où

$$W = \sum_{(i,j) \in A} W_{ij} \Phi_{ij}$$

2 Restrictions et extensions

- Cas de plusieurs sources ou de plusieurs puits : cf cours.
- Cas de disponibilité limitée de la source
- cas de demande limitée du puit : cf cours.
- Cas de capacités sur les nœuds : cf cours.

3 Coupe, Coupe minimale

Définition Une coupe est un couple d'ensembles de sommets (S, T) tels que $s \in S$, $t \in T$, $S \cap T = \emptyset$ et $S \cup T = X$.

On appelle *arcs sortants* les arcs orientés de S vers T et *arcs entrants* les arcs orientés de T vers S. On appelle capacité de la coupe $C(S, T)$ la somme des capacités des arcs sortants : $C(S,T) = \sum_{(i,j) \in A, i \in S, j \in T} C_{ij}$.

Théorème 1 Le débit maximal d'un flot de s à t est égal à la capacité minimale des coupes séparant s et t.

4 Graphe d'écart

Soit un flot Φ. On définit le graphe d'écart $G_e(\Phi)$ à partir de G contenant les sommets de X et les arcs suivants :

- on crée dans $G_e(\Phi)$ un arc *avant* (i,j) de *capacité résiduelle* $C_{ij} - \Phi_{ij}$ et de coût W_{ij} pour tout arc (i,j) de G tel que $\Phi_{ij} < C_{ij}$ (arc non saturé).
- on crée dans $G_e(\Phi)$ un arc *arrière* (i,j) de *capacité résiduelle* Φ_{ij} et de coût $-W_{ij}$ pour tout arc (j,i) de G tel que $\Phi_{ij} > 0$.

Les algorithmes qui résolvent les problèmes de flot maximal et de flot à coût minimum sont basés sur les théorèmes suivants :

Théorème 2 Un flot Φ de s vers t est maximal si et seulement si il n'existe pas de chemin de s vers t dans le graphe d'écart $G_e(\Phi)$.

Théorème 3 Un flot Φ de s à t et de débit F est de coût minimal parmi tous les flots de débit F si et seulement s'il n'existe pas de circuit de coût négatif dans le graphe d'écart $G_e(\Phi)$.

5 Algorithme de Ford et Fulkerson (1956)

Cet algorithme résout le problème du flot maximal. Sa complexité (poour la version basique) st en $\mathcal{O}(MNU)$ où U le maximum des capacités des arcs.

$F = 0$. $\Phi_{ij} = 0$, $\forall (i,j) \in A$. Construire $G_e(\Phi)$.

`Répéter`

a) Chercher un chemin μ de s vers t dans le graphe d'écart $G_e(\Phi)$.

b) Calculer la capacité résiduelle c du chemin μ : le minimum des capacités résiduelles de ses arcs.

c) $\forall (i,j)$ arc avant de μ, $\Phi_{ij} = \Phi_{ij} + c$.

d) $\forall (i,j)$ arc arrière deμ, $\Phi_{ij} = \Phi_{ij} - c$.

e) Mettre à jour $G_e(\Phi)$. $F = F + c$.

`tant qu'`un tel chemin existe.

La coupe de capacité minimale (S,T) est telle que S est l'ensemble des sommets marqués par la dernière exploration.

6 Algorithme de Busacker et Gowen (1961)

Cet algorithme résout le problème de calcul d'un flot de débit $\tilde{F}$ et de coût minimal. Sa complexité est en $\mathcal{O}(MN^2U)$. L'algorithmle est semblable à l'algorithme précédent avec les modifications suivantes. On initialise le coût total K à 0. On remplace l'étape a) par :

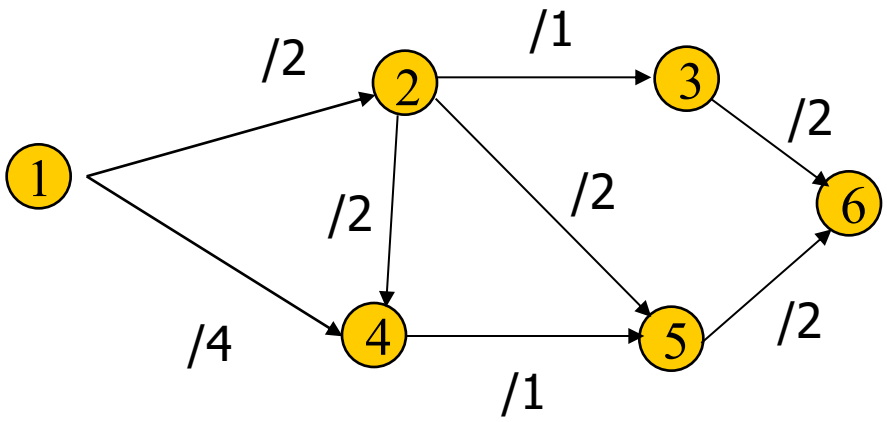
a) Chercher un chemin μ de coût minimal z de s vers t dans le graphe d'écart $G_e(\Phi)$.

A l'étape e), on augmente le coût total K de zc. La condition d'arrêt devient :

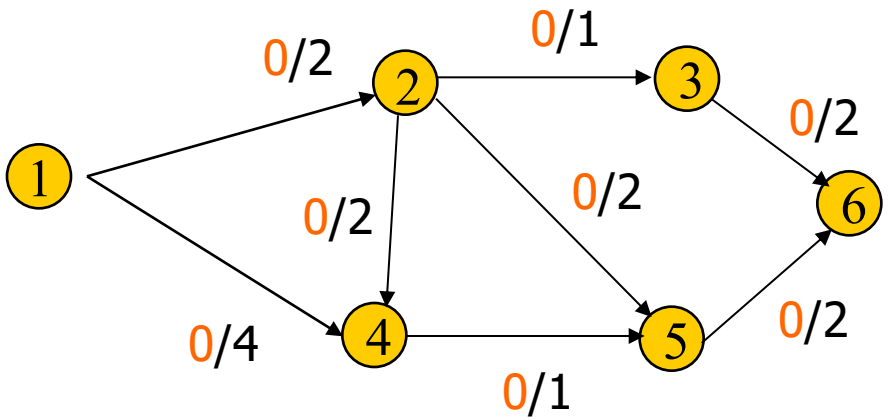
`tant qu'`un tel chemin existe ou $F = \tilde{F}$.

Algorithme de Ford et Fulkerson

Appliquer l'algorithme de Ford et Fulkerson à ce graphe



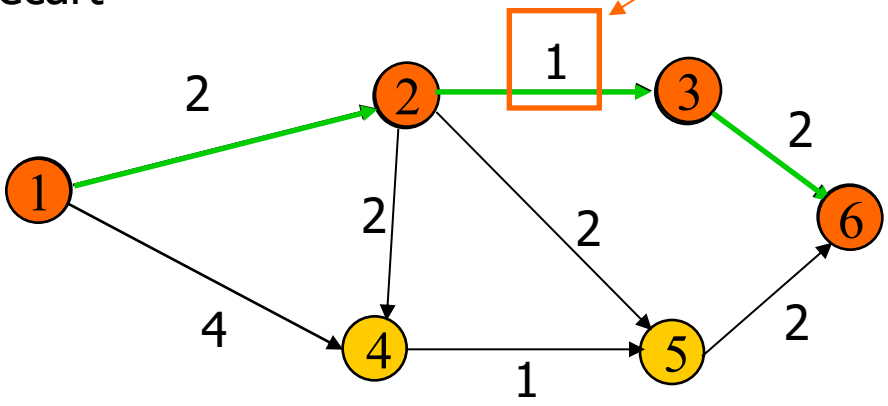
Algorithme de Ford et Fulkerson



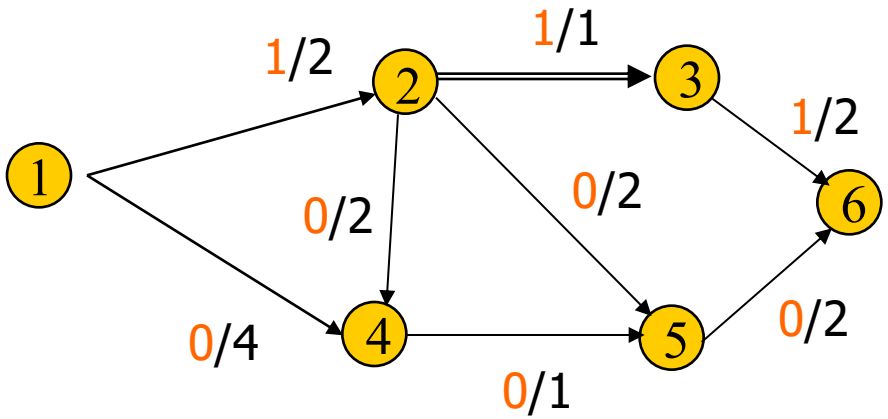
F = 0

Graphe d'écart

Capacité résiduelle minimale



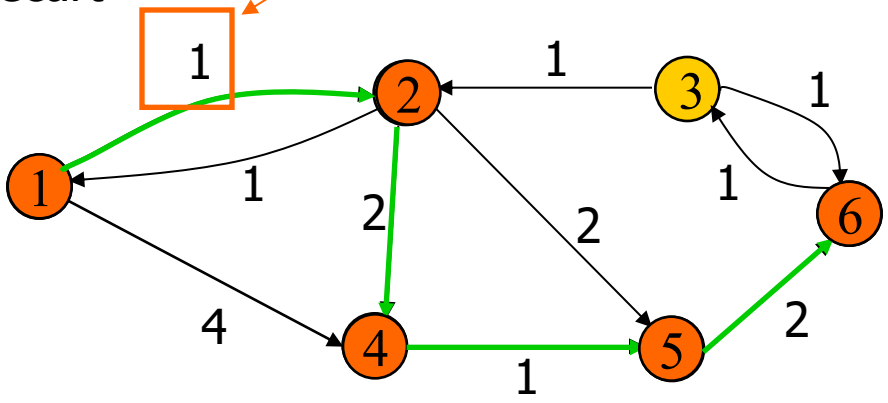
Algorithme de Ford et Fulkerson



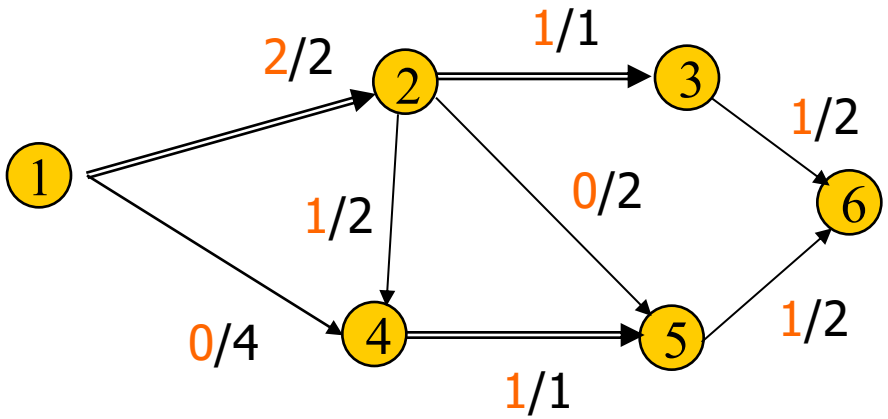
F = 1

Graphe d'écart

Capacité résiduelle minimale

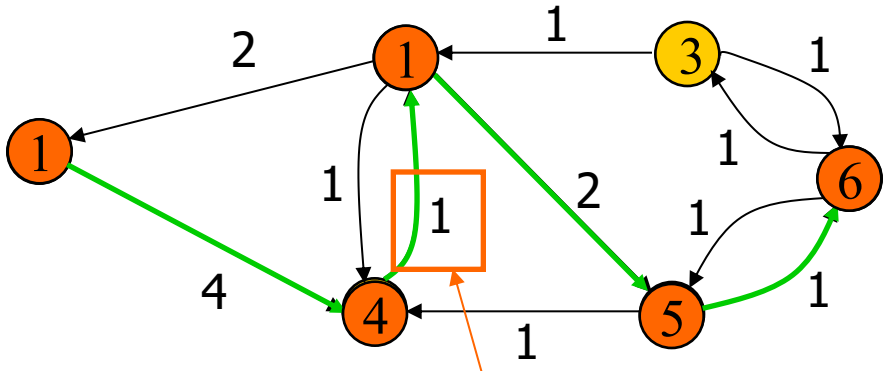


Algorithme de Ford et Fulkerson



F = 2

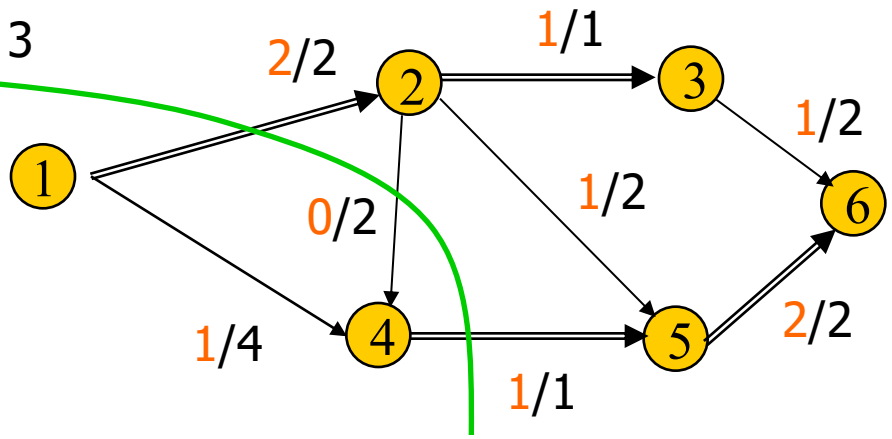
Graphe d'écart



Capacité résiduelle minimale

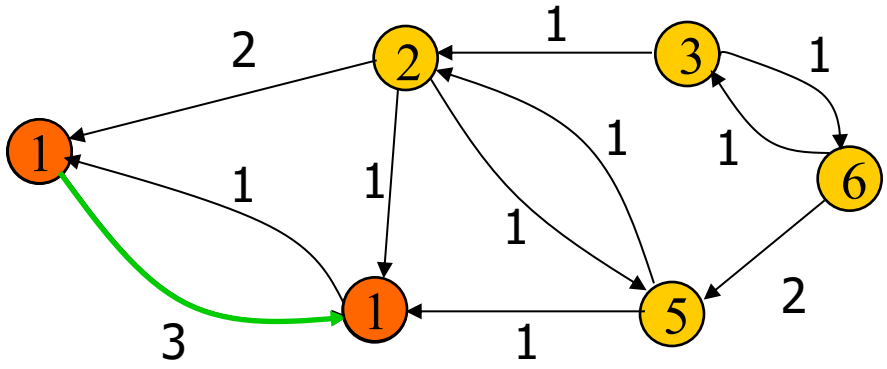
Algorithme de Ford et Fulkerson

Coupe min
Capacité = 3



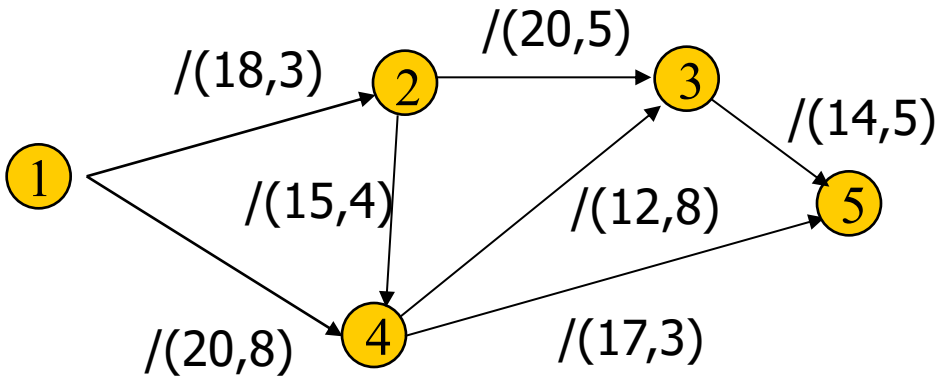
F = 3

Graphe d'écart

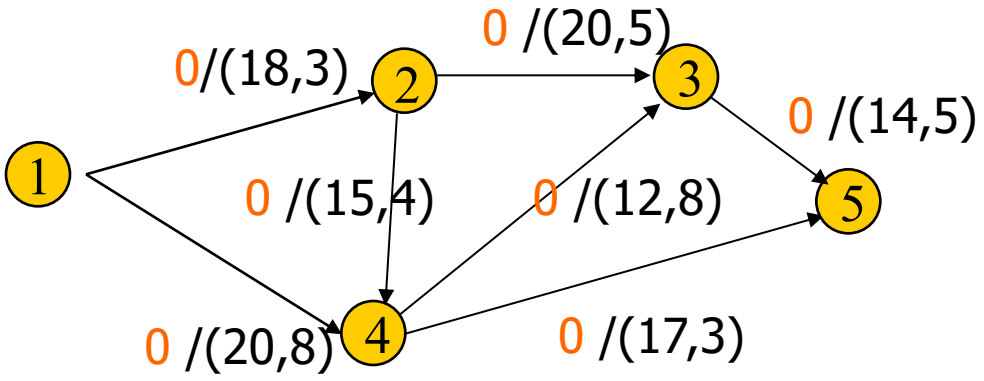


Algorithme de Busacker-Gowen

Appliquer l'algorithme de Busacker et Gowen à ce graphe

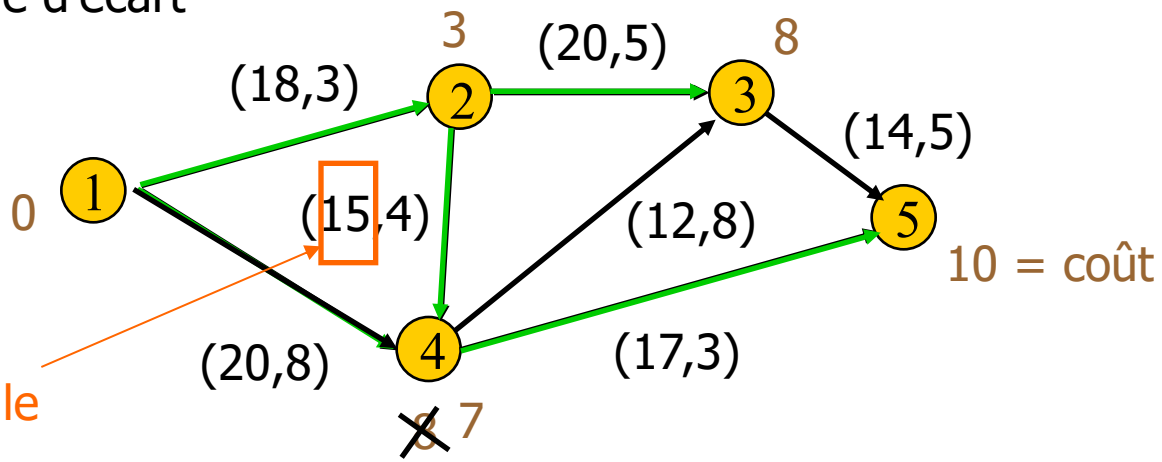


Algorithme de Busacker-Gowen



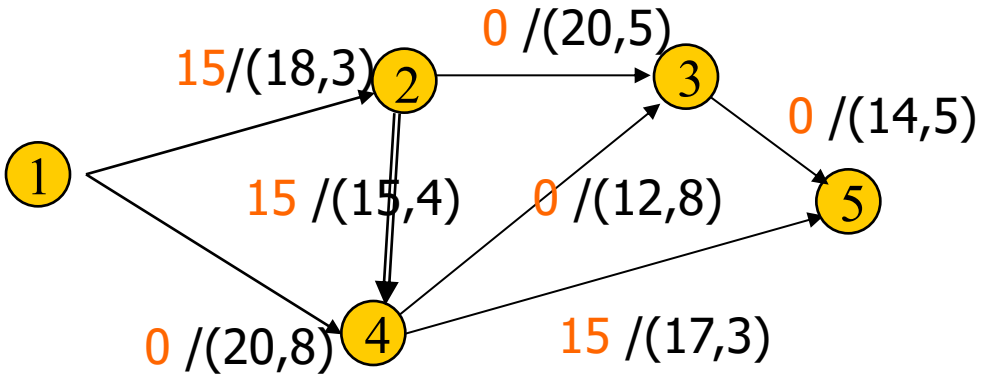
F = 0
W = 0

Graphe d'écart



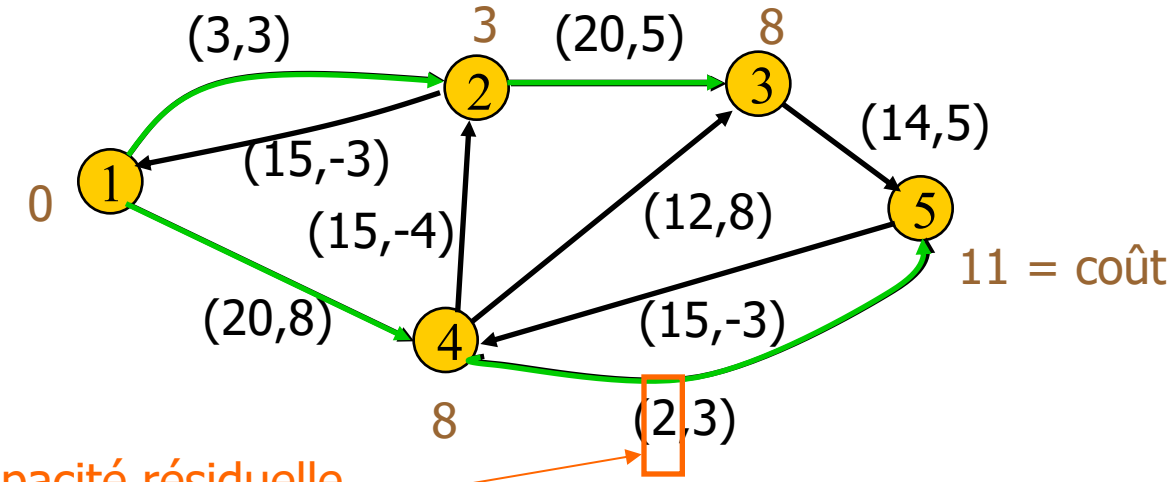
Capacité résiduelle minimale

Algorithme de Busacker-Gowen



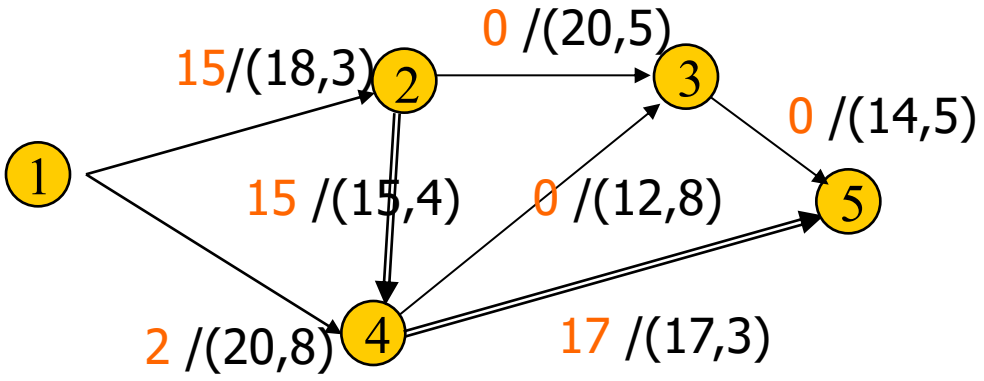
F = 15
W = 150

Graphe d'écart



Capacité résiduelle minimale

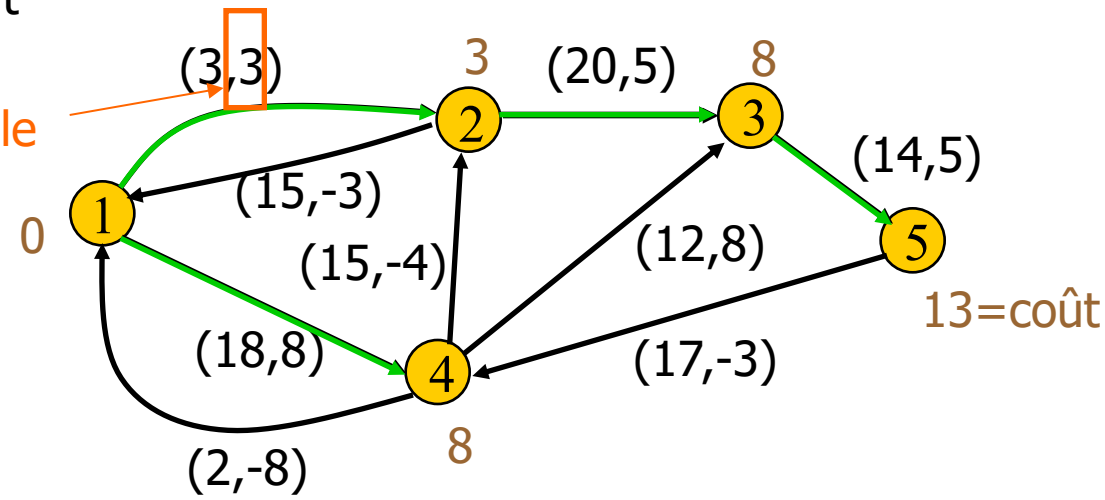
Algorithme de Busacker-Gowen



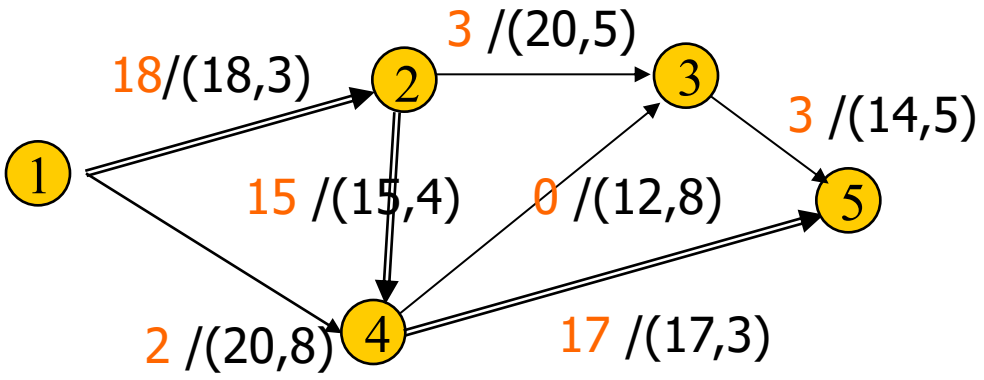
F = 17
W = 172

Graphe d'écart

Capacité résiduelle minimale

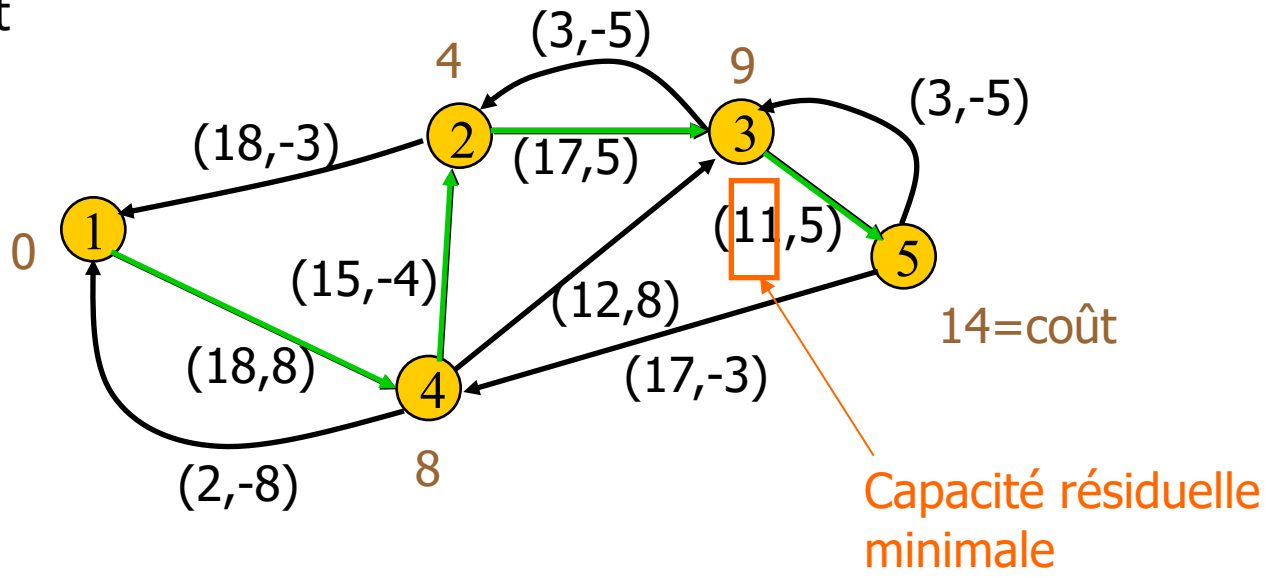


Algorithme de Busacker-Gowen

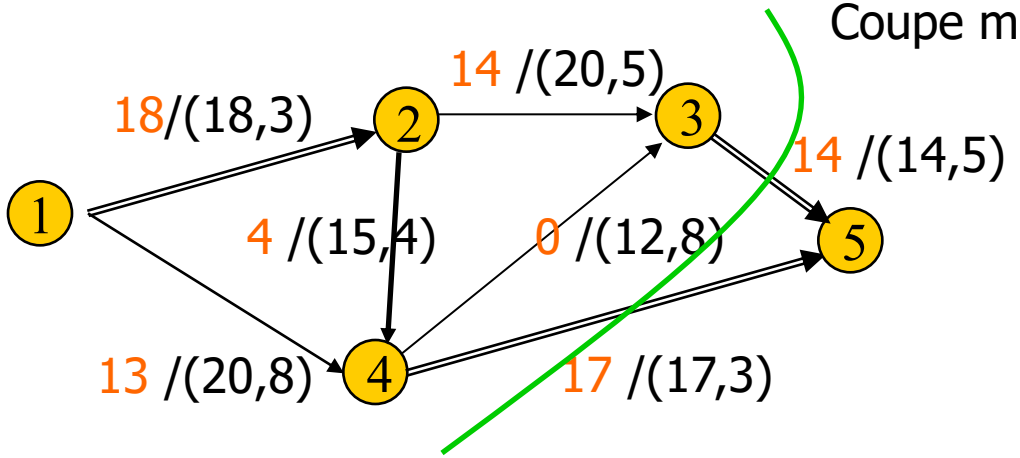


F = 20
W = 211

Graphe d'écart

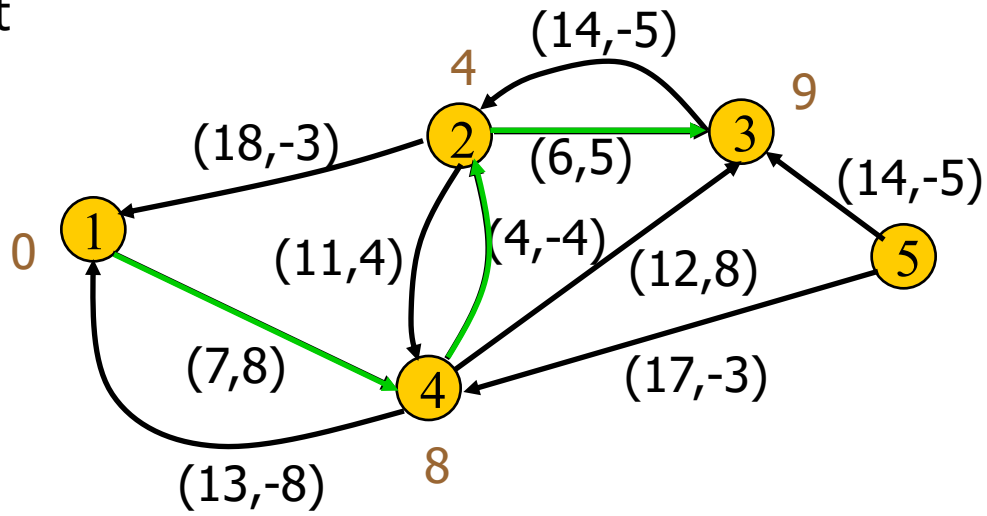


Algorithme de Busacker-Gowen



F = 31
W = 365

Graphe d'écart

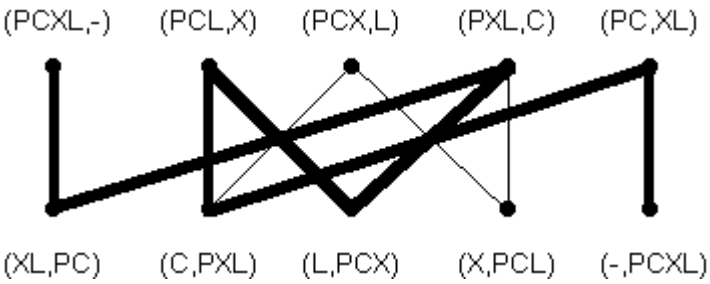


Solutions

Exo 1 : Loup, chèvre et chou

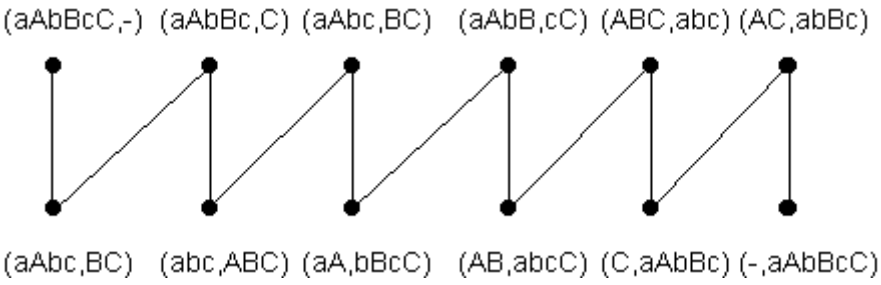
Il faut tracer le graphe de transition entre les états possibles. Le graphe n'est pas orienté puisqu'on peut toujours revenir à l'état précédent.

P : Batelier
C : Chèvre
X : Chou
L : Loup
Une fois que le graphe (biparti puisque les sommets pour lesquel le passeur est sur une rive ne sont reliés qu'aux sommets pour lesquel il est sur l'autre rive) est tracé, on voit qu'il y a deux chemins équivalents.



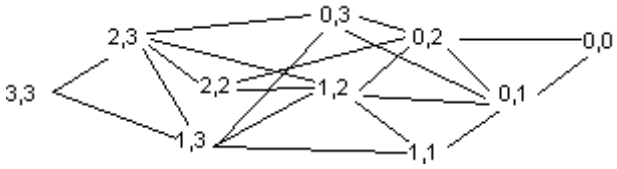
Exo 2 : Trois ménages

Selon le même principe on peu tracer touts les états
Avec A,B, C les femmes et a,b,c, les maris et b le bateau.
Voici une solution :



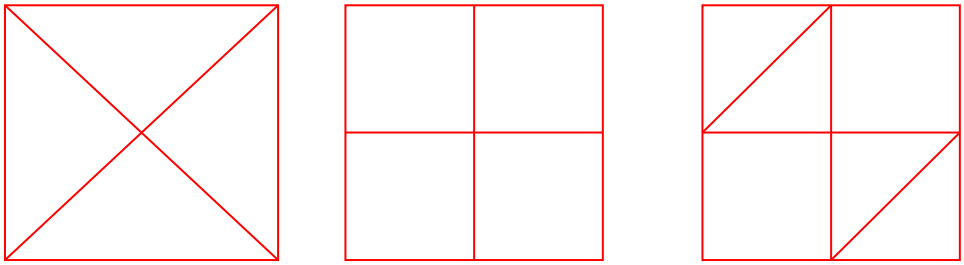
Exo3 jeu

Le jeu avec 2 tas de trois allumettes est décrit par le graphe suivant (tous les arcs sont orientés de gauche à droite) :



Le joueur qui atteint la configuration 0,0 perd la partie. Pour gagner, on doit donc atteindre la configuration 0,1. On peut vérifier qu'en jouant 1,3 au premier coup, quelle que soit la réponse de l'adversaire, on peut atteindre ensuite 0,1. Le coup gagnant au départ est donc « enlever 2 allumettes dans un tas ».

Exo 4 Figures



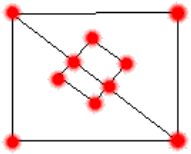
On peut dessiner uniquement la dernière figure car elle n'a aucun sommet de degré impair. Les deux premières ont 4 sommets de degré impair.

Leur rappeler le théorème vu en cours

Th 1

Il y a un circhuit eulérien dans un graphe s'il n'y a aucun sommet de degré impair (départ=arrivée)

Pour résoudre



Leur donner le théorème suivant (pas vu en cours !)

Th 2

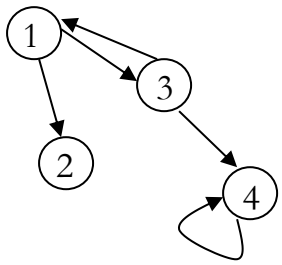
il y a un chemin eulérien dans un graphe s'il n'y a pas plus de 2 sommets de degré impair (départ = l'un des sommets impairs, arrivée = l'autre sommet impair).

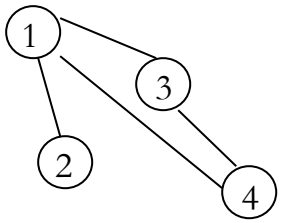
Graphes et Algorithmes

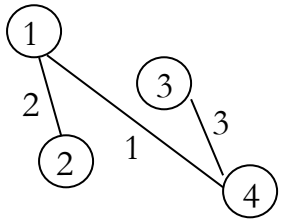
TD 2

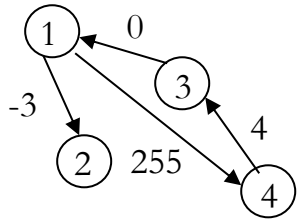
Exercice 1

Donner les représentations par matrices et listes d'adjacence des graphes suivants.









Exercice 2

Calculer les complexités (en temps de calcul et dans le pire des cas) des algorithmes suivants selon qu'on représente le graphe sous la forme d'une matrice ou d'une liste d'adjacence .

Algorithme 1

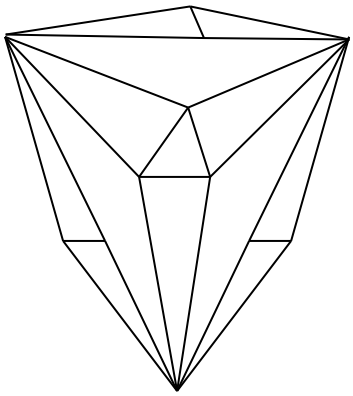
```
Pour chaque sommet i
    Pour chaque successeur j de i
        Afficher "(i,j)"
```

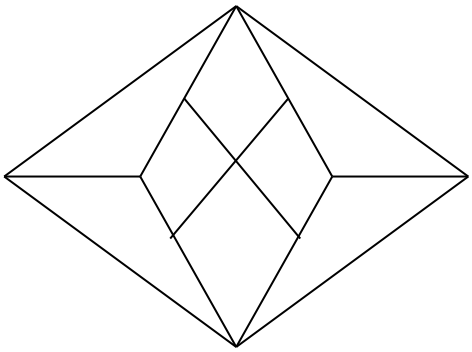
Algorithme 2

```
Pour chaque sommet i
    Pour chaque sommet j
        Si l'arc (i,j) existe
            Afficher "(i,j)"
        Sinon
            Afficher "i et j ne sont pas reliés"
```

Exercice 3

Un graphe qui admet un cycle hamiltonien (cf cours) est dit "graphe hamiltonien". Les graphes suivants sont ils hamiltoniens ? Démontrer ou infirmer





Remarque : Répondre à la question suivante peut vous aider.
Dans le cas où un graphe admet un cycle hamiltonien, combien de composantes connexes peut-on au plus obtenir en enlevant x sommets du graphe ?

Graphes et Algorithmes

TD 2

Exercice 1

Orienter cet exercice représentation informatique des graphes

Solution des matrices (4 ème : on ne peut pas mettre 0 pour l'absence d'arc).

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 2 & 0 & 1 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 \\ 1 & 0 & 3 & 0 \end{pmatrix} \begin{pmatrix} \infty & -3 & \infty & 255 \\ \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty \\ \infty & \infty & 4 & \infty \end{pmatrix}$$

Solution des listes :

Pas de pb. Créér d'abord le graphe orienté symétrique pour les graphes non orientés. Pour les graphes valués, faire deux tableaux de listes ou bien un seul tableau de listes de couples (successeur, poids)

Exercice 2

N nombre de sommets. M nombre d'arcs

Algorithme 1

Listes : $O(M)$ car $\sum_{i=1}^{N} d^+(i)=M$ (leur expliquer)

Matrices $O(N^2)$

Algorithme 2

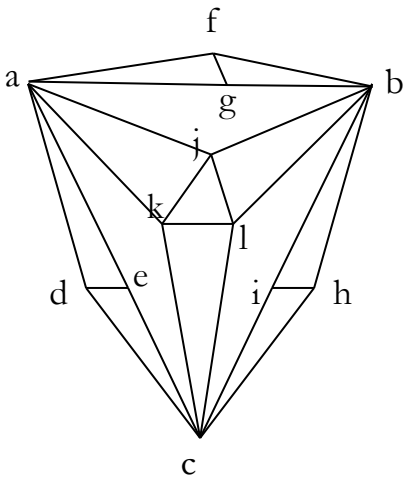
Matrices $O(N^2)$

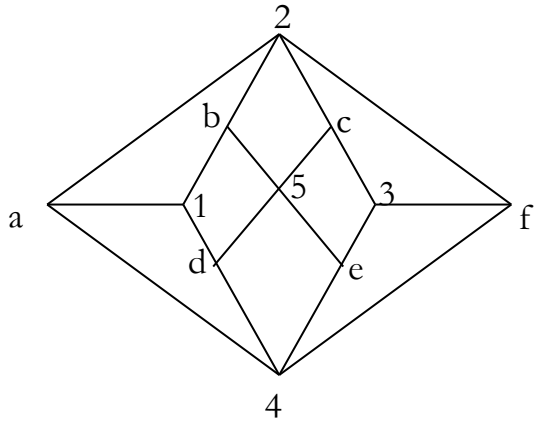
Listes ~~$O(N^2D)$ avec~~ $D=\max_{i=1,\ldots,N} d^+(i)$ NM

Exercice 3

Préambule : Rappeler ce qu'est un cycle hamiltonien et une composante connexe !!

- Réponse à la question : au plus x composantes connexes pour x sommets enlevés puisqu'il existe un cycle passant par tous les sommets.





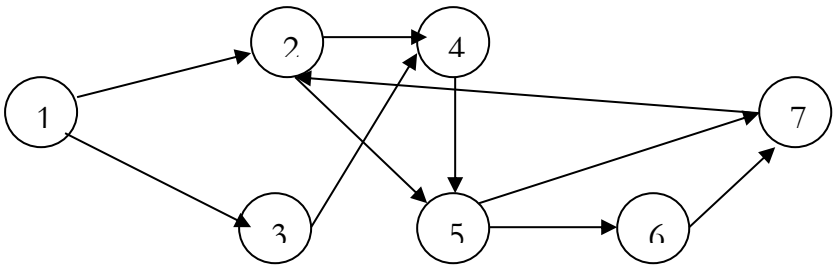
Pour le graphe de gauche. En enlevant a,b,c on obtient 4 composantes connexes {f,g}, {d,e}, {i,h}, {j,k,l} => pas hamiltonien

Pour le graphe de droite, on remarque qu'il est biparti. Séparer {a,b,c,d,e,f} et {1,2,3,4,5}. Si on enlève tous les sommets {1,2,3,4,5} on obtient les 6 composantes connexes {a},{b},{c},{d},{e} et {f} => pas hamiltonien

Graphes et Algorithmes

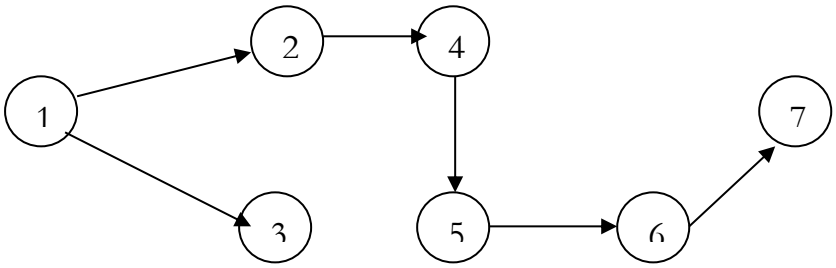
TD 3 (solutions)

Exercice 1 Algorithmes d'exploration

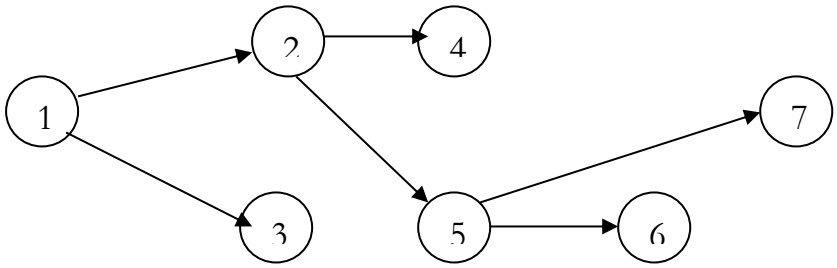


a) simuler l'exécution des algorithmes en montrant l'évolution de la pile pour l'algo enb profondeur et de la file pour l'algo en largeur.

On obtient



Pour l'algo en profondeur et



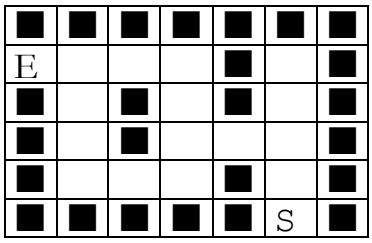
Pour l'algo en largeur.

b) Dans l'algo en profondeur on stocke dans la pile tous les sommets issus du sommet source tant qu'on arrive au « bout » d'un chemin. Lorsqu'on traite un sommet *x*, il suffit de regarder si un successeur *y* déjà marqué de *x* est dans la pile.

Par exemple lorsqu'on traite 7, la pile contient 1,2,4,5,6. 2 qui est un successeur de 7 déjà marqué est dans la pile. Il y a donc un circuit et on peut même afficher ce circuit 2,4,5,6,7.

c) algo en largeur
mple du graphe ci-dessus pour le plus court chemin entre 1 et 7.

Exercice 2 Le labyrinthe



Numéroter les cases. Une case = 1 sommet et mettre un arc entre deux sommets si on peut passer de la case à l'autre. C'est l'algo en largeur qui permet de résoudre le problème. En pratique on aura du mal à appliquer l'algo en largeur !!

Exercice 3 Composantes connexes et fortement connexes
Rappeler ce que c'est !
Pour calculer les composantes connexes :
a) on dessine le graphe orienté symétrique correspondant

b) on lance l'exploration (en largeur ou en profindeur) à partir d'un sommet et tous les sommets marqués sont de la même composante (on utilise un compteur de composante)
c) on incrémente le compteur et on remance b) à partir d'un sommet non marqué jusqu'à ce qu'il n'y ait plus de sommets non marqués.

Pour les composantes fortement connexes. Exemple d'algorithme simple (cf Prins)

a) On calcule le graphe H inverse de G (les arcs dans l'autre sens).
b) On part d'un sommet s. On lance une exploration dans G à partir de s et on marque les sommets atteints. On lance l'exploration dans H à partir de s et les sommets marqués dans les deux explorations font partie de la même composante connexe que s (compteur).
c) on incrémente le compteur des CFC et on remance b) à partir d'un sommet non marqué jusqu'à ce qu'il n'y ait plus de sommets non marqués.

Complexité O(NM)

Graphes et Algorithmes

TD 4

Exercice 1 : Remplacement de matériel

Une entreprise dispose en permanence de deux machines. Toute machine doit être changée chaque mois ou bien subir une opération de maintenance. Le coût de l'opération de maintenance diffère chaque mois en fonction de l'usure prévisionnelle de la machine qui dépend de l'activité de l'entreprise. De même les coûts d'achat varient en fonction des prix du marché. Le tableau suivant résume les coûts cumulés d'achat de 0, 1 ou 2 machines pour les trois premiers mois de l'année.[1] Il est toutefois impossible de remplacer plus de deux machines pendant la période considérée. L'entreprise veut minimiser son coût de maintenance. Modélisez ce problème à l'aide d'un graphe. Quel problème doit-on résoudre dans ce graphe pour répondre aux besoins de l'entreprise ? Résoudre le problème avec l'algorithme approprié.

coût/mois	janvier	février	mars
0	10	30	20
1	20	15	7
2	25	5	15

Exercice 2 : Complexité de l'algorithme de Dijkstra

Calculer la complexité de l'algorithme de Dijkstra en représentant le graphe

- avec les listes de successeurs,
- avec la matrice d'adjacence

Exercice 3 : Preuve d'optimalité de l'algorithme de Dijkstra

Au cours de l'exécution de l'algorithme (cf cours), V_i ne varie plus dès que i est marqué. Or, à chaque itération, le sommet marqué est celui de plus petit V_i parmi les sommets non marqués. Démontrer par l'absurde que V_i est bien la longueur du plus court chemin de s à i, sous certaines conditions.

TSVP

[1] Par exemple changer une machine et en réparer une en février coûte 15000 euros à l'entreprise. Changer les deux coûte 5000 euros.

Aide :

- Montrer par récurrence sur les itérations qu'il existe un chemin de s à i de longueur V_i pour tout sommet i (marqué ou non) tel que $V_i \neq +\infty$.
- A une itération donnée, on considère maintenant le sommet i non marqué vérifiant $V_i = \min \{V_j | j \text{ non marqué}\}$. Considérer un chemin de s à i de longueur L. Peut-on avoir $V_i > L$? Pour répondre, décomposer ce chemin en deux parties : un chemin de s à h avec h non marqué (pourquoi un tel sommet h existe-t-il ?) et un chemin de h à i…

Exercice 4

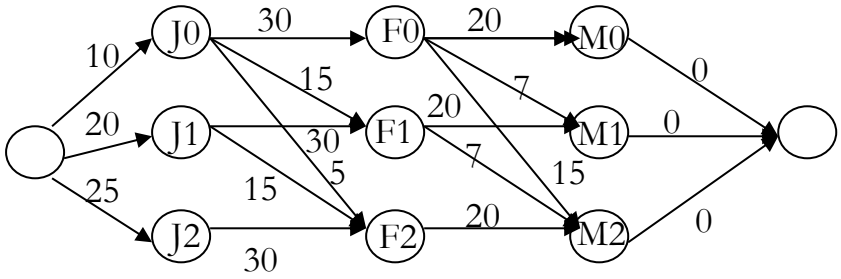
Dans un graphe dont les arcs ont des valeurs strictement positives, on souhaite calculer le plus court chemin d'un sommet de départ s vers tout autre sommet.
Supposons qu'on ne dispose que de l'algorithme d'exploration en largeur. Indiquer une façon d'obtenir tout de même le résultat souhaité. Quelle est la complexité de l'algorithme ?

Graphes et Algorithmes

TD 4 solution

Exercice 1 : Remplacement de matériel

coût/mois	janvier	Février	mars
0	10	30	20
1	20	15	7
2	25	5	15



Sommets : couple (mois, nombre de machines changées)
Arc : coût.

Résoudre avec l'algorithme de Dijkstra

Exercice 2 : Complexité de l'algorithme de Dijkstra

Listes de successeurs :
- init N itérations
_ Boucle principale N itérations

- Recherche du Min : N-q tests où i est l'indice d'itération
- d+(i) ou i est le sommet marqué à l'itération q

Nb total d'iterations

$$N + \sum_{i=1}^{N}(N-i)+d^{+}(i) = N + N(N-1)/2 + M = O(N^2) \text{ car } M<=N^2$$

Matrice d'adjacence pareil sauf que le parcours des successeurs coute O(N) => même complexité

Exercice 3 : Preuve d'optimalité de l'algorithme de Dijkstra

- Montrer par récurrence sur les itérations qu'il existe un chemin de s à i de longueur Vi pour tout sommet i (marqué ou non) tel que $Vi \neq +\infty$.

a) Itération 1

Sommet s marqué à la première itération, tous ses successeurs j obtiennent un Vj > 0 et il existe un chemin de s à j : l'arc (s,j).

b) Si vrai à l'itération q–1, itération q

Soit i marqué à l'itéraition q, Vi a été calculé à une itération < q => il y a un chemin de s à i. Pour les autres sommets j, Si $Vj \neq +\infty$, alors Vj a été calculé à une itération < q => il y a un chemin de s à j. . Sinon, Vj est mis à jour à l'itération q et i est un successeur de i => il existe un chemin de s à i de valeur Vj .

- suite

Puisque h est non marqué Vh <= Vi. Or l(h,i) >= 0 (valeurs positives)

D'où l >= Vi.

Exercice 4

Décomposer un arc de longueur V>1 en V arcs de longueur 1.
Complexité =O(Nombre d'arcs) = O(somme des valeurs de tous les arcs) !

Graphes et Algorithmes

TD 5 : ordonnancement

On considère un projet de construction d'une maison dont les tâches sont données dans le tableau ci-dessous. Pour chaque tâche, on donne sa durée en jours et les tâches qui doivent être achevées avant de pouvoir la démarrer.

Tâche		Durée	Tâches précédentes
1	Murs	7	-
2	Charpente	3	1
3	Toiture	1	2
4	Plomberie	8	1
5	Electricité	2	3.4
6	Fenêtres	1	3.4
7	Jardins	1	3.4
8	Peinture	3	6
9	Cuisine	2	8
10	installation	1	5.7.9

On cherche à déterminer la durée minimale du projet et les plus petites dates de début possibles des tâches (appelées dates de début au plus tôt).

Q. 1 : Modéliser comme un problème de graphe.

Q. 2 : Résoudre avec l'algorithme approprié.

Q. 3 : Modéliser comme un problème de graphe le problème de calcul pour chaque tâche de la plus grande date de début permettant de respecter la durée minimale du projet calculée à l'étape précédente. On appelle cette date la date de début au plus tard.

Q.4 : Calculer les dates de début au plus tard avec l'algorithme approprié.

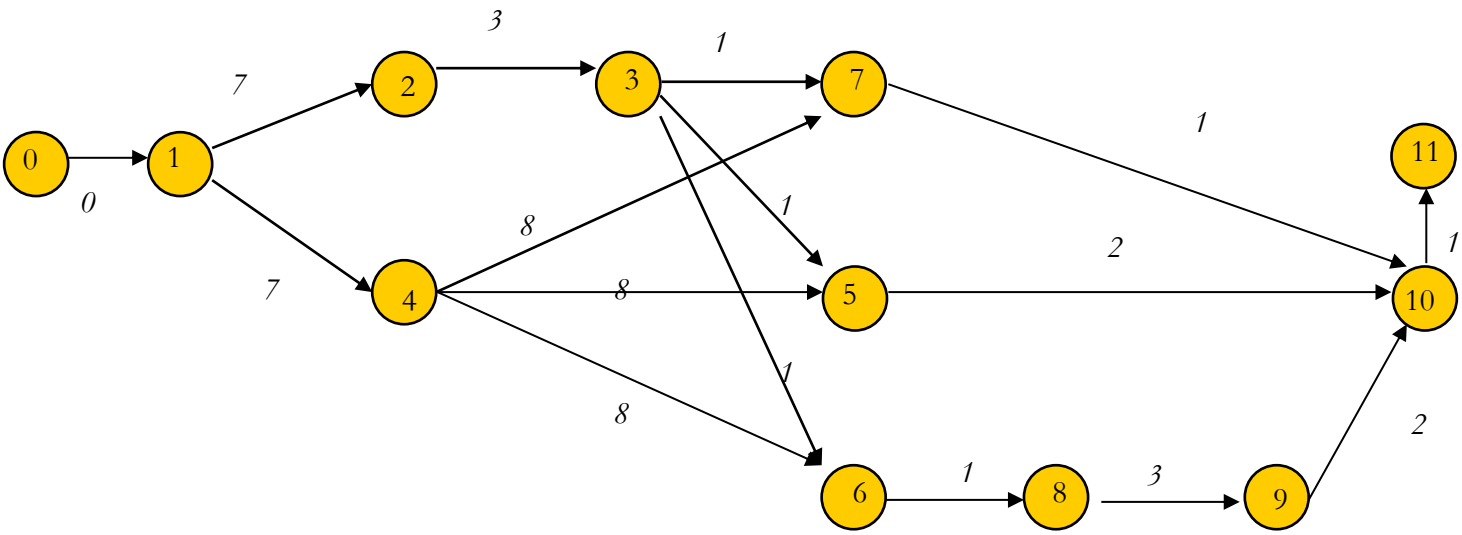
Q.5 : Les tâches qui ont une date de début au plus tôt égale à la date de début au plus tard sont appelées *critiques*. Quelle est la particularité des tâches critiques par rapport au graphe.

Q.6 : On souhaite modifier les contraintes qui lient la tâche 3 et la tâche 6 comme suit : 6 ne peut démarrer avant la fin de 3 mais doit être aussi démarrée **au plus** 2 jours après la fin de 3. Que devient le problème de graphe associé ? Résoudre.

Graphes et Algorithmes

TD 5 : ordonnancement : correction

Q. 1 : Modéliser comme un problème de graphe.



- Date de début de la tâche i : plus long chemin entre 0 et i
- Durée minimale du projet : plus long chemin entre 0 et 10

Q. 2 : Résoudre avec l'algorithme approprié.

On résout avec la version de l'algorithme de Bellman appliquée aux graphes sans circuit. Nous avons vu en cours la version calculant le plus court chemin. Au lieu d'appliquer l'algorithme au graphe avec des poids négatif, l'algorithme de Bellman est facilement transposable au plus longs chemins :

a) on décompose le graphe en niveaux et on associe un rang aux tâches (le rang d'une tâche est le plus long chemin en nombres d'arcs de 0 vers la tâche)
Rang des tâches :
Rang 0 : 0, rang 1 : 1, rang 2 : 2 et 4, rang 3 : 3, rang 4 : 7, 5 et 6, rang 5 : 8, rang 6 : 9, rang 7 : 10

b) on applique l'algorithme de Bellman en sélectionnant les tâches classées dans l'ordre des rangs croissants

$V_0=0$.
$V_1= V_0+0=0$.
$V_2= V_1+7=7$
$V_4= V_1+7=7$

$V_3 = V_2+3=10$
$V_7 = \max(V_3+1, V_4+8)=15$
$V_5 = \max(V_3+1, V_4+8)=15$
$V_6 = \max(V_3+1, V_4+8)=15$
$V_8 = V_6+1=16$
$V_9 = V_8+3=19$
$V_{10} = \max(V_7+1, V_5+2, V_9+2)=21$
$V_{11} = V_{10}+1 = 22$ (durée minimale du projet)

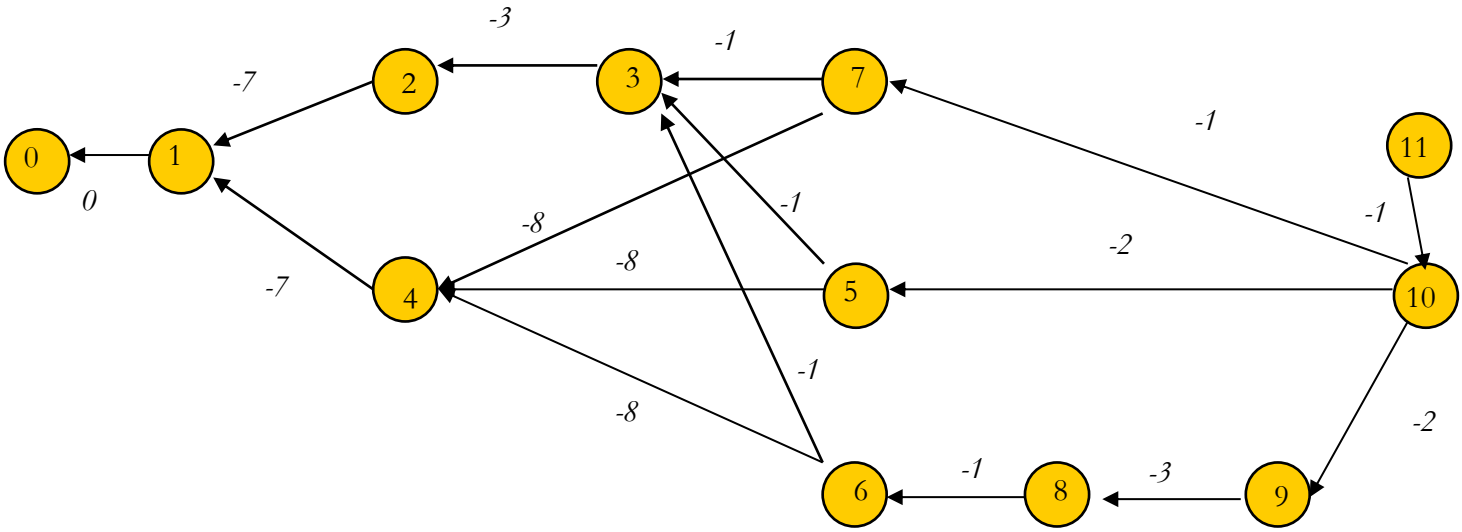
Q. 3 : Modéliser comme un problème de graphe le problème de calcul pour chaque tâche de la plus grande date de début permettant de respecter la durée minimale du projet calculée à l'étape précédente. On appelle cette date la date de début au plus tard.

Le temps minimal qui sépare le début d'une tâche i avec la fin du projet est donné par le plus long chemin entre i et le sommet final.

La date de début au plus tard de la tâche i est égale à la durée minimale du projet moins la longueur de ce plus long chemin.

Q.4 : Calculer les dates de début au plus tard avec l'algorithme approprié.

Il faut inverser le graphe et calculer les plus longs chemins à partir de 10. On peut obtenir directement la date de début au plus tard en appliquant l'algorithme de recherche des plus courts chemins dans le graphe inverse avec les poids opposés en donnant un potentiel initial égal à la durée minimale à la tâche de fin.



$V_{11}=22$
$V_{10} = V_{11}-1=21$
$V_9 = V_{10}-2 = 19$

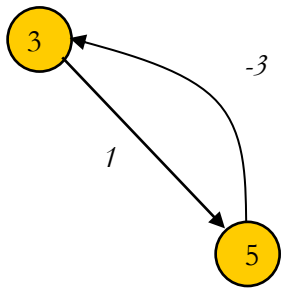
$V_8 = V_9 - 3 = 16$
$V_6 = V_8 - 1 = 15$
$V_5 = V_{10} - 2 = 19$
$V_7 = V_{10} - 1 = 20$
$V_3 = MIN(V_7 - 1, V_5 - 1, V_6 - 1) = 14$
$V_4 = MIN(V_7 - 8, V_5 - 8, V_6 - 8) = 7$
$V_2 = V_3 - 3 = 11$
$V_1 = MIN(V_2 - 7, V_4 - 7) = 0$
$V_0 = V_1 = 0$

Q.5 : Les tâches qui ont une date de début au plus tôt égale à la date de début au plus tard sont appelées *critiques*. Quelle est la particularité des tâches critiques par rapport au graphe.

Elles sont sur un chemin de longueur maximale. Si on allonge la longueur de ce chemin, la longueur du plus long chemin augmente => la durée minimale de réalisation du projet augmente.

Q.6 : On souhaite modifier les contraintes sui lient la tâche 3 et la tâche 6 comme suite. 6 ne peut démarrer avant la fin de 3 mais doit être démarrée au plus 2 jours après la fin de 3. Que devient le problème de graphe associé ? Résoudre.

On ajoute un arc de valeur –(1+2) entre 5 et 3. en effet la contrainte ajoutée implique que V5 ≤ V3 +1 + 2, c'est à dire V3 ≥ V5 - 3



Le graphe contient un circuit (de longueur négative). On applique l'algorithme de Bellman classique et la date de début de 3 est décalée à la deuxième itération : V3=12.

Graphes et Algorithmes

TD 6 : problèmes de flots

Exercice 1 : fiabilité dans un réseau de télécommunications

Un réseau de télécommunications est composé de nœuds et de liaisons. On considère un nœud de départ *s* et un nœud d'arrivée *t*. On cherche à évaluer la fiabilité de ce réseau. Pour cela, on construit le graphe correspondant et on cherche à déterminer la plus grande valeur *k* pour laquelle ce graphe est *k*-connexe, c'est à dire qu'il existe *k* chemins disjoints entre *s* et *t*. En effet, s'il y a *k* chemins disjoints dans le graphe, alors *s* peut transmettre à *t* même si des pannes surviennent et coupent jusqu'à *k*-1 chemins d'accès.

Construire un réseau de transport tel que le calcul du flot maximal de *s* à *t* résolve le problème.

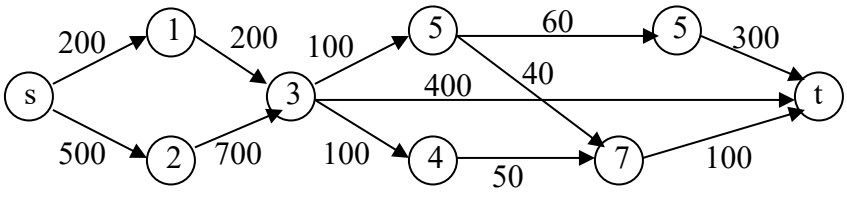
Exercice 2 Problème d'adduction d'eau

Considérons *r* points d'eau (barrages, sources) connectés à *v* villes par un réseau de canalisations. On connaît pour chaque point d'eau la quantité maximale en m3/jour qu'il peut fournir et pour chaque ville la consommation totale d'eau en m3/jour prévue dans 5 ans. On connaît les débits possibles en m3/jour pour chaque canalisation du réseau actuel. On veut savoir si le réseau actuel pourra satisfaire les consommations prévues dans 5 ans.

Construire un réseau de transport tel que le calcul du flot maximal de *s* à *t* résolve le problème.

Exercice 3

Appliquer l'algorithme de Ford et Fulkerson au réseau de transport suivant pour calculer le flot maximum entre *s* et *t*. Déterminer la coupe minimale.

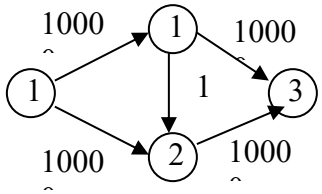


Exercice 4

Montrer que le débit de tout flot ϕ est inférieur ou égal à la capacité de toute coupe C(S,T)

Indication : sommer les lois de conservation du flot ϕ membre à membre sur les nœuds de *S* et simplifier

Exercice 5



Calculer le flot maximum entre 1 et 3 dans le graphe ci-dessus au moyen de l'algorithme de Ford et Fulkerson en utilisant l'algorithme d'exploration en profondeur, puis en largeur. En combien d'itérations l'algorithme converge-t-il dans chacun des deux cas ?

Graphes et Algorithmes

TD 6 solution

Exercice 1 : fiabilité dans un réseau de télécommunications

Il suffit de considérer que les nœuds du réseau ont une capacité de 1 : un seul chemin pourra passer par le nœud. On transforme ensuite chaque nœud en 2 somemts reliés par un arc de capacité 1. Les autres arcs ont une capacité infinie. Le flot max dans ce graphe donne le plus grand nombre de chemins disjoints chemins disjoints.

Exercice 2 Problème d'adduction d'eau

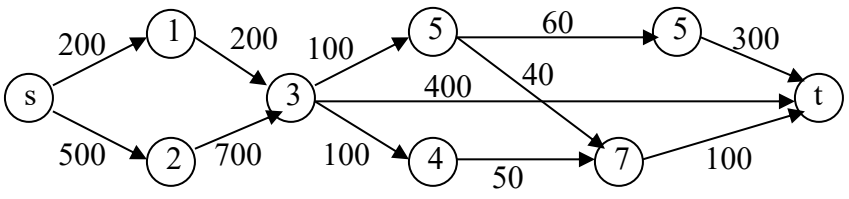
On crée un graphe dont les sommets sont
- les points d'eau
- les villes
- les intersections entre canalisations

et les arcs les canalisations.

On crée en plus
- une source fictive *s* reliée à chaque point d'eau par un arc dont la capacité est la quantité maximale disponible / jour
- un puit fictif *t* relié à chaque ville par un arc dont la capacité est égale à la consommation requise.

Exercice 2

Appliquer l'algorithme de Ford et Fulkerson au réseau de transport suivant pour calculer le flot maximum entre *s* et *t*. Déterminer la coupe minimale.



Les chemins sont cherchés « à la main » :
(pompé sur le web !)

$$A \xrightarrow[0]{200} 1 \xrightarrow[0]{200} 3 \xrightarrow[200]{400} B \qquad \varphi = 200$$

$$A \xrightarrow[300]{500} 2 \xrightarrow[500]{700} 3 \xrightarrow[0]{200} B \qquad \varphi = 200$$

$$A \xrightarrow[240]{300} 2 \xrightarrow[440]{500} 3 \xrightarrow[40]{100} 5 \xrightarrow[0]{60} 6 \xrightarrow[240]{300} B \qquad \varphi = 60$$

$$A \xrightarrow[200]{240} 2 \xrightarrow[400]{440} 3 \xrightarrow[0]{40} 5 \xrightarrow[0]{40} 7 \xrightarrow[60]{100} B \qquad \varphi = 40$$

$$A \xrightarrow[150]{200} 2 \xrightarrow[350]{400} 3 \xrightarrow[50]{100} 4 \xrightarrow[0]{50} 7 \xrightarrow[10]{60} B \qquad \varphi = 50$$

Exercice 4

En sommant les équations de conservations du flot sur les sommets de s on obtient

$$\sum_{i \in S} \sum_{j \in \Gamma(i)} \phi_{ij} = \sum_{i \in S} \sum_{j \in \Gamma^{-1}(i)} \phi_{ji}$$

d'où

$$\sum_{i \in S} \sum_{j \in \Gamma(i)} \phi_{ij} = F + \sum_{i \in S, i \neq s} \sum_{j \in \Gamma^{-1}(i)} \phi_{ji}$$

On peut simplifier l'expression en soustrayant les flux des arcs dont les deux extrémités sont dans S qui apparaissent des deux côtés. On ontient

$$\sum_{i \in S} \sum_{j \in T} \phi_{ij} = F + \sum_{i \in S, i \neq s} \sum_{j \in T} \phi_{ji}$$

or on sait que $\sum_{i \in S} \sum_{j \in T} \phi_{ij} \leq C(s,t)$

d'où la propriété.

Exercice 5

- 2000 itérations en profondeur (les successeurs sont pris dans l'ordre lexicographique et on trouve toujours la même chaine améliorante passant par l'arc de capacité 1) et 2 seulement en largeur.