Graphes, Algorithmes

et modélisation

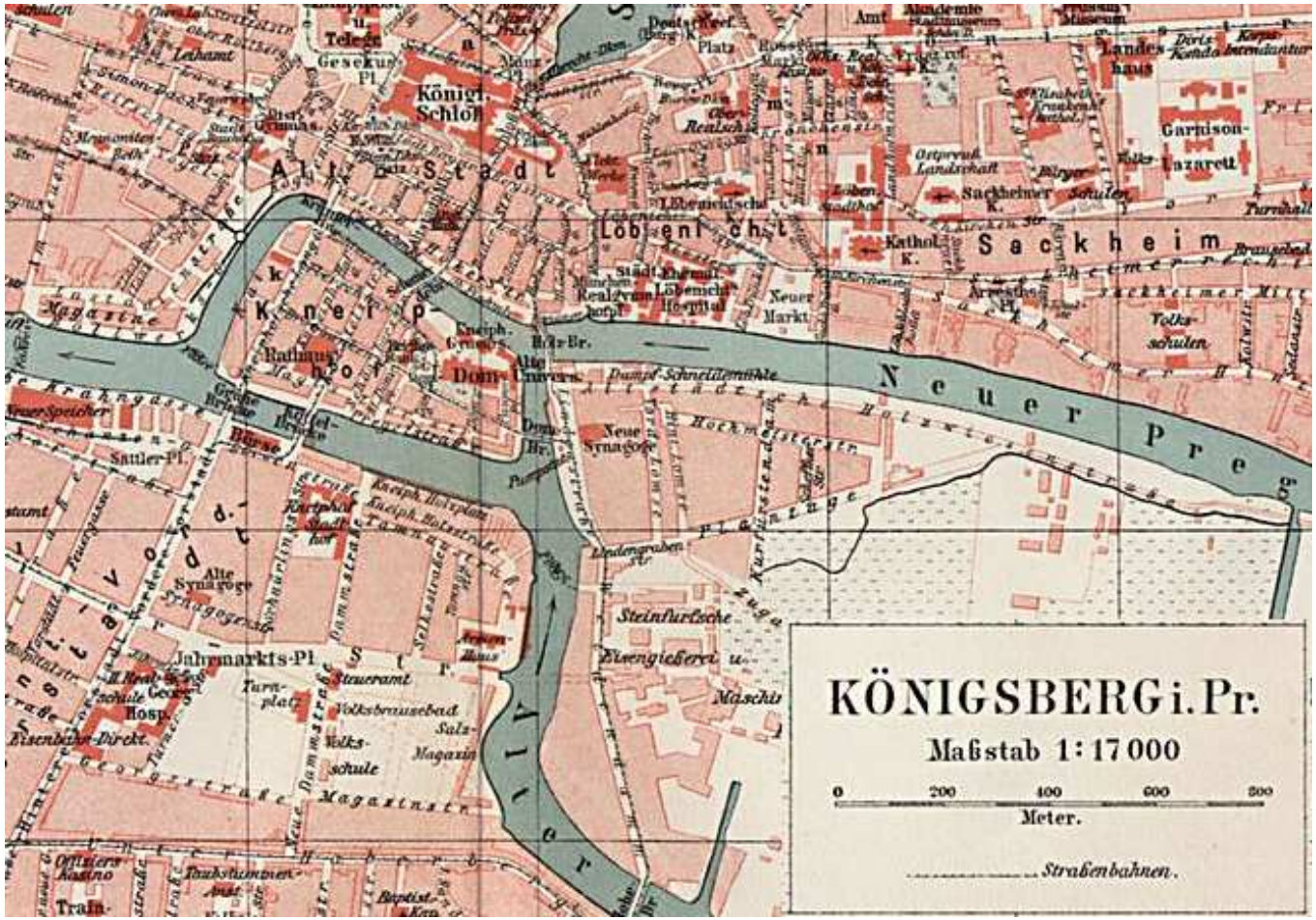
Thomas Schiex, Simon de Givry

INRA [1]

2 octobre 2012

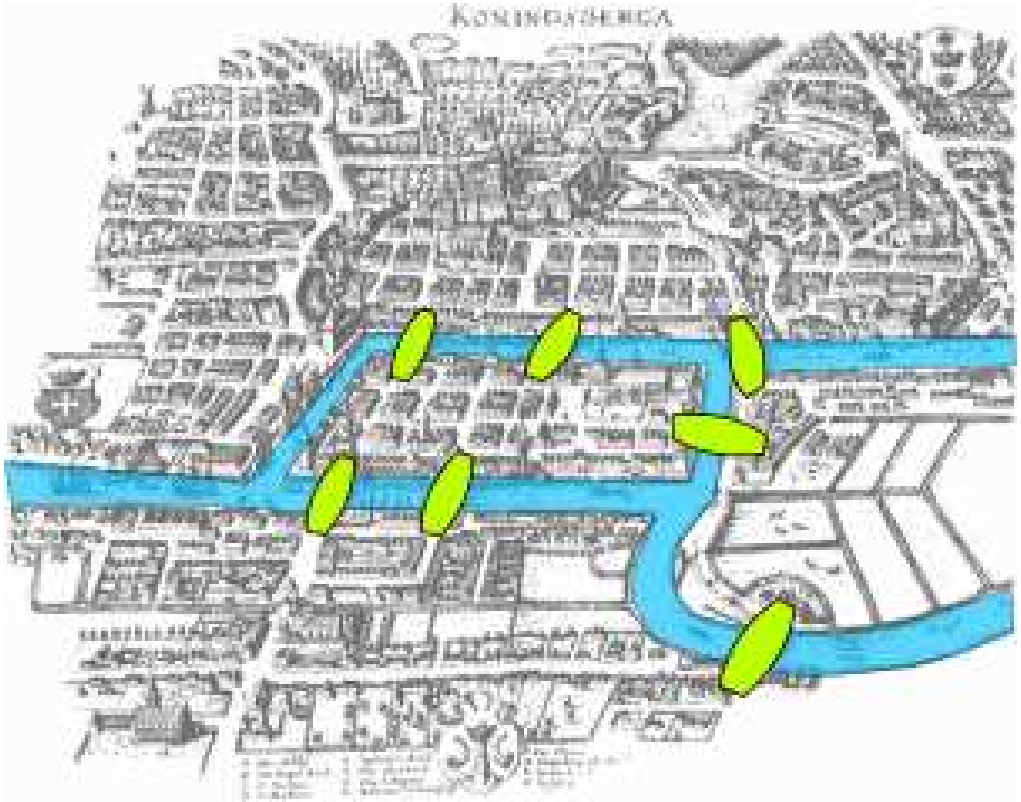
1. Remerciements à J-P. Métivier pour ses transparents animés. Certaines figures proviennent de Cormen, Leiserson et Rivest ©MIT, 2001

Graphes



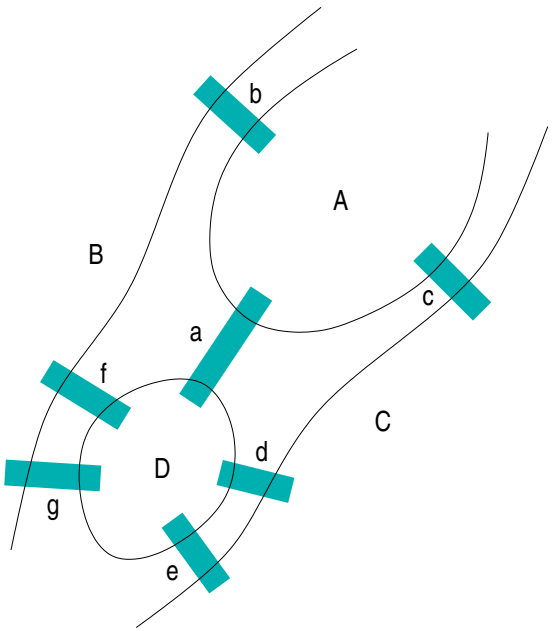
Visiter la ville en passant une seule fois par chaque pont ?

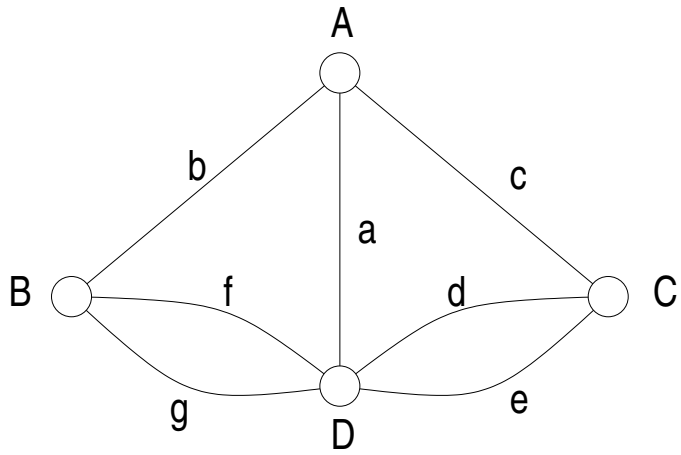
Graphes



Les éléments importants. . .

Graphes

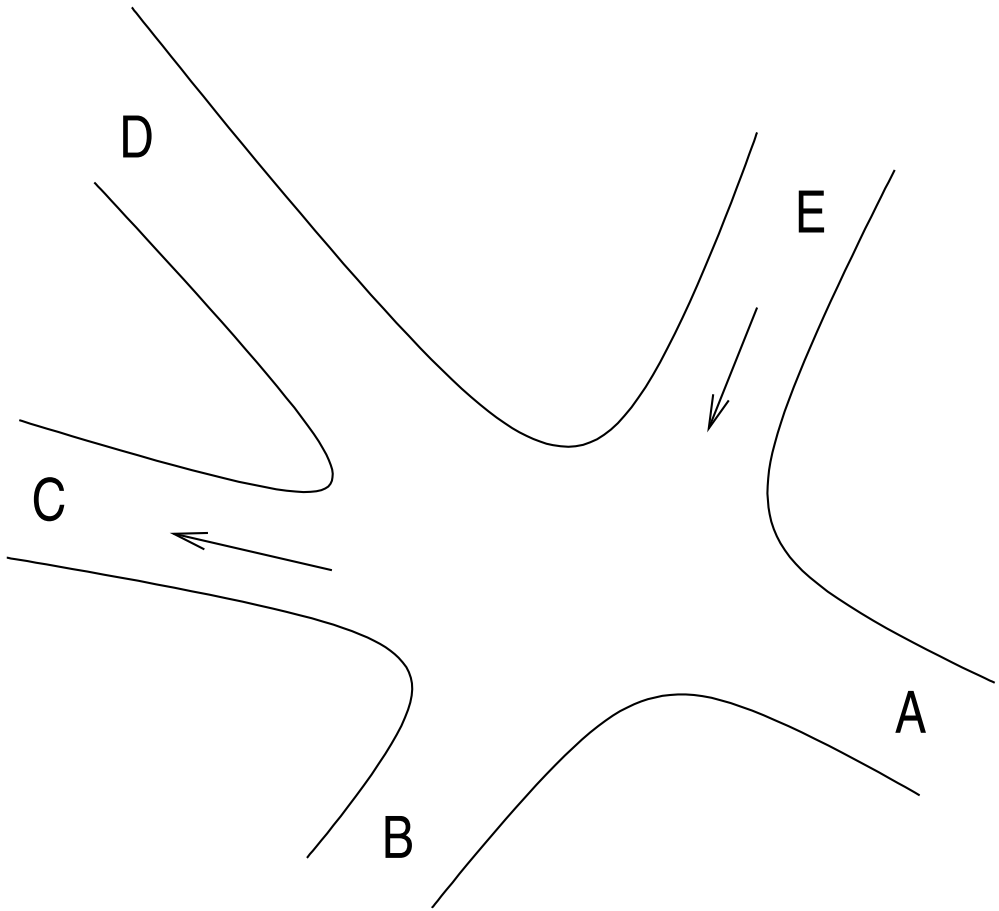




Les dernières étapes...

Un graphe admet un circuit eulérien ssi tout sommet est connecté à un nombre pair de sommets.

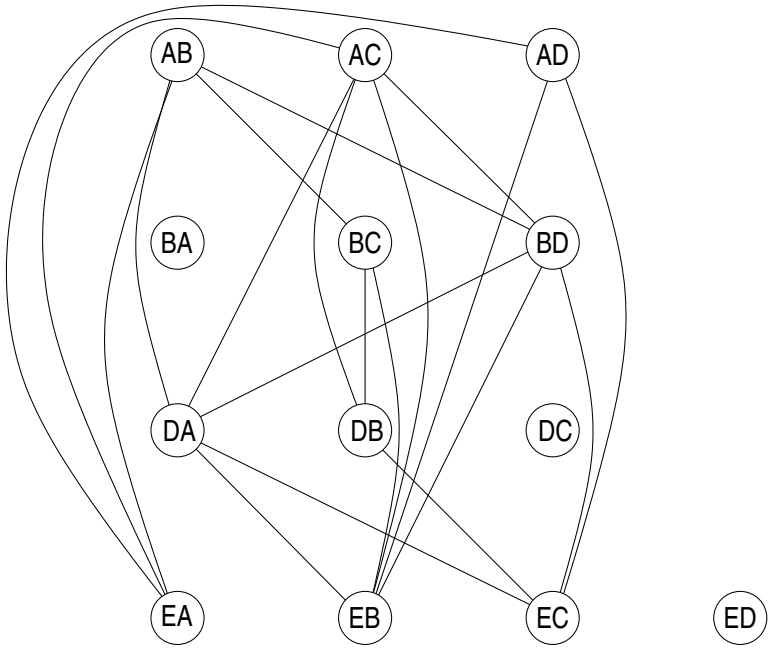
Carrefour (Princeton)



Déterminer les différentes phases des feux de circulation.

Graphes. . .

1. lister les flots possibles (sommets).
2. lister les incompatibilités entre flots (arêtes).
3. Problème de détermination d'un nombre minimum de "phases : coloriage (NP-dur).



Le graphe d'incompatibilité

Graphes non orientés

Définition (Graphe non orienté)

un graphe non orienté $G = (S, A)$ est défini par :

- *un ensemble S fini de sommets.*

Graphes non orientés

Définition (Graphe non orienté)

un graphe non orienté $G = (S, A)$ est défini par :

- *un ensemble S fini de sommets.*
- *un ensemble A d'arêtes, chaque arête connectant deux sommets (pas nécessairement différents).*

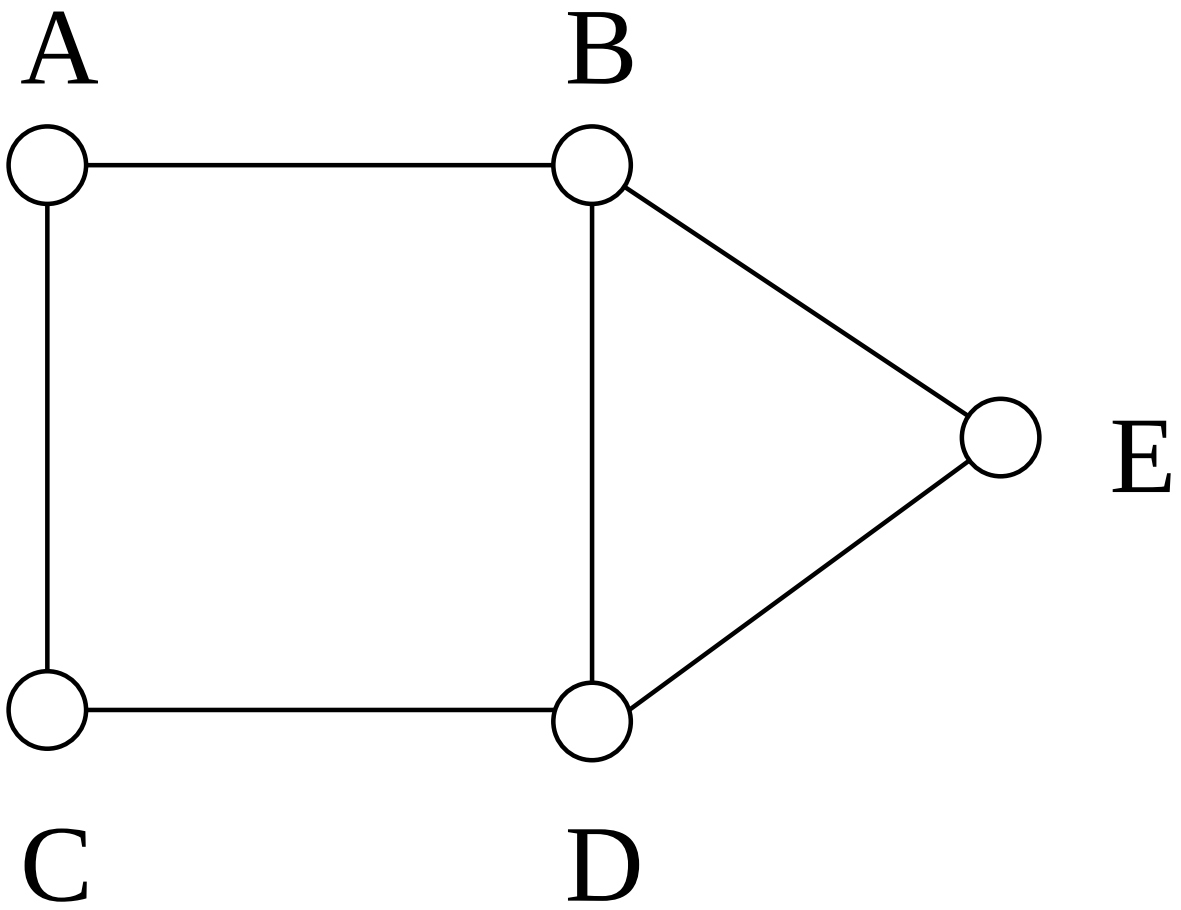
Graphes non orientés

Définition (Graphe non orienté)

un graphe non orienté $G = (S, A)$ est défini par :

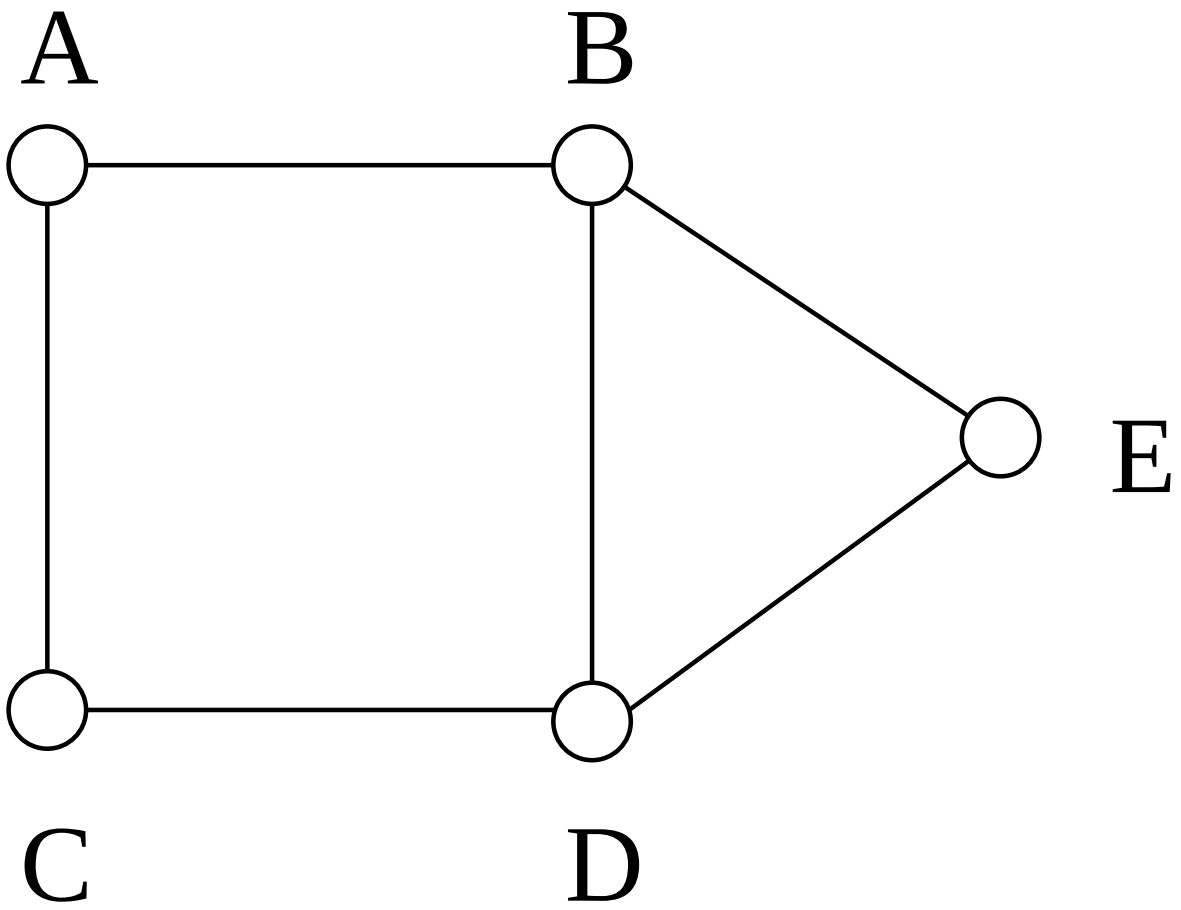
- *un ensemble S fini de sommets.*
- *un ensemble A d'arêtes, chaque arête connectant deux sommets (pas nécessairement différents).*
- *Une arête $a = \{x, y\}$ est aussi notée $x - y$. On dit que x et y sont adjacents.*

Un graphe non orienté



Un graphe non orienté, G_2

Un graphe non orienté



Un graphe non orienté, G_2

N.B. en anglais :

- graph ≡ graphe
- vertex/vertices ≡ sommet/sommets
- edge ≡ arête

Graphes orientés

Définition (Graphe orienté)

un graphe orienté $G = (S, A)$ est défini par :

- *un ensemble S fini de sommets (vertices, un sommet : a vertex)*

Graphes orientés

Définition (Graphe orienté)

un graphe orienté $G = (S, A)$ est défini par :

- *un ensemble S fini de sommets (vertices, un sommet : a vertex)*
- *un ensemble A d'arcs, chaque arc ayant une origine et un but dans S.*

Graphes orientés

Définition (Graphe orienté)

un graphe orienté $G = (S, A)$ est défini par :

- *un ensemble S fini de sommets (vertices, un sommet : a vertex)*
- *un ensemble A d'arcs, chaque arc ayant une origine et un but dans S.*
- *Un arc $a = (s, t)$: t est un successeur de s, s un prédécesseur de t.*

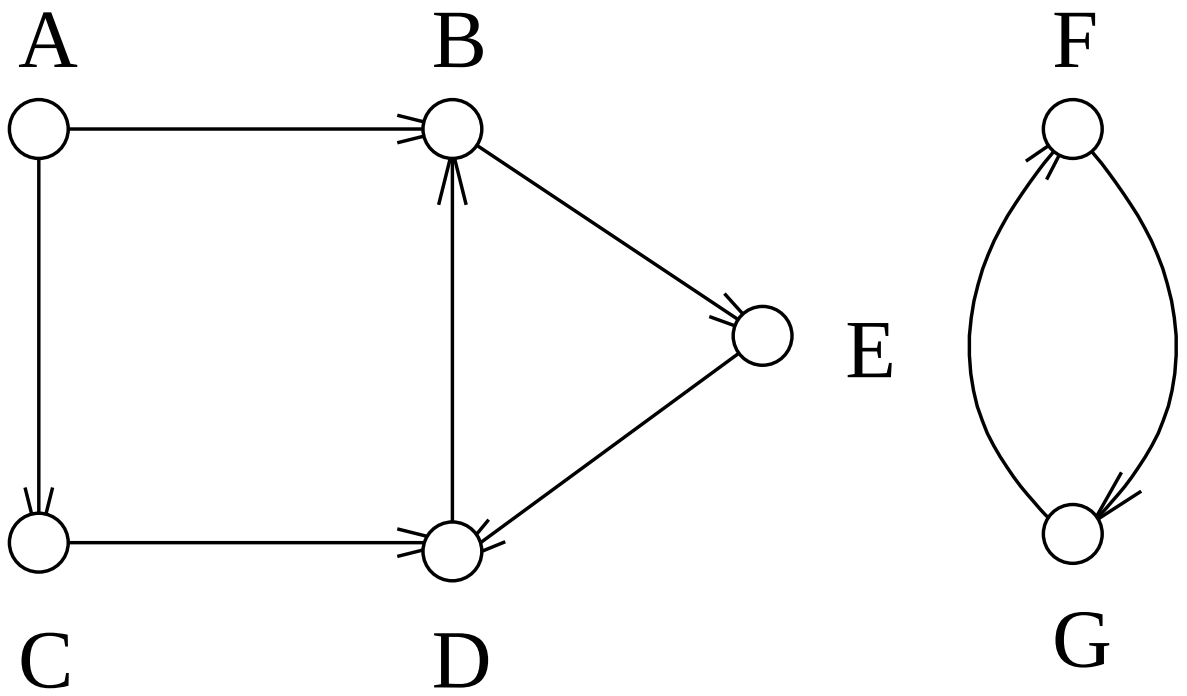
Graphes orientés

Définition (Graphe orienté)

un graphe orienté $G = (S, A)$ est défini par :

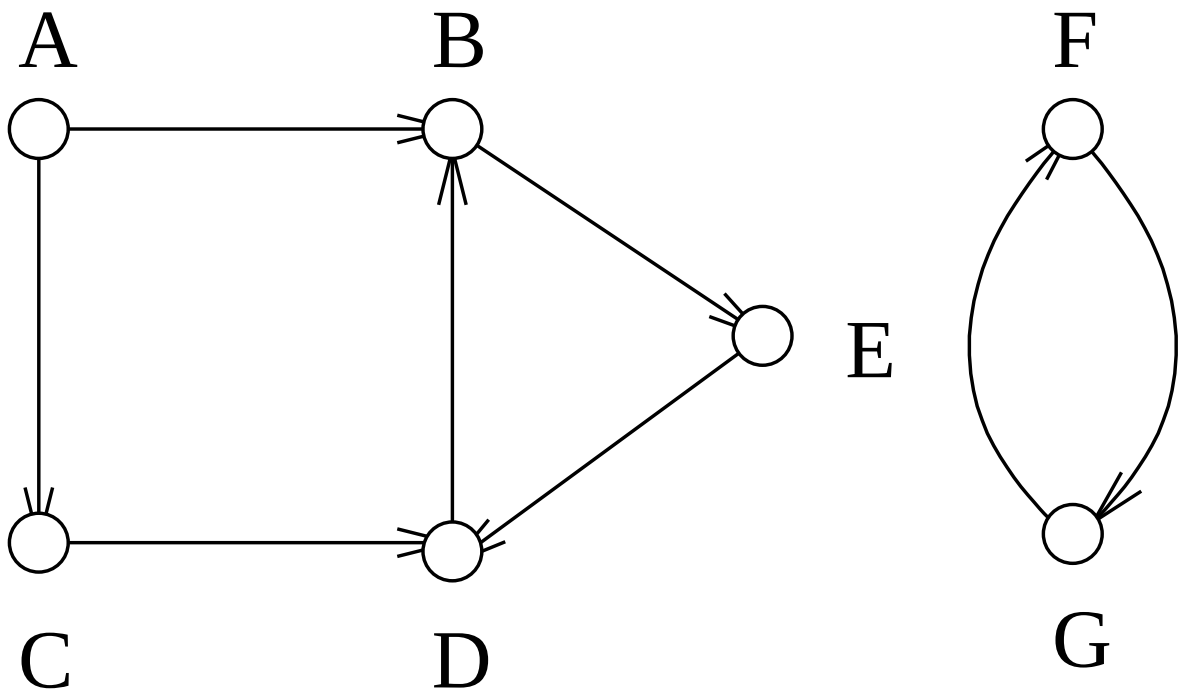
- *un ensemble S fini de sommets (vertices, un sommet : a vertex)*
- *un ensemble A d'arcs, chaque arc ayant une origine et un but dans S.*
- *Un arc $a = (s, t)$: t est un successeur de s, s un prédécesseur de t.*
- *Un arc $s \to s$ est une boucle. On appelle $|S|$ l'ordre du graphe.*

Un graphe orienté



Un graphe orienté, G_1

Un graphe orienté



Un graphe orienté, G_1

N.B. en anglais :

- directed graph/digraph $\equiv$ graphe orienté
- (directed) edge/arc $\equiv$ arc

Graphes simples, degrés

Définition (Graphe simple)

- *Un graphe est dit simple s'il existe au plus un arc (arête) d'origine et de buts donnés et s'il est sans boucle.*

Définition (degrés)

Graphes simples, degrés

Définition (Graphe simple)

- *Un graphe est dit simple s'il existe au plus un arc (arête) d'origine et de buts donnés et s'il est sans boucle.*
- *A définit dans ce cas une relation binaire sur S (une partie de $S \times S$).*

Définition (degrés)

Graphes simples, degrés

Définition (Graphe simple)

- *Un graphe est dit simple s'il existe au plus un arc (arête) d'origine et de buts donnés et s'il est sans boucle.*
- *A définit dans ce cas une relation binaire sur S (une partie de $S \times S$).*

Définition (degrés)

- *Dans un graphe orienté, le degré sortant (resp. entrant) d'un sommet s est le nombre de sommets qui sont successeurs (resp. prédécesseurs) de s. Le degré est la somme des degrés rentrants et sortants.*

Graphes simples, degrés

Définition (Graphe simple)

- *Un graphe est dit simple s'il existe au plus un arc (arête) d'origine et de buts donnés et s'il est sans boucle.*
- *A définit dans ce cas une relation binaire sur S (une partie de $S \times S$).*

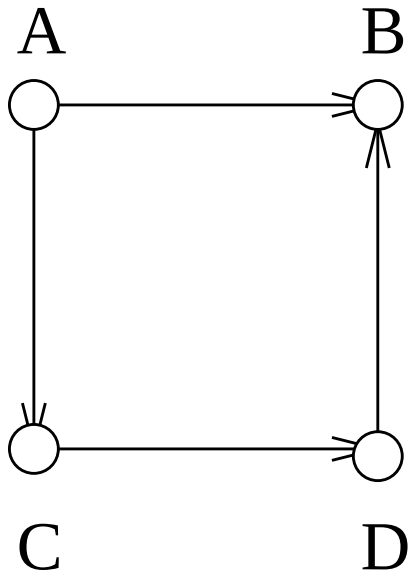
Définition (degrés)

- *Dans un graphe orienté, le degré sortant (resp. entrant) d'un sommet s est le nombre de sommets qui sont successeurs (resp. prédécesseurs) de s. Le degré est la somme des degrés rentrants et sortants.*
- *Dans un graphe non orienté, le degré de s est le nombre de sommets adjacents à s*

Sous-graphe

Définition (Sous-graphe)

Soit un graphe $G = (S, A)$, et $S' \subset S$, le sous-graphe induit par S' est le graphe (S', A') où $A' = \{(u, v) \in A / u \in S' \wedge v \in S'\}$.



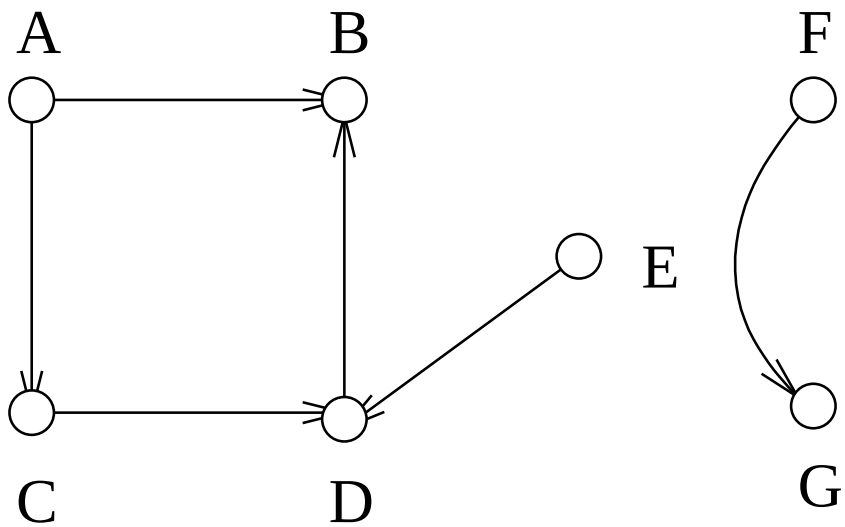
Le sous-graphe de G_1 induit par $\{A, B, C, D\}$

Isomorphisme de sous-graphe : NP-difficile.

Graphe partiel

Définition (Graphe partiel)

Soit un graphe $G = (S, A)$, et $A' \subset A$, le graphe (S, A') est un graphe partiel de G.

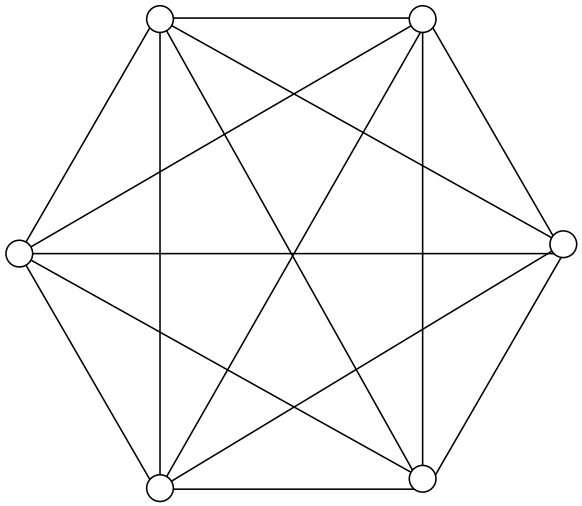


Un graphe partiel de G_1

Graphe complet, clique

Définition (Graphe complet)

- *Un graphe est complet si pour toute paire de sommets (u, v) il existe au moins un arc (arête) entre u et v.*

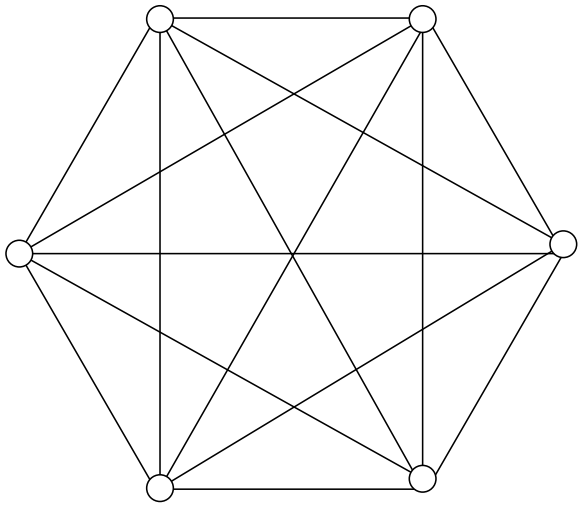


Le graphe simple complet non orienté K_6

Graphe complet, clique

Définition (Graphe complet)

- *Un graphe est complet si pour toute paire de sommets (u, v) il existe au moins un arc (arête) entre u et v.*
- *Si le sous-graphe induit par $S' \subset S$ est complet, S' forme une clique. Un graphe simple complet d'ordre n est noté K_n.*



Le graphe simple complet non orienté K_6

Chaînes et cycles

Définition (Chaînes,cycles)

Soit un graphe non orienté $G = (S, A)$, une chaîne de longueur k entre un sommet u et un sommet u' appelé but est une séquence $\langle v_0, \ldots, v_k \rangle$ telle que $v_0 = u, v_k = u'$ et

$$\forall i, 0 \leq i \leq k-1, (v_i, v_{i+1}) \in A$$

- *Simple : ne passe pas deux fois par la même arête.*

Chaînes et cycles

Définition (Chaînes,cycles)

Soit un graphe non orienté $G = (S, A)$, une chaîne de longueur k entre un sommet u et un sommet u' appelé but est une séquence $\langle v_0, \ldots, v_k \rangle$ telle que $v_0 = u, v_k = u'$ et

$$\forall i, 0 \leq i \leq k-1, (v_i, v_{i+1}) \in A$$

- *Simple : ne passe pas deux fois par la même arête.*
- *Élémentaire : ne passe pas deux fois par le même sommet.*

Chaînes et cycles

Définition (Chaînes,cycles)

Soit un graphe non orienté $G = (S, A)$, une chaîne de longueur k entre un sommet u et un sommet u' appelé but est une séquence $\langle v_0, \ldots, v_k \rangle$ telle que $v_0 = u, v_k = u'$ et

$$\forall i, 0 \leq i \leq k-1, (v_i, v_{i+1}) \in A$$

- *Simple : ne passe pas deux fois par la même arête.*
- *Élémentaire : ne passe pas deux fois par le même sommet.*
- *Une chaîne dont les deux extrémités sont confondues est appelé un cycle.*

Chemin, circuit

Définition (Chemins,circuits)

Soit un graphe orienté $G = (S, A)$, un chemin de longueur k d'un sommet u appelé extrémité initiale vers un sommet u' appelé extrémité finale/terminale est une séquence $\langle v_0, \ldots, v_k \rangle$ telle que $v_0 = u, v_k = u'$ et $\forall i, 0 \leq i \leq k-1, v_i \rightarrow v_{i+1} \in A$.

- *Simple : ne passe pas deux fois par le même arc.*

Circuit : chemin qui boucle. Détails sur élémentaire.

Chemin, circuit

Définition (Chemins,circuits)

Soit un graphe orienté $G = (S, A)$, un chemin de longueur k d'un sommet u appelé extrémité initiale vers un sommet u' appelé extrémité finale/terminale est une séquence $\langle v_0, \ldots, v_k \rangle$ telle que $v_0 = u, v_k = u'$ et $\forall i, 0 \leq i \leq k-1, v_i \rightarrow v_{i+1} \in A$.

- *Simple : ne passe pas deux fois par le même arc.*
- *Élémentaire : ne passe pas deux fois par le même sommet.*

Circuit : chemin qui boucle. Détails sur élémentaire.

Graphe connexe

Définition (Connexité)

- *Un graphe (non orienté) $G = (S, A)$ est connexe s'il existe une chaîne entre chaque paire de sommets $s, t \in S$.*

Graphe connexe

Définition (Connexité)

- *Un graphe (non orienté) $G = (S, A)$ est connexe s'il existe une chaîne entre chaque paire de sommets $s, t \in S$.*
- *La composante connexe d'un sommet $s \in S$ est le sous graphe induit par tous les sommets reliés à s par une chaîne au moins.*

Graphe connexe

Définition (Connexité)

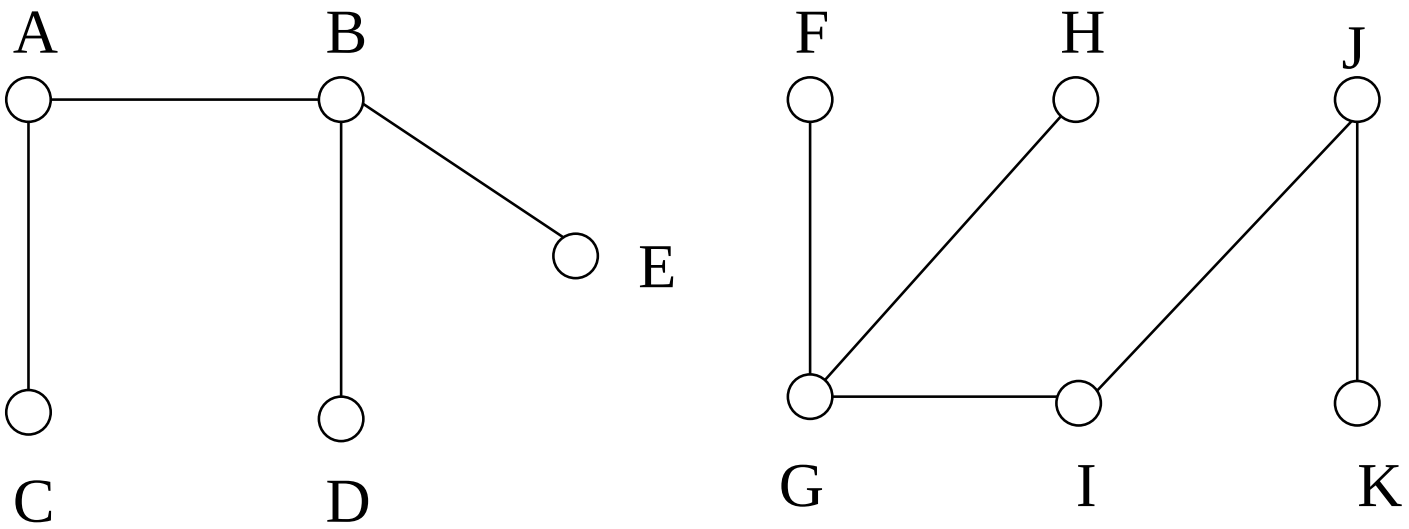
- *Un graphe (non orienté) $G = (S, A)$ est connexe s'il existe une chaîne entre chaque paire de sommets $s, t \in S$.*
- *La composante connexe d'un sommet $s \in S$ est le sous graphe induit par tous les sommets reliés à s par une chaîne au moins.*
- *Classe d'équivalence de la relation d'existence d'une chaîne.*

Arbre et Forêts

Définition (Arbre)

Un graphe non orienté et sans cycle simple (acyclique) est aussi appelé une forêt.

Chaque composante connexe est appelé un arbre.



Arbres. . .

Soit $G = (S, A)$ un graphe de $n \geq 2$ sommets.

- Il est connexe et a $n - 1$ arêtes.
- Il est connexe et sans cycle simple.
- Il sans cycle simple et a $n - 1$ arêtes.
- Il est sans cycle simple et l'ajout d'une arête crée un cycle simple.
- Il existe une et une seule chaîne simple entre chaque paire de sommets.
- Il est connexe et la perte d'une arête quelconque le rend non connexe

Graphe fortement connexe

Définition

- *Un graphe orienté $G = (S, A)$ est fortement connexe si pour tous sommets $s, t \in S$, il existe un chemin allant de s à t.*

Graphe fortement connexe

Définition

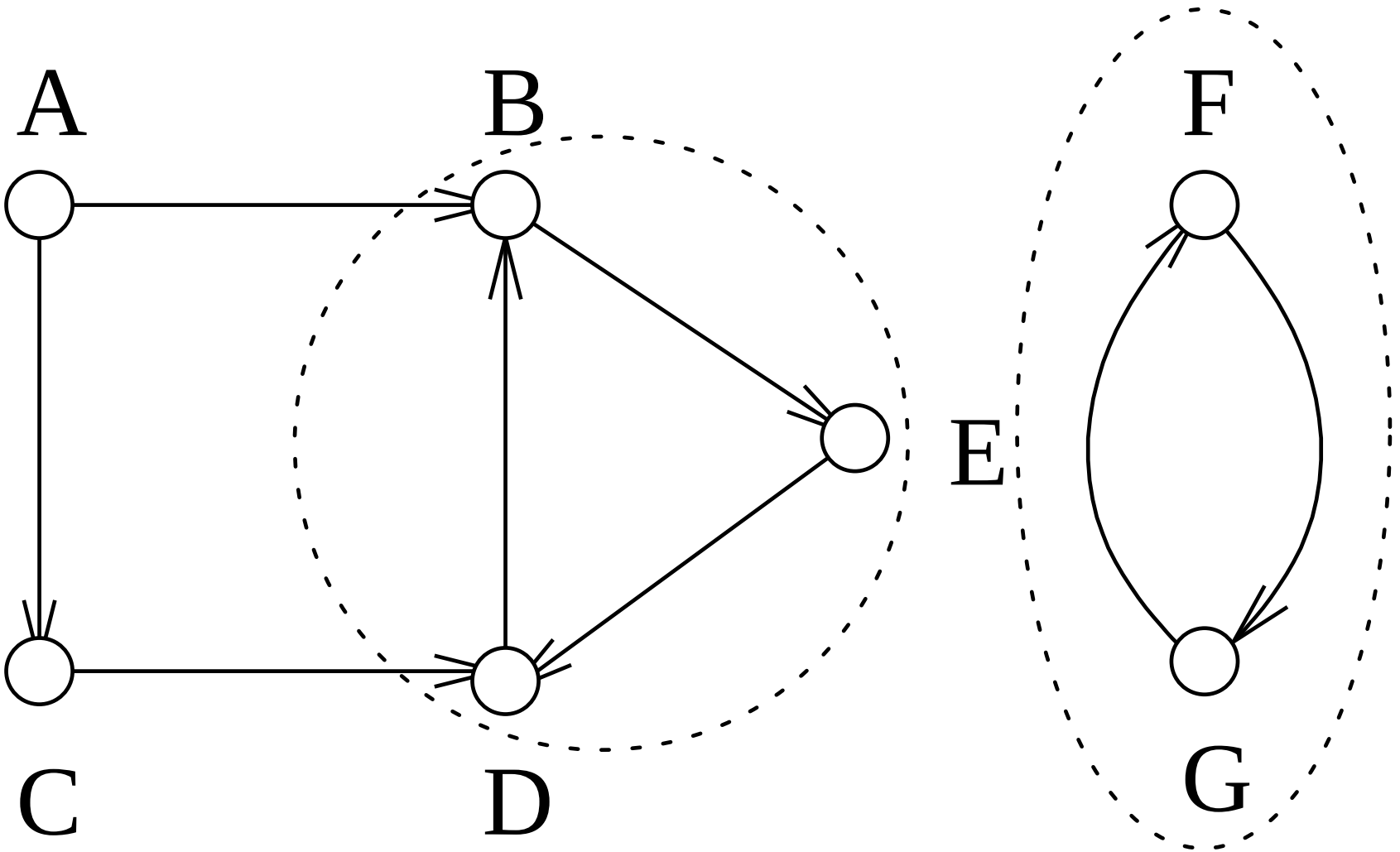
- *Un graphe orienté $G = (S, A)$ est fortement connexe si pour tous sommets $s, t \in S$, il existe un chemin allant de s à t.*
- *La composante fortement connexe d'un sommet $s \in S$ est le sous-graphe induit par l'ensemble des sommets $t \in S$ tels qu'il existe une chemin de s à t et de t à s.*

Graphe fortement connexe

Définition

- *Un graphe orienté $G = (S, A)$ est fortement connexe si pour tous sommets $s, t \in S$, il existe un chemin allant de s à t.*
- *La composante fortement connexe d'un sommet $s \in S$ est le sous-graphe induit par l'ensemble des sommets $t \in S$ tels qu'il existe une chemin de s à t et de t à s.*
- *Classe d'équivalence de la relation d'accessibilité mutuelle.*

Composantes fortement connexes



Les composantes fortement connexes de G_1

Des racines et des arbres. . .

Définition (Arborescence)

- *Un arbre enraciné ou arborescence est un graphe orienté possédant un sommet privilégié appelé racine et tel qu'il existe un chemin et un seul de la racine à tout autre sommet.*

Des racines et des arbres. . .

Définition (Arborescence)

- *Un arbre enraciné ou arborescence est un graphe orienté possédant un sommet privilégié appelé racine et tel qu'il existe un chemin et un seul de la racine à tout autre sommet.*
- *À partir d'un arbre : choisir une racine. Vocabulaire : nœuds, fils, descendants, ascendants. . .*

Des racines et des arbres. . .

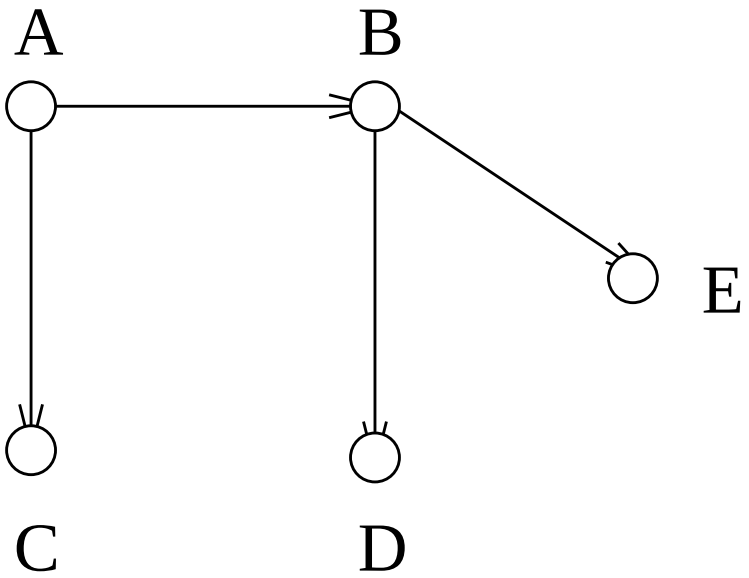
Définition (Arborescence)

- *Un arbre enraciné ou arborescence est un graphe orienté possédant un sommet privilégié appelé racine et tel qu'il existe un chemin et un seul de la racine à tout autre sommet.*
- *À partir d'un arbre : choisir une racine. Vocabulaire : nœuds, fils, descendants, ascendants. . .*

Définition (Arborescence ordonnée)

Une arborescence est ordonnée si les ensemble des fils de chaque sommet sont ordonnés.

Une arborescence



Arborescence obtenue à partir de la composante connexe $\{A, B, C, D, E\}$ de la forêt précédente, en isolant A comme racine

En machine : matrice d'adjacence

Définition (Matrice d'adjacence)

La matrice d'adjacence $\mathcal{A}$ d'un graphe $G = (S, A)$ d'ordre n est une matrice $n \times n$ à coefficients dans $\{0, 1\}$ où chaque ligne et chaque colonne correspond à un sommet de G et où :

$$\mathcal{A}_{ij} = 1 \Leftrightarrow (i, j) \in A$$

Dans le cas non orienté, la matrice d'adjacence est symétrique.

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

La matrice d'adjacence du sous-graphe de G_1 induit par $\{A, B, C, D, E\}$

Graphes. . .

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

La matrice d'adjacence du sous-graphe de G_1 induit par $\{A, B, C, D, E\}$

Accès en temps constant à l'existence d'arc/arête entre 2 sommets mais parcours des arêtes en $O(n^2)$.

Une alternative plus usuelle

Définition (Liste d'adjacence)

- *un tableau Adj de $|S|$ listes, une pour chaque sommet.*

Une alternative plus usuelle

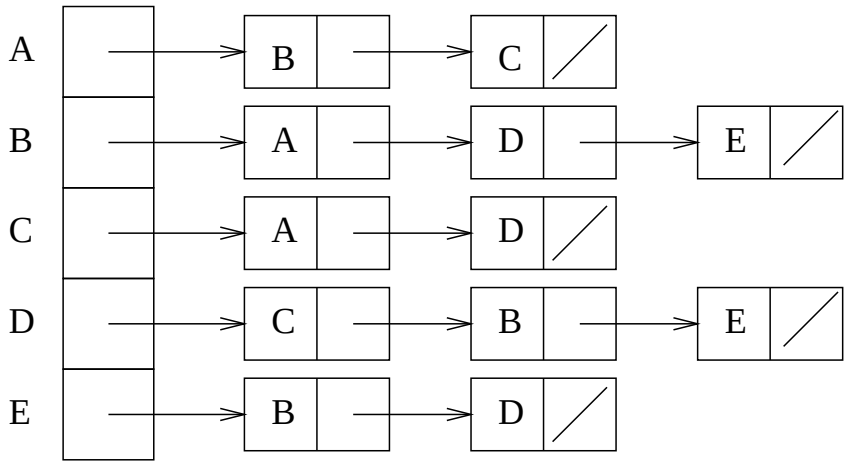
Définition (Liste d'adjacence)

- *un tableau Adj de $|S|$ listes, une pour chaque sommet.*
- *pour chaque sommet u, la liste $Adj[u]$ contient les (ou des références aux) sommets v adjacents à u (tels qu'il existe un arc/arête $(u, v) \in A$).*

Une alternative plus usuelle

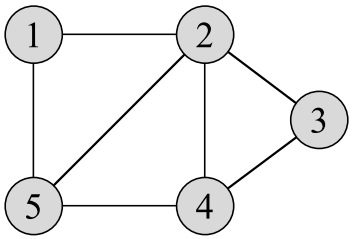
Définition (Liste d'adjacence)

- *un tableau Adj de $|S|$ listes, une pour chaque sommet.*
- *pour chaque sommet u, la liste $Adj[u]$ contient les (ou des références aux) sommets v adjacents à u (tels qu'il existe un arc/arête $(u, v) \in A$).*

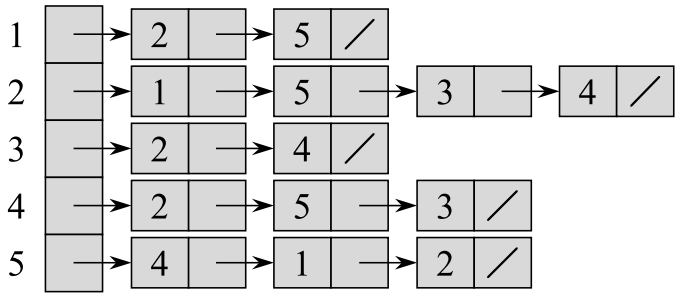


Listes d'adjacence de G_2

Exemples



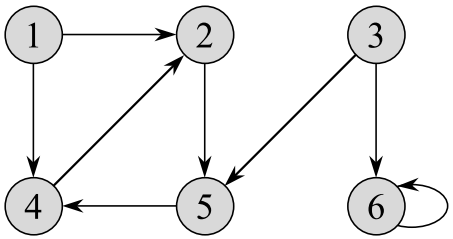
(a)



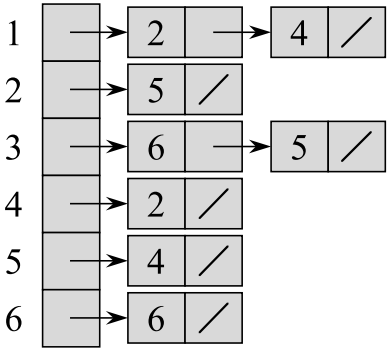
(b)

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

(c)



(a)



(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

Matrices et graphes

Produit de deux matrices $n \times n$ naïf est en $O(n^3)$. Pas d'améliorations importantes (eg. algorithme de Sollen en $O(n^{\log_2(7)})$).

Pour manipuler les matrices, on utilise l'algèbre de Boole (multiplication $= \wedge$, addition $= \vee$).

Définition (Fermeture transitive)

La fermeture transitive d'une graphe (simple) $G = (S, A)$ est le graphe $G' = (S, A')$ tel que $(i, j) \in A' \Leftrightarrow$ il existe un chemin/chaîne de i à j.

Matrices et Chemins

M matrice d'adjacence de G. M^p est telle que $M^p_{ij} = 1 \Leftrightarrow \exists$ un chemin de p arcs allant de i à j.

Entrées : M, matrice, p puissance;
si $p = 1$ **alors**
  retourner M;
sinon
  $J \leftarrow M^{\lfloor \frac{p}{2} \rfloor}$;
  $K \leftarrow J \times J$;
  si p *est pair* **alors**
    retourner K;
  sinon
    retourner $K \times M$;

Algorithme de Warshall

S'il existe un chemin entre deux sommets de G, il existe un chemin élémentaire en ces sommets, de longueur $< n$.

La matrice de la fermeture transitive du graphe est la somme (le ou logique) $M + M^2 + \cdots + M^{|S|-1}$.

Entrée : M, matrice d'adjacence;
pour *i allant de* 1 *à* n **faire**
  pour *j allant de* 1 *à* n **faire**
    si $M_{ji} = 1$ **alors**
      pour *k allant de* 1 *à* n **faire**
        si $M_{ik} = 1$ **alors** $M_{jk} \leftarrow 1$

Correction

Récurrence, sur la boucle extérieure

À la $i$ème itération, pour toute paire de sommets j, k, on a $M_{jk} = 1$ ssi il existe un chemin de j à k qui utilise uniquement des sommets (en sus) de $\{1, \ldots, i\}$.

1. Vrai au début de l'algorithme car M est la matrice d'incidence.

Correction

Récurrence, sur la boucle extérieure

À la $i$ème itération, pour toute paire de sommets j, k, on a $M_{jk} = 1$ ssi il existe un chemin de j à k qui utilise uniquement des sommets (en sus) de $\{1, \ldots, i\}$.

1. Vrai au début de l'algorithme car M est la matrice d'incidence.
2. à l'itération i, il existe un chemin de j à k qui passe par des sommets de $\{1, \ldots, i\}$ ssi il existe un chemin *élémentaire* de j à k qui passe par des sommets de $\{1, \ldots, i\}$.

Correction

Récurrence, sur la boucle extérieure

À la $i$ème itération, pour toute paire de sommets j, k, on a $M_{jk} = 1$ ssi il existe un chemin de j à k qui utilise uniquement des sommets (en sus) de $\{1, \ldots, i\}$.

1. Vrai au début de l'algorithme car M est la matrice d'incidence.
2. à l'itération i, il existe un chemin de j à k qui passe par des sommets de $\{1, \ldots, i\}$ ssi il existe un chemin *élémentaire* de j à k qui passe par des sommets de $\{1, \ldots, i\}$.
3. il se décompose en 2 parties de j à i et de i à k.

Correction

Récurrence, sur la boucle extérieure

À la $i$ème itération, pour toute paire de sommets j, k, on a $M_{jk} = 1$ ssi il existe un chemin de j à k qui utilise uniquement des sommets (en sus) de $\{1, \ldots, i\}$.

1. Vrai au début de l'algorithme car M est la matrice d'incidence.
2. à l'itération i, il existe un chemin de j à k qui passe par des sommets de $\{1, \ldots, i\}$ ssi il existe un chemin *élémentaire* de j à k qui passe par des sommets de $\{1, \ldots, i\}$.
3. il se décompose en 2 parties de j à i et de i à k.
4. chaque partie utilise des sommets de $\{1, \ldots, i-1\}$. On applique l'hypothèse de récurrence.

Correction

Récurrence, sur la boucle extérieure

À la $i$ème itération, pour toute paire de sommets j, k, on a $M_{jk} = 1$ ssi il existe un chemin de j à k qui utilise uniquement des sommets (en sus) de $\{1, \ldots, i\}$.

1. Vrai au début de l'algorithme car M est la matrice d'incidence.
2. à l'itération i, il existe un chemin de j à k qui passe par des sommets de $\{1, \ldots, i\}$ ssi il existe un chemin *élémentaire* de j à k qui passe par des sommets de $\{1, \ldots, i\}$.
3. il se décompose en 2 parties de j à i et de i à k.
4. chaque partie utilise des sommets de $\{1, \ldots, i-1\}$. On applique l'hypothèse de récurrence.
5. l'algorithme fixe M_{jk} à 1 ssi ces deux conditions sont vérifiées.

La version graphe

Entrée : G, un graphe;
pour chaque *sommet* s *de* G **faire**
    pour chaque *arc* $s_1 \to s$ *de* G **faire**
        pour chaque *arc* $s \to s_2$ *de* G **faire**
            Ajouter à G l'arc $s_1 \to s_2$;

En utilisant une autre algèbre que $(\wedge, \vee)$, l'algorithme peut calculer d'autres propriétés. Avec $(\min, +)$, il calcule la distance minimum entre toute paire de sommets (plus courts chemins).

Parcourir les graphes : arborescences

- Visiter chaque sommet, chaque arc/arête UNE fois.

Parcourir les graphes : arborescences

- Visiter chaque sommet, chaque arc/arête UNE fois.
- **Profondeur d'abord** : pour visiter un nœud que l'on découvre, il faut d'abord visiter tous ses fils. Il est alors fermé.

Parcourir les graphes : arborescences

- Visiter chaque sommet, chaque arc/arête UNE fois.
- **Profondeur d'abord** : pour visiter un nœud que l'on découvre, il faut d'abord visiter tous ses fils. Il est alors fermé.
- Structures : $d[u], f[u]$ date de découverte/fermeture.

Parcourir les graphes : arborescences

- Visiter chaque sommet, chaque arc/arête UNE fois.
- **Profondeur d'abord** : pour visiter un nœud que l'on découvre, il faut d'abord visiter tous ses fils. Il est alors fermé.
- Structures : $d[u], f[u]$ date de découverte/fermeture.

---

u racine, t variable globale initialisée à 0;

Procedure *DFS-Tree(u)*

```
    d[u] ← (t ← (t + 1));
    pour chaque sommet v, fils de u faire
        DFS-Tree(v);
    f[u] ← (t ← (t + 1));
```

Appliqué aux graphes

Arborescence DFS : à la découverte d'un sommet s, les fils de ce sommet dans l'arborescence sont les sommets accessibles depuis s et non encore visités.

Appliqué aux graphes

Arborescence DFS : à la découverte d'un sommet s, les fils de ce sommet dans l'arborescence sont les sommets accessibles depuis s et non encore visités.

Appel sur le graphe $G = (S, A)$ (orienté ou non);

Procedure *DFS(G)*

- **pour chaque** *sommet* $u \in S$ **faire**
  - $couleur[u] \leftarrow b$;
  - $\pi[u] \leftarrow \text{NIL}$;
- $t \leftarrow 0$;
- **pour chaque** *sommet* $u \in S$ **faire**
  - **si** $couleur[u] = b$ **alors** DFS-Visit(u)

Appliqué aux graphes

Arborescence DFS : à la découverte d'un sommet s, les fils de ce sommet dans l'arborescence sont les sommets accessibles depuis s et non encore visités.

Appel sur le graphe $G = (S, A)$ (orienté ou non);

Procedure *DFS(G)*

```
pour chaque sommet u ∈ S faire
    couleur[u] ← b;
    π[u] ← NIL;
t ← 0;
pour chaque sommet u ∈ S faire
    si couleur[u] = b alors DFS-Visit(u)
```

N.B. en anglais :

- DFS ≡ Depth First Search

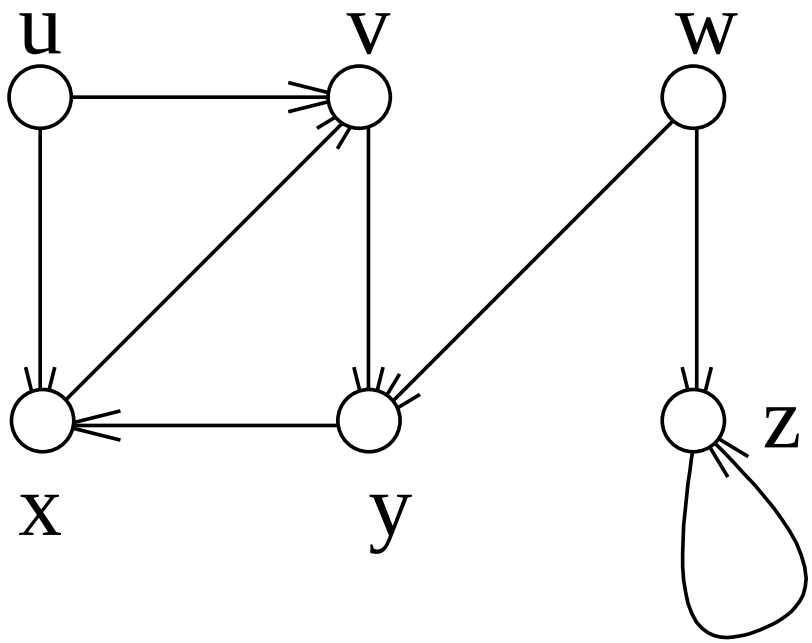
Algorithme de parcours

Procedure *DFS-Visit(u)*

```
couleur[u] ← g;
d[u] ← (t ← (t + 1));
pour chaque sommet v ∈ Adj[u] faire
    si couleur[v] = b alors
        π[v] ← u;
        DFS-Visit(v);
couleur[u] ← n;
f[u] ← (t ← (t + 1));
```

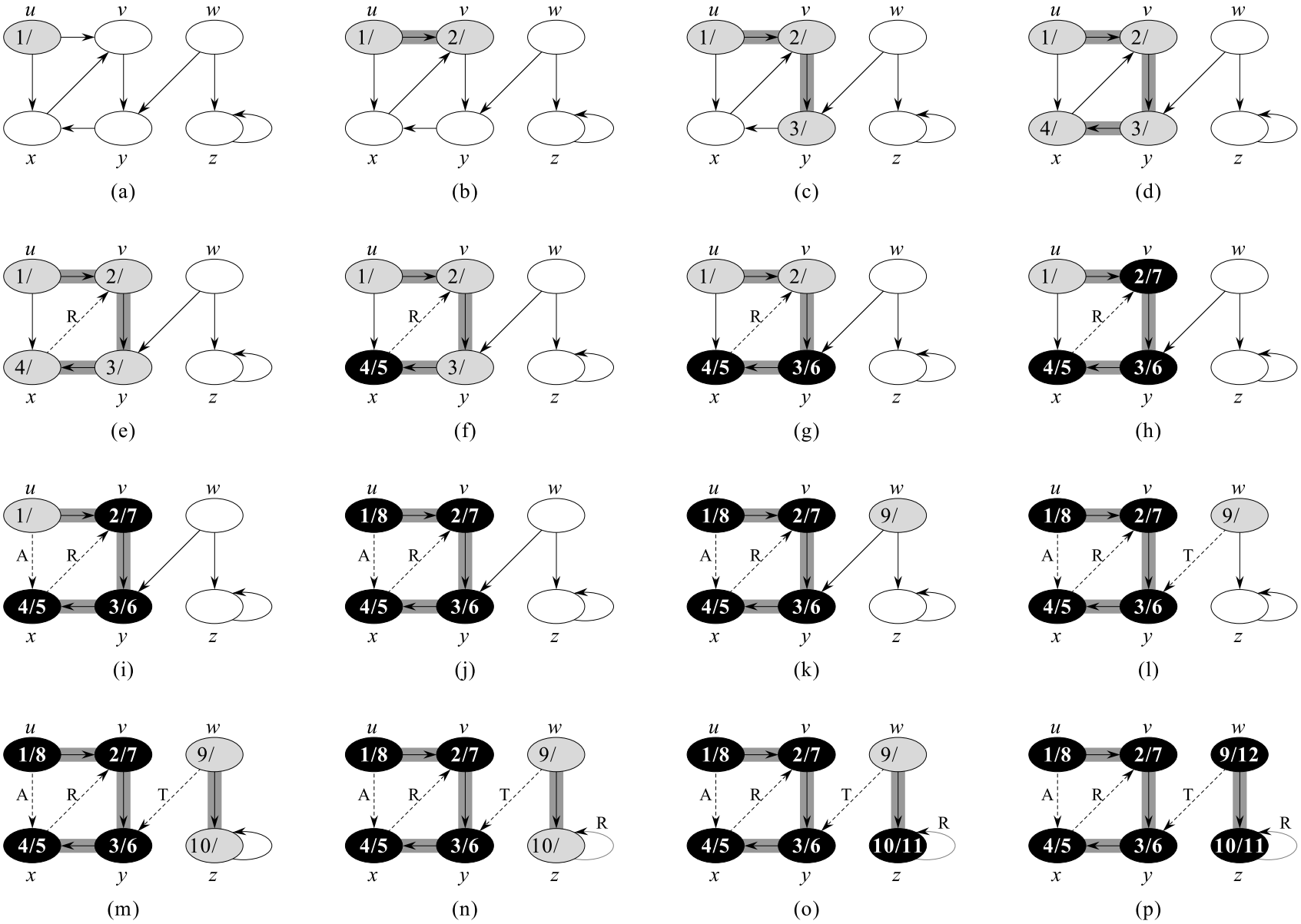
Initialisation $\theta(|S|)$,DFS-Visit une fois par sommet : globalement en $\theta(|S| + |A|)$

Exemple



Parcourir ce graphe en profondeur d'abord

Exemple


u v w
1/ x y z
(a)
u v w
1/ 2/ x y z
(b)
u v w
1/ 2/ 3/ x y z
(c)
u v w
1/ 2/ 4/ 3/ x y z
(d)
u v w
1/ 2/ R 4/ 3/ x y z
(e)
u v w
1/ 2/ R 4/5 3/ x y z
(f)
u v w
1/ 2/ R 4/5 3/6 x y z
(g)
u v w
1/ 2/7 R 4/5 3/6 x y z
(h)
u v w
1/ 2/7 A R 4/5 3/6 x y z
(i)
u v w
1/8 2/7 A R 4/5 3/6 x y z
(j)
u v w
1/8 2/7 9/ A R 4/5 3/6 x y z
(k)
u v w
1/8 2/7 9/ A R T 4/5 3/6 x y z
(l)
u v w
1/8 2/7 9/ A R T 4/5 3/6 10/ x y z
(m)
u v w
1/8 2/7 9/ A R T 4/5 3/6 R 10/ x y z
(n)
u v w
1/8 2/7 9/ A R T 4/5 3/6 R 10/11 x y z
(o)
u v w
1/8 2/7 9/12 A R T 4/5 3/6 R 10/11 x y z
(p)

Les arcs

Et les couleurs du sommet v à sa première rencontre.

- **arcs d'arbre (blanc)** : ceux qui sont suivis par DFS. Un arc $u \to v$ est un arc d'arbre si v a été découvert à partir de u.
- **arcs arrières (gris)** : ce sont les arcs qui connectent un sommet u à un ancêtre dans l'arborescence de profondeur d'abord.
- **arcs avants (noir)** : ce sont les arcs qui connecte un sommet u à un descendant de u mais qui n'ont pas été suivies par DFS.
- **arcs traversiers (noir)** : tous les autres arcs.

Détection de circuits : existence d'arcs arrière (gris) dans un parcours DFS.

Exemple : le tri topologique

Un tri topologique d'un Graphe Orienté Sans Circuit (GOSC) G consiste à déterminer un ordre des sommets tel que si un arc $u \rightarrow v$ existe dans G, alors v est placé après u dans l'ordre des sommets.

Procedure *Tri-Topologique(G)*

Appeler DFS(G) pour calculer les $f[u]$;

À chaque fois qu'un sommet est fermé, l'insérer en tête d'une liste L;

retourner L;

Exemple : le tri topologique

Un tri topologique d'un Graphe Orienté Sans Circuit (GOSC) G consiste à déterminer un ordre des sommets tel que si un arc $u \rightarrow v$ existe dans G, alors v est placé après u dans l'ordre des sommets.

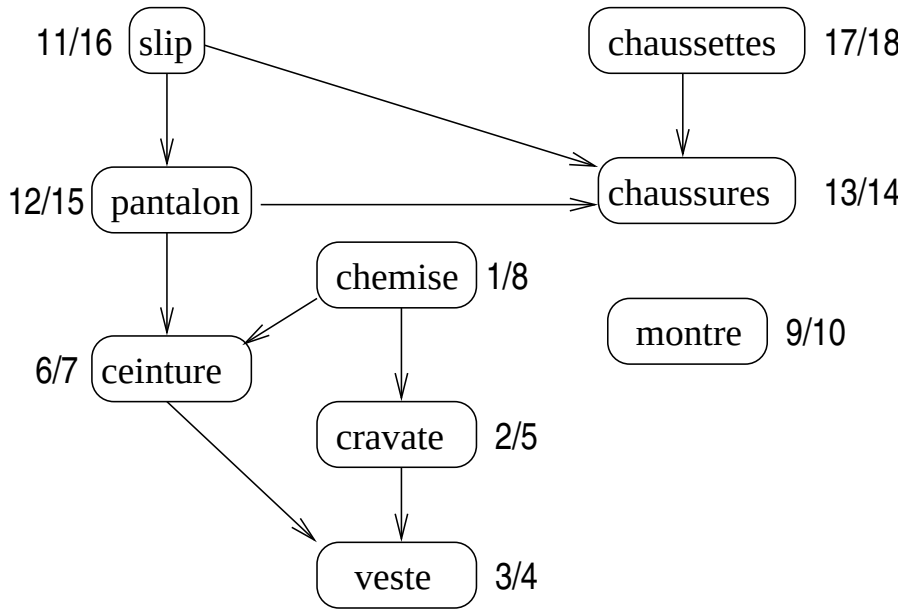
Procedure *Tri-Topologique(G)*

Appeler DFS(G) pour calculer les $f[u]$;
À chaque fois qu'un sommet est fermé, l'insérer en tête d'une liste L;
retourner L;

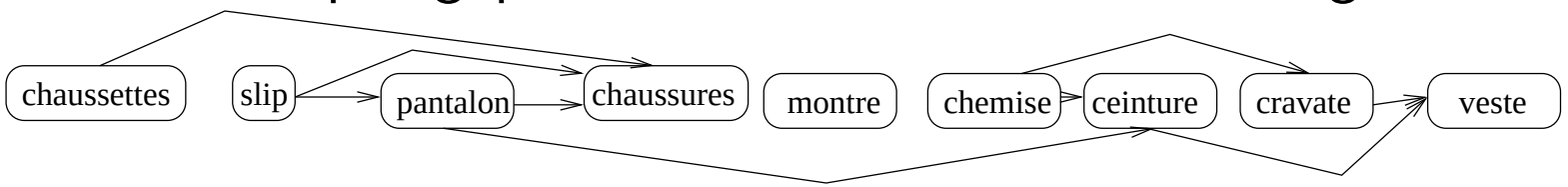
N.B. en anglais :

- Graphe Orienté Sans Circuit (GOSC) $\equiv$ Directed Acyclic Graph (DAG).

Exemple : habillage du Pr. Cosinus



Tri topologique du Pr. Cosinus avant l'habillage



Largeur d'abord

On découvre les sommets par distance croissante avec la "racine".
Utilise une file (FIFO)

Procedure *BFS*(G, s)

pour chaque *sommet* $u \in S - \{s\}$ **faire**
- $couleur[u] \leftarrow b$;
- $d[u] \leftarrow +\infty$;
- $\pi[u] \leftarrow \text{NIL}$;

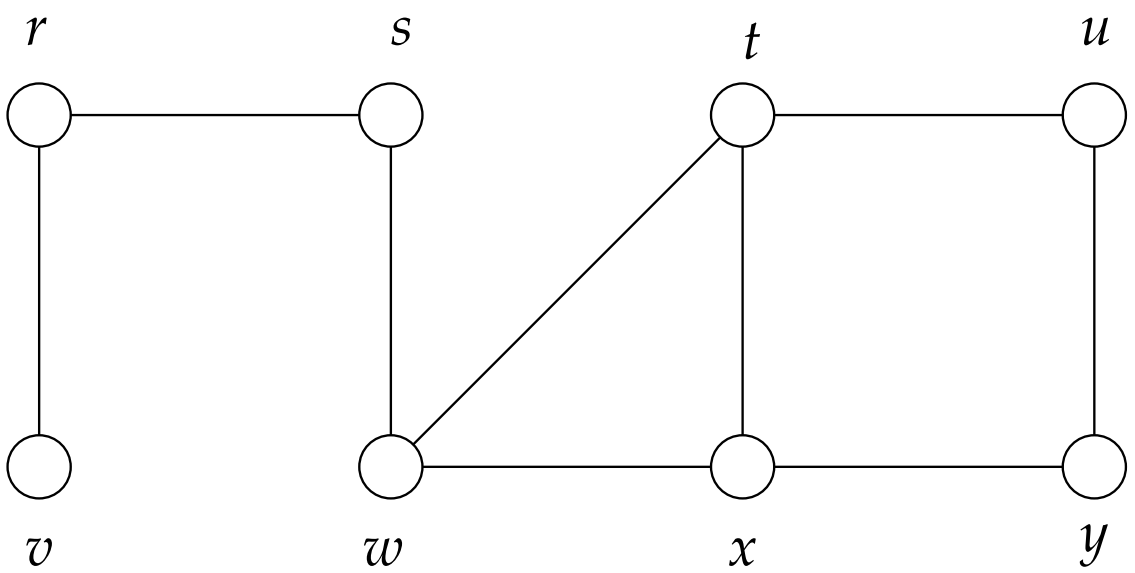
$couleur[s] \leftarrow g$;
$d[s] \leftarrow 0$;
$\pi[s] \leftarrow \text{NIL}$;
$Q \leftarrow \{s\}$;

La suite

tant que $Q \neq \varnothing$ **faire**
    $u \leftarrow tete(Q)$;
    pour chaque *sommet* $v \in Adj[u]$ **faire**
        si $couleur[v] = b$ **alors**
            $\pi[v] \leftarrow u$;
            $couleur[v] \leftarrow g$;
            $d[v] \leftarrow d[u] + 1$;
            Enfiler(Q, v);
    Défiler(Q);
    $couleur[u] \leftarrow n$;

Exemple

L'algorithme est globalement en $O(|S| + |A|)$.

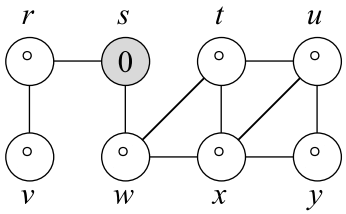


Un graphe à parcourir en largeur

Plus courts chemins depuis s identifiés (une arête $= 1$).

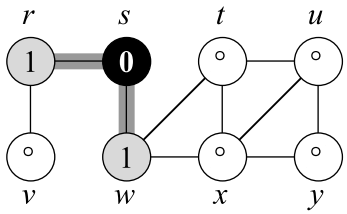
Exemple

(a)



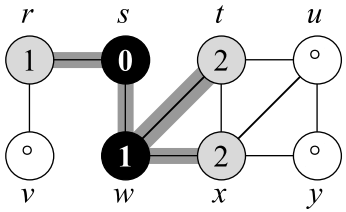
F	s
	0

(b)



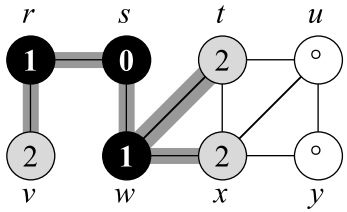
F	w	r
	1	1

(c)



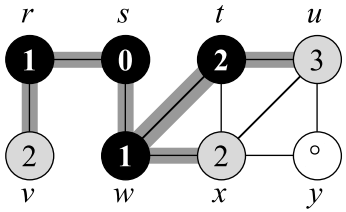
F	r	t	x
	1	2	2

(d)



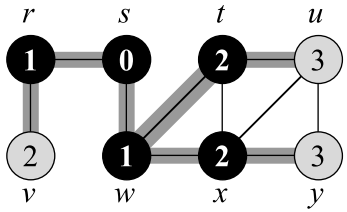
F	t	x	v
	2	2	2

(e)



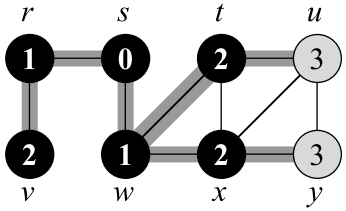
F	x	v	u
	2	2	3

(f)



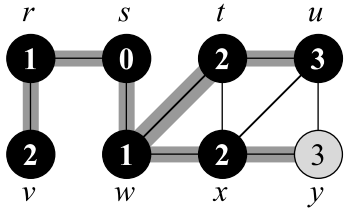
F	v	u	y
	2	3	3

(g)



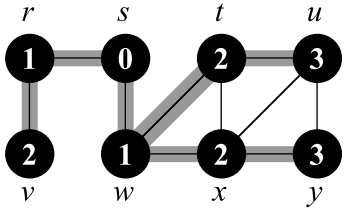
F	u	y
	3	3

(h)



F	y
	3

(i)



F Ze

Minimum Spanning Tree : MST

Définition (Graphe pondéré)

Graphe pondéré *: pour tout arc ou arête* (u, v)*, on note* $w(u, v)$ *le poids qui lui est associé. Il s'agit en général d'un entier.*

Problème modélisé : connecter plusieurs ≪ composants ≫ ensemble en utilisant le moins de longueur de fils possible.

Un sommet : un composant
Le poids d'une arête : longueur de fil.

Définition (Arbre couvrant)

Étant donné un graphe $G = (S, A)$*, un arbre couvrant de* G *est un graphe partiel de* G *qui définit un arbre et qui inclut tous ses sommets.*

Connecter et minimum de fil : sans cycle simple, donc un arbre.
Poids minimum.

Un algorithme glouton

Définition (Algorithme glouton)

Algorithme qui construit un objet en faisant à chaque étape le choix qui rapporte le plus instantanément.

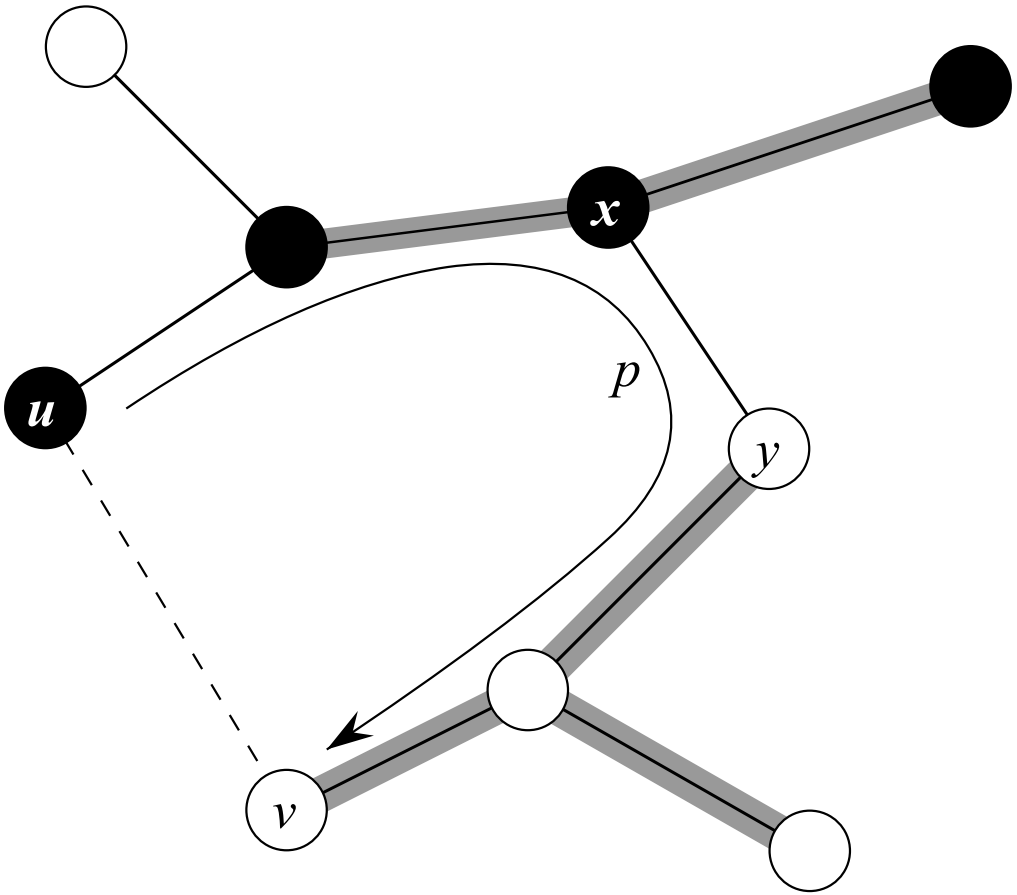
Définition

Soit un ensemble de sommets $T \subset S$ quelconque. On appelle cocycle de T, et on note $\omega(T)$, l'ensemble des arêtes de A dont une des extrémités est dans T et l'autre dans $S - T$.

Pour tout $T \subset S$, soit e une arête de poids minimum de $\omega(T)$. Alors $e \in$ à un MST de G.

Glouton = correct (ici).

Pourquoi la gloutonnerie marche, ici



L'algorithme de Prim

Procedure *MST-Prim*(G, w, r)

$Q \leftarrow$ ensemble des sommets du graphe;
pour chaque $u \in Q$ **faire** $key[u] \leftarrow +\infty$;
$key[r] \leftarrow 0$; $\pi[r] = \text{NIL}$;
tant que $Q \neq \varnothing$ **faire**
  $u \leftarrow$ Extrait-Minimum(Q);
  pour chaque $v \in Adj[u]$ **faire**
    si $v \in Q$ *et* $w(u, v) < key[v]$ **alors**
      $\pi[v] \leftarrow u$;
      $key[v] \leftarrow w(u, v)$;

Implémentation

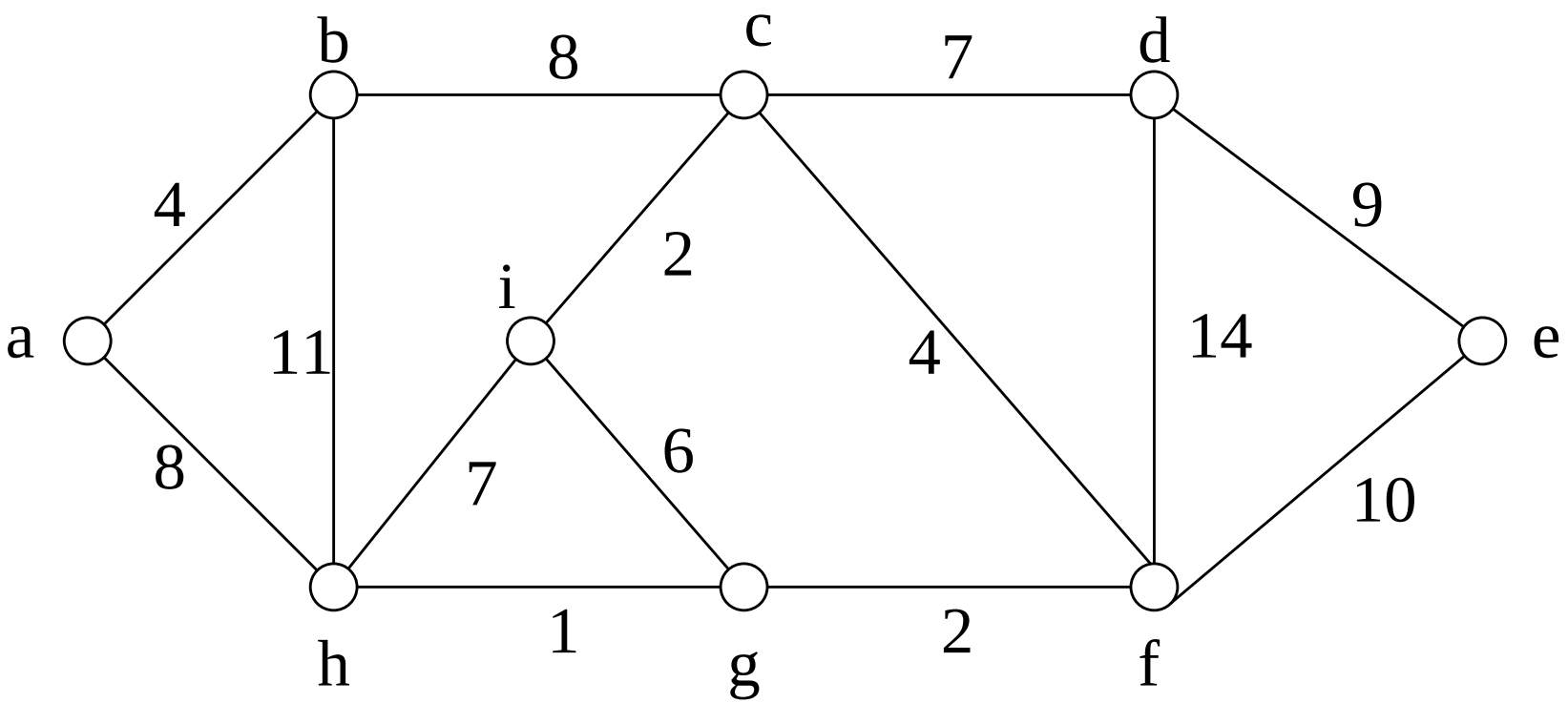
File de priorité Q : permet de conserver une liste d'objets avec une priorité $key[u]$ et un accès aisé à l'élément le plus prioritaire.

	Liste	Tas
Construction	$O(n)$	$O(n)$
Ajout	$O(1)$	$O(\log(n))$
key chg.	$O(1)$	$O(\log(n))$
Extrait-Minimum	$O(n)$	$O(\log n)$

Avec un tas (heap) : en $O(|S| \log |S| + |A| \log |S|) = O(|A| \log |S|)$.

Tas de Fibonacci : complexité amortie en $O(1)$ pour la diminution d'une *key*, donc Prim en $O(|A| + |S| \log |S|)$.

Exemple

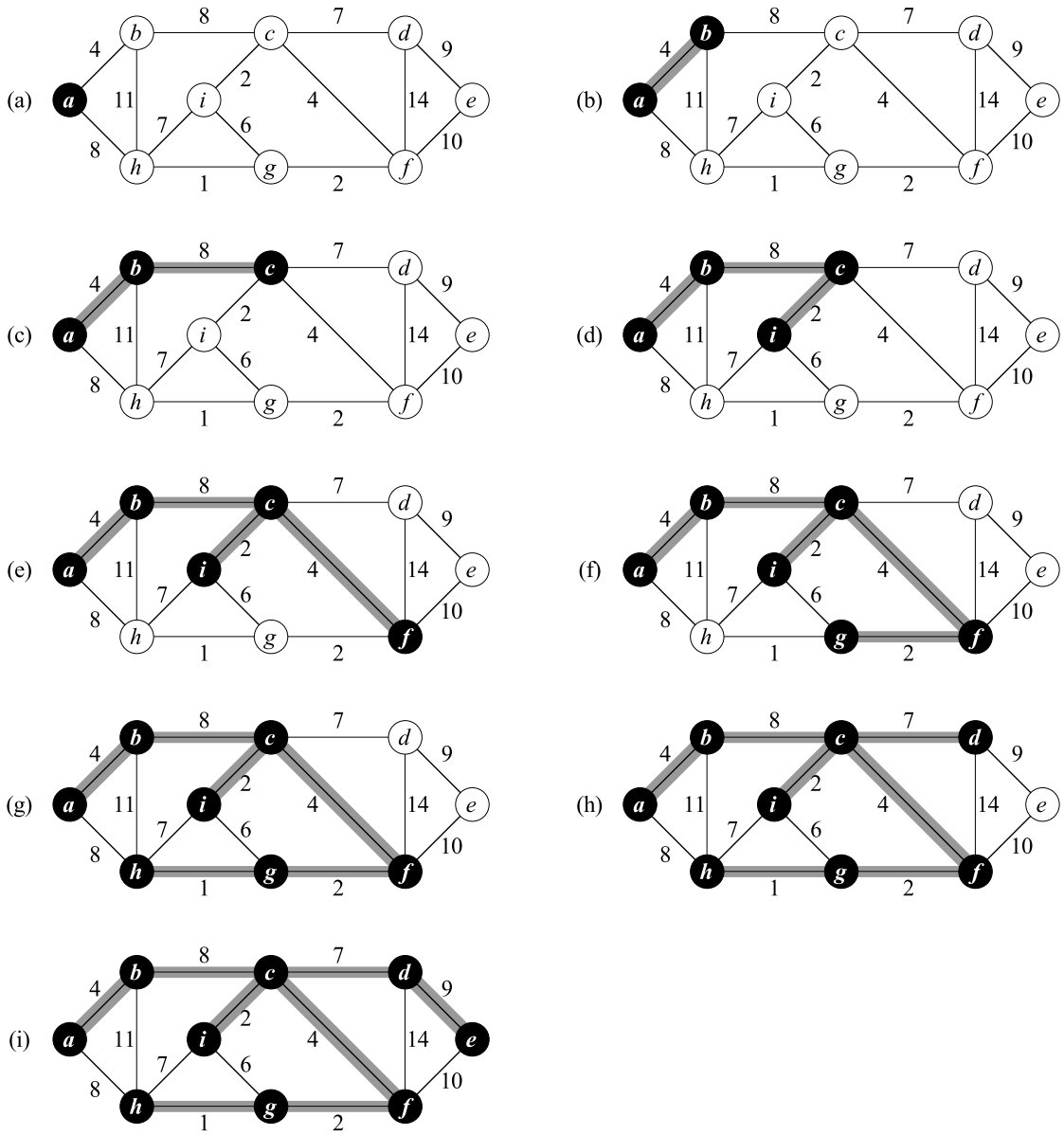


Un graphe non orienté pondéré

Maximum $\equiv$ Minimum

Held et Karp / WSP

Exemple



Plus court chemins

Definition

Étant donné un graphe (orienté), avec une pondération des arcs w, on définit le poids d'un chemin $c = \langle v_0, \ldots, v_k \rangle$ comme la somme des poids des arcs qui le constituent.

$$\delta(u, v) = \begin{cases} \min_{c \text{ chemin de } u \text{ à } v}\{w(c)\} & \text{si } \exists \text{ chemin de } u \text{ à } v \\ +\infty & \text{sinon} \end{cases}$$

- Une source, une destination
- Une source, toutes destinations
- Toutes paires
- $(\min, +)$ peut se remplacer par $(\wedge, \vee), (+, *) \ldots$

Circuits et coûts négatifs

Définition (Circuit absorbant)

Un circuit de longueur négative : si un tel circuit existe la distance minimum entre 2 sommets peut ne pas être définie.

- Poids positifs, graphe quelconque : Dijkstra
- Poids quelconques, graphe sans circuit : Bellman-Kalaba
- Poids quelconques mais sans circuit absorbant : Belmann-Ford-Moore.

$$\text{Plus court} \not\equiv \text{Plus long !}$$

Propriété de Bellman

≪ Tout sous-chemin d'un chemin optimal est aussi un chemin optimal ≫

Pour connaître la longueur ℓ du plus court chemin de Toulouse à Lille, il suffit par exemple de connaître la longueur d'un plus court chemin ℓ_i de Toulouse aux villes i reliées directement à Lille et de choisir d'aller vers la ville i^* qui minimise la somme $\ell_i + w(i, \text{Lille})$.

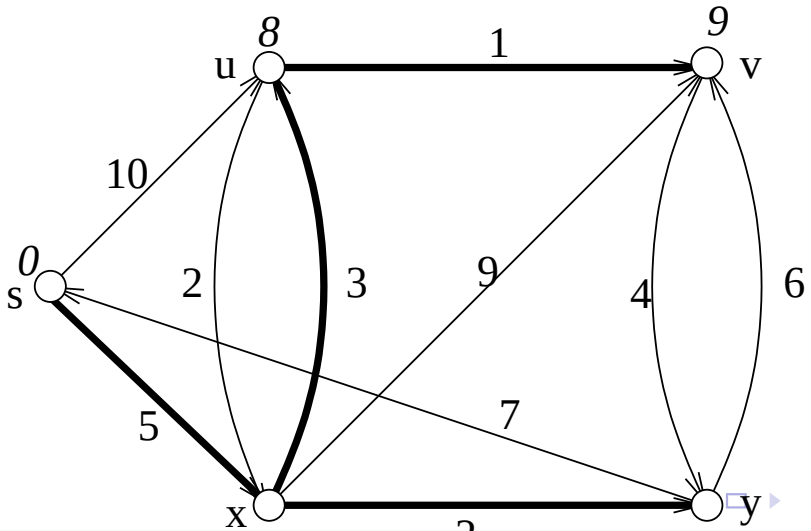
Arborescence des PCC

Définition (Arborescence des PCC de racine s)

Un sous-graphe orienté $G' = (S', A')$, où $S' \subseteq S$ et $A' \subseteq A$:

1. *S' est l'ensemble des sommets accessibles à partir de s dans G,*
2. *G' forme une arborescence de racine s,*
3. *pour tout $v \in S'$, le chemin de s à v dans G' est un plus court chemin de s vers v dans G.*

$\pi[u]$: prédécesseur de u par lequel passer si l'on veut aller de s à u.



Les procédures de base

$d[u]$: estimation pessimiste du PCC de s à u.

Procedure *Initialisation-PCC*(G, s)

```
pour chaque sommet v de G faire
    d[v] ← +∞;
    π[v] ← NIL;
d[s] ← 0;
```

Amélioration éventuelle de l'estimation de la distance de s à v.;

Procedure *Relaxation*(u, v, w)

```
si d[v] > d[u] + w(u, v) alors
    d[v] ← d[u] + w(u, v);
    π[v] ← u;
```

Quelques propriétés fondamentales

Propriété (Inégalité triangulaire)

Pour tout arc $(u, v) \in A$, on a $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Propriété (Propriété du majorant)

On a toujours $d[v] \geq \delta(s, v)$ pour tous les sommets $v \in S$, et une fois que $d[v]$ a atteint la valeur $\delta(s, v)$, elle ne change plus.

Propriété (Propriété aucun-chemin)

S'il n'y a pas de chemin de s à v, alors $d[v] = \delta(s, v) = +\infty$.

Propriété (Propriété de convergence)

Si $s \rightsquigarrow u \rightarrow v$ est un plus court chemin dans G pour un certain $u, v \in S$ et si $d[u] = \delta(s, u)$ à un certain instant antérieur au relâchement de l'arc (u, v), alors $d[v] = \delta(s, v)$ en permanence après le relâchement.

Quelques propriétés fondamentales

Propriété (Propriété de relâchement de chemin)

Si $p = \langle v_0, v_1, \ldots, v_k \rangle$ est un pcc de $s = v_0$ à v_k et si les arcs de p sont relâchés dans l'ordre (v_0, v_1), $(v_1, v_2), \ldots (v_{k-1}, v_k)$, alors $d[v_k] = \delta(s, v_k)$. Cette propriété est vraie indépendamment de toutes autres étapes de relâchement susceptibles de se produire, même si elles s'entremêlent avec des relâchements d'arcs de p.

Propriété (Propriété de sous-graphe prédécesseur)

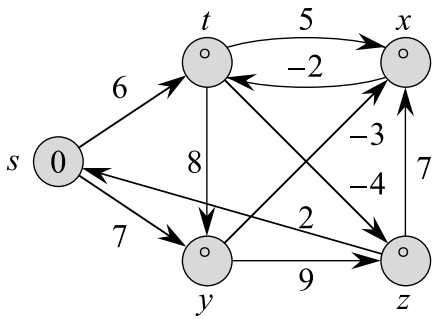
Une fois que $d[v] = \delta(s, v)$ pour tout $v \in S$, le sous-graphe prédécesseur est une arborescence de ppc de racine s.

Belmann-Ford-Moore

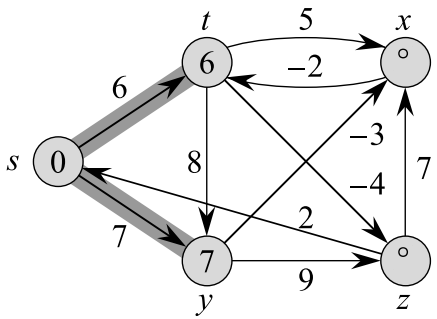
Procedure *Bellman-Ford*(G, w, s)

```
Initialisation-PCC(G, s);
pour chaque i ← 1 à |S| − 1 faire
    pour chaque (u, v) ∈ A faire
        Relaxation(u, v, w);
pour chaque (u, v) ∈ A faire
    si d[v] > d[u] + w(u, v) alors
        retourner faux;
retourner vrai;
```

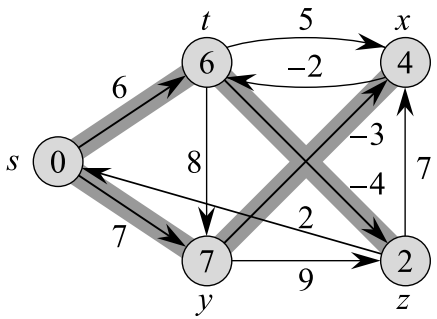
Exemple



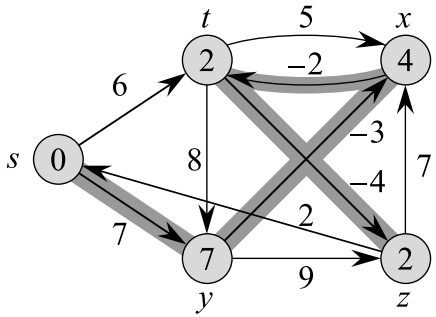
(a)



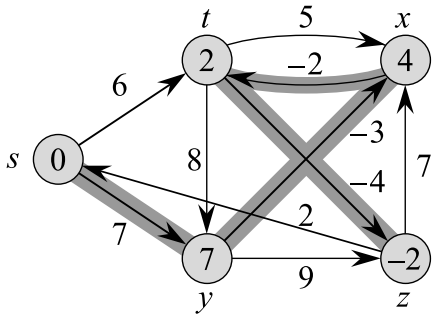
(b)



(c)



(d)



(e)

Complexité

1. L'initialisation est en $\theta(|S|)$.

Complexité

1. L'initialisation est en $\theta(|S|)$.
2. La boucle principale s'exécute $O(|S|)$ sur A

Complexité

1. L'initialisation est en $\theta(|S|)$.
2. La boucle principale s'exécute $O(|S|)$ sur A
3. La boucle finale s'exécute en $O(|A|)$

Complexité

1. L'initialisation est en $\theta(|S|)$.
2. La boucle principale s'exécute $O(|S|)$ sur A
3. La boucle finale s'exécute en $O(|A|)$

L'algorithme est en $O(|S||A|)$.

Correction

Si le graphe n'a pas de circuit absorbant.

1. Soit $p = \langle v_0, v_1, \ldots, v_k \rangle$, où $v_0 = s$ et $v_k = v$, un plus court chemin élémentaire de s à v.

Correction

Si le graphe n'a pas de circuit absorbant.

1. Soit $p = \langle v_0, v_1, \ldots, v_k \rangle$, où $v_0 = s$ et $v_k = v$, un plus court chemin élémentaire de s à v.
2. p a au plus $|S| - 1$ arcs.

Correction

Si le graphe n'a pas de circuit absorbant.

1. Soit $p = \langle v_0, v_1, \ldots, v_k \rangle$, où $v_0 = s$ et $v_k = v$, un plus court chemin élémentaire de s à v.
2. p a au plus $|S| - 1$ arcs.
3. A chaque itération i de la boucle, l'arc (v_{i-1}, v_i) est relaxé donc $d[v] = \delta(s, v)$.

Correction

Si le graphe n'a pas de circuit absorbant.

1. Soit $p = \langle v_0, v_1, \ldots, v_k \rangle$, où $v_0 = s$ et $v_k = v$, un plus court chemin élémentaire de s à v.
2. p a au plus $|S| - 1$ arcs.
3. A chaque itération i de la boucle, l'arc (v_{i-1}, v_i) est relaxé donc $d[v] = \delta(s, v)$.
4. l'algorithme retourne *vrai* par inégalité triangulaire.

Correction (2)

Si le graphe contient un circuit absorbant $\langle v_0, v_1, \ldots, v_k \rangle$ accessible depuis $v_0 = v_k = s$ et retourne *vrai*.

1. Pour tout $i = 1, \ldots, k$, on a $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$

Correction (2)

Si le graphe contient un circuit absorbant $\langle v_0, v_1, \ldots, v_k \rangle$ accessible depuis $v_0 = v_k = s$ et retourne *vrai*.

1. Pour tout $i = 1, \ldots, k$, on a $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$
2. Sommons toutes ces inégalités. On a :

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Correction (2)

Si le graphe contient un circuit absorbant $\langle v_0, v_1, \ldots, v_k \rangle$ accessible depuis $v_0 = v_k = s$ et retourne *vrai*.

1. Pour tout $i = 1, \ldots, k$, on a $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$
2. Sommons toutes ces inégalités. On a :

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

3. mais c'est un circuit donc

$$\sum_{i=1}^{k} d[v_{i-1}] = \sum_{i=1}^{k} d[v_i]$$

Correction (2)

Si le graphe contient un circuit absorbant $\langle v_0, v_1, \ldots, v_k \rangle$ accessible depuis $v_0 = v_k = s$ et retourne *vrai*.

1. Pour tout $i = 1, \ldots, k$, on a $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$
2. Sommons toutes ces inégalités. On a :

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

3. mais c'est un circuit donc

$$\sum_{i=1}^{k} d[v_{i-1}] = \sum_{i=1}^{k} d[v_i]$$

4. donc $\sum_{i=1}^{k} w(v_{i-1}, v_i) \geq 0$. Impossible.

Dijkstra

Cas de graphes avec des poids tous positifs ou nuls.

1. Maintient un ensemble V de sommets pour lesquels l'estimation $d[\cdot]$ est correcte (égale à $\delta(s, \cdot)$)

Dijkstra

Cas de graphes avec des poids tous positifs ou nuls.

1. Maintient un ensemble V de sommets pour lesquels l'estimation $d[\cdot]$ est correcte (égale à $\delta(s, \cdot)$)
2. Sélectionne un sommet u de $S - V$ dont l'estimation $d[u]$ est la plus faible.

Dijkstra

Cas de graphes avec des poids tous positifs ou nuls.

1. Maintient un ensemble V de sommets pour lesquels l'estimation $d[\cdot]$ est correcte (égale à $\delta(s, \cdot)$)
2. Sélectionne un sommet u de $S - V$ dont l'estimation $d[u]$ est la plus faible.
3. L'insère dans V

Dijkstra

Cas de graphes avec des poids tous positifs ou nuls.

1. Maintient un ensemble V de sommets pour lesquels l'estimation $d[\cdot]$ est correcte (égale à $\delta(s, \cdot)$)
2. Sélectionne un sommet u de $S - V$ dont l'estimation $d[u]$ est la plus faible.
3. L'insère dans V
4. Relâche tous les arcs qui quittent u

Dijkstra

Dijkstra(G, w, s);
Initialisation-PCC(G, s);
$V \leftarrow \varnothing$;
$Q \leftarrow S$;
tant que $Q \neq \varnothing$ **faire**
    $u \leftarrow$ Extrait-Minimum(Q);
    pour chaque *sommet* $v \in Adj[u]$ **faire**
        Relaxation(u, v, w);
    $V \leftarrow V \cup \{u\}$;

Dijkstra : correction

Lorsqu'un sommet u est inséré dans V, $d[u] = \delta(s, u)$. Par l'absurde.

1. Soit u le premier sommet inséré dans V tel que $d[u] \neq \delta(s, u)$.

Dijkstra : correction

Lorsqu'un sommet u est inséré dans V, $d[u] = \delta(s, u)$. Par l'absurde.

1. Soit u le premier sommet inséré dans V tel que $d[u] \neq \delta(s, u)$.
2. $u \neq s$ donc V non vide.

Dijkstra : correction

Lorsqu'un sommet u est inséré dans V, $d[u] = \delta(s, u)$. Par l'absurde.

1. Soit u le premier sommet inséré dans V tel que $d[u] \neq \delta(s, u)$.
2. $u \neq s$ donc V non vide.
3. il y a un chemin de s à u (sinon $d[u] = \infty = \delta[s, u]$)

Dijkstra : correction

Lorsqu'un sommet u est inséré dans V, $d[u] = \delta(s, u)$. Par l'absurde.

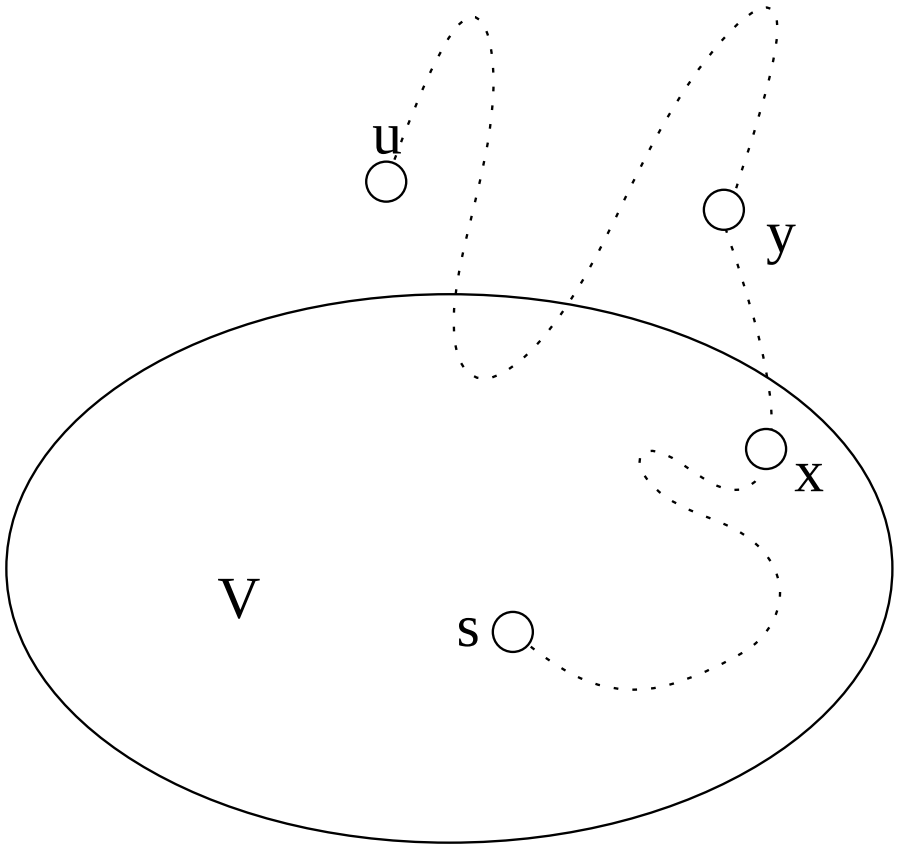
1. Soit u le premier sommet inséré dans V tel que $d[u] \neq \delta(s, u)$.
2. $u \neq s$ donc V non vide.
3. il y a un chemin de s à u (sinon $d[u] = \infty = \delta[s, u]$)
4. donc un plus court chemin de s à u.

Dijkstra : correction

Lorsqu'un sommet u est inséré dans V, $d[u] = \delta(s, u)$. Par l'absurde.

1. Soit u le premier sommet inséré dans V tel que $d[u] \neq \delta(s, u)$.
2. $u \neq s$ donc V non vide.
3. il y a un chemin de s à u (sinon $d[u] = \infty = \delta[s, u]$)
4. donc un plus court chemin de s à u.
5. Soit y le premier sommet de ce pcc qui n'est pas dans V et x son prédécesseur dans V.

Dijkstra



Mise à jour des sommets de V

Dijkstra

1. $d[x] = \delta(s, x)$

Dijkstra

1. $d[x] = \delta(s, x)$
2. $d[y] = \delta(s, y)$ (plus court chemin et relaxation).

Dijkstra

1. $d[x] = \delta(s, x)$
2. $d[y] = \delta(s, y)$ (plus court chemin et relaxation).
3. Poids positifs : $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$

Dijkstra

1. $d[x] = \delta(s, x)$
2. $d[y] = \delta(s, y)$ (plus court chemin et relaxation).
3. Poids positifs : $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$
4. u de poids minimum donc $d[u] \leq d[y]$

Dijkstra

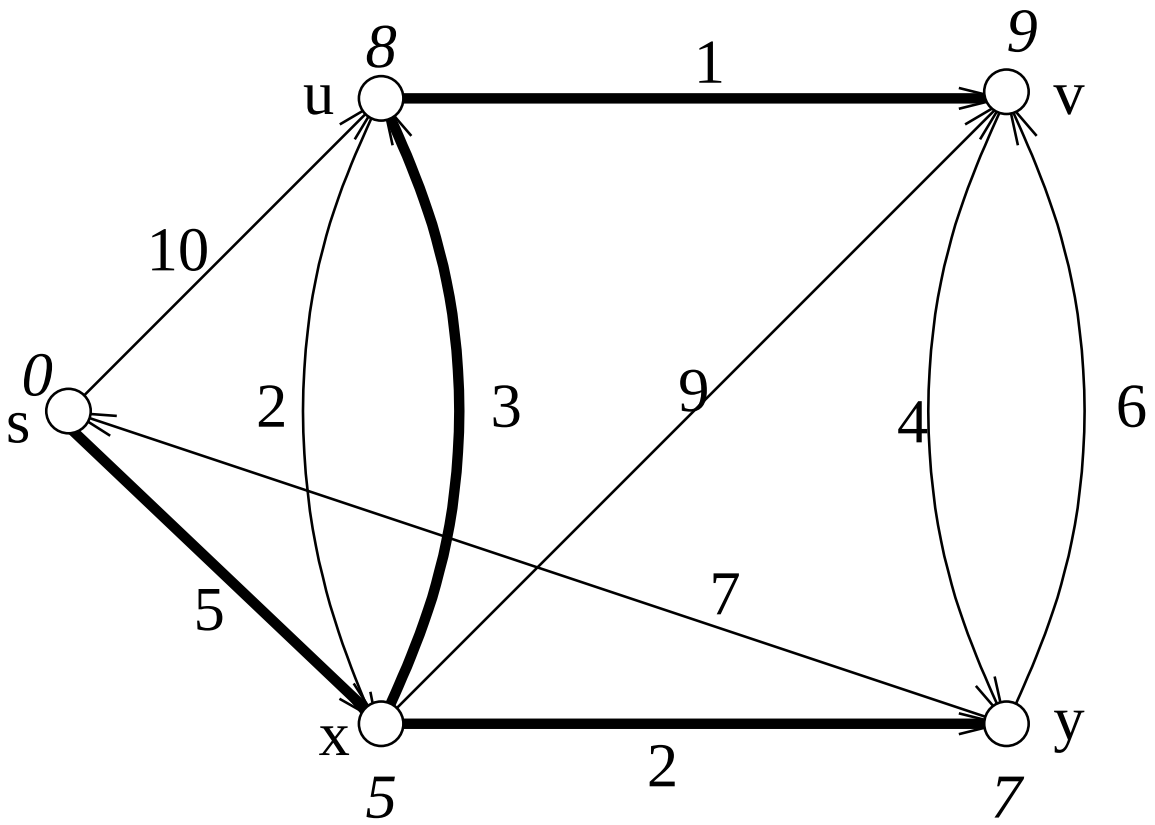
1. $d[x] = \delta(s, x)$
2. $d[y] = \delta(s, y)$ (plus court chemin et relaxation).
3. Poids positifs : $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$
4. u de poids minimum donc $d[u] \leq d[y]$
5. donc $d[y] = \delta(s, y) = \delta(s, u) = d[u]$

Dijkstra

1. $d[x] = \delta(s, x)$
2. $d[y] = \delta(s, y)$ (plus court chemin et relaxation).
3. Poids positifs : $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$
4. u de poids minimum donc $d[u] \leq d[y]$
5. donc $d[y] = \delta(s, y) = \delta(s, u) = d[u]$

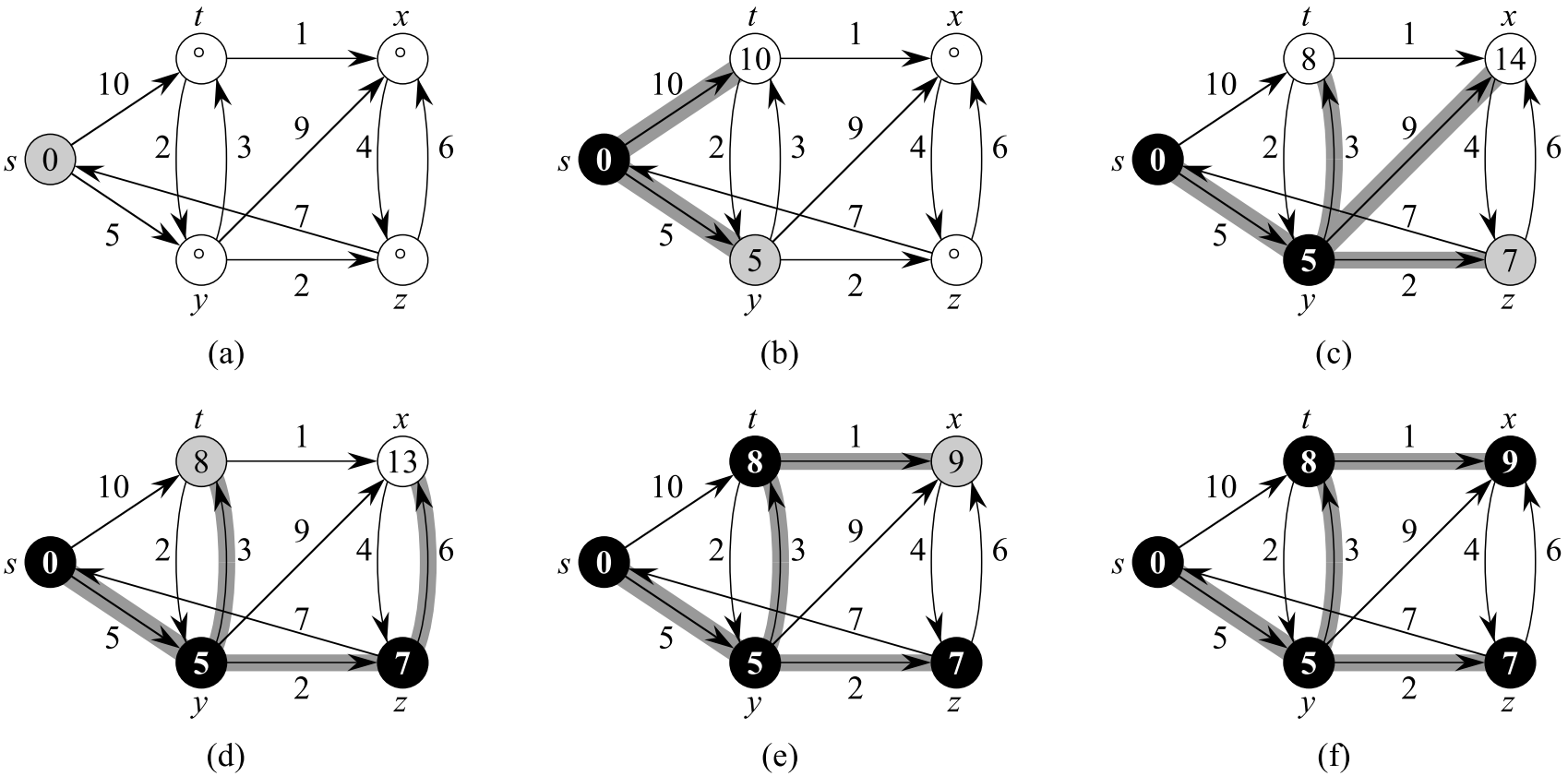
Impossible puisque $d[u] \neq \delta(s, u)$.

Un exemple



Arbre des plus courts chemins, en italique les $\delta(s, u)$.

Un exemple



Arbre des plus courts chemins, dans les sommets $\delta(s, u)$.

Complexité

On utilise une file de priorité.

- $|S|$ appels à Extrait-Minimum

Complexité

On utilise une file de priorité.

- $|S|$ appels à Extrait-Minimum
- Chaque $Adj[\cdot]$ exploré

Complexité

On utilise une file de priorité.

- $|S|$ appels à Extrait-Minimum
- Chaque $Adj[\cdot]$ exploré
- En $O(|S|^2 + |A|)$ (file = tableau non trié)

Complexité

On utilise une file de priorité.

- $|S|$ appels à Extrait-Minimum
- Chaque $Adj[\cdot]$ exploré
- En $O(|S|^2 + |A|)$ (file = tableau non trié)
- En $O(|S| + |A|.\log(|S|))$ (Tas),

Complexité

On utilise une file de priorité.

- $|S|$ appels à Extrait-Minimum
- Chaque $Adj[\cdot]$ exploré
- En $O(|S|^2 + |A|)$ (file = tableau non trié)
- En $O(|S| + |A|.\log(|S|))$ (Tas),
- en $O(|S|.\log(|S|) + |A|)$ (Tas de Fibonacci).

Bellman-Kalaba

Adapté aux graphes orientés sans circuit (DAG). Poids de signes quelconques.

1. Pas de circuit, donc tri topologique possible depuis s.

Bellman-Kalaba

Adapté aux graphes orientés sans circuit (DAG). Poids de signes quelconques.

1. Pas de circuit, donc tri topologique possible depuis s.
2. Un chemin de s à u respecte l'ordre du tri.

Bellman-Kalaba

Adapté aux graphes orientés sans circuit (DAG). Poids de signes quelconques.

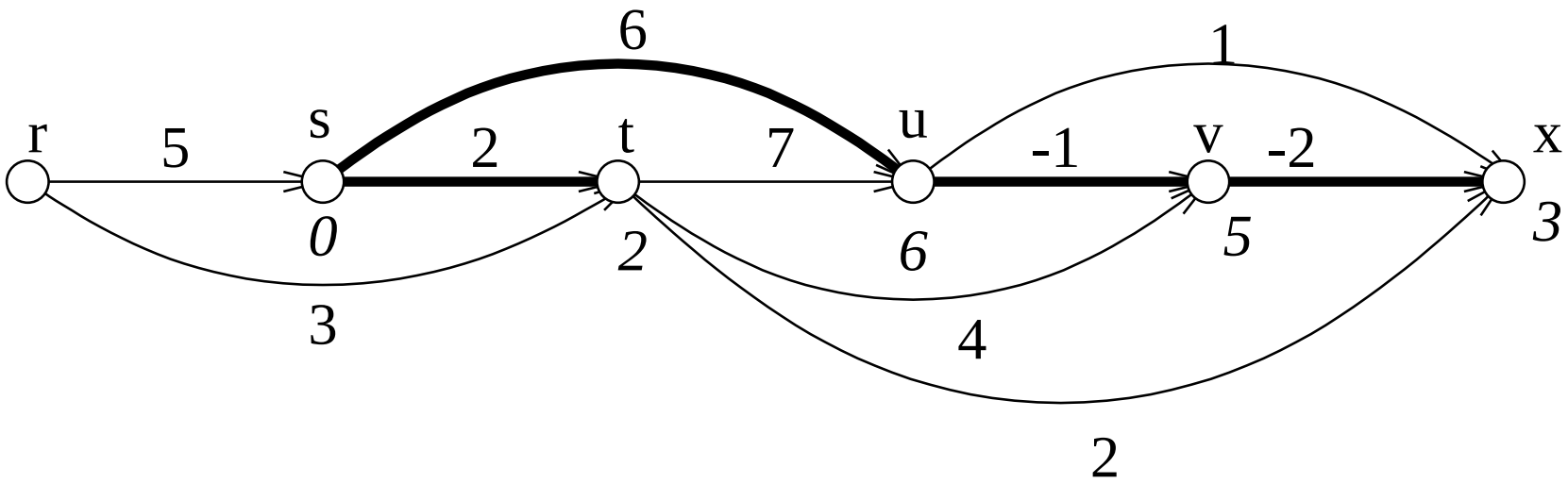
1. Pas de circuit, donc tri topologique possible depuis s.
2. Un chemin de s à u respecte l'ordre du tri.
3. Il suffit de connaître la longueur d'un PCC de s aux prédécesseurs de u (inductivement).

Exemple

GOSC-PlusCourt(G, w, s);
Tri-Topologique(G);
Initialisation-PCC(G, s);
pour chaque *sommet u de G pris dans l'ordre du tri topologique* **faire**
    pour chaque *sommet* $v \in Adj[u]$ **faire**
        Relaxation(u, v, w);

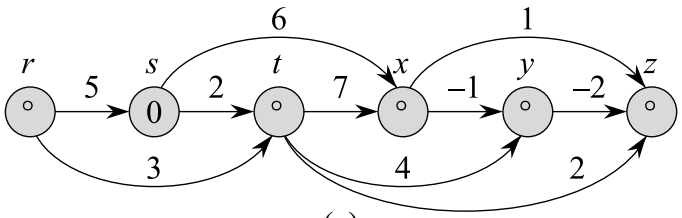
En $\theta(|S| + |A|)$.

Exemple

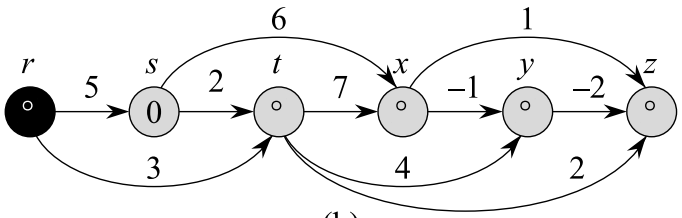


Un GOSC, un arbre des plus courts chemins.

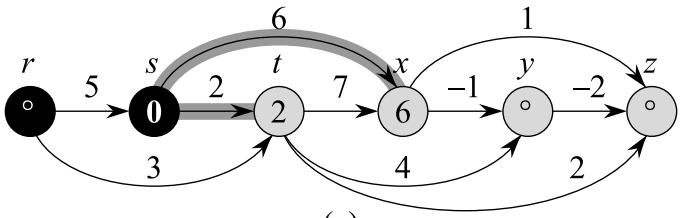
Exemple



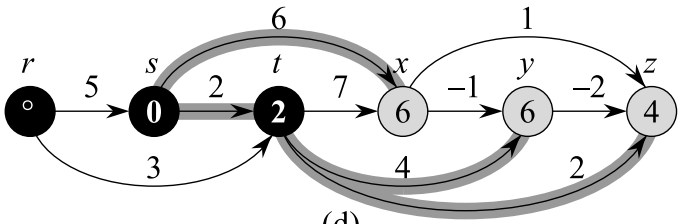
(a)



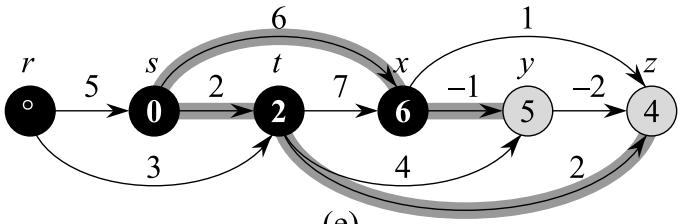
(b)



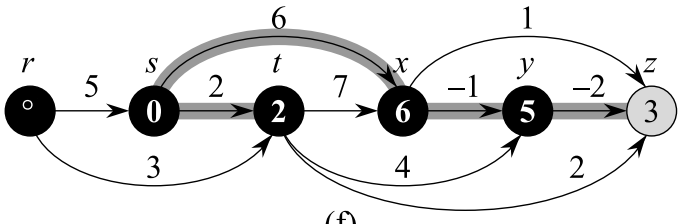
(c)



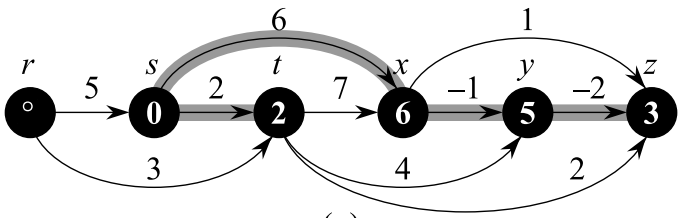
(d)



(e)



(f)



(g)

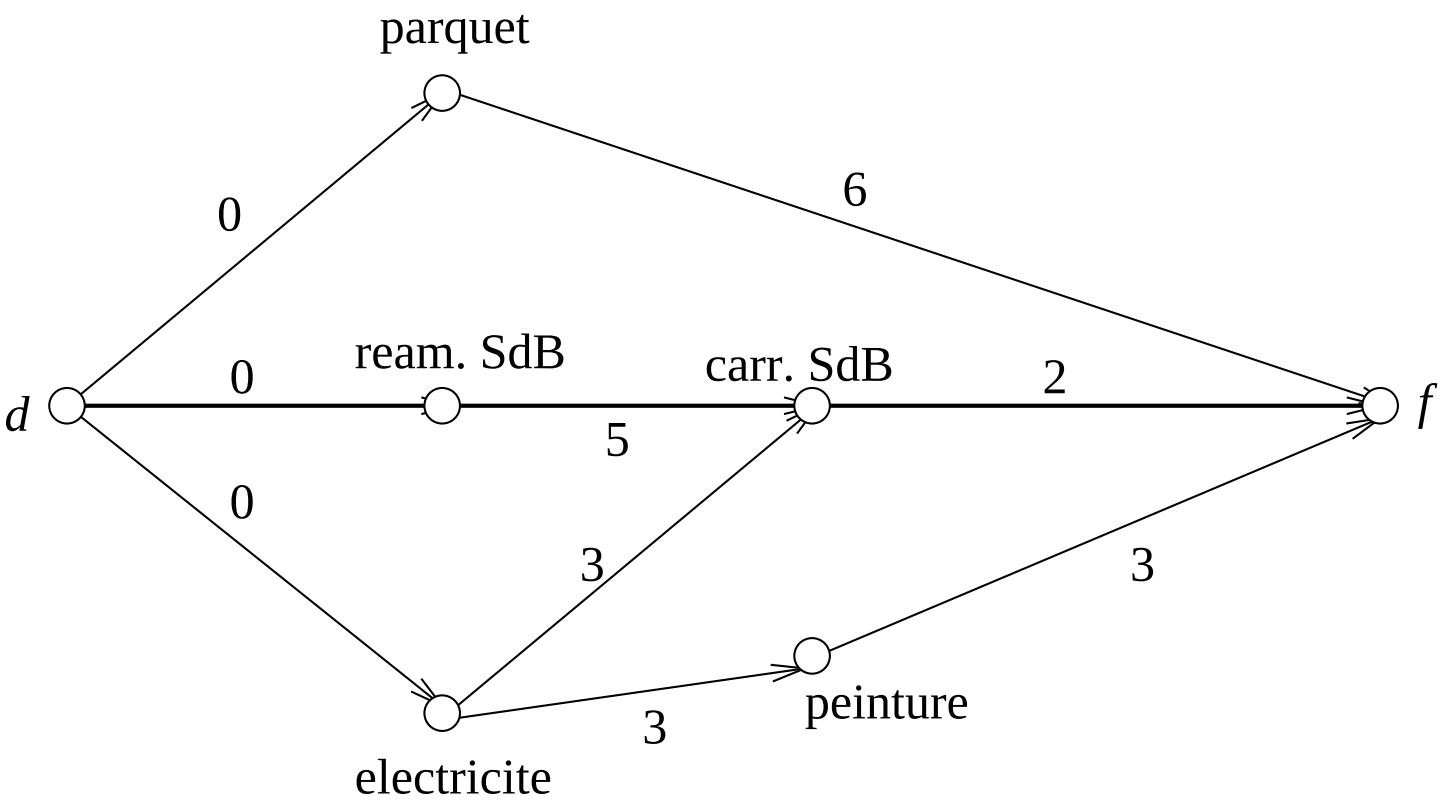
Ordonnancement simple

Pour rénover une maison, il est prévu de

- refaire l'installation électrique (3 jours)
- réaménager (5 jours) la salle de bains
- de carreler (2 jours) la salle de bains
- de refaire le parquet de la salle de séjour (6 jours)
- de repeindre les chambres (3 jours).

La peinture et le carrelage ne pourront être faits qu'après la réfection de l'installation électrique.

La méthodes Potentiel/Tâche (et PERT)



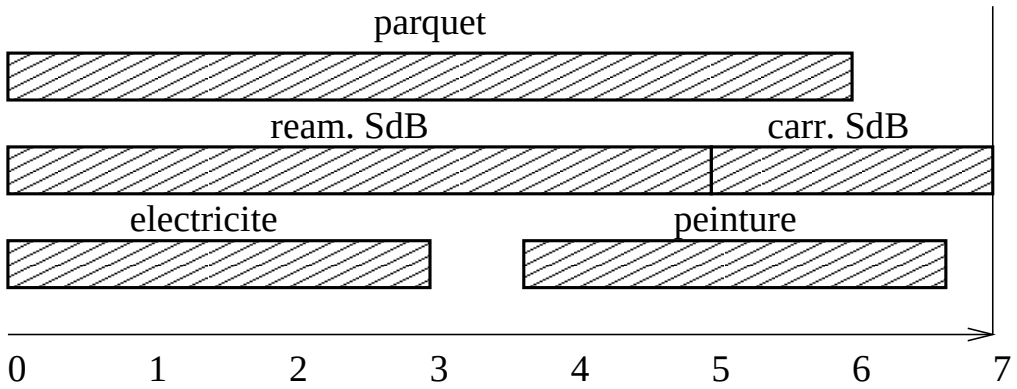
Graphe potentiel/tâches et chemin critique

Les dates importantes

- *date de début au plus tôt* de chaque tâche : longueur d'un plus long chemin de la source d à chacun des sommets
- chemins critiques : chemins les plus longs de d à f.
- *date de début au plus tard* : durée globale moins la longueur d'un plus long chemin de f à la tâche.
- *marge totale* d'une tâche : différence entre sa *date de début au plus tôt* et sa *date de début au plus tard*

Exemple

Tâches	début au plus tôt	début au plus tard	marge totale
parquet	0	1	1
réam. SdB	0	0	0
carr. SdB	5	5	0
electricité	0	1	1
peinture	3	4	1



Exemples d'utilisation de ces algorithmes

1. Bellman-Kalaba : Chaîne de Markov cachées et Viterbi.
2. Bellman-Kalaba : Alignement de séquences d'ADN et Needleman et Wunsch.
3. Prim/Kruskal et MST : k-clustering.

Couplages

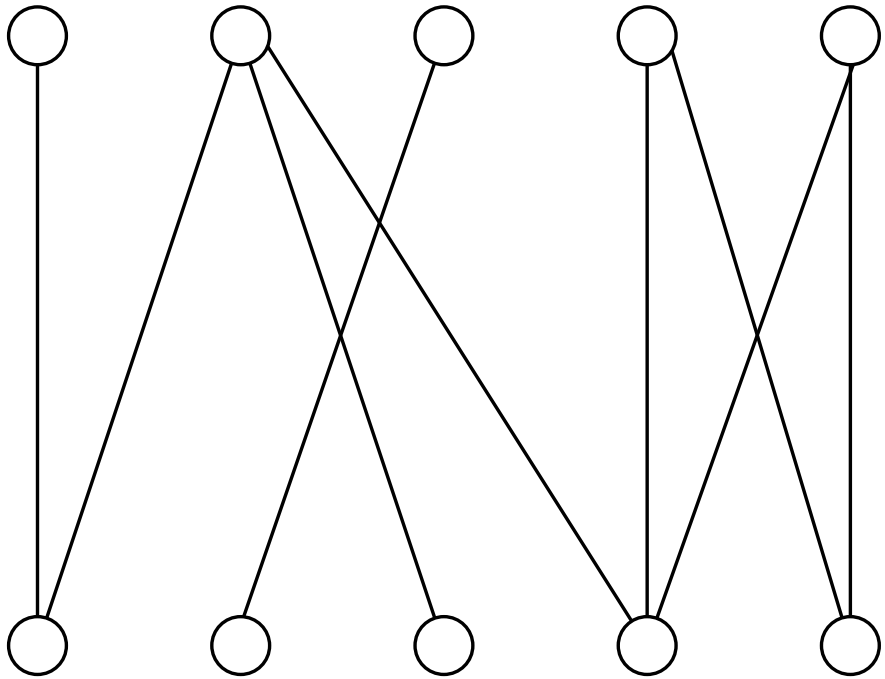
On doit affecter un certain nombre de cours à un certain nombre d'enseignants.

- Chaque enseignant est qualifié dans certaines disciplines.
- On désire attribuer un cours à chaque enseignant de tel façon qu'un même enseignement ne soit pas affecté à deux professeurs différents, et vice-versa.
- S'il n'est pas possible de satisfaire tous les enseignants, on veut satisfaire le maximum d'entre eux.

Graphe biparti

Définition (Graphe biparti)

Un graphe non orienté $G = (S, A)$ est biparti si $S = S_1 \cup S_2$, $S_1 \cap S_2 = \emptyset$, s'il n'existe pas d'arête entre les sommets de S_1 et s'il n'existe pas d'arête entre les sommets de S_2.



Couplage

Définition

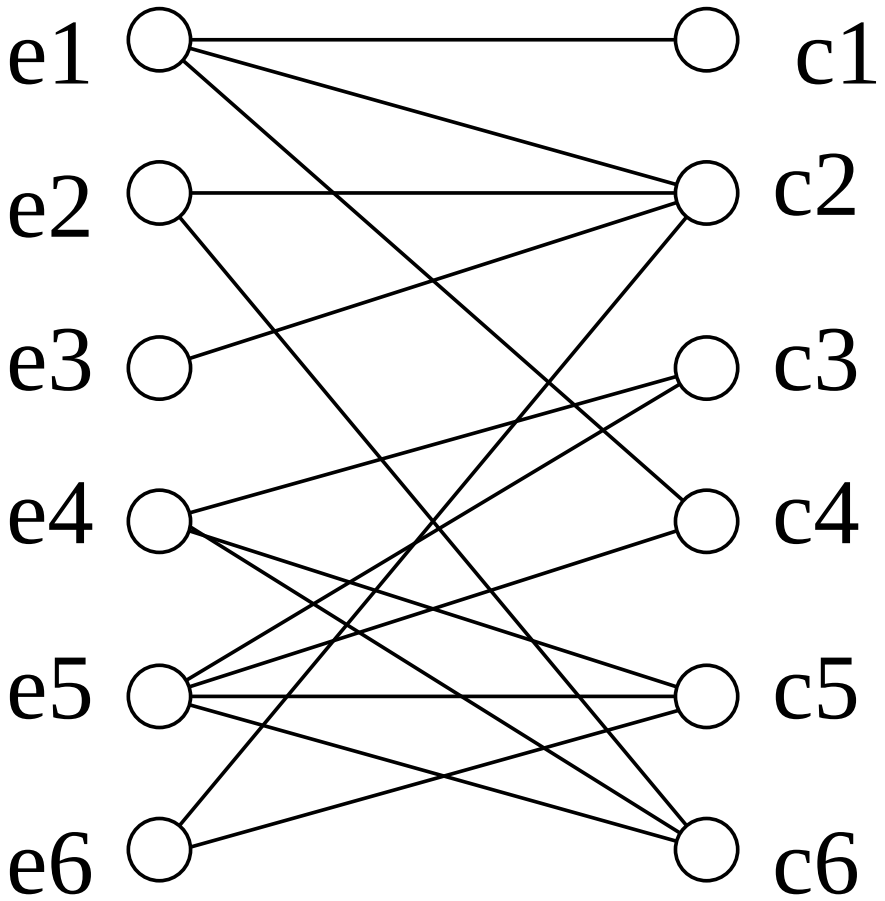
Étant donné un graphe non orienté $G = (S, A)$, un couplage sur G est formé d'un ensemble d'arêtes C tel qu'aucune paire d'arêtes de C ne partage un sommet commun.

La taille du couplage est le nombre d'arêtes de C.

Le couplage est **parfait** *si tous les sommets du graphe sont couverts par le couplage.*

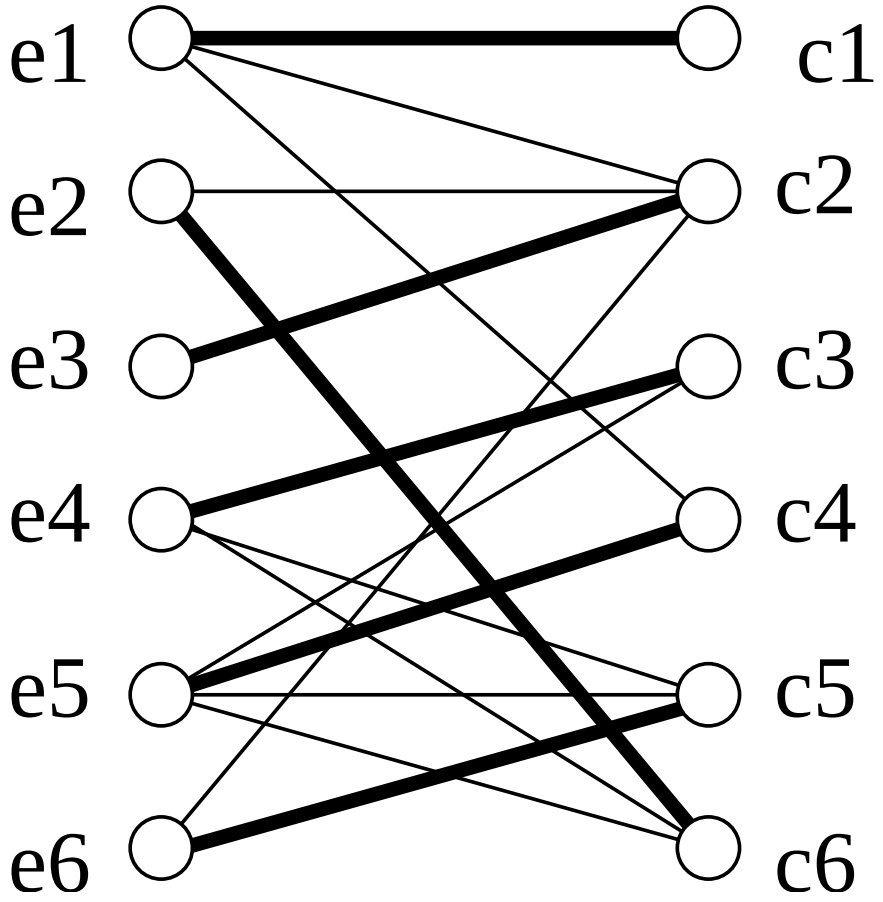
Objectif : trouver un couplage de taille maximum ($\leq \lfloor \frac{|S|}{2} \rfloor$)

Exemple



Un graphe biparti

Exemple



Un graphe biparti et un couplage maximum (parfait)

Terminologie

Soit $C \subset A$ un couplage de G.

- un sommet $s \in S$ est *saturé* si c'est l'extrémité d'une arête de C. Sinon, *insaturé*.
- Une chaîne dans laquelle une arête sur deux est dans C et les autres dans $A - C$ est dite *alternée*.
- Une chaîne alternée est dite *améliorante* si elle relie deux sommets insaturés à ses extrémités (longueur impaire).

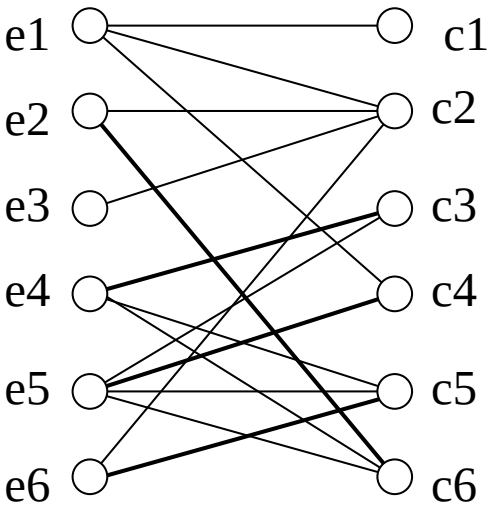
Théorème de Berge

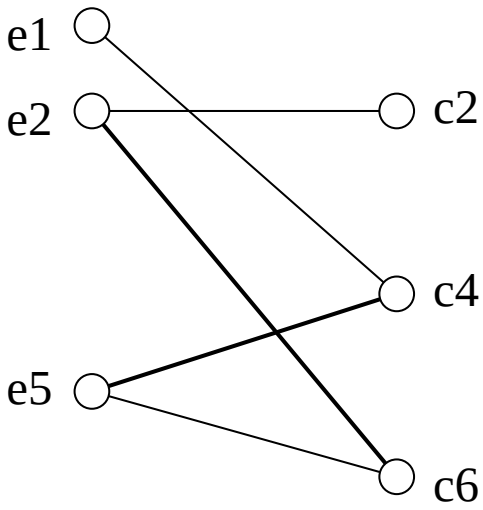
Théorème

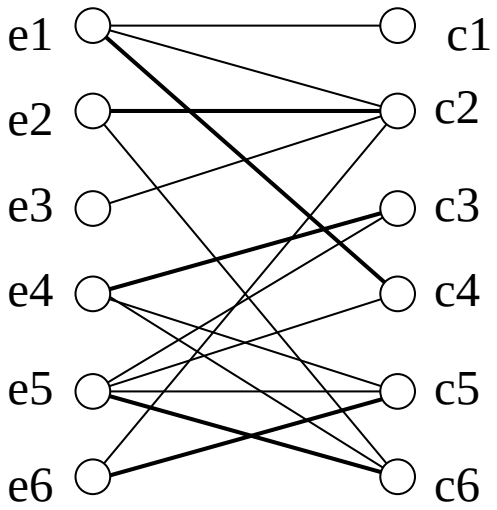
Un couplage C est maximum si et seulement s'il n'existe pas de chaîne alternée améliorante relativement à C.

Chaîne améliorante : suffisant

À partir d'un couplage C et d'une chaîne améliorante C_A :
on améliore le couplage (de 1) par "ou exclusif" $\oplus$ (suppression dans C des arêtes $C \cap C_A$ et ajout des arêtes $C_A \backslash (C \cap C_A)$).







Chaîne améliorante : nécessaire

Soit C et D, 2 couplages de G, $|C| < |D|$ et $G' = (S, C \oplus D)$.

C et D sont deux couplages : chaque sommet de S est l'extrémité d'au plus une arête de C et d'au plus une arête de D.

Une composante connexe de G' est une chaîne simple (évent. un cycle) dont une arête sur deux est dans C et les autres dans D.

Comme $|C| < |D|$, $C \oplus D$ contient plus d'arêtes de D que de C. Les cycles : autant d'arêtes de C que de $D \Rightarrow G'$ a une chaîne qui n'est pas un cycle, avec plus d'arêtes de D que de C : c'est une chaîne améliorante.

Algorithme des chaînes améliorantes (algorithme d'Hopcroft-Karp)

On peut chercher les chaînes améliorantes en parcourant le graphe biparti en alternant le parcours d'arêtes hors et dans C à partir des sommets insaturés de S_1.

$\Longrightarrow$ Construction du graphe des chaînes améliorantes par un parcours en largeur

Au plus il y a $\lfloor \frac{|S|}{2} \rfloor$ chaînes améliorantes.

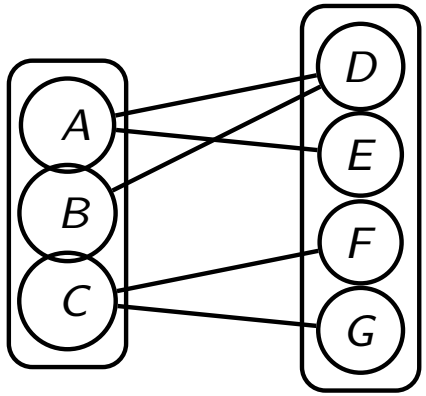
Complexité (majorant) de l'algorithme en $O(|S|.|A|)$

Algorithme d'Hopcroft-Karp

Entrée : Un graphe biparti $G = (S_1 \cup S_2, A)$
Sortie : Un couplage maximal C

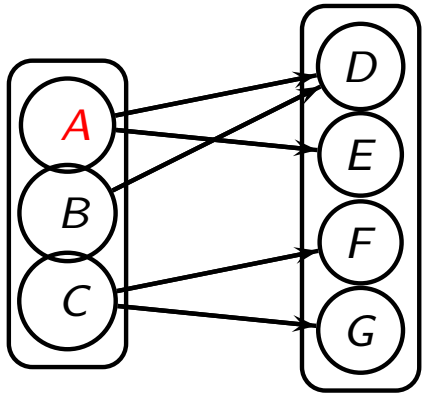
```
/* initialisation */
```
anciennetaille = -1 ;
$C = \emptyset$;
```
/* traitement */
```
Tant que *anciennetaille* $\neq |C|$ **faire**
    anciennetaille $= |C|$;
    orienter le graphe (S_1 vers S_2, excepté pour C, ordre inverse) ;
    Pour tout $u \in S_1$ tel que $u \notin C$ **faire**
        Si $\exists\ v \in S_2$ connecté à u par un chemin tel que $v \notin C$ **alors**
            chemin = chemin améliorant de u vers v ;
            $C = C \oplus$ *chemin* ;
Retourner C

Algorithme d'Hopcroft-Karp

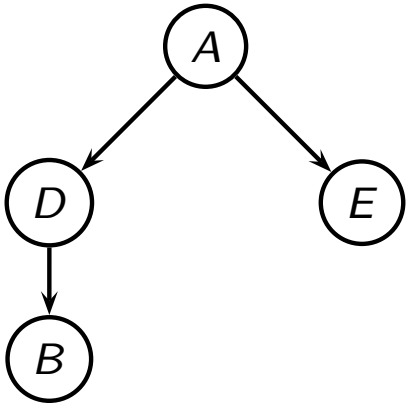


C={ }

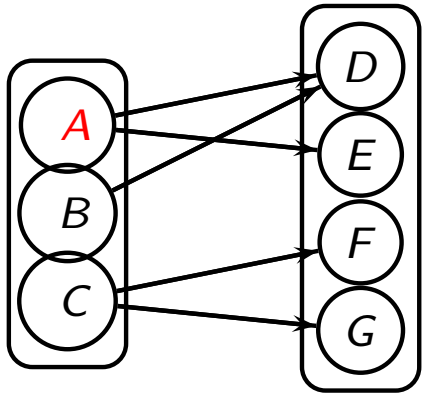
Algorithme d'Hopcroft-Karp



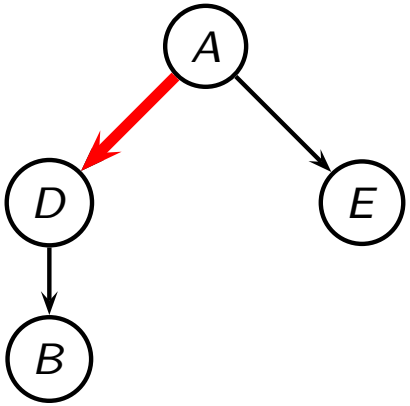
C={ }



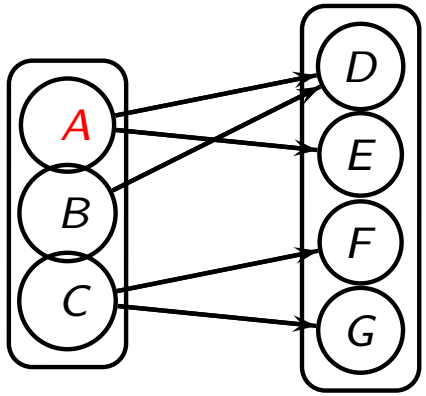
Algorithme d'Hopcroft-Karp



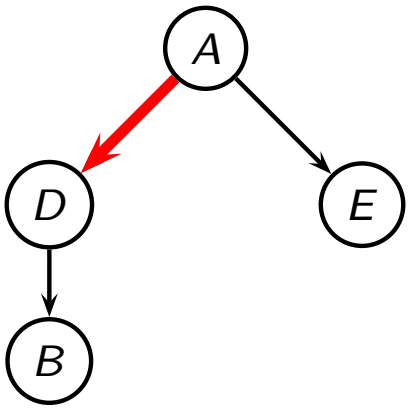
C={ }



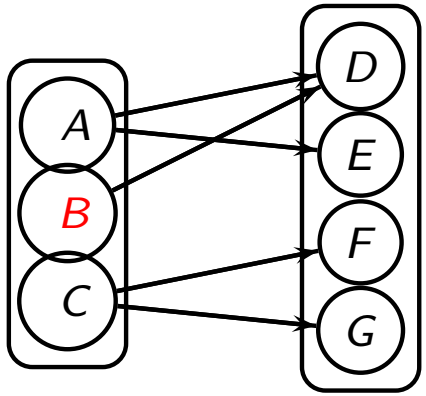
Algorithme d'Hopcroft-Karp



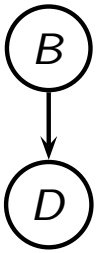
C={(A,D) }



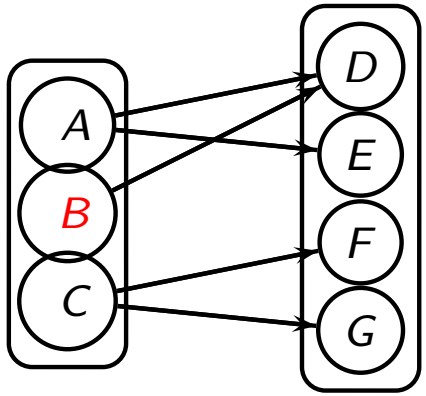
Algorithme d'Hopcroft-Karp



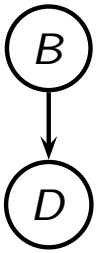
C={(A,D) }



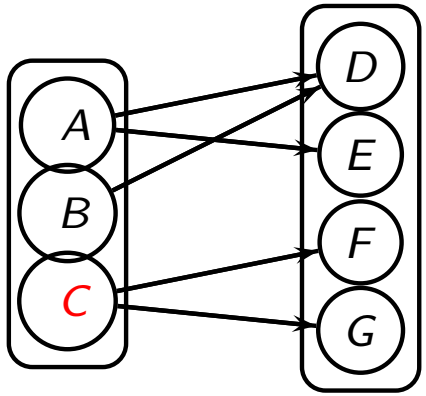
Algorithme d'Hopcroft-Karp



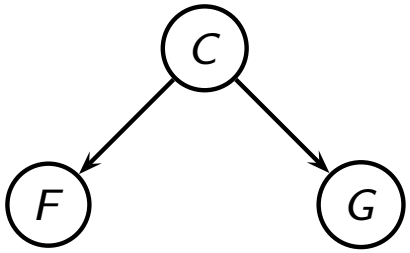
C={(A,D) }



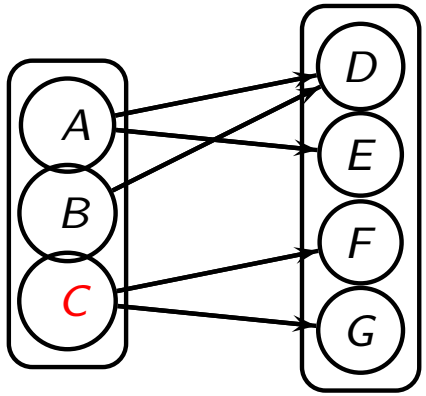
Algorithme d'Hopcroft-Karp



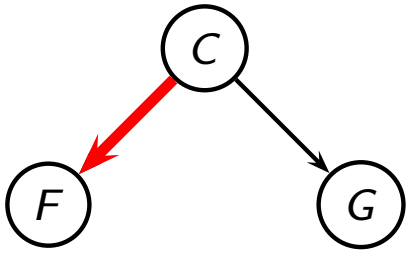
C={(A,D) }



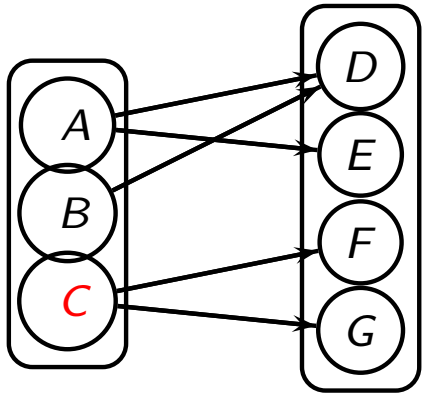
Algorithme d'Hopcroft-Karp



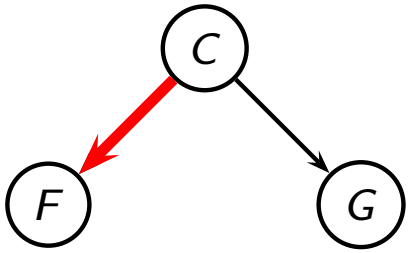
C={(A,D) }



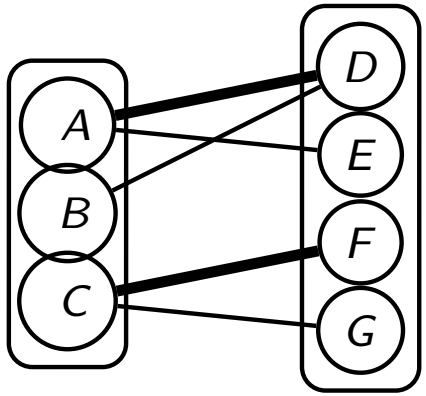
Algorithme d'Hopcroft-Karp



C={(A,D) (C,F) }

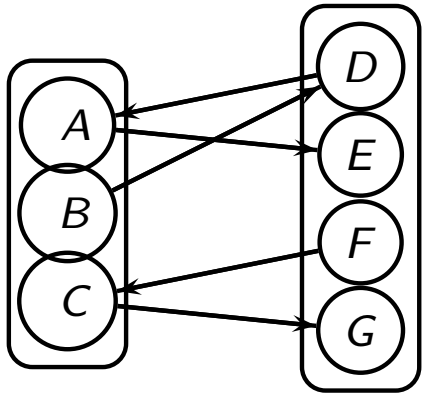


Algorithme d'Hopcroft-Karp



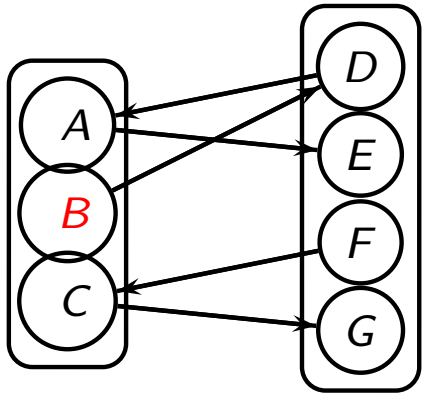
C={(A,D) (C,F) }

Algorithme d'Hopcroft-Karp

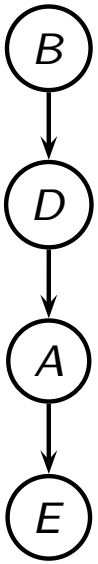


C={(A,D) (C,F) }

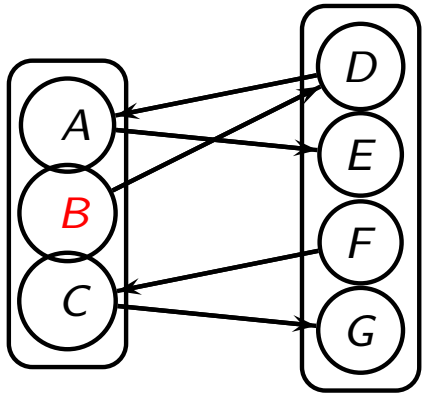
Algorithme d'Hopcroft-Karp



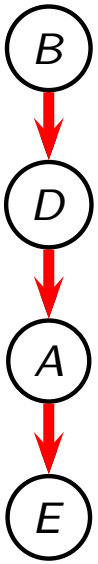
C={(A,D) (C,F) }



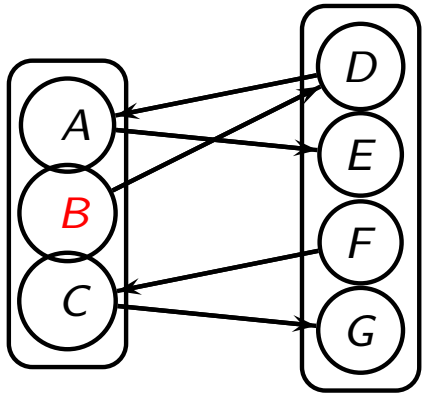
Algorithme d'Hopcroft-Karp



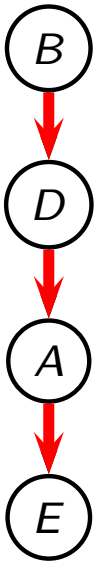
C={(A,D) (C,F) }



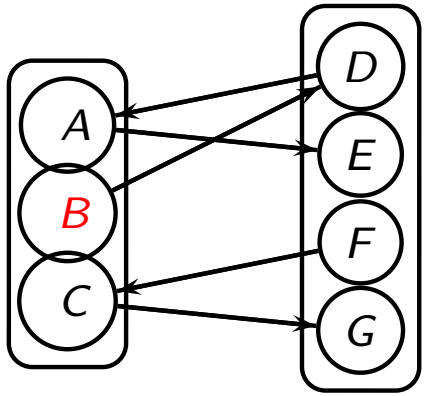
Algorithme d'Hopcroft-Karp



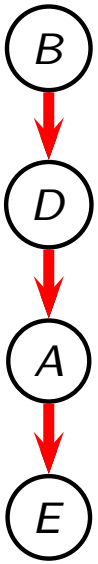
C={(A,D) (C,F) (B,D) }



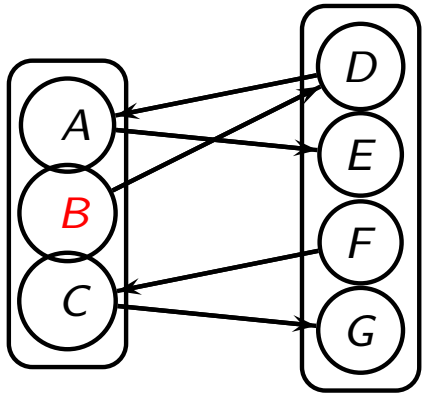
Algorithme d'Hopcroft-Karp



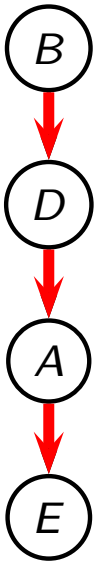
C={ (C,F) (B,D) }



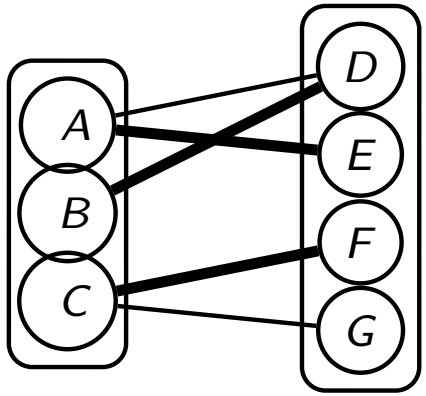
Algorithme d'Hopcroft-Karp



C={ (C,F) (B,D) (A,E)}

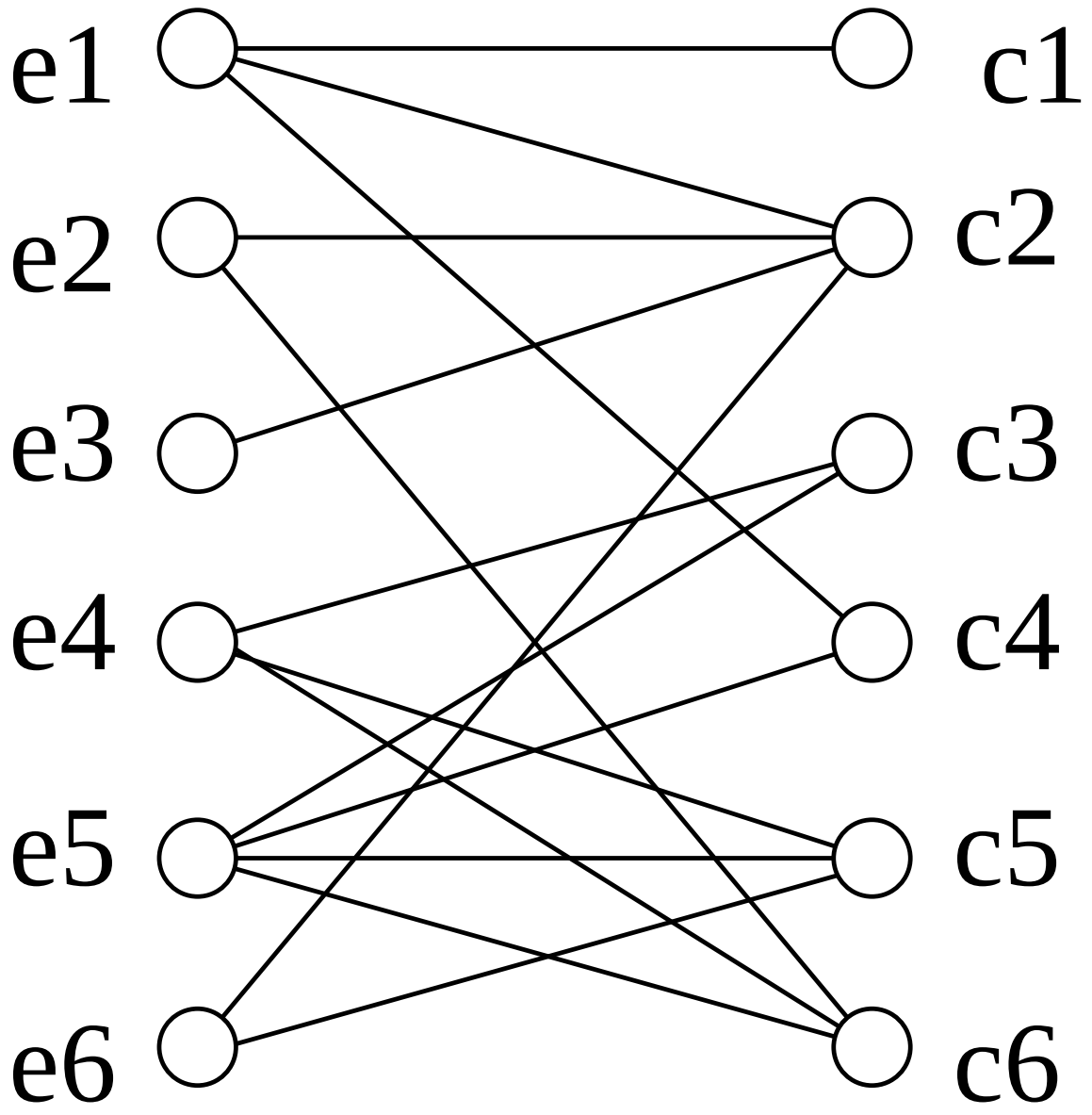


Algorithme d'Hopcroft-Karp

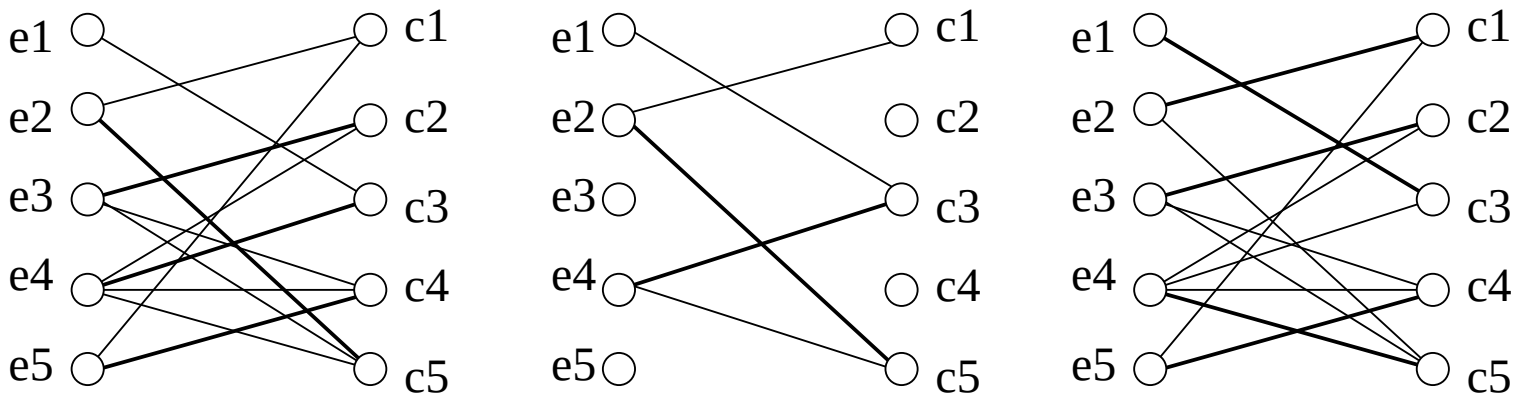


C={ (C,F) (B,D) (A,E)}

Exemple



Exemple



Un couplage, chaîne améliorante, couplage amélioré

Finalement. . .

A chaque itération, la taille du plus court chemin augmente de 1.

A l'itération $\sqrt{|S|}$, il y a au plus $\sqrt{|S|}$ chemins améliorants de taille supérieure à $\sqrt{|S|}$.

Algorithme de Hopcroft et Karp : complexité en $O(\sqrt{|S|}.|A|)$

Extensions possibles : couplages pondérés, graphes quelconques

Exercice : problème du Sudoku

9		8		7			4	2
			9				6	
		6	8		4		9	
5	8	3						4
				6	3			7
3	5				2	8		
	6			3	7			9
2		7			5			6

Solution : problème du Sudoku

Problème Sudoku NP-complet.

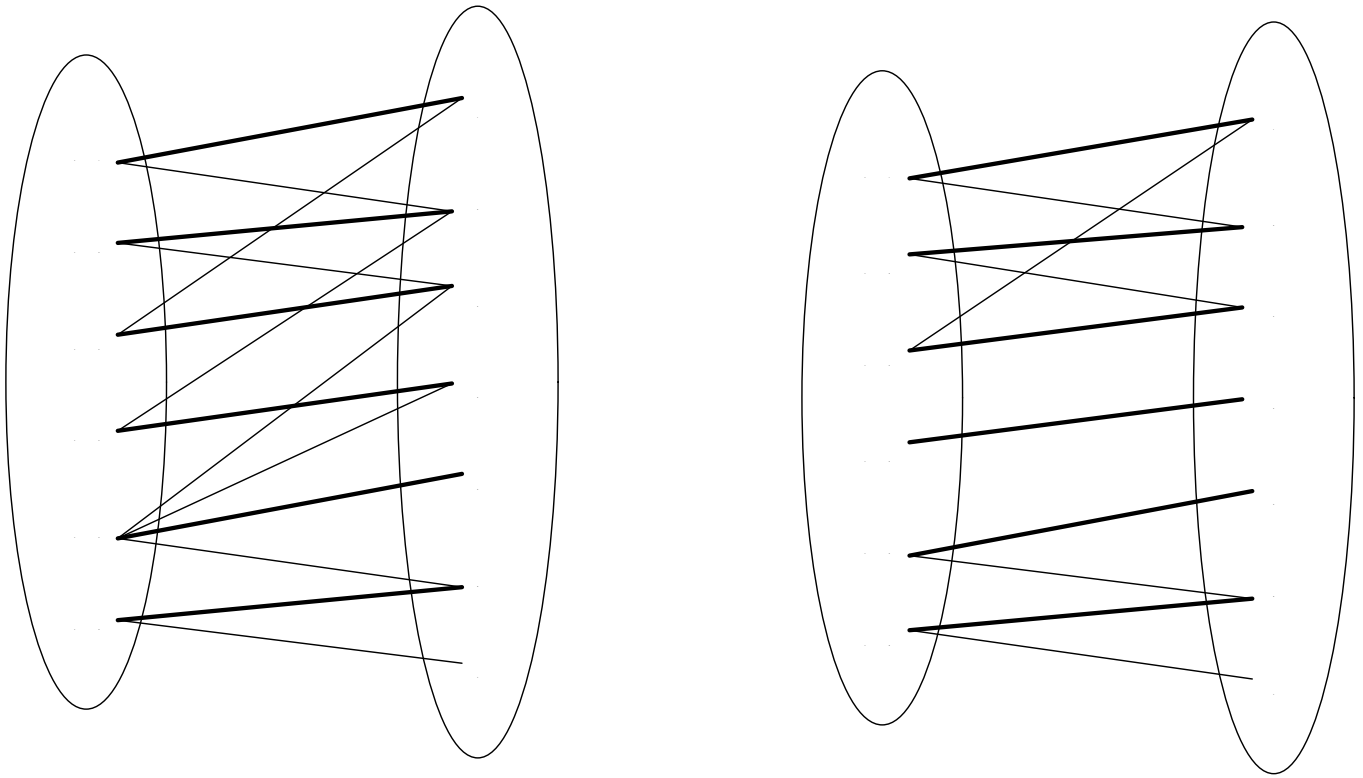
Modélisation par 27 graphes bipartis.

Test si aucune solution n'existe :
taille d'un couplage maximum $< |X|$

Suppression des valeurs n'appartenant à aucune solution :
retrait des sommets x_i,a et test d'existence d'un couplage couvrant $X - x_i$.
$\Longrightarrow$ complexité $O(|A|^2)$

Possibilité d'améliorer la complexité en $O(|A|)$ (Régin, 1994).

Filtrage des domaines



Un couplage couvrant l'ensemble des variables et le résultat du filtrage

Théorème de Hall

Théorème

Un graphe biparti $G = (S_1 \cup S_2, A)$ admet un couplage parfait si et seulement si pour tout sous-ensemble X de S_1 (de S_2, respectivement), le nombre de sommets de S_2 (de S_1, respectivement) adjacents à un sommet de X est supérieur ou égal à la cardinalité de X.

Autre problème : transport

Une entreprise a une usine de grues au sommet s et un entrepôt au sommet t.

Le transport des grues qu'elle fabrique nécessitent l'utilisation de convois exceptionnels. Un ensemble de routes existent entre les sommets s et t chaque route permettant un certain nombre de transports exceptionnels par mois.

L'entreprise aimerait ajuster sa production mensuelle de façon à produire exactement le maximum d'objets qu'elle peut transporter dans son entrepôt chaque mois.

La notion de réseau

Un réseau ou graphe de transport est :

- un graphe *orienté* $G = (S, A)$ pondéré par $c(u, v)$ appelée capacité de l'arc (u, v), toujours positive ou nulle.
- deux sommets particuliers : s et t (respectivement source et puits).
- Pour toute paire de sommets (u, v), si $(u, v) \notin A$, on étend $c(u, v) = 0$.

Par commodité, on suppose que chaque sommet se trouve sur un certain chemin reliant la source au puits.

Ce que l'on cherche

Un flot *net* sur ce graphe est une pondération f des arcs du graphe qui :

- **Respect des capacités** : $f(u, v) \leq c(u, v)$
- **Symétrie** : pour tout $u, v \in S$, $f(u, v) = -f(v, u)$
- **Conservation des flots** : pour tous les sommets $u \in S - \{s, t\}, \sum_{v \in S} f(u, v) = 0$ (tout ce qui rentre dans un sommet en sort).

Valeur d'un flot f, notée $|f| = \sum_{v \in S} f(s, v)$

Analogie avec un flot de matériau (liquide, électricité, communication). capacité = débit maximal.

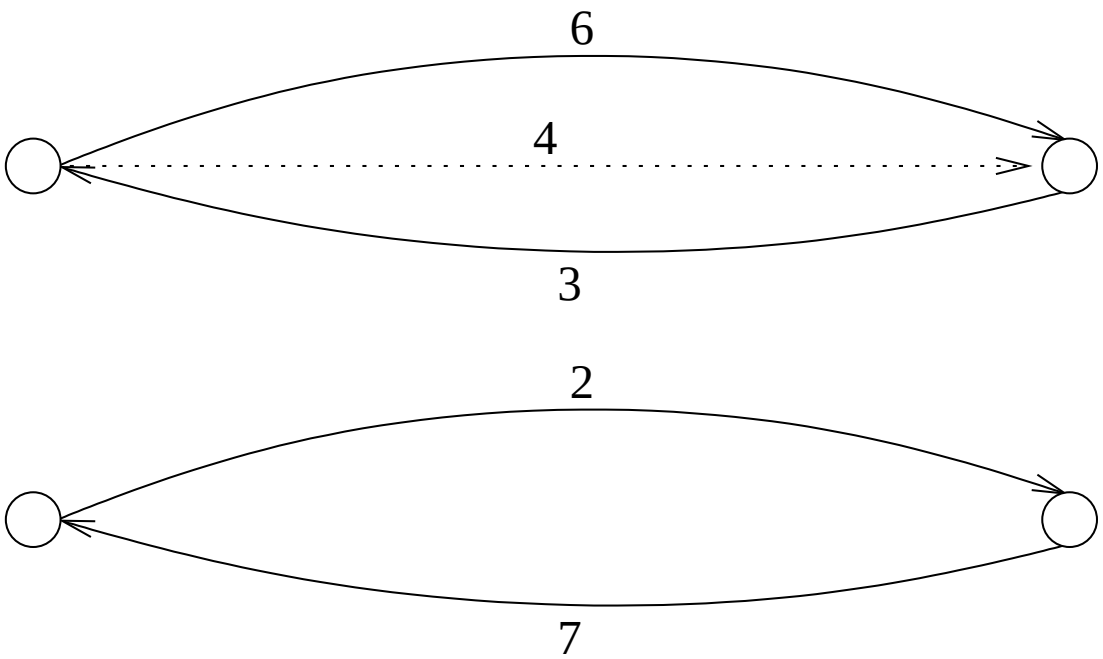
Objectif : trouver un flot de valeur maximum.

La bonne idée

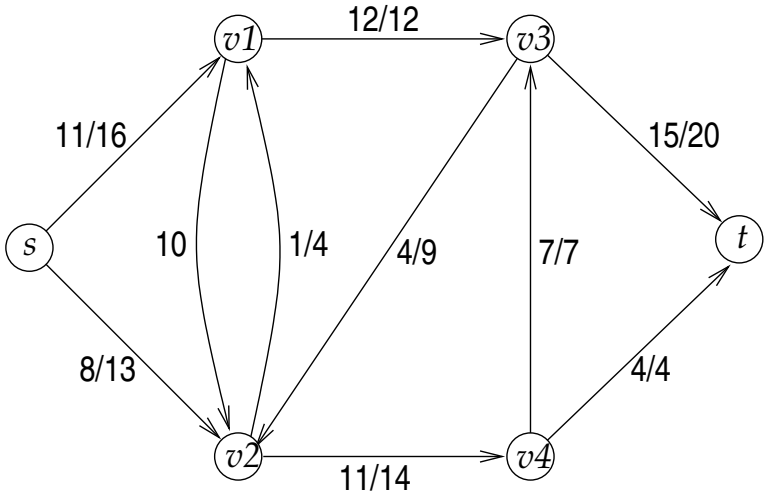
Définition

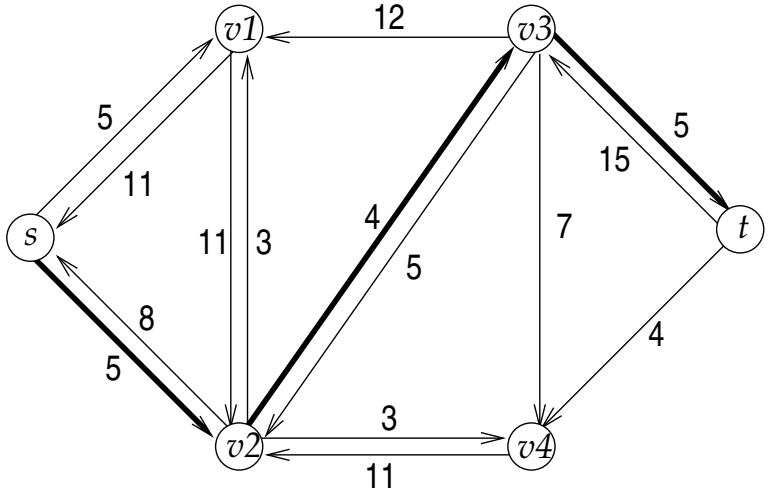
Capacité résiduelle : $c_f(u, v) = c(u, v) - f(u, v)$
(toujours positive ou nulle, par définition).

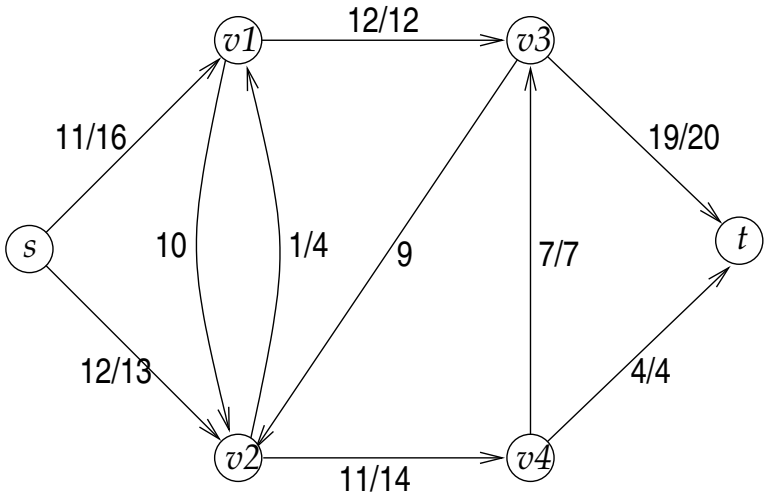
Réseau résiduel : $G_f = (S, A_f)$ *avec*
$A_f = \{(u, v) \in S \times S : c_f(u, v) > 0\}$

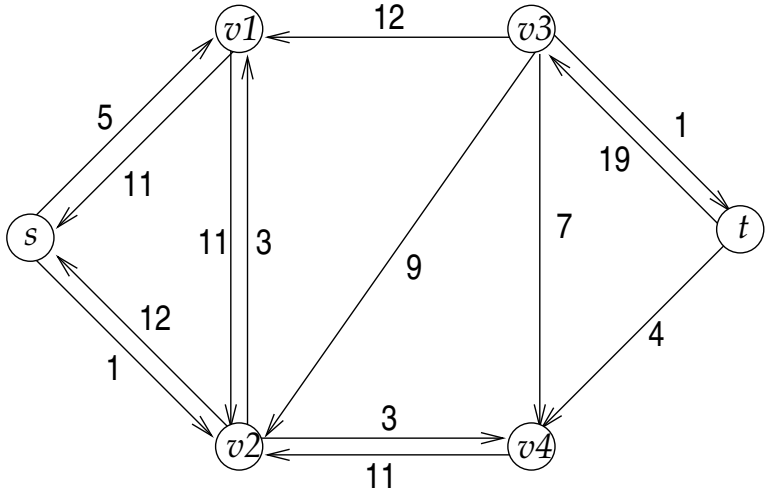


Exemple : flot, réseau résiduel, chemin améliorant









Méthode de Ford-Fulkerson

Procedure *FordFulkerson*(G, s, t)

- **pour chaque** *arc* $(u, v) \in A$ **faire**
  - $f[u, v] \leftarrow 0$;
  - $f[v, u] \leftarrow 0$;
- **tant que** $\exists$ *un chemin* p *de* s *vers* t *dans le réseau résiduel* **faire**
  - $c_p \leftarrow \min\{c_f(u, v) \ : \ (u, v) \in p\}$;
  - **pour chaque** *arc* (u, v) *de* p **faire**
    - $f[u, v] \leftarrow f[u, v] + c_p$;
    - $f[v, u] \leftarrow -f[u, v]$;

Théorème de Ford-Fulkerson

Théorème

Soit G un réseau de source s et de puits t.

Un flot f est maximum si et seulement s'il n'existe pas de chemin améliorant dans le réseau résiduel G_f.

La valeur du flot maximum est égale à une coupe de capacité minimum dans G.

Notation ensembliste et propriétés sur les flots

Définition

Notation de sommation implicite :

$$f(U,V) = \sum_{u \in U} \sum_{v \in V} f(u,v)$$

Conservation des flots : $\forall u \in S \setminus \{s,t\}$, $f(u,S) = 0$.
Valeur du flot $|f| = f(s,S) = f(S,t)$.

Propriété

- *pour tout* $U \subseteq S$, *on a* $f(U,U) = 0$.
- *pour tout* $U, V \subseteq S$, *on a* $f(U,V) = -f(V,U)$.
- *pour* $U, V, W \subseteq S$ *et* $U \cap V = \varnothing$, *alors*
  $f(U \cup V, W) = f(U,W) + f(V,W)$
  $f(W, U \cup V) = f(W,U) + f(W,V)$

Coupe d'un réseau

Définition

Une coupe (E, T) *d'un réseau de transport* $G = (S, A)$ *est une partition de* S *en* E *et* $T = S - E$ *telle que* $s \in E$ *et* $t \in T$.

Capacité de la coupe : $c(E, T) = \sum_{(u,v) \in A : u \in E, v \in T} c(u, v)$

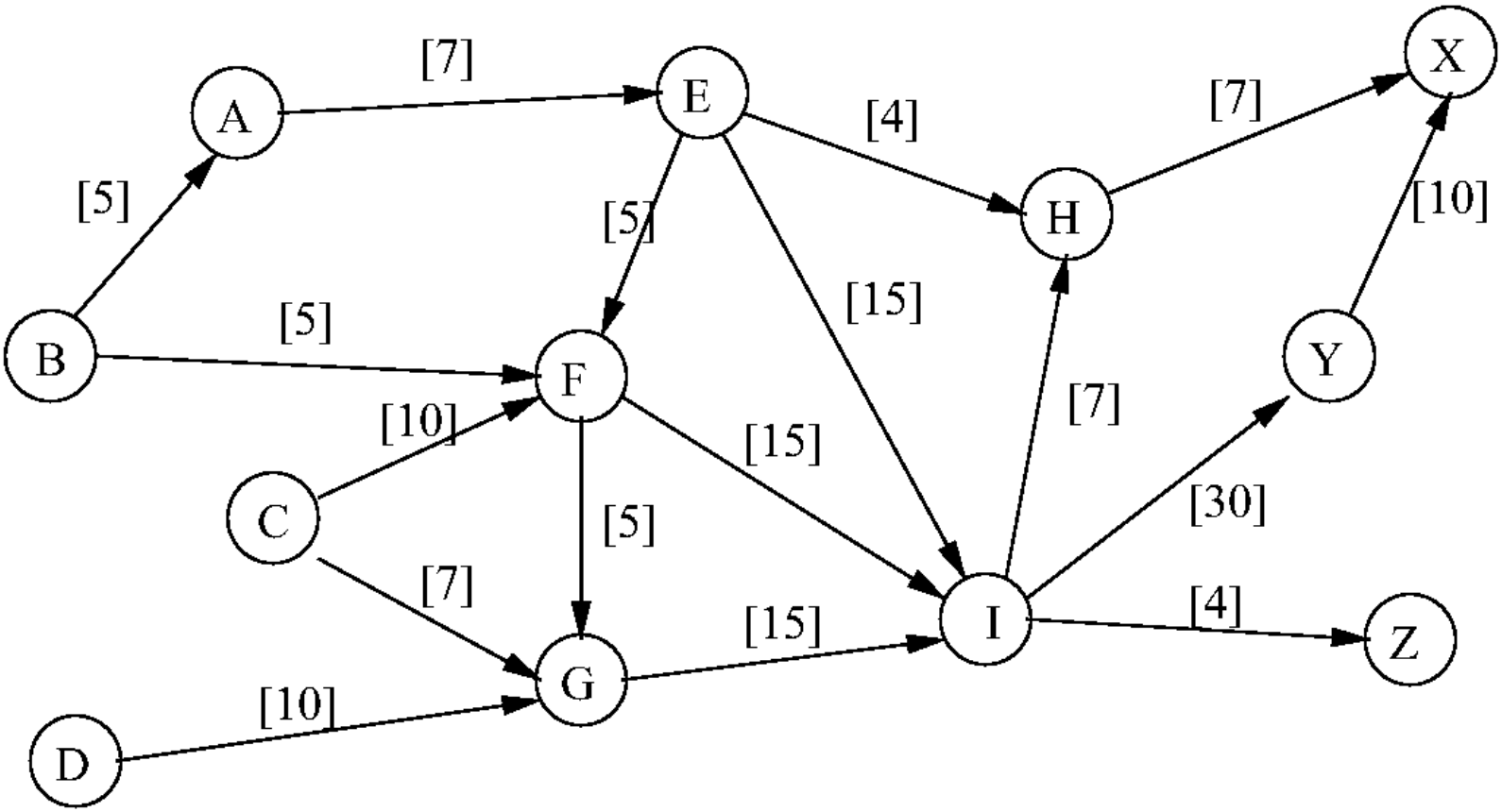
Théorème

Pour toute coupe (E, T) *et tout flot* f,

$$|f| \leq c(E, T)$$

Preuve par récurrence : $f(E, T) = |f|$. Si $|E| = 1$, c'est immédiat. Par récurrence, soit e un élément pris dans $T \setminus \{s, t\}$ et mis dans E. On a $f(E \cup e, T - e) = f(E, T) - f(E, e) + f(e, T) = f(E, T) + f(e, S) = f(E, T)$. Par définition, $f(E, T) \leq c(E, T)$.

Coupe d'un réseau



Preuve du Théorème de Ford-Fulkerson

Théorème

$f^* \equiv \nexists$ *chemin améliorant dans* $G_f \equiv \exists\ c(E,T) = |f^*|$

Preuve :

Hypothèse f flot maximum. Si G_f contient un chemin améliorant alors on peut améliorer le flot f ce qui contredit l'hypothèse.

Hypothèse G_f *ne contient pas de chemin améliorant.*

Soit (E,T) une coupe telle que

$E = \{v \in S \ : \ \exists \text{ chemin } s \rightsquigarrow v \in G_f\}$ et $T = S - E$.

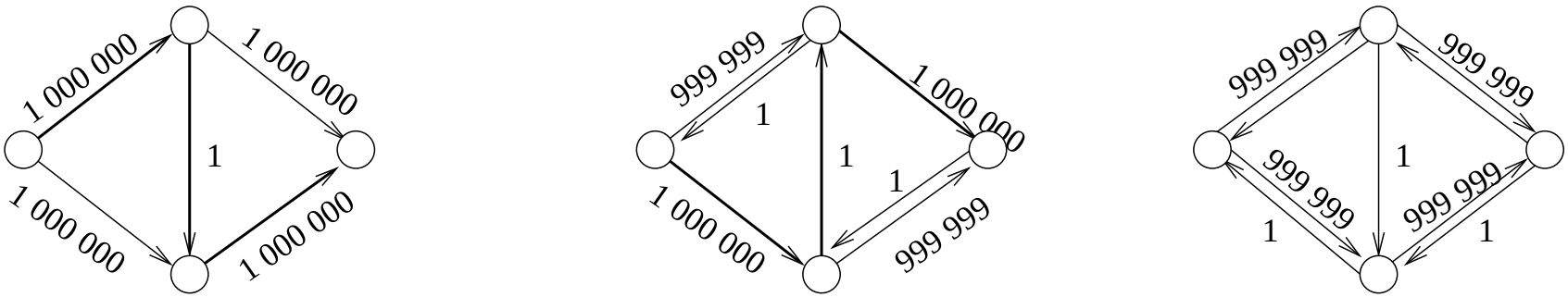
$\forall (u,v) \in E \times T, \qquad f(u,v) = c(u,v)$ (sinon une capacité résiduelle non nulle existe entre u et v et $v \in E$).

$\implies |f| = f(E,T) = c(E,T)$.

Or $|f| \leq c(E',T')$ quelque soit la coupe (théorème précédent), donc f est forcément maximum.

FF peut être lent

Complexité de l'algorithme en $O(|f^*|.|A|)$ où f^* est la valeur du flot maximum et ce pour des capacités entières (théorème de l'intégralité de $|f|$).



Un réseau difficile pour Ford-Fulkerson : réseau initial et réseaux résiduels successifs

Edmonds-Karp : $O(|S|.|A|^2)$ (FF avec recherche en largeur du plus court chemin améliorant).

Goldberg et Tarjan : $O(|S|.|A|.\log(\frac{|S|^2}{|A|}))$

`http://www.boost.org/libs/graph/`

Preuve de complexité d'Edmonds-Karp

Théorème

$\forall v \in S - \{s, t\}$, la distance de plus court chemin $\delta_f(s, v)$ dans G_f augmente de façon monotone après chaque augmentation de flot.

Preuve par l'absurde : $\exists v \in S \setminus \{s, t\}$ tel qu'une augmentation de flot de f à f' provoque $\delta_{f'}(s, v) < \delta_f(s, v)$.
On suppose que $\delta_{f'}(s, v) \leq \delta_{f'}(s, u)$ pour tout $u \in S \setminus \{s, t\}$ tel que $\delta_{f'}(s, u) < \delta_f(s, u)$ (*ie.* v le plus proche de s). Sa contraposée donne : $\delta_{f'}(s, u) < \delta_{f'}(s, v) \implies \delta_f(s, u) \leq \delta_{f'}(s, u)$.
Soit $p' = s \rightsquigarrow u \to v$, un plus court chemin de s à v dans $G_{f'}$. On a $(u, v) \in A_{f'}$ et $\delta_{f'}(s, u) = \delta_{f'}(s, v) - 1$.
Par la contraposée, on a $\delta_f(s, u) \leq \delta_{f'}(s, u)$.
Cas $(u, v) \in A_f$. Alors $\delta_f(s, v) \leq \delta_f(s, u) + 1$ (inégalité triangulaire).
Donc $\delta_f(s, v) \leq \delta_{f'}(s, u) + 1 \leq \delta_{f'}(s, v)$. Contradiction.
Cas $(u, v) \notin A_f$. Il faut un plus court chemin p dans G_f ayant augmenté $f(v, u)$. $\delta_f(s, v) = \delta_f(s, u) - 1 \leq \delta_{f'}(s, u) - 1 \leq \delta_{f'}(s, v) - 2 < \delta_{f'}(s, v)$.
Contradiction.

Preuve de complexité d'Edmonds-Karp

Théorème

L'algorithme d'Edmonds-Karp effectue un nombre total d'augmentations en $O(|S|.|A|)$.

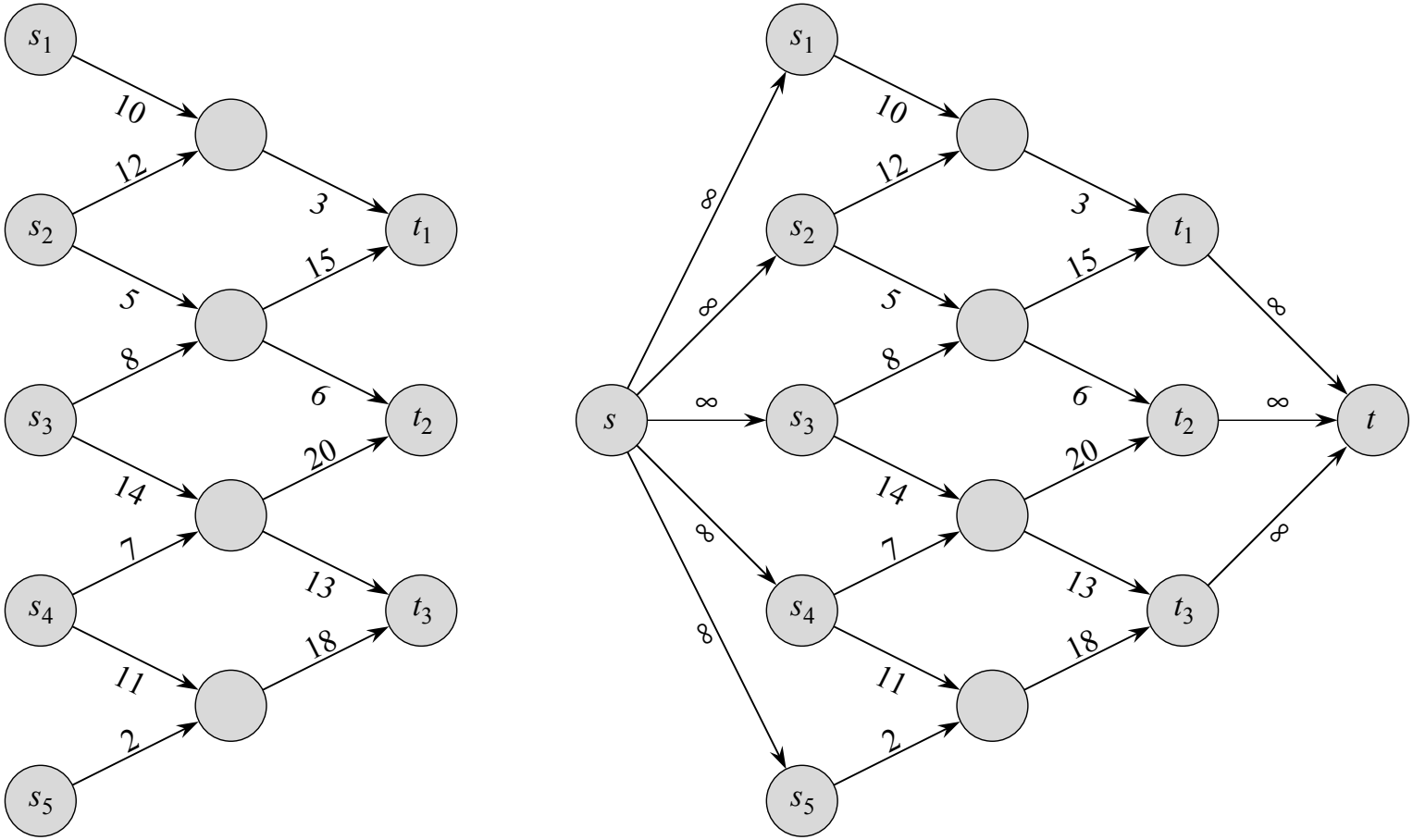
Definition : arc (u, v) *critique* sur un chemin améliorant p de G_f si $c_f(p) = c_f(u, v)$.

$\Longrightarrow$ tout arc critique disparait après avoir augmenté le flot

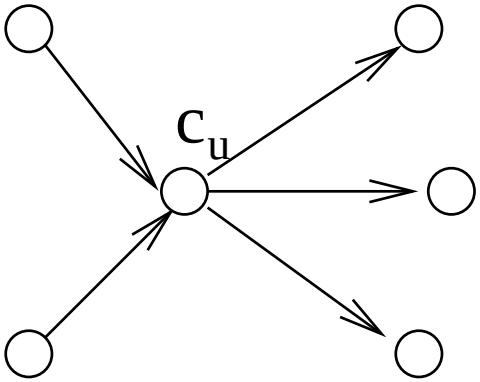
$\Longrightarrow$ il existe au moins un arc critique par chemin améliorant

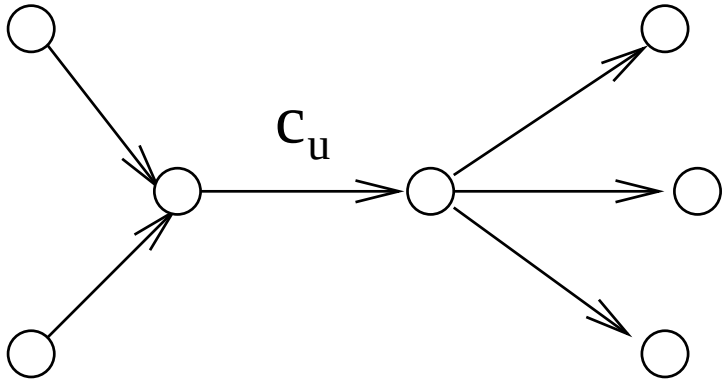
Preuve : chacun des $2.|A|$ *arcs peut devenir critique au plus* $|S|/2 - 1$ *fois.*
Soit $(u, v) \in A$. Lorsque (u, v) devient critique, on a $\delta_f(s, v) = \delta_f(s, u) + 1$ car (u, v) appartient à un plus court chemin.
Ensuite (u, v) réapparaîtra ssi $(v, u) \in p'$, chemin améliorant de $G_{f'}$.
On a $\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1 \geq \delta_f(s, v) + 1 \geq \delta_f(s, u) + 2$ (cf. théorème précédent).
Donc la distance entre u et la source augmente au moins de 2 entre deux moments critiques et ne peut pas dépasser $|S| - 2$. Ainsi (u, v) ne devient critique au plus $|S|/2 - 1$ fois.

Cas d'un problème à sources et puits multiples

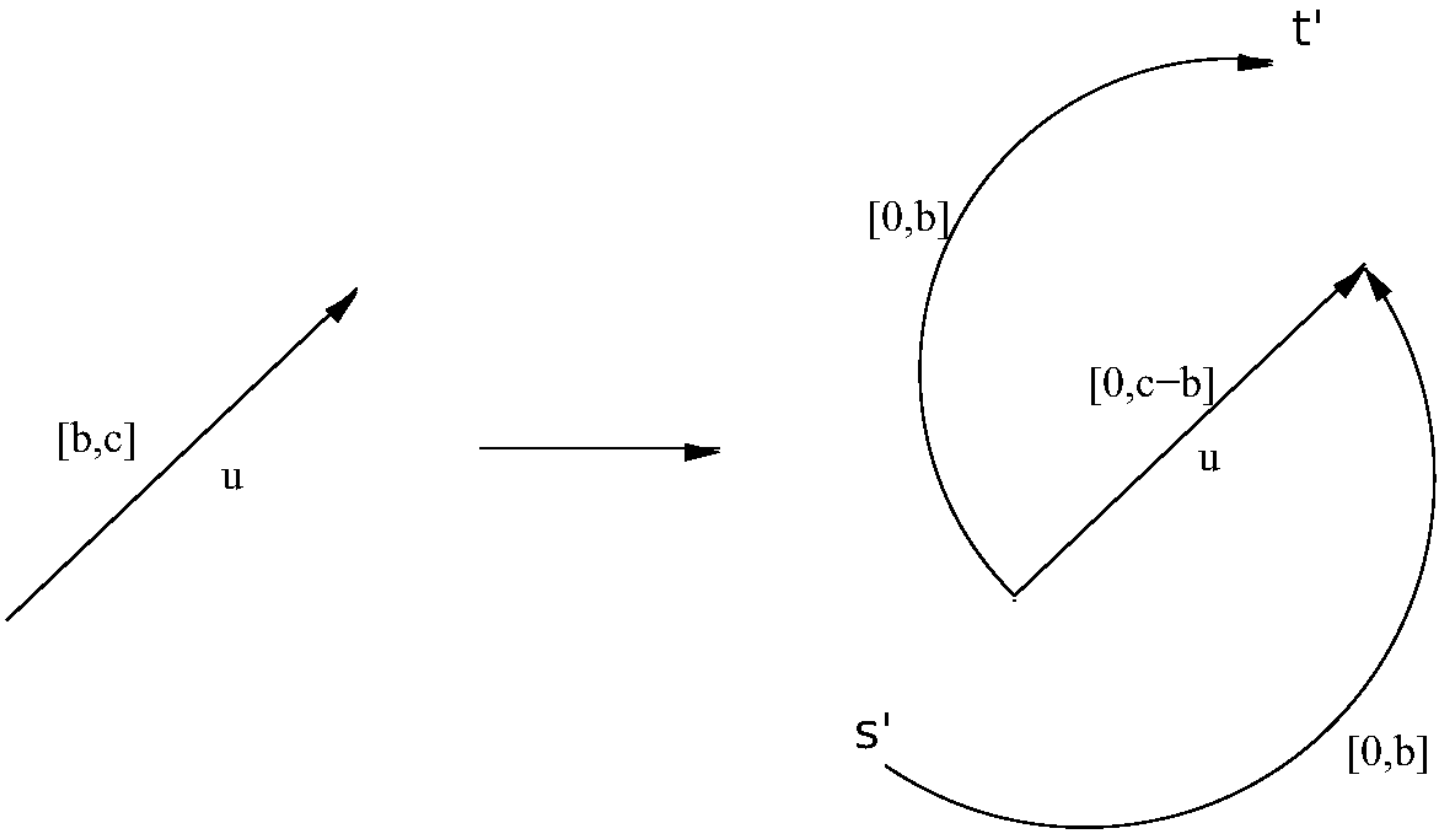


Cas d'un problème à capacités sur les sommets





Cas d'un problème avec demandes et capacités

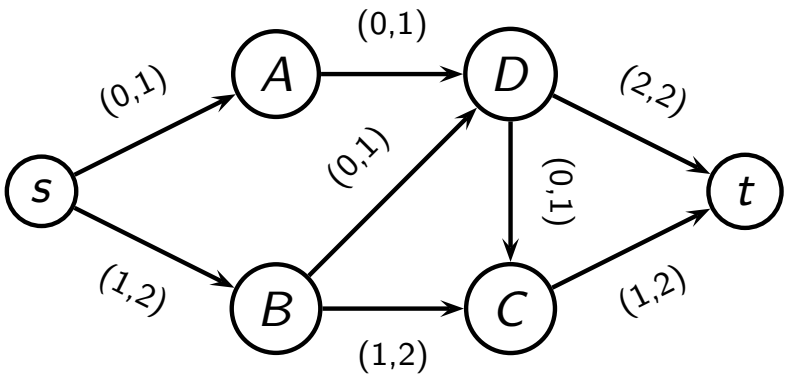


graphe initial

graphe modifié

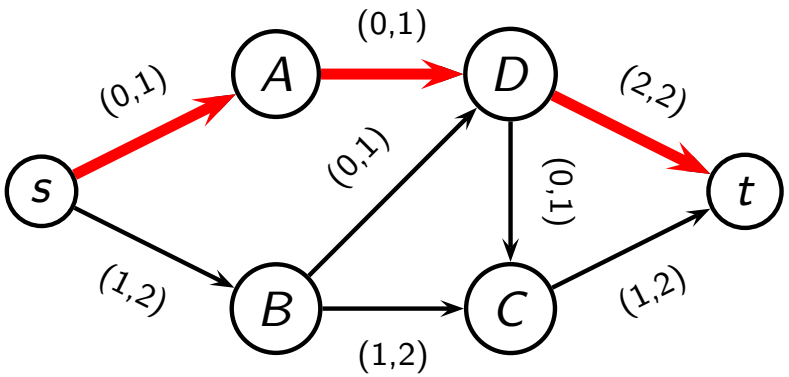
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



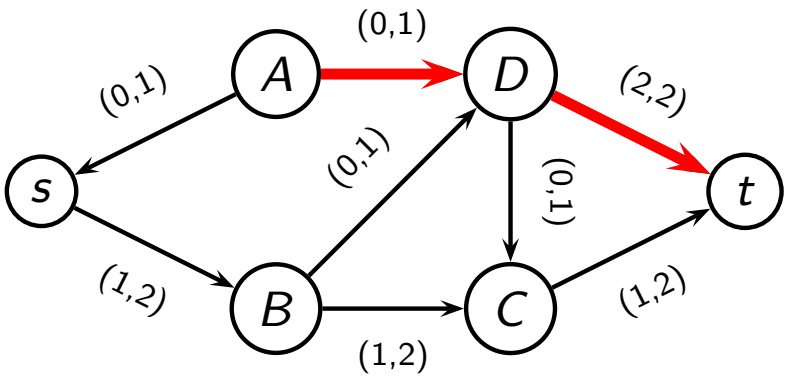
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



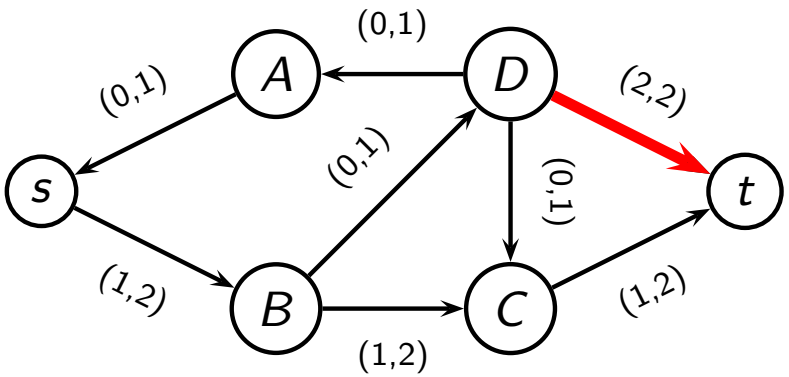
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



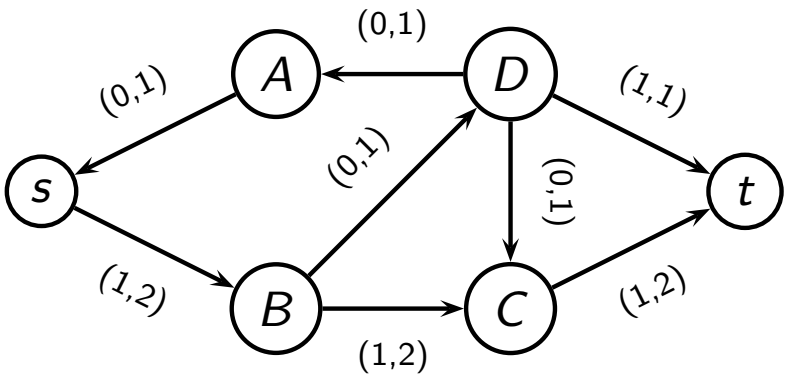
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



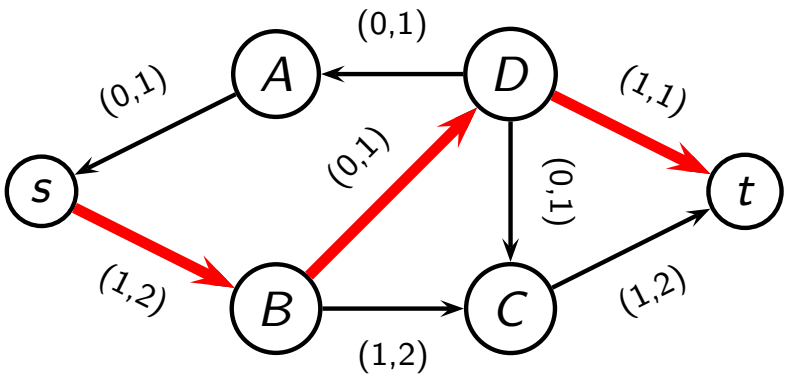
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



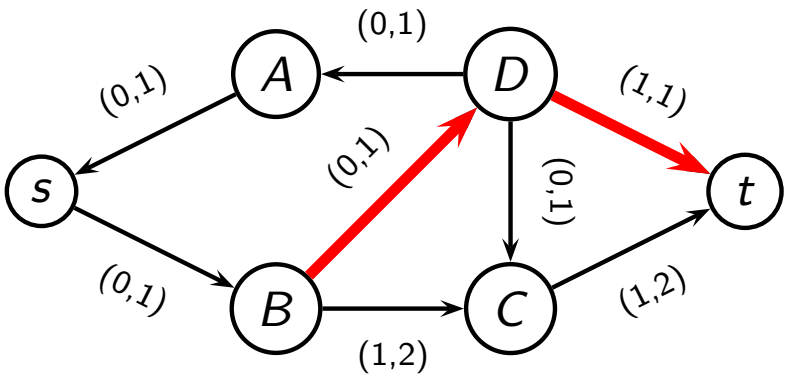
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



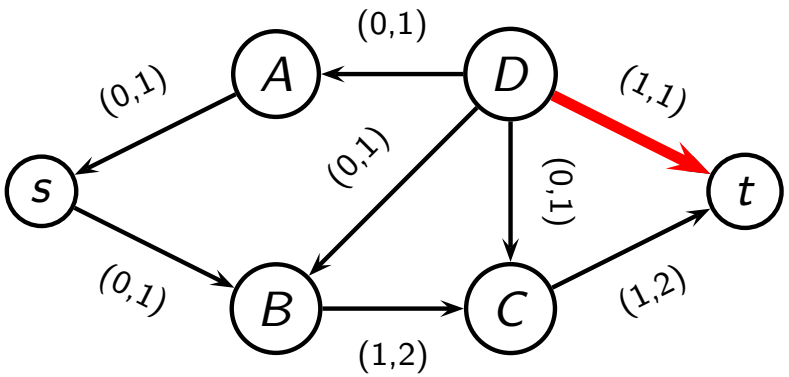
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



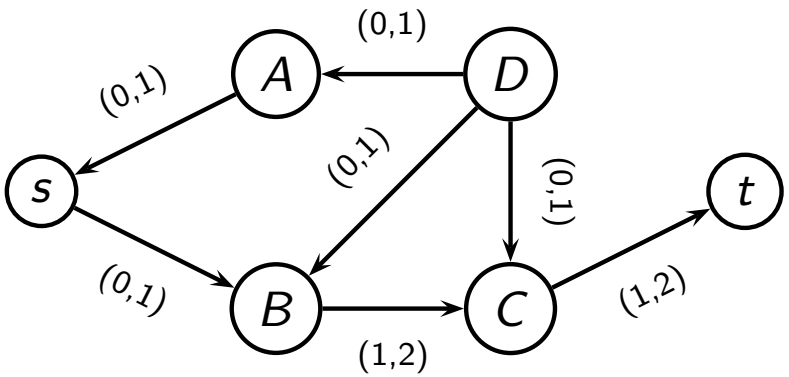
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



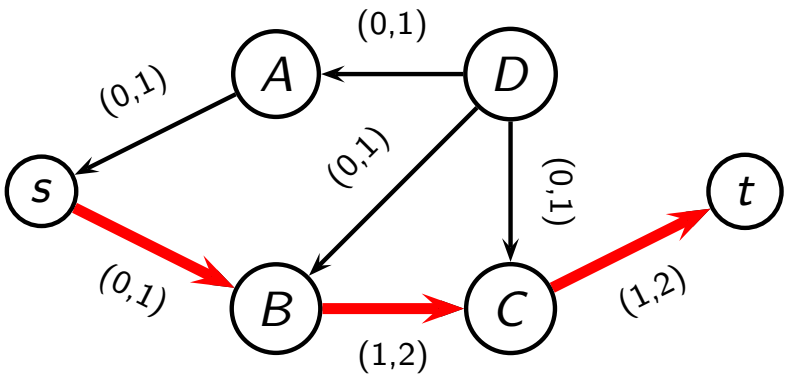
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



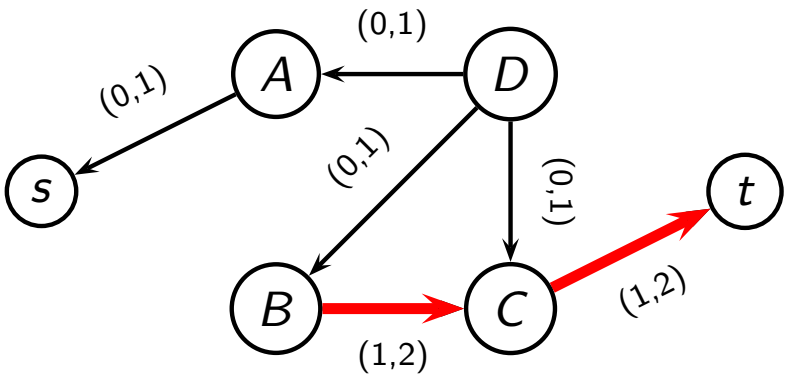
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



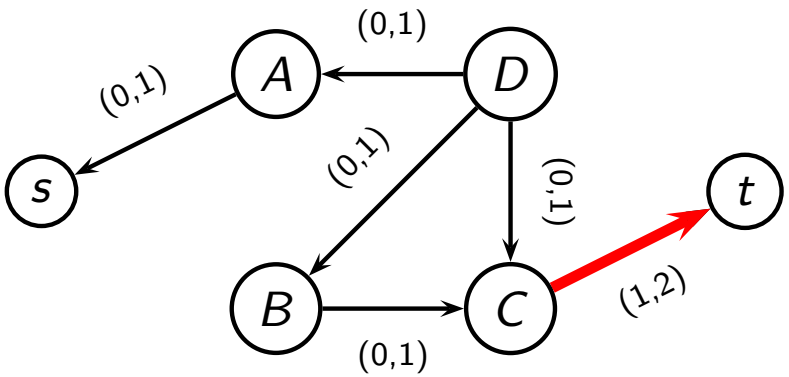
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



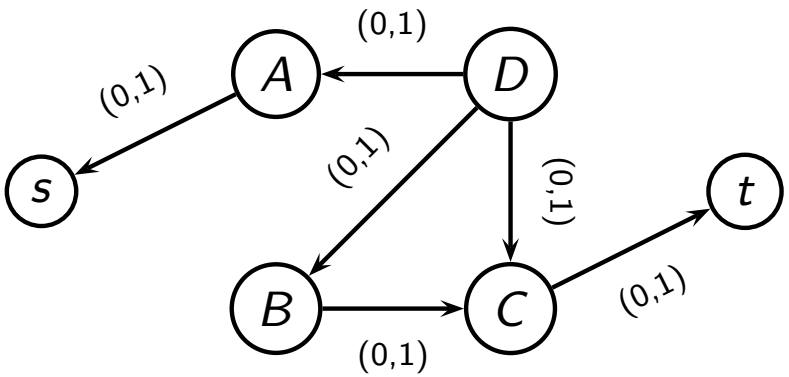
Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



Méthode de Ford-Fulkerson avec demandes et capacités

Si la demande $d(u, v)$ est non nulle et une unité de flot passe dans (u, v), alors diminuer $d(u, v)$ et $c(u, v)$ de 1 dans le réseau résiduel.



Exercice : problème du centre aéré

Un centre aéré propose un ensemble d'activités encadrées par des moniteurs agréés. Réaliser le planning des moniteurs pour satisfaire les contraintes d'encadrement énoncées ci-dessous.

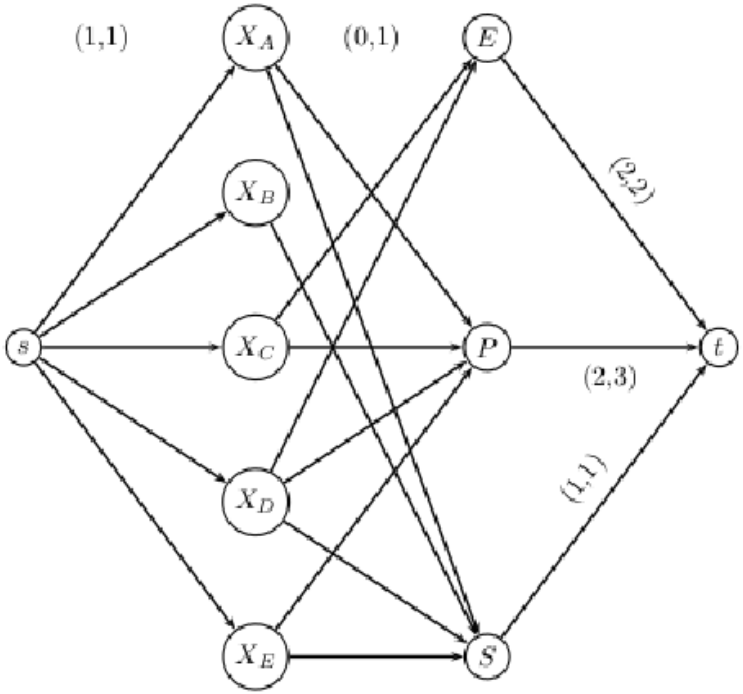
Activité	min	max
Escalade	2	2
Piscine	2	3
Sieste	1	1

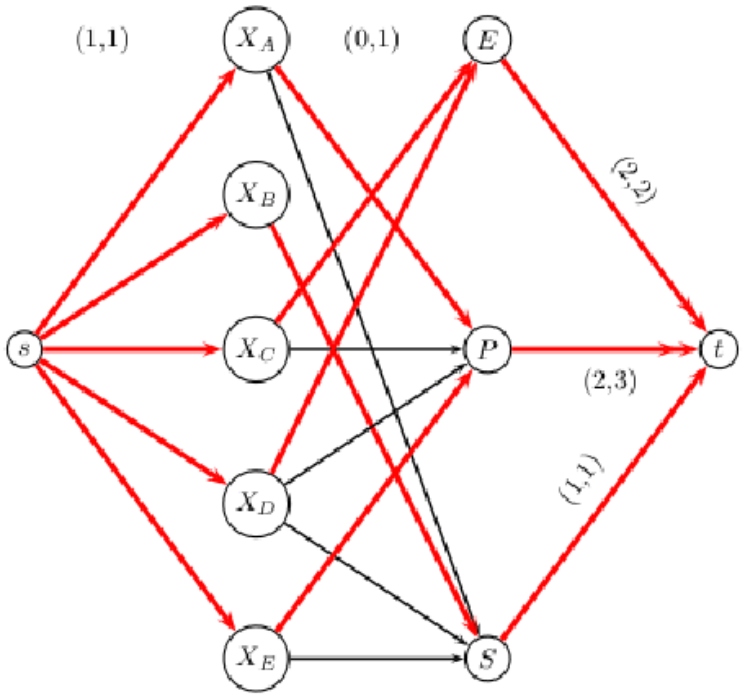
Moniteur	E	P	S
Alice		X	X
Bob			X
Charles	X	X	
Daphné	X	X	X
Eve		X	X

Solution : problème du centre aéré

Activité	min	max
Escalade	2	2
Piscine	2	3
Sieste	1	1

Moniteur	E	P	S
Alice		X	X
Bob			X
Charles	X	X	
Daphné	X	X	X
Eve		X	X





Exercice : problème de fermeture de bénéfice maximal

Un certain nombre de tâches $j \in J$ avec un bénéfice associé $b(j)$ sont définies. Le bénéfice peut être négatif (une perte).

Les tâches sont soumises à des contraintes de fermeture : le choix d'une tâche j_1 peut impliquer le choix d'une tâche j_2. On représente cette relation dite de fermeture (ou implication) par un graphe dont les sommets sont les tâches, et où il existe un arc (j_1, j_2) si et seulement si le le choix de j_1 implique le choix de j_2. L'objectif est de trouver un ensemble de tâches cohérentes (respectant les contraintes) et de bénéfices maximal. Plus formellement :

Déterminer un sous ensemble $U \subset J$ tel que (1) aucun arc ne sorte de U (cohérence) et (2) $\sum_{j \in U} b(j)$ soit maximum.

Exercice : modéliser ce problème par un graphe de transport tel qu'un flot maximum (ou une coupe minimum) dans ce graphe donne une solution optimale du problème.

Solution : fermeture de bénéfice maximal

Réseau dont les sommets sont les tâches plus source s et puits t.

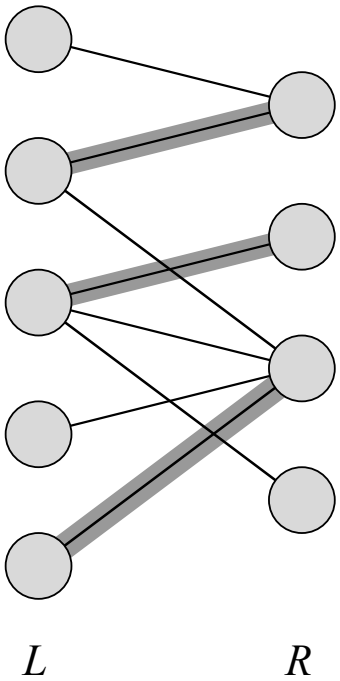
- Cas $b(j) \geq 0$, ajouter l'arc (s, j) de capacité $b(j)$
- Cas $b(j) < 0$, ajouter l'arc (j, t) de capacité $-b(j)$
- Cas $j_1 \implies j_2$, ajouter l'arc (j_1, j_2) avec une capacité infinie

Soit (E, T), coupe de capacité finie, donc cohérente.
Soit $H^+ = \{j \in H : b(t) \geq 0\}$ et $H^- = \{j \in H : b(t) < 0\}$ $\quad \forall H \subseteq J$

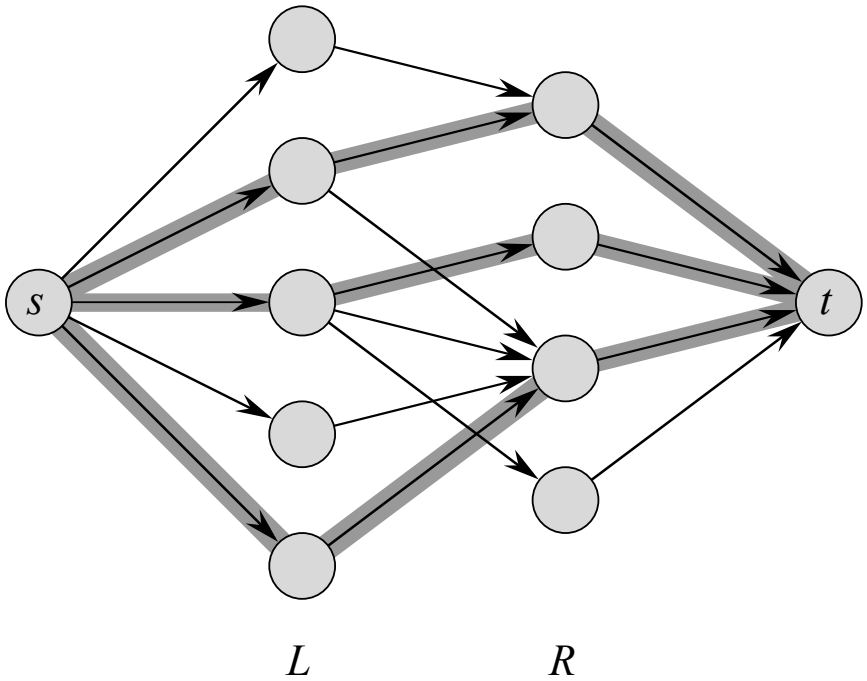
$$
\begin{aligned}
c(E,T) &= \sum_{j \in T^+} c(s,j) + \sum_{j \in E^-} c(j,t) && (1)\\
&= \sum_{j \in T^+} b(j) - \sum_{j \in E^-} b(j) && (2)\\
&= \sum_{j \in J^+} b(j) - \sum_{j \in E^+} b(j) - \sum_{j \in E^-} b(j) && (3)\\
&= \sum_{j \in J^+} b(j) - \sum_{j \in E} b(j) && (4)
\end{aligned}
$$

Minimiser la coupe $\equiv$ maximiser le bénéfice de E ($\sum_{j \in J^+} b(j)$ constant)

Lien entre couplage maximum dans un graphe biparti et flot maximum



(a)



(b)

Complexité de Ford-Fulkerson sur un graphe biparti en $O(|S|.|A|)$ car $f^* \leq \frac{|S|}{2}$.

Flot maximum de coût minimum

Réseau $G = (S, A)$ pondéré pour chaque arc $(u, v) \in A$ par :

- une capacité $c(u, v)$
- un *coût de transport unitaire* $w(u, v)$

Un flot *brut* (perte de la symétrie si $w(u, v) \neq w(v, u)$) sur ce graphe est une pondération g des arcs du graphe qui :

- **Respect des capacités** : $g(u, v) \leq c(u, v)$
- **Conservation des flots** :
  $\forall u \in S - \{s, t\}, \quad \sum_{(u,v)\in A} g(u, v) = \sum_{(v,u)\in A} g(v, u)$

Coût total d'un flot g, noté $K = \sum_{(u,v)\in A} w(u, v).g(u, v)$

Objectif : parmi les flots de valeur maximum, trouver celui de coût minimum

Graphe des capacités résiduelles

Définition

Soit un réseau $G = (S, A)$ *de capacité* c *et de coût* w *et un flot brut* g *sur* G.
Le graphe des capacités résiduelles avec coûts $G_g = (S, A_g)$ *tel que :*

- $(u, v) \in A$ *tel que* $g(u, v) < c(u, v) \implies (u, v)^+ \in A_g$ *avec* $c_g(u, v) = c(u, v) - g(u, v)$ *et* $w_g(u, v) = w(u, v)$
  (appelé arc avant*)*
- $(u, v) \in A$ *tel que* $g(u, v) > 0 \implies (v, u)^- \in A_g$ *avec* $c_g(v, u) = g(u, v)$ *et* $w_g(v, u) = -w(u, v)$
  (appelé arc inverse*)*

Chemin améliorant p de s à t dans G_g :
augmenter le flot de $c_g(p) = \min_{(u,v)\in p} c_g(u, v)$ sur chaque arc avant de p et diminuer de $c_g(p)$ sur chaque arc inverse.
$\implies$ coût du flot augmente de $w_g(p) = c_g(p) \times$ longueur du chemin p

Circuit de coût strictement négatif

Théorème

Un flot brut g de valeur $|g|$ est de coût minimum parmi tous les flots de valeur $|g|$ ssi il n'existe pas un circuit négatif dans G_g.

Preuve par la contraposée ($\Longrightarrow$) : S'il existe un circuit négatif c dans G_g, modifier le flot g selon ce circuit en augmentant g de 1 le long des arcs avant du circuit et en le diminuant de 1 pour les arcs inverses.

$\Longrightarrow$ valeur du flot inchangée et coût diminué de $w_g(c)$.

Deux approches possibles :

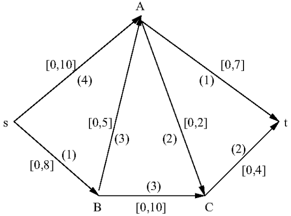
1. Partir d'un flot d'une valeur désirée et le rendre minimum en supprimant progressivement les circuits négatifs par augmentation locale du flot dans les circuits.
2. Construire une série de flots de valeurs croissantes, mais tous de coût minimum pour leur valeur (sans circuit négatif).

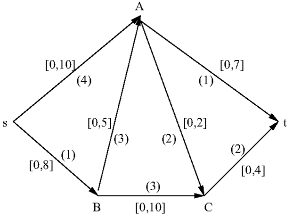
Algorithme de Busacker et Gowen

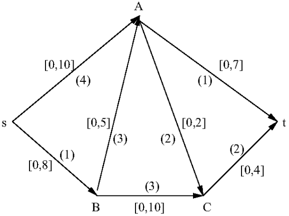
Procedure *BusackerGowen*(G, s, t, c, w)

$F \leftarrow 0$;
$W \leftarrow 0$;
pour chaque *arc* $(u, v) \in A$ **faire**
  $g[u, v] \leftarrow 0$;
tant que $\exists$ *un chemin* p *de coût minimum* z *de* s *à* t *dans* G_g **faire**
  $c_p \leftarrow \min\{c_g(u, v) \ : \ (u, v) \in p\}$;
  $F \leftarrow F + c_p$;
  $W \leftarrow W + c_p.z$;
  pour chaque *arc avant* $(u, v)^+$ *de* p **faire**
    $g[u, v] \leftarrow g[u, v] + c_p$;
  pour chaque *arc inverse* $(u, v)^-$ *de* p **faire**
    $g[v, u] \leftarrow g[v, u] - c_p$;

Exercice


A
s
t
B
C
[0,10]
(4)
[0,7]
(1)
[0,5]
(3)
(2)
[0,2]
(1)
[0,8]
(2)
[0,4]
(3)
[0,10]


A
s
t
B
C
[0,10]
(4)
[0,7]
(1)
[0,5]
(3)
(2)
[0,2]
(1)
[0,8]
(2)
[0,4]
(3)
[0,10]

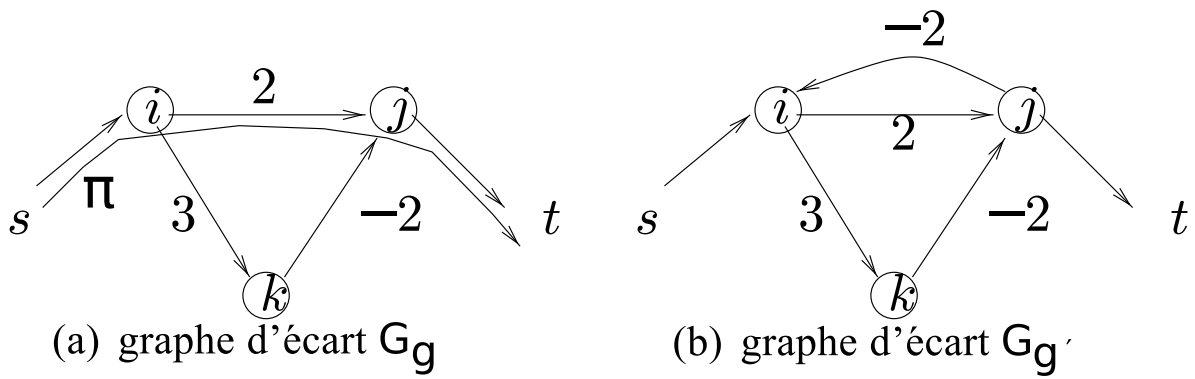

A
s
t
B
C
[0,10]
(4)
[0,7]
(1)
[0,5]
(3)
(2)
[0,2]
(1)
[0,8]
(2)
[0,4]
(3)
[0,10]

Preuve de correction de Busacker et Gowen

Théorème

A chacune des itérations, le flot courant est de coût minimum parmi tous ceux de valeur F.

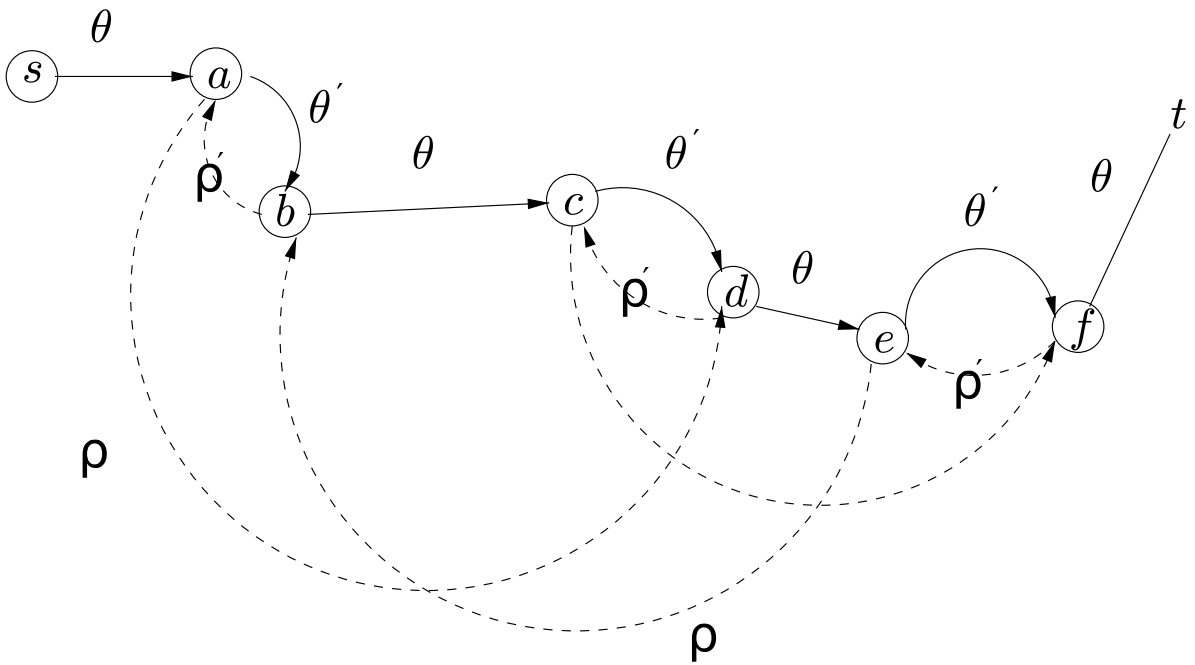
Preuve par récurrence sur la valeur du flot : Vrai pour un flot nul.
Soit un flot g de coût minimum et un chemin améliorant π de coût minimum conduisant à un flot g' sous-optimal.
Donc il existe un circuit négatif ρ dans $G_{g'}$ (et pas dans G_g) passant par (i, j) tel que $(j, i) \in \pi$.



(a) graphe d'écart G_g (b) graphe d'écart $G_{g'}$

Suite de la preuve de Busacker et Gowen

Soit $\pi' = \{(u, v) \in \pi : (v, u) \in \rho\}$ et $\rho' = \{(v, u) \in \rho : (u, v) \in \pi\}$.
Alors $\theta = (\pi - \pi') \cup (\rho - \rho')$ forme un *chemin de s à t* et une union de circuits (cas $|\pi'|$ pair) dans $G_{g'}$ et G_g.
Ces circuits sont forcément de coûts positifs (dans G_g).
On a $w(\pi') = -w(\rho')$ par construction de $G_{g'}$, donc
$w(\theta) = w(\pi) + w(\rho)$.
Puisque $w(\rho) < 0$ alors $w(\theta) < w(\pi)$.
Donc il existe un chemin de coût inférieur à π dans G_g. Contradiction !



Complexité de l'algorithme de Busacker et Gowen

Soit $B = \max_{(u,v)\in A} c(u,v)$ la capacité maximale du réseau.

Alors la capacité de toute coupe est au plus de $n = B.|S|$
$\Longrightarrow$ au plus n itérations si $c_p = 1$ pour tous les plus courts chemins trouvés.

Calculer un chemin de coût minimal en présence de coûts négatifs (mais pas de circuit négatif) dans le graphe résiduel : algorithme de Bellman-Ford-Moore en $O(|S|.|A|)$

Donc complexité de Busacker et Gowen : $O(|S|^2.|A|.B)$ (pseudo-polynomial)

Goldberg et Tarjan : $O(|S|^2.|A|^3.\log_2(|S|))$
Orlin : $O(|A|.\log(|S|).(|A| + |S|.\log|S|))$

Exercice : problème de transport par cargo

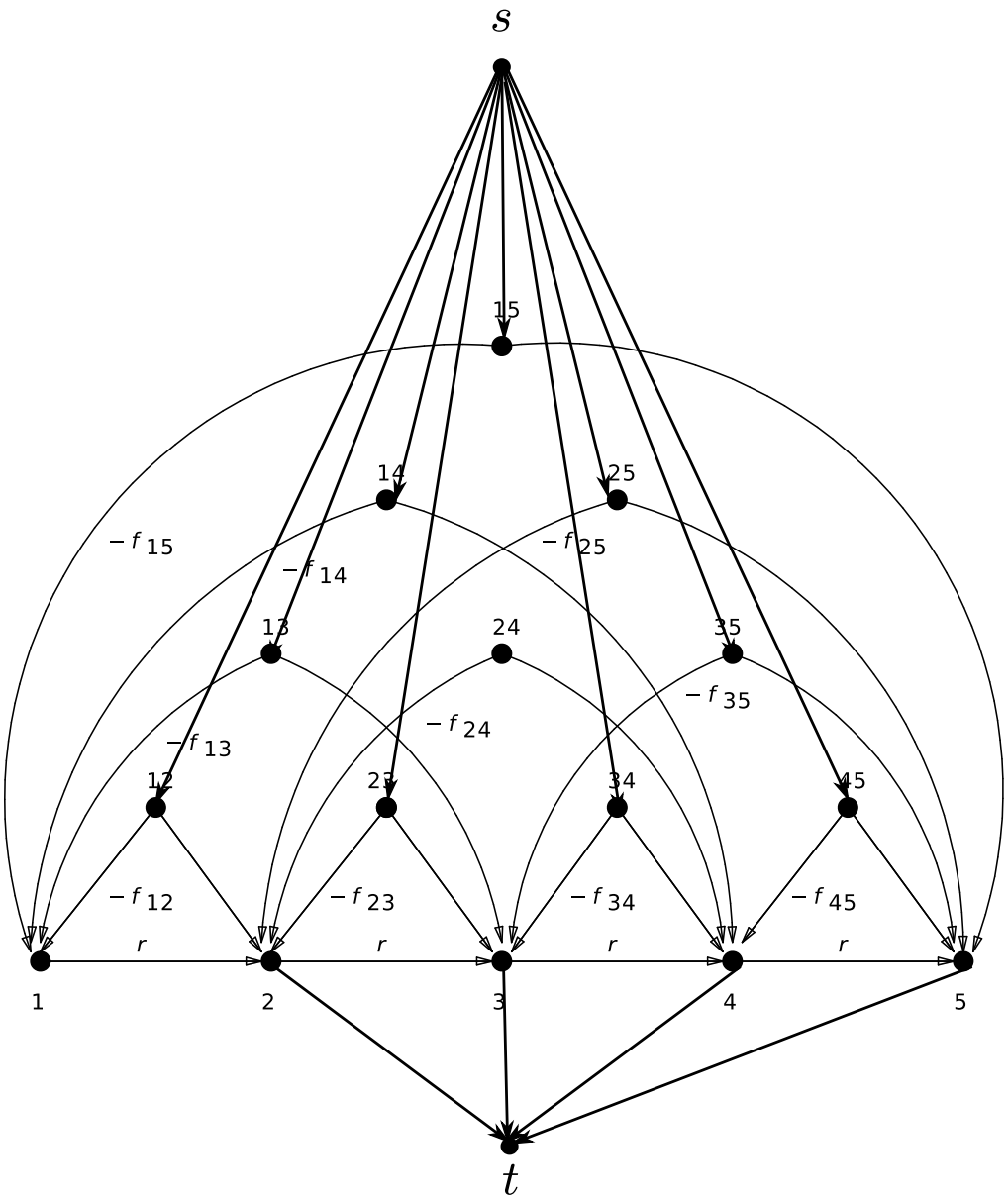
Une petite entreprise de transport utilise un bateau qui est capable de transporter r unités d'un bien donné. Le bateau suit une longue route avec plusieurs arrêts entre le début et la fin de son voyage. A chacun de ces ports, il est possible de décharger et de recharger des unités.

Chaque port i a une quantité b_{ij} à transporter vers le port $j > i$ (les ports sont numérotés dans l'ordre de leur visite). Soit f_{ij} le revenu obtenu en transportant un container de i à j.

Le but de l'entreprise est de décider combien de containers il faut charger à chaque port de façon à maximiser son revenu et sans dépasser la capacité du bateau.

Modéliser ce problème sous forme d'un réseau de transport avec coûts.

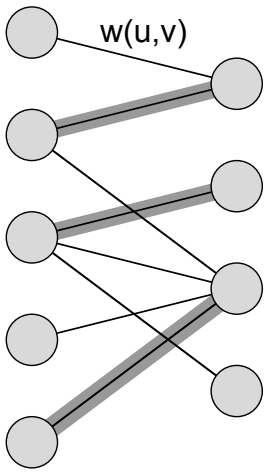
Solution


s
15
14
25
13
24
35
12
23
34
45
1
2
3
4
5
t
$-f_{15}$
$-f_{14}$
$-f_{25}$
$-f_{13}$
$-f_{24}$
$-f_{35}$
$-f_{12}$
$-f_{23}$
$-f_{34}$
$-f_{45}$
r
r
r
r

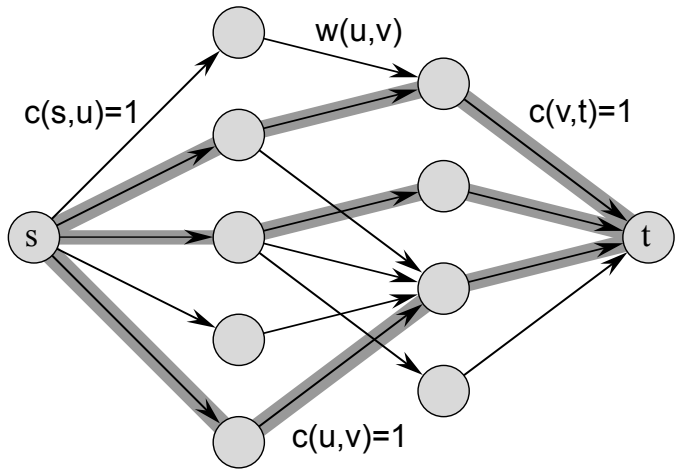
Lien entre problème d'affectation, couplage parfait pondéré et flot maximum de coût minimum

Définition

Soit un graphe biparti $G = (S_1 \cup S_2, A)$ *pondéré pour chaque arête* $(u, v) \in A$ *par un coût* $w(u, v)$ *et tel que* $|S_1| = |S_2| = n$. *Le* problème d'affectation *consiste à trouver un couplage parfait* C *de coût minimum* $w(C) = \sum_{(u,v)\in C} w(u, v)$.



(a)



(b)

Busacker et Gowen : $O(n^4)$; algorithme Hongrois : $O(n^3)$