Ecole d'ingénieurs du Canton de Vaud

INFORMATIQUE ORIENTATION LOGICIELS

ÉLÉMENTS DE LA THÉORIE DES GRAPHES

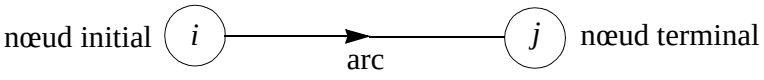
Prof. É. D. Taillard

DÉFINITIONS

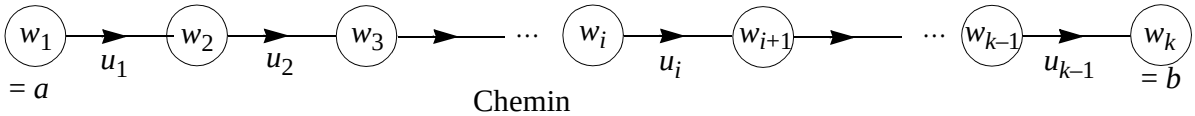
Un *graphe* G est constitué d'un ensemble X de *sommets* ou *nœuds* et d'une famille U d'éléments appelés *arcs*.

Deux sommets sont *adjacents* s'il existe un arc entre eux.

Chaque arc est caractérisé par un couple ordonné (i, j) de nœuds. i est l'extrémité *initiale* et j l'extrémité *terminale*.



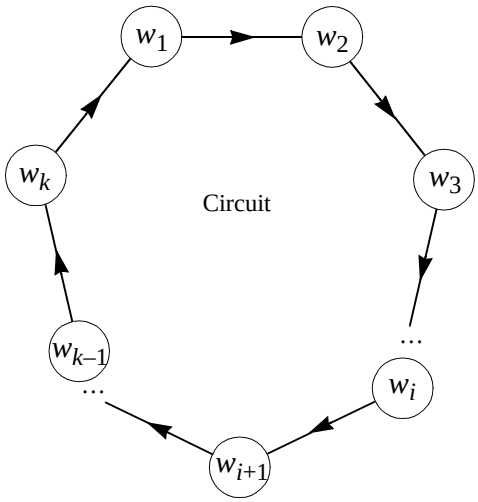
Un *chemin* reliant les sommets a et b est un graphe de sommets $a = w_1, w_2, \ldots, w_k = b$ et d'arcs $u_1, u_2, \ldots, u_{k-1}$ tels que $u_i = (w_i, w_{i+1})$.



Chemin

DÉFINITIONS (2)

Si on ajoute un arc entre w_k et w_1, on obtient un *circuit* (chemin fermé).



Graphe non orienté

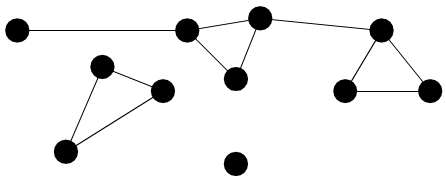
Dans certains cas, on ne se préoccupe pas de l'orientation des arcs. On parle alors d'*arête* (= arc non orienté), de *chaîne* (= chemin non orienté) et de *cycle* (= circuit non orienté)

DÉFINITIONS (3)

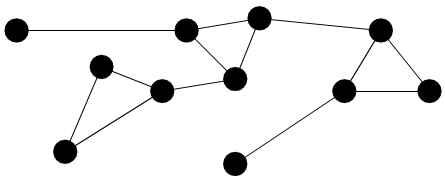
Un graphe est *connexe* s'il existe une chaîne entre toute paire de sommets.

Un graphe sans cycle est une *forêt*.

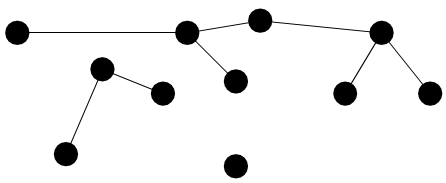
Un graphe connexe sans cycle est un *arbre*.



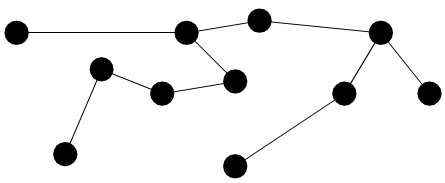
Graphe non connexe
(3 *composantes connexes*)



Graphe connexe



Forêt



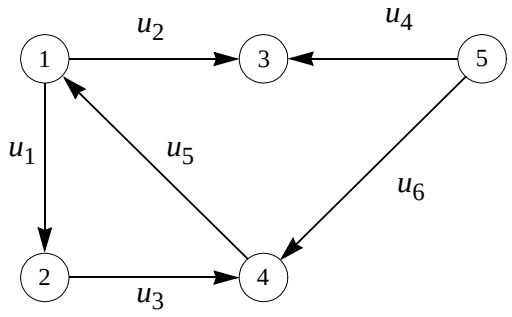
Arbre

REPRÉSENTATION D'UN GRAPHE

a) Matrice d'incidence sommet-arc.

Soit $G = (X, U)$, un graphe à n sommets et m arcs. On lui associe une matrice A à n lignes (correspondant aux sommets) et m colonnes (associées aux arcs). $a_{ik} = \begin{cases} 1 & \text{si } i \text{ est l'extrémité terminale de } u_k \\ -1 & \text{si } i \text{ est l'extrémité initiale de } u_k \\ 0 & \text{sinon} \end{cases}$

Exemple



Sommet	Arc u_1	u_2	u_3	u_4	u_5	u_6
1	–1	–1			1	
2	1		–1			
3		1		1		
4			1		–1	1
5				–1		–1

b) Listes d'adjacence

Un graphe peut être représenté par 2n listes : n listes de successeurs et n listes de prédécesseurs.

Exemple

Sommet	Listes des prédécesseurs	Listes des successeurs
1	4	2, 3
2	1	4
3	1, 5	∅
4	2, 5	1
5	∅	3, 4

L'utilisation de l'une ou l'autre de ces représentations dépend du graphe que l'on considère et de l'utilisation que l'on désire en faire. Une formulation mathématique d'un problème sur un graphe sera par exemple souvent représenté par une matrice d'incidence alors que la représentation interne dans un programme d'un graphe peu dense (qui a peu d'arcs) se fera souvent à l'aide de listes d'adjacence.

EXPLORATION D'UN GRAPHE

Question : Quels sont les sommets que l'on peut atteindre depuis un sommet de départ *a* ?

Algorithme d'exploration

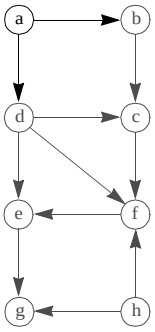
```
Donnée: G avec listes S(n) des successeurs de u; a: un sommet particulier
Résultat: G avec sommets atteignables à partir de a marqués.
Début:
   Q = {a}
   Répéter
      Choisir u ∈ Q;
      Retirer u de Q;
      Marquer u;
      ∀ v ∈ S(u):
         Si v n'est pas marqué: introduire v dans Q
   Tant que Q ≠ ∅
Fin.
```

L'ensemble *Q* peut être géré en pile ou en queue.

Dans le premier cas, on a une exploration en *profondeur* (on va aussi loin que possible avant de revenir en arrière).

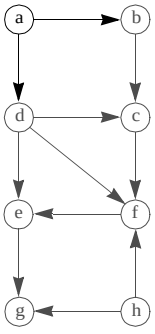
Dans le second cas, on a une exploration en *largeur.*

Exemple d'exploration en profondeur (pile)



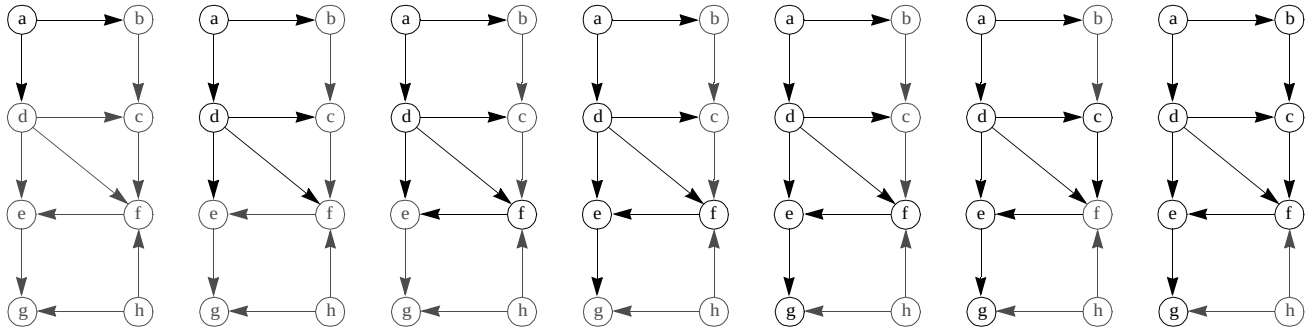
Q = {a}

Exemple d'exploration en largeur (queue)



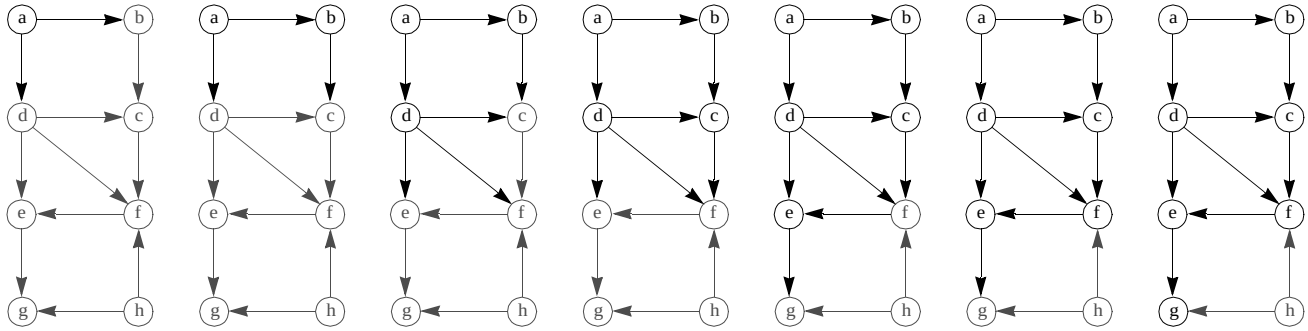
Q = {a}

Exemple d'exploration en profondeur (pile)



Q = {a} {b, d} {b, c, e, f} {b, c, e} {b, c g} {b, c} {b}

Exemple d'exploration en largeur (queue)



Q = {a} {b, d} {d, c} {c, e, f} {e, f} {f, g} {g}

COMPLEXITÉ D'UN ALGORITHME

Notation O(.)

On dit que les fonctions $f(m)$ et $g(m)$ satisfont $f(m) = O(g(m))$

s'il existe une constante $c > 0$ telle que pour tout m assez grand on a $f(m) \leq c \cdot g(m)$.

Exemple :

$f(m) = 1/m + \log(m) + 300m + 5m^2$ est en $O(m^2)$

car pour $c = 6$ (par exemple) et pour m assez grand on a bien $1/m + \log(m) + 300m + 5m^2 \leq 6m^2$

Motivation de la notation O(·)

Évaluer le temps de calcul d'un algorithme en fonction de la taille m des données

Avoir une mesure de ce temps indépendante de la machine sur laquelle l'algorithme est évalué

Avoir une idée de l'augmentation du temps de calcul si on cherche à résoudre

COMPLEXITÉ DE L'EXPLORATION D'UN GRAPHE

On a n nœuds et m arcs.

Chaque sommet est mis une fois (au plus) dans Q.

Il en est ressorti une fois et est marqué une fois.

$\Rightarrow O(n)$.

Chaque arc est également examiné une fois au plus (ligne « $\forall\ v \in S(u)$ »)

$\Rightarrow O(m)$.

La complexité est donc en $O(n + m)$.

RECHERCHE D'UN CHEMIN DE *S* À *T*

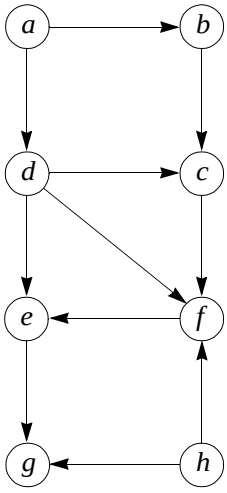
Idée :

Marquer les sommets du nom de leur prédécesseur (pour retrouver le chemin). S'arrêter dès que t est marqué.

```
Données: G avec liste des successeurs, deux sommets particuliers s et t.
Résultat: Chemin de s à t.
Début:
   Q = {s}; m(s) = 0;
   Tant que Q ≠ ∅ et m(t) non marqué
      Retirer v de Q;
      ∀ v ∈ S(v), si w est non marqué:
         marquer w: m(w) = v
         Introduire w dans Q
   Si m(t) non marqué, écrire «Pas de chemin de s à t»
   Sinon
      Écrire «Chemin de s à t, dans l'ordre inverse:»
      v = t;
      Tant que v ≠ 0
         Écrire v;
         v = m(v)
Fin
```

Exemple :

Recherche d'un chemin de *a* à *f* dans :



Exploration en largeur

$m(a) = 0$, $Q = \{a\}$

$m(b) = a$, $m(d) = a$, $Q = \{b, d\}$

$m(c) = b$, $Q = \{d, c\}$

$m(e) = d$, $m(f) = d$

f marqué, stop !

Chemin : $f \leftarrow d \leftarrow a$.

RECHERCHE D'UN PLUS COURT CHEMIN

Définition

On appelle *réseau* un graphe $G(X, U)$ dans lequel on a attribué une valeur numérique pour chaque arc de U.
Cette structure est notée $R(X, U, C)$.
C représente la famille de longueurs c_{ij} associées aux arcs (i, j).

Problème :

Étant donné un réseau $R(X, U, C)$, et deux sommets particuliers s et t,
trouver un plus court chemin de s à t.

ALGORITHME DE DIJKSTRA

Données:
$R = (X, U, C)$ avec $c_{ij} \geq 0$
Sommet particulier s.

Résultat:
λ_i $(1 \leq i \leq n = |X|)$: longueurs des plus courts chemins de s à i.

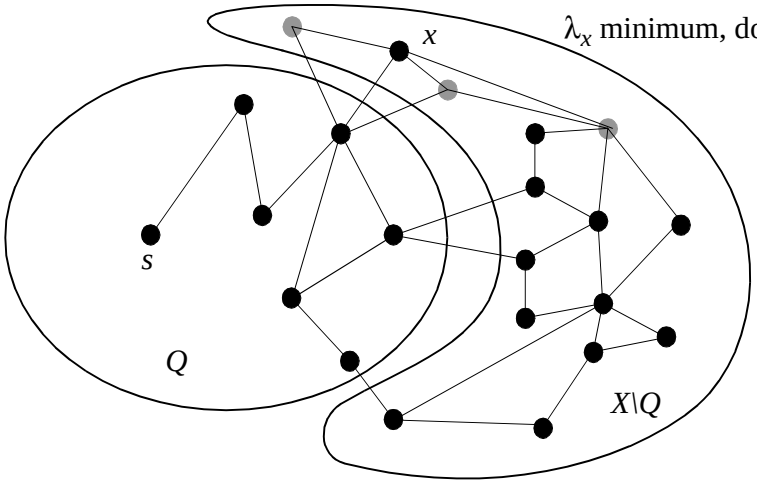
Début:
$Q = \{s\}$; $\lambda_s = 0$
$\forall\ v \in X \setminus \{s\}$ poser $\lambda_v = c_{sv}$

Tant que $Q \neq X$
Trouver x tel que $\lambda_x = \min(\lambda_v \mid v \notin Q)$
$Q = Q \cup \{x\}$
$\forall\ v \in X \setminus Q$ poser $\lambda_v = \min(\lambda_v, \lambda_x + c_{xv})$
Fin.

JUSTIFICATION DE L'ALGORITHME DE DIJKSTRA

Q représente l'ensemble des sommets pour lesquels on connaît déjà les plus courts chemins

On ajoute les sommets dans Q par ordre croissant des longueurs des plus courts chemins, puisqu'on choisit le sommet x dont le λ_x est le plus petit et que $c_{ij} \geq 0$.



λ_x minimum, donc choisi pour entrer dans Q

Valeurs de λ mises à jour

ALGORITHME DE BELLMAN

Données: $R=(X, U, C)$; sommet 1.

Résultat:
λ_i longueur d'un plus court chemin de 1 à i **ou**
Indication de l'existence d'un circuit de longueur négative.

Début:
$\lambda_1 = 0$; $\lambda_2, \ldots, \lambda_n = \infty$
Compteur = 0
Répéter
Fini = vrai
Compteur = *Compteur* + 1
Pour $k = 1, \ldots, m$
Soient i, j les extrémités de u_k
Si $\lambda_j > \lambda_i + c_{ij}$
Poser $\lambda_j = \lambda_i + c_{ij}$
Fini = faux
Jusqu'à ce que *Compteur* > n **ou** Fini
Si *Compteur* > n, il existe un circuit de longueur négative
Fin.

FONCTIONNEMENT DE L'ALGORITHME

À chaque itération on passe en revue tous les arcs.

On teste s'il est plus court de passer par i pour aller à j.

On s'arrête dès qu'on ne trouve plus d'améliorations

Le plus court chemin, s'il existe, a au plus $n - 1$ arcs. Si *Compteur* atteint la valeur de n, c'est qu'il y a un circuit de longueur négative, donc qu'il n'existe pas de plus court chemin.

Complexité de l'algorithme

L'algorithme fait au plus n itérations (*Répéter … Tant que…*).

Chaque itération peut s'effectuer en $O(m)$

Complexité globale : $O(nm)$

FORMALISATION MATHÉMATIQUE DU PCC

Données (c_{ij}), matrice des distances.

Variables de décision : x_{ij}, valant 1 si on utilise l'arc (i, j) pour aller de s à t, 0 sinon.

minimiser	$\sum_{(i,j) \in U} c_{ij} x_{ij}$		Somme des distances des arcs choisis
sous	$\sum_{i \in P(s)} x_{is} - \sum_{k \in S(s)} x_{sk} = -1$		On sort une fois de s (sans y entrer)
contraintes	$\sum_{i \in P(t)} x_{it} - \sum_{k \in S(t)} x_{tk} = 1$		On entre une fois dans t (sans en sortir)
	$\sum_{i \in P(j)} x_{ij} - \sum_{k \in S(j)} x_{jk} = 0$	$\forall j \neq s, t$	On entre et on sort le même nombre de fois pour tous les autres sommets
	$x_{ij} \in \{0, 1\}$		

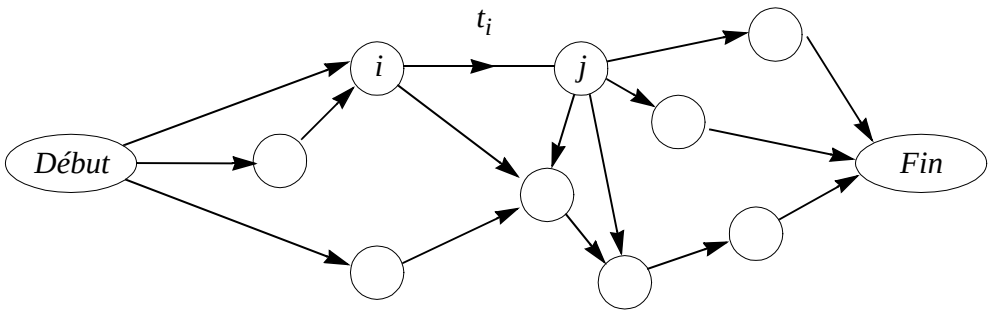
APPLICATIONS UTILISANT UN PLUS «COURT» CHEMIN

Chemin à débit maximum à partir d'un sommet *s*.

Planification de tâches.

La tâche *i* doit précéder la tâche *j* et elle prend un temps t_i pour être réalisée.

Modélisation en termes de graphes :



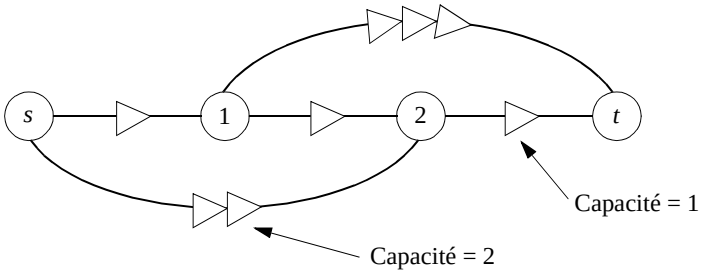
Résolution du problème : trouver un plus long chemin de *début* à *fin*.

FLOT DE VALEUR MAXIMALE

Soit un réseau **R** = (*X*, *U*, *C*), un sommet-source $s \in X$ et un sommet-puits *t*. Pour chaque arc $(i, j) \in U$, on a une capacité c_{ij} (débit maximum, par exemple des bauds pour une ligne de transmission).

Problème :

Trouver le flot maximum que l'on peut faire transiter de *s* à *t*.

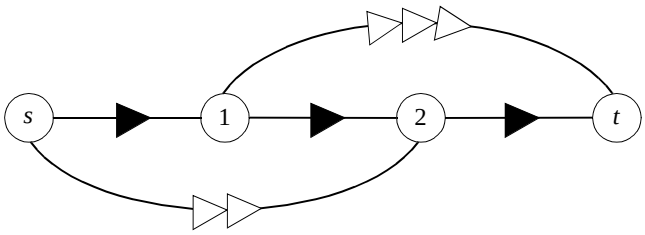


1er essai d'algorithme

```
Répéter:
   Trouver un chemin de s à t (n'empruntant que des arcs non saturés)
   Si un tel chemin existe: Augmenter au maximum le flot sur ce chemin.
Tant qu'on réussit à augmenter le flot.
```


Application de l'algorithme :

On trouve en premier le chemin *s*–1–2–*t* et on obtient :



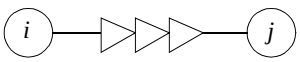
Il n'existe plus de chemin de *s* à *t* et pourtant on n'a pas un flot maximum !

Remède :

Construire un réseau intermédiaire **R*** des augmentations de flots possibles en remarquant qu'il est possible d'augmenter le flot de *i* à *j* par la diminution d'un flot existant de *j* à *i* :

Flot existant dans *R*

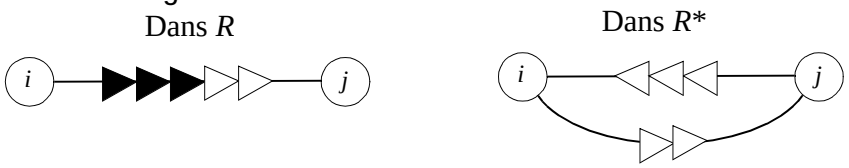
Dans *R**



ALGORITHME DE FORD & FULKERSON

Données: Réseau **R** = (*X*, *U*, *C*), sommets *s* et *t*.
Résultat: Flot de valeur maximum de *s* à *t*
Début:
  Partir d'un flot nul dans tous les arcs
  Répéter:

    Construire le réseau d'augmentation **R***



    Chercher un chemin de *s* à *t* dans **R***

    Si un tel chemin existe: augmenter au maximum le flot le long de ce chemin (diminuer le flot (*i*, *j*) dans les arcs de **R** qui apparaissent inversés (*j*, *i*) dans **R***)

  Tant qu'un tel chemin existe
Fin.

JUSTIFICATION DE L'ALGORITHME

Définitions

L'ensemble des arcs *sortants* d'un sous-ensemble de sommets *Y* est appelé une *coupe* de *Y* à $\bar{Y} = X \backslash Y$

La *capacité de la coupe* est la somme des capacités des arcs de la coupe.

On vérifie que l'ensemble des sommets marqués dans **R*** à la dernière étape de l'algorithme de Ford & Fulkerson forme une coupe séparant *s* de *t* telle que sa capacité est égale au flot que l'on a trouvé (sinon, on aurait pu marquer d'autres sommets).

Comme il est impossible d'avoir un flot supérieur à la capacité d'une coupe de *s* à *t*, on a établit le

Théorème du flot maximum et de la coupe minimum

La valeur d'un flot maximum de *s* à *t* est égale à la capacité minimale d'une coupe séparant *s* de *t*.

COMPLEXITÉ DE L'ALGORITHME :

Définition

La *taille* d'un exemple de problème est le nombre de bits qu'il faut pour le stocker en mémoire.

Exemple :

Un réseau à n sommets peut être représenté avec n^2 nombres de b bits (il suffit de mémoriser les valeurs des capacités c_{ij} dans une matrice). La taille du problème est donc de bn^2

Si l'on n'a pas de chance en appliquant l'algorithme de Ford & Fulkerson, on n'augmente le flot que d'une unité à chaque itération de l'algorithme. En augmentant d'un bit la taille du problème, par exemple pour doubler la valeur de la capacité la plus élevée, on peut doubler le temps d'exécution de l'algorithme.

Par conséquent, l'algorithme de Ford & Fulkerson peut prendre un temps qui augmente exponentiellement avec la taille du problème !

AMÉLIORATION DE L'ALGORITHME

Pour garantir un nombre d'étapes polynomial en la taille du problème, on peut procéder comme suit :

1) Au lieu de chercher un chemin dans R^*, on va chercher un chemin avec le nombre minimum d'arcs de s à t. (Exploration en largeur ou construction d'un *réseau en couches* R^c)
2) On sature ce chemin et on recommence.

Cet algorithme est polynomial car :

- Chercher un tel chemin est polynomial $O(n^2)$
- D'une étape à la suivante, le nombre d'arcs du chemin ne peut décroître
- On a au plus n étapes avec un nombre donné d'arcs de s à t
- On a au minimum 1 arc et au maximum n

On en déduit que la complexité totale est $O(n^4)$.

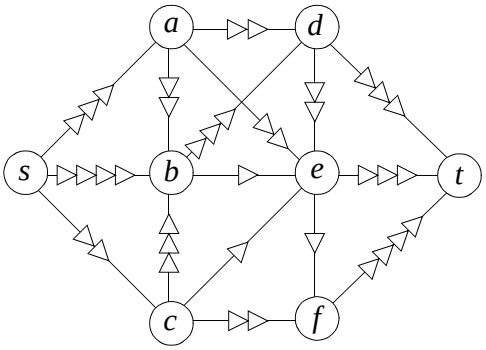
ALGORITHME MKM

Données: Réseau **R** = (*X*, *U*, *C*), sommets *s* et *t*.
Résultat: Flot de valeur maximum de *s* à *t*
Début:
  ∀ *u* ∈ *U* *Flot*(*u*) = 0;
  Continuer := vrai
  Tant que *Continuer*, **répéter**:
    Construire le réseau d'augmentation en couches $\mathbf{R^c}$
    Si *t* n'est pas atteignable dans $\mathbf{R^c}$
      Continuer := faux
    Sinon
      Répéter, **tant que** *t* est atteignable:
        Calculer *traverse*(*i*), la quantité maximale de flot pouvant traverser le sommet *i* dans $\mathbf{R^c}$ ($traverse(i) = min\left(\sum_{j \in S(i)} c_{ij}, \sum_{k \in P(i)} c_{ki}\right)$)
        Éliminer tous les sommets *i* avec *traverse*(*i*) = 0
        Trouver un sommet *j* avec *traverse*(*j*) minimum
        Pousser un flot de valeur *traverse*(*j*) de *j* à *t*
        Tirer un flot de valeur *traverse*(*j*) de *s* à *j*.
Fin.

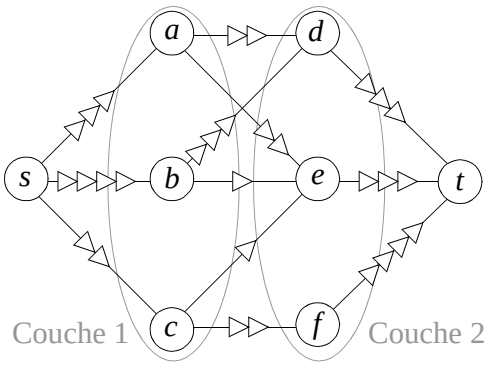
Il est possible de vérifier que cet algorithme est en $O(n^3)$.

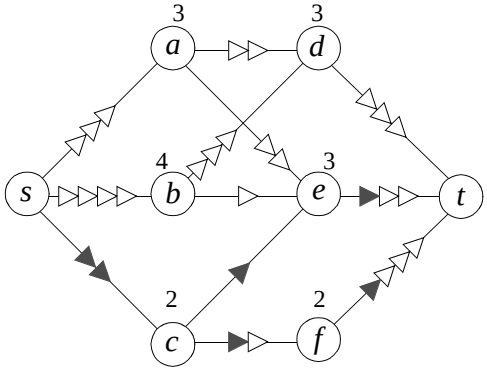
Illustration de l'algorithme MKM

Réseau en initial

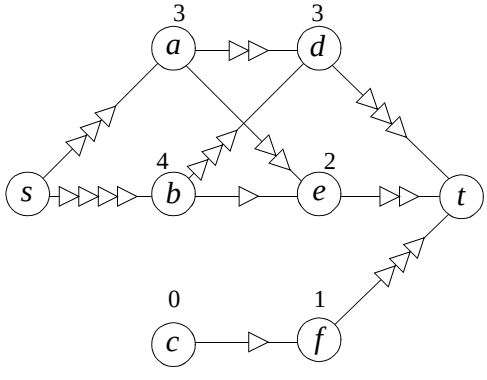


1er réseau en couches

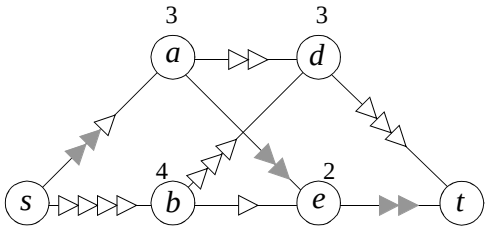


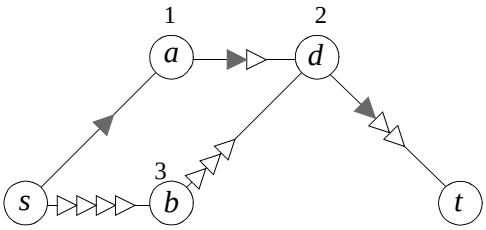


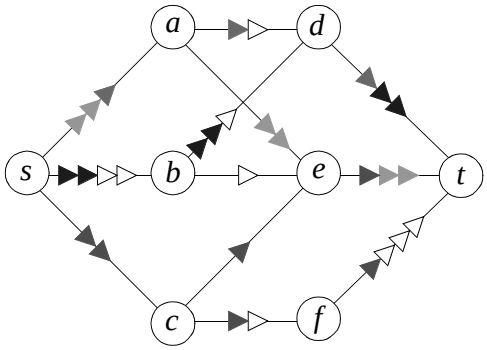
1er flot dans le 1er réseau en couches



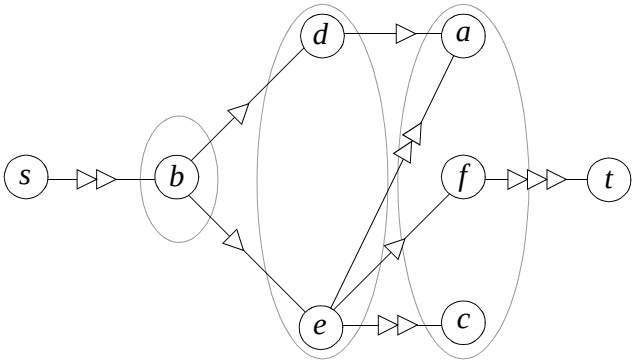
Mise à jour des capacités restantes







Flot maximal dans le 1er réseau en couches



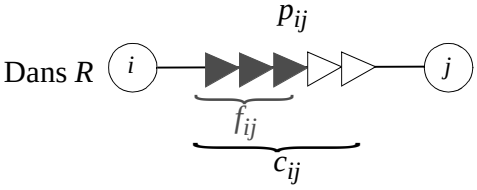
2e réseau en couches

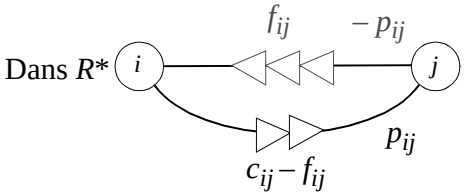
FLOT À COÛT MINIMUM

Problème : En plus d'une capacité c_{ij} pour chaque arc (i, j), on a un *coût unitaire d'utilisation* p_{ij}. On cherche un *flot maximum* entre deux sommets s et t de *coût total minimum*.

Algorithme :

Données: Réseau $R = (X, U, C, P)$, 2 sommets particuliers s et t.
Résultat: Flot maximum à coût minimum.
Début:
  Répéter:
    Construire R^*, le réseau d'augmentation





    Chercher un plus court chemin de s à t

    Augmenter le flot au maximum le long de ce chemin
  Tant qu'on trouve un chemin de s à t dans R^*

Problème rencontré dans l'implantation de cet algorithme :

On a des coûts négatifs dans R^*. On ne peut donc pas utiliser l'algorithme rapide de Dijkstra directement.

Remèdes possibles :

1) Utiliser l'algorithme de Bellman, au détriment du temps de calcul.

2) Utiliser Dijkstra, mais modifier la définition des coûts de sorte à ne pas avoir de coûts négatifs.

Modification de la définition des coûts :

À la première itération, on peut utiliser Dijkstra sans autre, puisque $R = R^*$ (on suppose que tous les coûts réels sont non négatifs).

À chaque itération, Dijkstra donne la longueur λ_i du plus court chemin de s à i dans R^*

Remplacer tous les p_{ij} par $\lambda_i + p_{ij} - \lambda_j$.

Avec ces nouveaux coûts, le plus court chemin de s à t n'est pas modifié, à l'exception de sa longueur qui vaut 0 ; aucun coût n'est négatif (car, par définition du plus court chemin, on doit avoir $\lambda_i + p_{ij} \geq \lambda_j$)

Donc, en renversant des arcs dans R^* et en changeant le signe de leur coût (= 0), on ne crée pas d'arcs de coût négatif.

APPLICATION DE LA NOTION DE FLOTS

Problème de l'affectation linéaire

Couplage d'un ensemble A d'éléments à un autre ensemble B

Formulation mathématique :

$$\text{minimiser} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad \text{Coût total de l'affectation}$$

$$\text{sous} \quad \sum_{j=1}^{n} x_{ij} \leq 1 \quad \forall i \qquad \text{Une personne peut faire au plus une tâche}$$

$$\text{contraintes} \quad \sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \qquad \text{Chaque tâche est réalisée}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j$$

Exemples :

Ensemble A : Personnes, Machines, Professeurs

Ensemble B : Tâches, Pièces, Classes.

EXERCICE

Les 4 étudiants (*A*, *B*, *C*, *D*) d'une classe doivent choisir parmi 4 projets de semestre (1, 2, 3, 4).

Chaque étudiant pronostique la note qu'il pense obtenir pour chacun des projets :

		Projet			
		1	2	3	4
Étudiant	*A*	6	5	5,8	5,5
	B	6	5,5	4,5	4,8
	C	4,5	6	5,4	4
	D	5,5	4,5	5	3,8

Quels doivent être les choix des étudiants pour maximiser la moyenne de classe ?

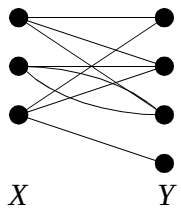
COMPLÉMENTS THÉORIQUES SUR LES GRAPHES

Définitions

Un *couplage* dans un graphe *G* est un sous-ensemble *C* d'arcs (ou d'arêtes, dans le cas non orienté) dont toutes les extrémités sont distinctes.

L'*indice chromatique* d'un graphe *G* est le nombre minimum de couleurs nécessaires à attribuer aux arêtes du graphe de telle manière que toutes les arêtes adjacentes au même sommet aient des couleurs différentes.

Un graphe *G* est *biparti* si l'ensemble des sommets peut être partitionné en deux sous-ensembles distincts *X* et *Y* de sorte que tout arc ait une extrémité dans *X* et l'autre dans *Y*.



On peut trouver un couplage maximum dans un graphe biparti en introduisant un sommet *s* avec des arcs (*s*, *i*) de capacité 1 pour tout $i \in X$ et un sommet *t* avec des arcs (*j*, *t*) de capacité 1 pour tout $j \in Y$. Les arcs (*i*, *j*) utilisés par un flot maximum de *s* à *t* donnent le couplage maximum.

ÉTABLISSEMENT D'HORAIRES

Théorèmes

Dans un graphe biparti, il existe un couplage C avec $|C| = |X|$ si, et seulement si tout sous-ensemble Z de X a au moins autant de voisins dans Y que $|Z|$.

(König II) Pour tout multigraphe biparti, l'indice chromatique est égal au maximum des degrés des sommets.

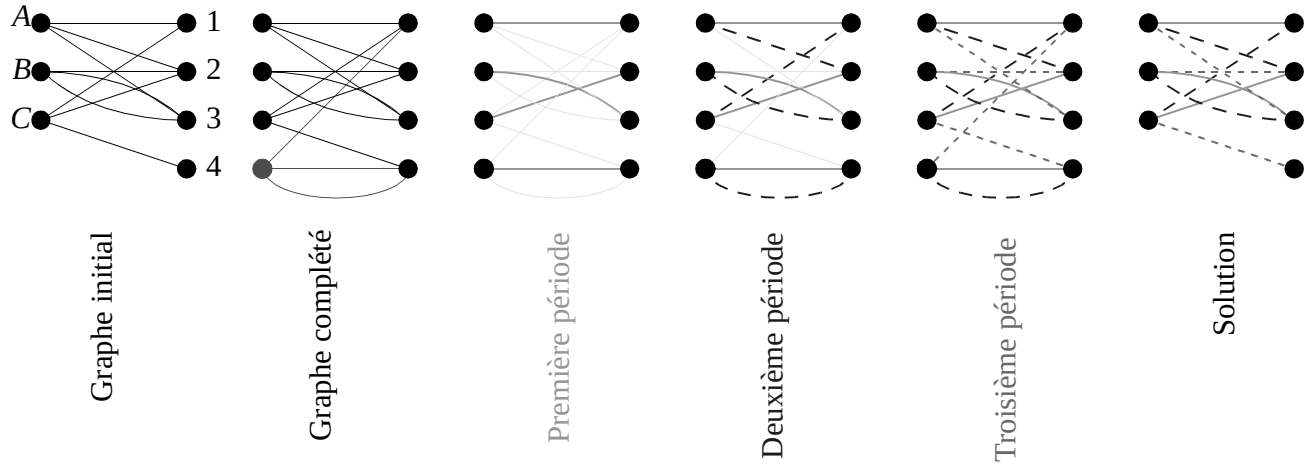
Application des théorèmes : coloration des arêtes d'un graphe biparti en un nombre minimum de couleurs.

Combien d'heures au minimum faut-il prévoir à la grille horaire pour satisfaire les nombres d'heures d'enseignements spécifiés dans le tableau suivant :

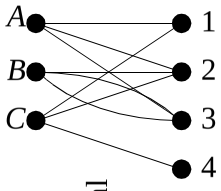
		Professeurs			
		1	2	3	4
Classes	*A*	1	1	1	
	B		1	2	
	C	1	1		1

DÉMONSTRATION KÖNIG II ; SOLUTION DU PROBLÈME

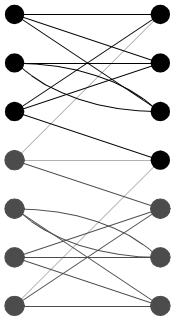
```
Compléter le graphe de sorte qu'il y ait le même nombre de sommets de
chaque côté et que tous les sommets aient le même degré (= degré maximum
d'un sommet dans le graphe initial).

Tant qu'il reste des arêtes, répéter
   Chercher un couplage maximum dans le graphe (les arêtes utilisées dans
   ce couplage correspondront aux cours à donner à une heure donnée)
   Supprimer les arête du couplage trouvé
```



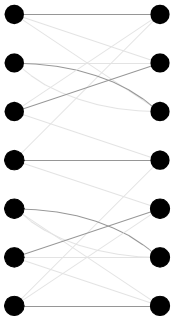
SOLUTION



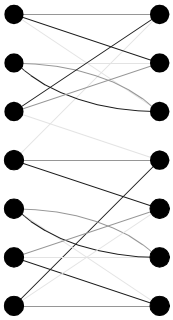
Graphe initial



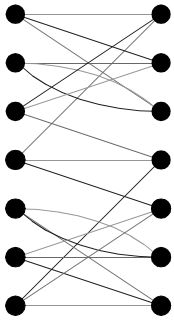
Graphe complété



Premier flot maximum (période 1)



Deuxième période



Troisième période

Solution :