

Déroulement

- 14h de cours-TD au S5
- 16h de cours-TD au S6
- 14h de projet tutoré
- Contrôle de connaissances :
 - S5 – 0,5 ECTS
 - S6 – 1 ECTS
 - Projet – 1 ECTS

Bibliographie (entre autres) :

- Poly de Clarisse Dhaenens
- Algorithmes de graphes, Ed. Eyrolles, Lacomme-Prins-Sevaux
- Optimisation combinatoire (vol 2 : programmation discrète), Ed. Hermann, Sakarovitch
- Wikipedia

Programme

- Concept de graphe
 - Définitions élémentaires
 - Représentation et implémentation
- Cheminement dans les graphes
 - Connexité
 - Exploration
- Graphes particuliers
 - Arbres / Arborescences
- Problèmes de graphes et algorithmes de résolution
 - Recherche du plus court chemin
 - Ordonnancement simple, PERT
 - Recherche du flot maximal

Objectifs

- Savoir modéliser un problème à l'aide de graphes et reconnaître le problème à résoudre
- Connaître les structures de données dédiées à la manipulation de graphes
- Connaître les différents types de graphes existants
- Maîtriser les principaux algorithmes de résolution pour les problèmes classiques

Principaux algorithmes

- Exploration en largeur, en profondeur
- Algorithme de Kruskal (arbre couvrant)
- Recherche d'une arborescence couvrante
- Algorithme de Bellman
- Algorithme de Dijkstra
- Algorithme de Ford-Fulkerson
- Algorithme de Roy

Concept de graphe

Graphe

Définition

Un **graphe G (non orienté)** est un couple (X, E) formé de deux ensembles disjoints :

$$X = \{x_1, x_2, \dots, x_n\} \text{ avec } n \geq 1$$

$$E = \{e_1, e_2, \dots, e_m\} \text{ avec } m \geq 0$$

tels que, pour tout i , e_i est une paire d'éléments de X .

X : ensemble des sommets du graphe G

E : l'ensemble des arêtes du graphe G

Exemple

$$G = (X, E)$$



$$X = \{1, 2, 3, 4\}$$

$$E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$$

$$e_1 = \{1, 2\}, e_2 = \{1, 2\}, e_3 = \{1, 3\},$$

$$e_4 = \{2, 3\}, e_5 = \{3, 4\}, e_6 = \{4, 4\}$$

Graphe : Terminologie (1)

- $e = \{x, y\}$ (ou $e = (x, y)$ ou $e = xy$)
Les sommets x et y sont les **extrémités** de e . On dit aussi que e relie x et y .
- Un sommet x et une arête e sont **incidents** si x est une extrémité de e .
- Deux sommets d'un graphe sont **adjacents** s'ils sont extrémités d'une même arête. L'ensemble des sommets adjacents au sommet x est noté $V(x)$. 
- Si $x = y$, on dit que e est une **boucle**. 
Deux arêtes sont dites **parallèles** lorsqu'elles ont les mêmes extrémités. Un ensemble d'arêtes parallèles est appelé **arête multiple**.
- Un graphe est dit **simple** lorsqu'il est sans boucle ni arêtes parallèles.
Le **graphe complémentaire** d'un graphe simple G , est le graphe simple H ayant les mêmes sommets et tel que deux sommets distincts de H soient adjacents si et seulement s'ils ne sont pas adjacents dans G . Il est généralement noté \bar{G} .

Exemple : graphe simple et son complémentaire

G

\bar{G}

Graphe : Terminologie (2)

- Un graphe (**simple**) est dit **complet** si chaque sommet est relié à tous les autres. On dénote par K_n le graphe complet à n sommets.
Remarque : un graphe complet s'appelle aussi une clique.

Exemple : graphes complets

K_1

K_2

K_3

K_4

Degré d'un sommet (1)

Définition

Le **degré d'un sommet** x , noté $d(x)$, est le nombre d'arêtes incidentes à x .

Remarque : une boucle est comptée deux fois.

- Si $d(x) = 0$, le sommet x est **isolé**.
- Si $d(x) = 1$, le sommet x est **pendant**. Une arête pendante est une arête incidente à un sommet pendant.

On pose :

$$\delta(G) = \min_{x \in X} \{d(x)\}$$

$$\Delta(G) = \max_{x \in X} \{d(x)\}$$

- Si $\delta(G) = \Delta(G) = k$ (tous les sommets ont le même degré), le graphe est dit **k-régulier**.
- Dans un graphe **simple** (sans boucle ni arête multiple), $\Delta(G) \leq n - 1$.

Degré d'un sommet (2)

Proposition

Pour tout graphe $G=(X,E)$, on a :

$$\sum_{x \in X} d(x) = 2|E|$$

Conséquence 1

Pour tout graphe G ayant n sommets et m arêtes, on a :

$$\delta(G) \leq \lfloor 2m/n \rfloor$$

$$\Delta(G) \geq \lceil 2m/n \rceil$$

Si G est simple, alors

$$m \leq n(n-1)/2$$

Rmq : l'égalité est vérifiée si G est un graphe complet.

Degré d'un sommet (3)

Conséquence 2

Dans un graphe, le nombre de sommets de degré impair est pair.

Démonstration :

Sous-graphe

Soient un graphe $G = (X, E)$ et deux sous-ensembles $Y \subset X$ et $F \subset E$.

Définitions

- $H = (Y, F)$ est un **sous-graphe de G** si pour toute arête $e \in F$, les extrémités de e sont dans Y .
- Le **sous-graphe engendré par Y** est le sous-graphe de G dont l'ensemble des arêtes F est formé de la totalité des arêtes de E dont les extrémités sont dans Y . On le note G_Y , car il est uniquement déterminé par Y .
- Le **sous-graphe engendré par F** , noté $G_{[F]}$, est le sous-graphe de G dont l'ensemble des arêtes est F et dont l'ensemble des sommets est formé de toutes les extrémités des arêtes appartenant à F .
- Un **graphe partiel de G** est un sous-graphe de G de la forme $H = (X, F)$.

Exemple : sous-graphes engendrés



G

$G_{\{1,3,4\}}$

$G_{[\{e_1, e_3, e_4\}]}$

Stable et Clique

Définitions

- Un **stable** dans un graphe $G = (X, E)$ est un sous-ensemble de sommets $S \subset X$ tel que le sous-graphe G_S engendré par S ne comporte pas d'arête.
- Une **clique** dans un graphe $G = (X, E)$ est un sous-ensemble de sommets $S \subset X$ tel que le sous-graphe G_S engendré par S est un graphe complet. Une p -clique désigne une clique contenant p sommets.

Exemple

G (graphe simple)

Clique de G

Stable de G

Graphe k -parti

Définition

- Un graphe est **k -parti**, si son ensemble de sommets admet une partition en k stables.
- En particulier, nous distinguerons les graphes bipartis où $k = 2$.
Un graphe **biparti complet** est un graphe biparti où chaque sommet d'un stable est relié à tous les sommets de l'autre stable. On note $K_{n,p}$ avec n et p la cardinalité de chaque partition.

Exemple

Graphe 3-parti

$K_{2,4}$ (biparti complet)

Graphe orienté

Définition

Un **graphe G orienté** est un couple (X, A) formé de deux ensembles disjoints :

$$X = \{x_1, x_2, \dots, x_n\} \text{ avec } n \geq 1$$

$$A = \{a_1, a_2, \dots, a_m\} \text{ avec } m \geq 0$$

tels que, pour tout i , a_i est un couple d'éléments de X .

X : ensemble des sommets du graphe G

A : l'ensemble des arcs du graphe G

Chaque arc est composé d'un sommet initial et d'un sommet terminal.

Exemple

$$G = (X, A)$$

$$X = \{1, 2, 3, 4\}$$

$$A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$$

$$a_1 = \{2, 1\}, a_2 = \{1, 2\}, a_3 = \{1, 3\},$$

$$a_4 = \{2, 3\}, a_5 = \{3, 4\}, a_6 = \{4, 4\}$$

Graphe orienté : Remarques

- Pour un graphe orienté, les relations d'adjacence et d'incidence sont définies de la même façon que pour les graphes non orientés.
- Idem pour les arcs parallèles et multiples.
- Un graphe orienté est simple, s'il est sans boucle et si entre deux sommets quelconques, il existe au plus un arc.
- La notion de sous-graphe est définie de la même façon en remplaçant *arêtes* par *arcs*.
- Pour les graphes orientés, on définit la notion de **prédécesseur** et de **successeur** d'un sommet. Soit x un sommet. Le sommet y est dit **successeur** de x (respectivement **prédécesseur**) s'il existe un arc xy (resp. yx). L'ensemble des successeurs de x est noté $V^+(x)$ et l'ensemble de ses **prédécesseurs**, $V^-(x)$.

Graphe orienté : Degrés

Soit x un sommet d'un graphe orienté.

Définition

On appelle **degré rentrant** noté $d^-(x)$ (resp. **degré sortant** noté $d^+(x)$), le nombre d'arcs ayant x comme extrémité terminale (resp. initiale).

On a :

$$d^-(x) + d^+(x) = d(x)$$

De plus, on a l'égalité suivante :

$$\sum_x d^-(x) = \sum_x d^+(x) = |A|$$

puisque chaque arc a exactement une extrémité initiale et une extrémité terminale.

Représentation d'un graphe

Pourquoi ?

- Nombres de sommets et d'arêtes très grands :
→ quasi impossible de dessiner le graphe sur une feuille
- Description (sous forme d'ensembles) des sommets et des arêtes/arcs :
→ ne permet pas d'appliquer des algorithmes de recherche, d'exploration...

Proposition :

- Représentation déterministe d'un graphe :
→ à une représentation correspond un et un seul graphe
 - Représentation matricielle
 - Représentation tabulaire

Remarque : on suppose dans cette partie que les graphes considérés n'ont pas de sommets isolés.

Matrice d'adjacence d'un graphe non orienté

Représentation matricielle

Soit $G = (X, E)$ un graphe à n sommets.

Définition

La **matrice d'adjacence** de G , notée $\mathcal{A}(G) = [a_{ij}]$, est une matrice de dimension $n \times n$ traduisant les relations d'adjacence entre ses sommets.

Ses lignes et colonnes sont indexées par X et

a_{ij} = nombre d'arêtes ayant x_i et x_j comme extrémités.

Remarques :

- $\mathcal{A}(G)$ est une matrice symétrique à éléments entiers.
- Si G est simple, la diagonale est formée de 0 (pas de boucle) et les autres éléments sont dans $\{0, 1\}$ (pas d'arête multiple).

Matrice d'incidence d'un graphe non orienté

Représentation matricielle

Soit $G = (X, E)$ un graphe à n sommets et m arêtes.

Définition

La **matrice d'incidence** de G , notée $\mathcal{B}(G) = [b_{ij}]$, est une matrice de dimension $n \times m$ traduisant les relations d'incidence entre ses sommets et ses arêtes.

Ses lignes sont indexées par X et ses colonnes par E

b_{ij} = nombre de fois où x_i est incident à e_j .

Remarques :

- Les éléments de $\mathcal{B}(G)$ appartiennent à $\{0, 1, 2\}$, et à $\{0, 1\}$ si le graphe ne contient pas de boucle.
- La somme des éléments d'une colonne est égale à 2 (chaque colonne correspond à une et une seule arête).

Représentation matricielle d'un graphe non orienté

Exemple

Représentation matricielle d'un graphe orienté

Soit $G = (X, A)$ un graphe à n sommets et m arcs.

Définition : Matrice d'adjacence

$\mathcal{A}(G) = [a_{ij}]$ définie par :

$$a_{ij} = \text{nombre d'arcs ayant} \begin{cases} x_i \text{ comme extrémité initiale et,} \\ x_j \text{ comme extrémité terminale.} \end{cases}$$

Définition : Matrice d'incidence

$\mathcal{B}(G) = [b_{ij}]$ définie par :

$$b_{ij} = \begin{cases} 1 & \text{si l'arc } a_j \text{ admet } x_i \text{ comme extrémité initiale,} \\ -1 & \text{si l'arc } a_j \text{ admet } x_i \text{ comme extrémité terminale,} \\ 0 & \text{si l'arc est une boucle et les autres cas.} \end{cases}$$

Remarques :

- Un arc est représenté par un et un seul 1 dans la matrice d'adjacence. La somme des a_{ij} doit donc être égale au nombre d'arcs.
- La matrice d'incidence n'est significative pour un graphe orienté que si ce graphe ne comporte pas de boucle car dans le cas contraire, deux graphes différents peuvent avoir la même matrice.

Représentation matricielle d'un graphe orienté

Exemple

Représentation matricielle : Remarques

- Les représentations d'un graphe à l'aide des matrices d'adjacence ou d'incidence sont simples.
- Mais...
 - Gourmandes en cases mémoire
→ matrices d'adjacence = $O(n^2)$, d'incidence = $O(n \times m)$
 - Graphes avec peu d'arêtes \Rightarrow matrices creuses
Idée : listes d'adjacence ou... représentation tabulaire

Tables des arcs

Représentation tabulaire

Soit $G = (X, A)$ un graphe à n sommets et m arcs.

Dans cette représentation on utilise 2 tableaux SI et ST de dimension m tels que :

- $SI[i] =$ sommet initial de l'arc a_i
- $ST[i] =$ sommet terminal de l'arc a_i

Remarques :

- Si le graphe est non orienté ($G = (X, E)$), l'ordre SI, ST n'a pas d'importance.
- Cette représentation nécessite $O(m)$ cases mémoire.

Exemple

Table des successeurs

Représentation tabulaire

Soit $G = (X, A)$ un graphe à n sommets et m arcs.

Dans cette représentation on utilise 2 tableaux Succ et Head de dimension m et n resp. tels que :

- Succ est une liste rangée contenant successivement pour chaque sommet x_i l'ensemble de ses sommets adjacents (successeurs) x_j .
- Head[i] = indice du tableau Succ à partir duquel sont rangés les successeurs du sommet x_i .

Remarque :

- Cette représentation nécessite $O(m + n)$ cases mémoire.

Exemple

Quelle représentation choisir ? (1)

Cas des prédécesseurs d'un sommet

Il est très courant, lors de l'exploration et/ou de l'exploitation d'un graphe, de vouloir connaître la liste des prédécesseurs d'un sommet. En fonction de la représentation choisie pour le graphe, on peut construire cette liste de différentes manières.

- Représentation par matrice d'adjacence : la construction de la liste des prédécesseurs est immédiate. Les prédécesseurs d'un sommet x_i sont les sommets x_j tels que $a_{ji} \geq 1$. Il faut donc regarder dans la colonne i de la matrice. La complexité de recherche est en $O(n)$.
- Représentation par matrice d'incidence : il faut repérer les arcs (en colonne) qui ont un -1 en ligne x_i et rechercher l'autre extrémité de l'arc. La complexité est donc en $O(m \times n)$.
- Représentation par table des arcs : il faut repérer dans ST les occurrences de x_i et sélectionner, en tant que prédécesseur, l'élément correspondant dans SI. La complexité de recherche est en $O(m)$.

Quelle représentation choisir ? (2)

Cas des prédécesseurs d'un sommet

- Représentation par table des successeurs : on peut obtenir les listes des prédécesseurs $\text{Pred}[x]$ par l'algorithme suivant :

Pour tout sommet x

Initialiser $\text{Pred}[x]$ à \emptyset

Pour tout sommet x

Pour tout successeur y de x ($y \in \text{Succ}[\text{Head}[x], \dots, \text{Head}[x+1]-1]$)

Ajouter x à $\text{Pred}[y]$

Cet algorithme est en $O(m + n)$.

Quelle représentation choisir ?

Cas des ajouts et suppressions de sommets et d'arcs

Les représentations matricielles et tabulaires sont simples et facilement acceptées par l'ensemble des langages algorithmiques. **Mais** la rigidité de l'allocation séquentielle supporte mal les transformations graphiques élémentaires comme la suppression ou l'ajout de sommets, d'arcs, ...

Il est alors possible, lorsque le graphe est amené à être souvent modifié, d'utiliser les listes chaînées où les pointeurs permettent de maintenir l'ordre de la liste.

Exemple : listes chaînées

Quelle représentation choisir ?

Conclusion

Les différents modes de représentation d'un graphe ont chacun leurs avantages et leurs inconvénients. **Il n'existe pas de meilleure représentation** de façon absolue.

Il faut alors trouver un compromis entre la facilité d'accès aux informations (et de modification éventuelle) et l'encombrement mémoire. Il conviendra à chaque problème, de se poser la question sur la représentation à adopter, car pour un graphe donné (avec ses particularités) et un problème donné (avec les traitements à faire), il existe des représentations mieux adaptées que d'autres.

Cheminement dans les graphes

Chaîne et cycle d'un graphe (1)

Soit un graphe $G = (X, E)$ non orienté.

Définition

Une **chaîne** est une suite alternée de sommets et d'arêtes $\Gamma = x_0 e_1 x_1 e_2 \dots x_{k-1} e_k x_k$ telle que pour $1 \leq i \leq k$, les extrémités de e_i sont x_{i-1} et x_i .

- $\Gamma = x_0 \dots x_k$ est une chaîne de longueur k (k arêtes sont parcourues).
- Une **chaîne fermée** est une chaîne pour laquelle $k > 0$ et $x_k = x_0$.
Pour une chaîne fermée, la numérotation des termes de Γ (sommets et arêtes) est donnée à une permutation près.
- Une **chaîne simple**, est une chaîne dont les arêtes sont distinctes ($e_1 \neq e_2 \neq \dots e_k$).
- Une **chaîne élémentaire** est une chaîne dont les sommets sont distincts ($x_0 \neq x_1 \neq \dots x_k$).
Une chaîne élémentaire a deux extrémités distinctes ($x_0 \neq x_k$).

Définition

Un cycle est une chaîne fermée simple.

- Un cycle est **élémentaire** si ses sommets sont distincts ($x_0 \neq x_1 \neq \dots x_{k-1}$).

Chaîne et cycle d'un graphe (2)

Exemple

G

Chaîne simple de G

Chaîne élémentaire de G

Cycle de G

Cycle élémentaire de G

Chaîne et cycle d'un graphe (3)

Soit un graphe $G = (X, E)$ à n sommets et m arêtes.

Remarques :



- Lorsque G est simple (pas d'arête multiple), une chaîne est parfaitement définie par la suite de ses sommets.
- Une chaîne élémentaire est une chaîne simple.
- Une boucle est un cycle élémentaire de longueur 1.
- Toute chaîne simple est de longueur inférieure ou égale à m .
- Toute chaîne élémentaire est de longueur inférieure à n .
Une **chaîne Hamiltonienne** est une chaîne élémentaire de longueur $n - 1$ passant exactement une fois par tous les sommets.
- Tout cycle élémentaire est de longueur inférieure ou égale à n .
Un **cycle Hamiltonien** est un cycle élémentaire de longueur n ie. il passe une et une seule fois par tous les sommets.
- Tout cycle est de longueur inférieure ou égale à m .
Un **cycle Eulérien** est un cycle de longueur m passant exactement une et une seule fois par chaque arête.

Chaîne et cycle d'un graphe (4)


Proposition 1

- (i) De toute chaîne de x à y ($x \neq y$) on peut extraire une chaîne élémentaire allant de x à y .
- (ii) De tout cycle passant par une arête e on peut extraire un cycle élémentaire passant par e .

Conséquence :

De tout cycle passant par un sommet x on peut extraire un cycle élémentaire passant par x .

Proposition 2

- (i) Si G est un graphe tel que $\delta(G) \geq 2$ alors G contient un cycle. 
- (ii) Si G est simple et $\delta(G) \geq 2$, alors G contient un cycle élémentaire de longueur supérieure ou égale à $\delta(G) + 1$ et une chaîne élémentaire de longueur supérieure ou égale à $\delta(G)$.

Proposition 3

- Si $m \geq n$ alors G contient au moins un cycle
(et si G ne contient pas de cycle, alors $m \leq n - 1$).

Coloration d'un graphe

Applications I



Définition

On appelle **k-coloration** d'un graphe $G = (X, E)$ (non orienté) une partition de ses sommets en k -classes (X_1, X_2, \dots, X_k) de sorte que deux sommets d'une même classe ne soient pas adjacents.

- On dit que les sommets de la classe X_j sont de la couleur j , avec $j = 1, 2, \dots, k$.
- Un graphe admettant une k -coloration est dit **k-coloriable**.
- Le **nombre chromatique**, noté $\gamma(G)$, d'un graphe G est le nombre minimum pour lequel G est k -coloriable.

Théorème (Cas particulier : graphe biparti)

Soit un graphe $G = (X_1, X_2, E)$, les conditions suivantes sont équivalentes :

- (i) G est biparti 
- (ii) $\gamma(G) = 2$
- (iii) G n'a pas de cycle élémentaire de longueur impaire 
- (iv) G n'a pas de cycle de longueur impaire

Couplage et recouvrement d'un graphe (1)

Applications II

Définition

Un sous-ensemble E' de l'ensemble des arêtes du graphe $G = (X, E)$ est un **couplage de G** si les arêtes de E' sont deux à deux non adjacentes.

Remarque : Si on considère le graphe partiel $G' = (X, E')$, E' est un couplage si et seulement si, $\forall x \in X, d_{G'}(x) \leq 1$.

- Un sommet x est dit **saturé** par un couplage si $d_{G'}(x) = 1$.
- Un **couplage maximum** est un couplage de cardinalité maximum.
- Un **couplage maximal** est un couplage tel que l'on ne peut pas rajouter d'arête.
- Un couplage est dit **parfait** s'il sature tous les sommets. Un couplage parfait est donc maximum.

Soit E' un couplage de $G = (X, E)$.

Une chaîne élémentaire de G est dite **alternée** par rapport au couplage E' si les arêtes appartiennent successivement à E' puis à $E \setminus E'$.

Couplage et recouvrement d'un graphe (2)

Applications II

Théorème [Berge]

Soit E' un couplage de G . Il y a équivalence entre :

- (i) E' est un couplage maximum
- (ii) Il n'existe pas de chaîne alternée par rapport à E' dont les extrémités sont des sommets non saturés

Exemple

Couplage et recouvrement d'un graphe (3)

Applications II

Définition

Un sous-ensemble E^* de l'ensemble des arêtes de $G = (X, E)$ est un **recouvrement de G** (des sommets par les arêtes) si tout sommet x est incident à une arête de E^* au moins.

Remarque : Si on considère le graphe partiel $G^* = (X, E^*)$, E^* est un recouvrement si et seulement si $\forall x \in X, d_{G^*}(x) \geq 1$.

On a les mêmes définitions de recouvrements minimal et minimum.

Exemple

Connexité d'un graphe (1)

Soit un graphe $G = (X, E)$.

Définition

G est **connexe** s'il possède la propriété suivante : $\forall x \in X, \forall y \in X$, il existe une chaîne reliant x et y .

- On appelle **composante connexe** tout sous graphe maximal (au sens de l'inclusion) de G vérifiant cette propriété.
- Un graphe est dit connexe s'il ne possède qu'une seule composante connexe.

Exemple



Graphe connexe

Graphe non connexe

Connexité d'un graphe (2)

Soit un graphe $G = (X, E)$.

Définitions

- Soit S un sous-ensemble de sommets.
On appelle **cocycle associé à S** , noté $\omega(S)$, l'ensemble des arêtes ayant exactement une extrémité dans S (i.e. $e \in \omega(S)$ si et seulement si e a une extrémité dans S et l'autre dans le complémentaire de S (noté \bar{S})).
- Un sous-ensemble F d'arêtes ($F \subset E$) est appelé **cocycle de G** s'il existe un ensemble $S \subset X$ tel que $F = \omega(S)$.

Remarque :

Un cocycle de cardinalité 1 est un **isthme**.

Exemple

Connexité d'un graphe (3)

Proposition 4

Un graphe $G = (X, E)$ est connexe si et seulement si pour tout $S \subset X$, on a $\omega(S) \neq \emptyset$.

Conséquences

- (i) Si G est connexe alors, pour tout $e \in E$, $G - e$ a au plus 2 composantes connexes.
- (ii) Si G est connexe, alors $m \geq n - 1$.

Chemin et circuit d'un graphe orienté

Soit un graphe orienté $G = (X, A)$.

Définition

Un **chemin** est une suite alternée de sommets et d'arcs de G , $\Gamma = x_0 a_1 x_1 \dots a_k x_k$ telle que pour tout $1 \leq i \leq k$, a_i admet x_{i-1} comme sommet initial et x_i comme sommet terminal.

Les définitions de **chemins fermés**, **simples**, **élémentaires** s'inspirent directement de celles correspondant au concept de graphes non orientés.

Définition

Un **circuit** est un chemin fermé simple.



Remarque : Les notions de chaîne et de cycle peuvent se définir sur un graphe orienté. En effet, la suite alternée de sommets et d'arcs $\Gamma = x_0 a_1 x_1 \dots a_k x_k$ est une chaîne de G si pour $1 \leq i \leq k$, l'arc a_i est incident à x_{i-1} et à x_i . On ne précise alors pas l'orientation des arcs.

Proposition 5

De tout chemin de x à y (resp. chemin fermé contenant un arc a) on peut extraire un chemin élémentaire (resp. circuit élémentaire) de x à y (resp. passant par a).

Graphe orienté

Chemin, circuit, chaîne et cycle

Exemple

G

Chemin élémentaire de G

Chaîne élémentaire de G

Circuit élémentaire de G

Cycle élémentaire de G

Connexité forte (1)

Soit un graphe orienté $G = (X, A)$.

Définition

La **connexité forte** est définie par : $\forall x \in X, \forall y \in X$, il existe un chemin de x à y .

Un graphe est dit **fortement connexe** s'il ne possède qu'une seule composante fortement connexe.



Remarques :

- Un graphe fortement connexe est connexe.
- Dans un graphe fortement connexe, il passe un circuit par chaque point (chemin de x à y et chemin de y à x).
- A tout graphe G , on associe le graphe simple $\tilde{G} = (\tilde{X}, \tilde{A})$ appelé **graphe réduit de G** et défini par :
 - \tilde{X} composante fortement connexe de G et
 - $(\tilde{X} \tilde{Y}) \in \tilde{A} \iff \exists x \in \tilde{X}, \exists y \in \tilde{Y} \text{ tq } (xy) \in A$.

Connexité forte (2)

Exemple : Graphe réduit

Proposition 6

G est fortement connexe si et seulement si $\forall S \subset X, \omega^+(S) \neq \emptyset$.
avec $\omega^+(S) = \{a \in A \mid \text{l'extrémité initiale de } a \text{ appartient à } S\}$
 $\omega^-(S) = \{a \in A \mid \text{l'extrémité finale de } a \text{ appartient à } S\}$

Exploration d'un graphe

Après avoir modélisé un problème sous forme de graphe, il est très souvent nécessaire d'examiner l'ensemble de ses sommets.

On appelle exploration d'un graphe, tout procédé déterministe permettant de choisir, à partir du dernier sommet visité, le sommet suivant.

Cela revient à fixer des règles de priorité dans l'examen des sommets.

Deux types d'exploration sont généralement utilisées :

- **Exploration en largeur** qui donne une partition en couche à partir de la racine.
- **Exploration en profondeur** qui revient à prolonger une chaîne élémentaire.

Remarques :

- Si G n'est pas connexe, l'exploration détermine la composante connexe contenant la racine.
- L'algorithme détermine pour chaque sommet x de la composante, une chaîne élémentaire unique de r à x .

Exploration d'un graphe : Principe

Le principe est de visiter tous les sommets du graphe. Dès qu'un sommet est visité, il est **marqué** ; ce qui permet de ne pas le considérer plusieurs fois. Un sommet est dit **exploré** quand l'ensemble de ses voisins (successeurs) sont marqués.

- Au départ, seul le sommet racine r est marqué.
- Puis, les marques se propagent : tant qu'un arc (xy) avec x marqué et y non marqué est trouvé, y est alors marqué à son tour.
- A la fin, les sommets marqués sont les descendants de r .

Les algorithmes suivants sont présentés pour un graphe G codés par les deux tableaux Succ et Head précédemment introduits.

- Un tableau de booléen Mark indique les noeuds visités : '0' si non visité, '1' sinon.
- Un ensemble Z contient les sommets dont il reste des successeurs à visiter. Cet ensemble Z sera codé différemment selon qu'on considère le parcours en largeur (file) ou en profondeur (pile).

Exploration en largeur

Principe

- Les sommets sont atteints par ordre de distance croissante à s , en nombre d'arcs : sont explorés les successeurs de r , puis les successeurs de ses successeurs...
- Progression en tâche d'huile
- Nom anglais : *breadth-first search* (BFS)

L'ensemble Z est représenté par une file.

- Le premier élément à rentrer dans la file est la racine r . Ce sommet est alors marqué.
- Dès qu'un sommet, non encore marqué, est atteint, il est marqué et inséré à la queue de la file.
- Le sommet en cours d'exploration est situé à la tête de la file. Dès qu'un sommet est exploré, il est retiré de la tête de la file.

Exploration en largeur

Pseudo-code

Initialiser Mark à Faux

Mark[r] := Vrai

Clearset(Z)

EnQueue(Z,r)

Répéter

 DeQueue(Z,x)

 Pour tout successeur y non marqué de x

 Mark[y] := Vrai

 EnQueue(Z,y)

 FinPour

Jusqu'à SetIsEmpty(Z)

Exploration en largeur

Complexité

- **Taille mémoire :**
 - $O(n + m)$ pour G
 - $O(n)$ pour Z et MarkTotal : $O(n + m)$
- **Temps :**
 - $O(n)$ pour l'initialisation
 - $O(m)$ pour l'exploration des sommetsTotal : $O(n + m)$

Remarque :

- Cet algorithme permet aussi de déterminer la partition en couches du graphe G associée à la racine r .

Partition en couches

Distance entre deux sommets

Soient x et y , 2 sommets d'un graphe connexe.

On pose $d(x, y)$, **distance de x à y** , la longueur de la plus courte chaîne entre x et y . Elle correspond au nombre d'arêtes minimal nécessaires pour se déplacer de x à y .

Partition en couches

Dans un graphe connexe, on appelle **partition en couches associée à une racine r** , la suite d'ensembles définie par :

$$C_0 = \{r\}$$

$$C_1 = \{x \in X \mid d(r, x) = 1\}$$

$$C_2 = \{x \in X \mid d(r, x) = 2\}$$

$$C_i = \{x \in X \mid d(r, x) = i\}$$

Remarque : si $|i - j| \geq 2$, il n'existe pas d'arête entre les couches C_i et C_j .

L'algorithme d'exploration en largeur d'un graphe connexe est basé sur une partition en couches des sommets. Il permet même de déterminer les différentes couches, Mark est alors représenté par un vecteur d'entiers contenant pour chaque sommet, le numéro de la couche à laquelle il appartient.

Exploration en profondeur

Principe

- L'exploration procède comme une sonde, en allant le plus loin possible le long d'un chemin d'origine r avant de rebrousser chemin.
- L'exploration d'un sommet se fait en plusieurs phases.
- Nom anglais : *depth-first search* (DFS)

L'ensemble Z est représenté par une pile.

- Le premier élément à rentrer dans la pile est la racine r . Ce sommet est alors marqué.
- Dès qu'un sommet, non encore marqué, est atteint, il est marqué et ajouté à la pile.
- Le sommet en cours d'exploration est situé au sommet de la pile.
- Dès qu'un sommet est exploré, il est retiré du sommet de la pile.

Un tableau *Next*, de taille n , pointe sur le prochain successeur de chaque sommet à explorer.

Exploration en profondeur

Pseudo-code

Initialiser Mark à Faux

Mark[r] := Vrai

Clearset(Z)

Push(Z,r)

Next := Head

Répéter

 x := Front(Z)

 Si Next[x] \geq Head[x+1] alors

 Pop(Z,x)

 Sinon

 y := Succ[Next[x]]

 Next[x] := Next[x]+1

 Si non Mark[y] alors

 Mark[y] := Vrai

 Push(Z,y)

 FinSi

 FinSi

Jusqu'à SetIsEmpty(Z)

Exploration en profondeur

Complexité

- **Taille mémoire :**

- $O(n + m)$ pour G

- $O(n)$ pour Z , Mark et Next

Total : $O(n + m)$

- **Temps :**

- $O(n)$ pour l'initialisation

- $O(m)$ pour l'exploration des sommets

Total : $O(n + m)$

Arbres et arborescence

Arbre (1)

Définition

- Un **arbre** est un graphe connexe sans cycle (on considère ici les graphes sans leur orientation éventuelle).
- Une **forêt** est un graphe sans cycle. Ainsi toute composante connexe d'une forêt est un arbre.



Exemples

Arbre (2)

Proposition 1

Tout arbre à $n \geq 2$ sommets possède au moins deux sommets de degré 1.

Conséquence 1

La classe \mathcal{T} des arbres est définie récursivement par :

- 1- Le graphe à 1 sommet appartient à \mathcal{T} .
- 2- Si $T \in \mathcal{T}$, alors le graphe obtenu à partir de T en ajoutant un sommet de degré 1, appartient à \mathcal{T} .

Remarque : Tout élément de \mathcal{T} s'obtient de cette manière.

Exemple

Arbre (3)

Conséquence 2

Les proposition suivantes sont équivalentes :

- (i) T est un arbre.
- (ii) T est connexe et $m = n - 1$.
- (iii) T est sans cycle et $m = n - 1$.

Proposition 2

Les propositions suivantes sont équivalentes :

- (i) T est un arbre.
- (ii) T est connexe minimal i.e. pour toute arête e , $T - e$ n'est plus connexe.
- (iii) Entre deux sommets quelconques, il existe une et une seule chaîne.

Arbre (4)

Proposition 3

Un graphe est connexe si et seulement si il contient un graphe partiel qui soit un arbre.
Un tel graphe est dit **arbre couvrant**.



Remarque : Les 2 algorithmes d'exploration construisent un arbre pour la composante connexe explorée.

Exemple

Arbre et co-arbre

Définition

Soit $G = (X, E)$ un graphe connexe. G contient un arbre $T = (X, F)$.

On appelle **co-arbre** associé à T le graphe $K = (X, E - F)$, ie. le complémentaire de T par rapport à G .

De manière générale, on appelle co-arbre de G , tout graphe partiel de G dont le complémentaire est un arbre de G .

Proposition 4

Soit $G = (X, E)$, un graphe connexe. Alors,

- $T = (X, F)$ est un arbre de G si et seulement si T est sans cycle et maximal pour cette propriété (i.e. $\forall e \in E \setminus F$, $T + e$ contient un cycle de G).
- $K = (X, U)$ est un co-arbre de G si et seulement si K ne contient pas de cocycle de G et est maximal pour cette propriété (i.e. $\forall e \in E - U$, $K + e$ contient un cocycle).

Graphes bipartis

Application I

Rappel : l'ensemble des sommets d'un graphe biparti peut être décomposé en 2 stables.

Proposition 5

L'ensemble des sommets de tout arbre ($n \geq 2$) admet une partition unique en 2 stables.

Démonstration :

Proposition 6

Un graphe est biparti si et seulement si il ne contient pas de cycle de longueur impaire. De plus, si le graphe est connexe, la partition est unique.

Problème de l'arbre de poids optimal (1)

Application II

Soit $G = (X, E)$ un graphe connexe à n sommets.

Soit $p : E \rightarrow \mathbb{R}$, une application qui associe à toute arête de G un poids réel.

Si T est un arbre de G , alors le poids total de T est défini par $p(T) = \sum_{e \in T} p(e)$.

On cherche alors à déterminer un **arbre couvrant de poids optimal** (par exemple de poids minimal, sinon, on changera p en $-p$).

Exemple

Problème de l'arbre de poids optimal (2)

Application II

Algorithme de Kruskal

1/ Trier la liste des arêtes par poids croissant

2/ Construire la séquence :

$$T_1 = \emptyset$$

$$T_i = T_{i-1} + e_{\lambda_i}$$

où λ_i est le plus petit entier k tel que e_k ne forme pas de cycle avec les arêtes de T_{i-1} .

Exemple



Problème de l'arbre de poids optimal (3)

Application II

Complexité de l'algorithme de Kruskal

Cet algorithme est très simple dans sa conception. Chaque arête est examinée une et une seule fois. C'est un **algorithme glouton**.

Tri des m arêtes : avec un bon algorithme de tri, $O(m \log m)$

Vérification de la non existence de cycle : peut être linéaire avec une bonne structure de données.

Complexité globale : $O(m \log m)$.

Arbre enraciné

Définition

Un **arbre enraciné** est un arbre où un de ses sommets, appelé racine de l'arbre (noté r), peut être distingué.

Ce sommet permet de partitionner les sommets en couches :

$$x \in C_i \leftrightarrow i \text{ est la longueur de la plus courte chaîne de } r \text{ à } x.$$

Remarque : i est aussi la longueur de la plus longue chaîne de r à x car dans un arbre il existe une et une seule chaîne reliant 2 sommets.

Exemple

Arborescence

Définition

Une **arborescence** est un graphe orienté $G = (X, A)$ ayant un sommet distingué r appelé racine de l'arborescence tel que $\forall x \in X$, il existe un chemin unique de r à x .

Une arborescence est connexe et sans circuit.

De plus, si $n \geq 2$, toute arborescence possède une source unique (la racine r) et au moins un puits (p tel que $d^-(p) = 1$ et $d^+(p) = 0$).

Proposition 7

Les propositions suivantes sont équivalentes :

- (i) A est une arborescence de racine r
- (ii) A est connexe et contient un sommet r tel que $d^-(r) = 0$ et $\forall x, x \neq r, d^-(x) = 1$
- (iii) A est sans circuit et contient un sommet r tel que $d^-(r) = 0$ et $\forall x, x \neq r, d^-(x) = 1$

Arborescence

Représentation

Les arborescences étant des graphes, il est possible d'utiliser les représentations présentées précédemment.

Mais les arborescences ont des propriétés spécifiques que l'on peut exploiter afin d'utiliser une représentation plus adaptée.

En effet, $\forall x \neq r, d^-(x) = 1$. Il existe donc une application $pred : X - \{r\} \rightarrow X$, fonction prédécesseur qui associe à chaque sommet un prédécesseur unique.

Ainsi, on peut utiliser une représentation qui associe à chaque sommet un enregistrement à deux champs : le premier contiendra le nom du sommet et le deuxième champ donnera l'adresse de son prédécesseur.

Exemple

Arborescence

Classe

Définition

La classe des arborescences \mathcal{A} est définie récursivement de la manière suivante :

- 1- Le graphe à 1 sommet appartient à \mathcal{A} .
- 2- Si $(A_1, r_1), \dots (A_p, r_p) \in \mathcal{A}$ ($p \geq 1$) et disjointes alors, en donnant à r_1, r_2, \dots, r_p un prédécesseur commun r , on obtient une nouvelle arborescence.

Remarque : Toute arborescence s'obtient en appliquant un nombre fini de fois les règles précédentes.

Exemple

Arborescence d'un graphe orienté

De la même manière que pour les graphes connexes non orientés, on veut savoir si un graphe orienté connexe admet un graphe partiel qui soit une arborescence.

Pour cela il est nécessaire de définir ce que serait la racine d'une arborescence d'un graphe orienté.

Définition

Un sommet r est une **racine** d'un graphe G orienté si pour tout sommet x de G , il existe un chemin de r à x .

Exemple

Arborescence d'un graphe orienté

Proposition 8

Un graphe G contient un graphe partiel qui soit une arborescence de racine r si et seulement si r est une racine de G .

Conséquence

Soit $G = (X, A)$ un graphe orienté fortement connexe. Alors pour tout arc $a \in A$, il existe un graphe partiel contenant a qui soit une arborescence dont la racine est l'extrémité initiale de a .

Recherche d'une arborescence couvrante (1)

Définition

On appelle **arborescence couvrante** tout graphe partiel de G (si il existe) qui soit une arborescence.

Soit $G = (X, A)$, un graphe orienté enraciné en $r \in X$ (ie. qui admet r comme racine).

- Si G est fortement connexe, la recherche d'une arborescence couvrante est simple.
- Dans le cas contraire, on utilise la construction suivante...

Recherche d'une arborescence couvrante (2)

Soit $X = X_1 \cup X_2 \cup \dots \cup X_p$, la partition de X où pour tout i , X_i engendre une composante fortement connexe de G .

Considérons le graphe réduit de G , \tilde{G} . C'est un graphe sans circuit, admettant une source unique (donc une racine) correspondant à la composante fortement connexe contenant r . Lorsque l'on connaît le graphe réduit \tilde{G} , on peut rechercher une arborescence de la manière suivante :

1/ Choisissons une arborescence dans \tilde{G} .

Cette arborescence détermine $p - 1$ arcs dans G où chaque arc va d'une composante fortement connexe à une autre.

2/ Soit $(x_i r_j)$, un arc allant de X_i à X_j .

X_j est fortement connexe, donc il existe une arborescence enracinée en r_j .

3/ L'ensemble ainsi obtenu forme une arborescence de G enracinée en 1 sommet quelconque de la composante de r .

Recherche d'une arborescence couvrante (3)

Exemple

Application I

Problème du plus court chemin

Problème du plus court chemin (1)

Soit $G = (X, A, w)$, un **graphe valué** (ou pondéré) tel que :

- $G = (X, A)$ est un graphe orienté
- $w : A \rightarrow \mathbb{R}$ est une application qui associe à tout arc de G un poids réel (coût, distance...)

Définition

La **longueur d'un chemin** Γ dans un graphe G est égale à la somme des longueurs des arcs qui constituent ce chemin.

$$l(\Gamma) = \sum_{a \in \Gamma} w(a)$$

Cette définition généralise la notion de longueur d'un chemin que l'on avait utilisée jusqu'à présent et qui correspondait *au nombre d'arcs*. Ce qui était équivalent à un poids w telle que $\forall a \in A, w(a) = 1$.

Problème du plus court chemin (2)

Etant donné deux sommets x et y d'un graphe G , 3 cas peuvent se présenter :

- Il n'existe pas de chemin entre x et y .
- Il existe un chemin entre x et y , mais pas de plus court chemin.
- Il existe un plus court chemin entre x et y .

Exemple

Problème du plus court chemin (3)

- L'existence d'un chemin entre deux sommets x et y peut être vérifiée en appliquant un algorithme d'exploration du graphe à partir de x (jusqu'à trouver y).
- L'existence d'un chemin ne garantit pas qu'il existe un plus court chemin : cas des circuits absorbants. Ce phénomène est dû à l'existence d'un circuit C pour lequel sa longueur est négative ($l(C) < 0$). Ainsi, au plus on parcourt ce circuit, au plus la longueur totale du chemin diminue.
- La recherche du plus court chemin élémentaire permet de s'affranchir des circuits absorbants.

Problème du plus court chemin (4)

Remarque : Combinatoire explosive...

Il existe dans un graphe fini un nombre fini de chemins élémentaires. La recherche de chemins élémentaires entre 2 sommets fixés sont des problèmes d'optimisation combinatoire non difficiles en principe car il "suffit" d'énumérer tous les chemins élémentaires possibles et de choisir celui de longueur minimale. La difficulté pratique provient de l'augmentation très rapide du nombre de chemins avec le nombre de sommets et d'arcs, ce qui rend l'énumération impossible.

Dans un graphe complet à n sommets, il existe $n!$ chemins élémentaires.

Si $n = 5$, il suffit d'énumérer 120 chemins.

Si $n = 10$, un peu moins de 4 millions de chemins existent.

Si $n = 25$, un peu plus de 10^{25} chemins sont possibles. Avec un ordinateur qui évaluerait 10^9 chemins par secondes, il faudrait l'âge de la Terre pour pouvoir tous les évaluer!!!

Nécessité de connaître des algorithmes qui donnent le plus court chemin (quand il existe) entre deux sommets sans énumérer toutes les possibilités.

Problème du plus court chemin (5)

Le problème du plus court chemin peut être formulé de 3 façons différentes :

- A/ Etant donné deux sommets r et t , trouver un plus court chemin de r à t .
- B/ Etant donné un sommet de départ r , trouver un plus court chemin de r vers tout autre sommet.
- C/ Trouver un plus court chemin entre tout couple de sommets, ie. calculer une matrice appelée *distancier*.

Le problème B est celui qui nous intéresse car c'est le plus général. En effet, la résolution du problème B permet de traiter directement le problème A : l'algorithme est stoppé dès que le sommet de destination t a été définitivement traité. De même, le problème C peut être traité en appliquant le problème B pour chaque sommet du graphe comme sommet de départ. Pour les problèmes A et C, il peut exister des algorithmes très spécifiques nécessitant certaines contraintes sur le graphe qui s'avèrent meilleurs en complexité que ceux proposés dans le cas du problème B. Ceux-ci ne seront pas étudiés dans ce cours.

Problème du plus court chemin (6)

Problème : étant donné un sommet de départ r , trouver un plus court chemin de r vers tout autre sommet du graphe $G = (X, A, w)$ (n sommets et m arcs).

- **w constante** : revient à trouver les plus courts chemins en nombre d'arcs.
Résolution avec une exploration de graphe en largeur, implémentable en $O(m)$.
- **$w \geq 0$** : algorithme de Dijkstra avec une complexité en $O(n^2)$.
- **G sans circuit** : algorithme de Bellman avec une complexité en $O(m)$ (si les sommets sont triés).
- sinon : algorithme de Bellman-Ford (programmation dynamique)

Résultats :

- π est un tableau contenant pour chaque sommet x , le coût du plus court chemin $\pi[x]$.
Initialisation : $\pi[x] = +\infty$ pour tout $x \neq r$ et $\pi[r] = 0$.
- Père est un tableau stockant le père de chaque sommet.
 $\text{Père}[x] = y$ signifie que y précède x dans le chemin optimal reliant r à x .
 $\text{Père}[x] = 0$ signifie que x n'a pas été atteint par un chemin d'origine r .
Initialisation : $\text{Père}[x] = 0$ pour tout $x \neq r$ et $\text{Père}[r] = r$.

Algorithme de Dijkstra

Conditions d'application de l'algorithme de Dijkstra :

- Poids positifs ou nuls
- Graphe avec ou sans circuit(s)

Principe

Calcule les plus courtes longueurs de proche en proche par ajustements successifs.

- Initialiser π , Père et Mark

Tant qu'il existe des sommets non marqués :

- Choisir parmi les sommets non marqués, le sommet x avec $\pi[x]$ minimum.
- Marquer x
- Pour tous les successeurs y de x ,
Si $\pi[y] > \pi[x] + w(xy)$ Alors $\pi[y] = \pi[x] + w(xy)$ et Père[y] = x .

Algorithme de Dijkstra

Pseudo-code

Initialiser Mark à Faux, π à $+\infty$, Père à 0 et Fini := False
 $\pi[r] := 0$, Père[r] := r

Répéter

 Fini := True

 Choisir x tel que Mark[x]=Faux et $\pi[x]$ minimum

 Si x existe Alors

 Fini := False

 Mark[x] := Vrai

 Pour k := Head[x] à Head[x+1]-1

 y := Succ[k]

 Si $\pi[y] > \pi[x] + w(xy)$ Alors

$\pi[y] := \pi[x] + w(xy)$

 Père[y] := x

 FinSi

 FinPour

FinSi

Jusqu'à (non Fini)

Algorithme de Dijkstra

Remarques :

- L'algorithme de Dijkstra fournit l'arborescence des plus courts chemins enracinés en r ainsi que toutes les longueurs $\pi(x)$ de r à x .
- Preuve de l'optimalité : par récurrence
- Complexité :

$$O(n + \sum_{i=1}^n (n + d^+(x))) = O(n^2)$$

Recherche du potentiel π minimum : $O(n)$

Implémentation avec un tas : $O(\log n)$

Algorithme de Dijkstra

Déroulement sur un exemple

Graphe sans circuit : ordre topologique

Théorème

Un graphe sans circuit contient au moins une source, ie. un sommet x tel que $d^-(x) = 0$.

Conséquence

Un graphe orienté $G = (X, A)$ est sans circuit si et seulement si X admet une partition en niveaux $X = N_0 \cup N_1 \cup \dots \cup N_p$ telle que pour tout x et pour tout i :

$x \in N_i \Leftrightarrow i$ est la longueur (taille) d'un plus **long** chemin se terminant en x .

Remarque : Les niveaux N_i forment des stables.

Ainsi, les sommets sont triés par un tri topologique tel que :

$$\forall x \in X, \forall y \in X \text{ tels que } (xy) \in A \text{ alors } n(x) < n(y)$$

Ici, on numérote d'abord les sommets appartenant à N_0 puis N_1 , ...

Algorithme de Bellman

Conditions d'application de l'algorithme de Bellman :

- Graphe sans circuit
- Toute valeur réelle de poids
- Les sommets doivent être triés dans l'ordre topologique (tableau Sorted)

Principe

Calcule les plus courtes longueurs en suivant la numérotation des sommets.

- Initialiser π , Père

Pour k de 2 à n :

- $y = \text{Sorted}[k]$
- Choisir le sommet x parmi les prédécesseurs de y tel que
$$\pi[y] = \min(\pi[x] + w(xy))$$

Algorithme de Bellman

Pseudo-code

Initialiser π à $+\infty$, Père à 0

$\pi[r] := 0$, Père[r] := r

Pour i := 2 à n

 y := Sorted[i]

 Pour tout prédécesseur x de y

 Si ($\pi[x] \neq \infty$) et ($\pi[x] + w(xy) < \pi[y]$) Alors

$\pi[y] := \pi[x] + w(xy)$

 Père[y] := x

 FinSi

 FinPour

FinPour

Algorithme de Bellman

Remarques :

- Complexité : $O(m)$
- Nécessite de trier les sommets par niveaux (en $O(n + m)$ avec un algorithme de parcours en profondeur)
- Variante : utiliser les niveaux pour éviter les prédécesseurs.

```
Initialiser  $\pi$  à  $+\infty$ , Père à 0
 $\pi[r] := 0$ , Père[r] := r
Pour tout x successeur de r (ie.  $x \in N[1]$ )
     $\pi[x] := w(rx)$ 
    Père[x] := r
FinPour
Pour i := 1 à L /* L est le nombre total de niveau */
    Pour tout x dans N[i]
        Pour tout successeur y de x tel que  $\pi[x] + w(xy) < \pi[y]$ 
             $\pi[y] := \pi[x] + w(xy)$ 
            Père[y] := x
        FinPour
    FinPour
FinPour
```

Algorithme de Bellman

Déroulement sur un exemple

Application II

Problème d'ordonnancement

Problème d'ordonnancement (1)

La conception et la gestion d'un projet complexe composé de multiples travaux élémentaires pose des problèmes délicats :

- de planification : définition du calendrier, gestion de l'enchaînement des tâches, ...
- de contrôle de l'exécution du projet (coordination).

Il est donc nécessaire de faire appel à des techniques d'ordonnancement qui permettent de déterminer dans quel ordre doivent être exécutées les tâches de façon à *optimiser* un objectif (minimiser les coûts, minimiser le temps, ...).

Définition

On se trouve face à un **problème d'ordonnancement** lorsque, en vue de la réalisation d'un objectif quelconque, il faut accomplir de multiples tâches, elles-mêmes soumises à un ensemble de contraintes :

- *contraintes de précedence / succession* qui imposent qu'une tâche j ne peut commencer avant une autre tâche i .
- *contraintes disjonctives* qui empêchent 2 tâches i et j de s'exécuter en même temps (parce qu'elles partagent une même ressource, par exemple).
- *contraintes cumulatives* qui permettent de tenir compte des moyens humains et matériels disponibles. Les contraintes cumulatives sont très difficiles à prendre en compte.

Problème d'ordonnancement (2)

Exemple et visualisation

Exemple

Une entreprise de jouets fabrique des jouets en bois. Chaque pièce doit être découpée, poncée puis teintée (dans cet ordre !!). L'atelier comporte des machines qui découpent (M1), poncent (M2) et teignent (M3). Voici pour trois types de pièces, les temps d'usinage en secondes pour chaque opération.

	T1	T2	T3
M1	40	80	70
M2	50	60	80
M3	70	60	50

Objectif : rechercher la durée minimale pour fabriquer ces trois pièces.

Pour cela on visualise l'avancement des travaux (occupation des machines et usinages des pièces) par un diagramme de GANTT dans lequel :

- l'axe horizontal représente le temps,
- l'axe vertical représente les ressources (ici les machines),
- une tâche élémentaire est représentée par un rectangle.

Problème d'ordonnancement (3)

Exemple et visualisation

Visualisation : 2 cas

- 3 machines de chaque type M1, M2, M3 sont disponibles
- 1 machine de chaque type M1, M2, M3 est disponible

Problème d'ordonnancement (4)

Définition

On appelle **ordonnancement simple** des problèmes d'ordonnancement pour lesquels :

- le nombre de ressources n'est pas limité,
- les tâches ont une durée d'exécution connue,
- seules des contraintes de précédence existent.

En général, l'**objectif** est de rechercher la **durée minimale du projet** ie. la durée minimale de l'exécution de l'ensemble des tâches.

Modélisation potentiel-tâche : les tâches sont représentées par les sommets

Date au plus tôt, date au plus tard

Plaçons nous dans le cas d'une minimisation de la longueur totale d'un projet.

Il est alors possible de définir pour chaque tâche x :

- La date au plus tôt, $\pi(x)$: c'est la date minimum à laquelle on peut démarrer l'exécution de la tâche x en fonction des données et des contraintes du problème (notamment en fonction des contraintes de précédence).
- La date au plus tard, $\eta(x)$: c'est la date limite à laquelle on doit commencer la tâche x sous peine de retarder l'exécution totale du projet.
- La marge, $m(x)$, qui représente le délai dont on dispose pour lancer la tâche x .
 $m(x) = \eta(x) - \pi(x)$.

Les tâches ayant une marge nulle ($\eta(x) = \pi(x)$) sont appelées **tâches critiques**, car pour ces tâches, un retard pris dans l'exécution entraîne un retard dans la réalisation du projet.

Méthode Potentiel-tâche

Exemple

La construction des fondations d'une maison se compose de 5 tâches élémentaires :

- T : Terrassement - Durée 3 jours
- G : Mise en place de la grue - Durée 1 jour
- R : Branchement au réseau EDF et d'eau - Durée 2 jours
- B : Coulage de la dalle en béton - Durée 3 jours
- S : Mise en place de la fosse septique - Durée 4 jours

Les tâches T , G , R peuvent commencer tout de suite.

Par contre, les tâches S et B nécessitent la tâche T .

B a besoin de l'eau et de la grue qui a besoin de l'EDF pour fonctionner.

Le problème se résume sous la forme d'un tableau :

Tâche	durée	précédence
T	3	-
G	1	-
R	2	-
B	3	T, G, R
S	4	T

Méthode Potentiel-tâche

Modélisation

Une tâche est représentée par un sommet. Un arc représente une contrainte de précédence.

- Ajout des sommets D et F pour modéliser resp. le début et la fin des travaux.
- Sur les arcs, on indique la durée de la tâche correspondant au sommet initial.

Résolution

- On calcule les valeurs $\pi(x)$, en utilisant l'algorithme de Bellman à partir du sommet de début. Ici, on cherche le potentiel maximum de chaque tâche.
- On calcule les valeurs $\eta(x)$ en appliquant l'algorithme de Bellman sur l'antiarborescence issue du sommet de fin. Ici, on cherche le potentiel minimum de chaque tâche par rapport à la durée minimale trouvée $\pi(F)$.

Durée des travaux :

Tâches critiques :

Tâches retardables :

Application III

Problème du flot maximum

Problème de flot (1)

Définitions

Un **réseau** est un graphe $R = (X, A, c)$, sans boucle comportant 2 sommets particuliers : s , la source et p , le puits. Une application $c : A \rightarrow \mathbb{R}^+ \cup \{+\infty\}$ associe à chaque arc a sa capacité $c(a)$.

En général, on ajoute à ce réseau, un arc de retour $a_r = (ps)$, qui revient du puits vers la source (cet arc forme un circuit avec le réseau), de capacité infinie.

Un **flot** dans un réseau est une application $f : A \rightarrow \mathbb{R}^+$ telle que :

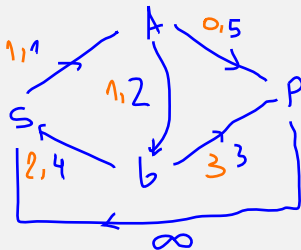
- $\forall a, f(a) \leq c(a)$
- $\forall w \in X, \sum_{u\vec{w} \in A} f(uw) = \sum_{w\vec{v} \in A} f(wv)$

La première contrainte impose un respect des capacités du réseau, tandis que la deuxième s'assure de la conservation du flux en chaque sommet.

Problème de flot (2)

Exemple

On considère un réseau de conduits (tuyaux, câbles électriques) ayant chacun une capacité (limite supérieure de la quantité de flux qui peut passer dans le conduit par unité de temps). On cherche à envoyer la plus grande quantité de flux d'un sommet origine donné s à un sommet p sachant que le flux est conservatif (la somme des flux entrant en un sommet intermédiaire du réseau est égale à la somme des flux sortant de ce sommet).



Réseau

Une solution

Objectif : On cherche la valeur maximale du flot qui peut circuler dans le réseau.

Problème de flot (3)

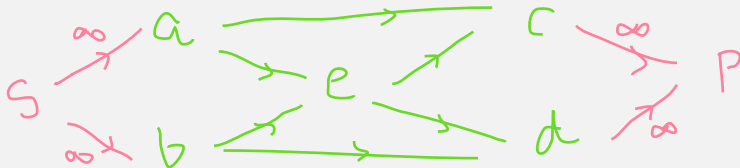
Généralisation

Sources et puits multiples

Une généralisation du problème précédent peut être considérée comme un problème de flot maximal : on suppose qu'il existe maintenant, non plus une seule source et un seul puits, mais tout un ensemble S de sommets sources et un ensemble P de sommets puits.

Ce cas se réduit au précédent en ajoutant au réseau initial un sommet s dit "super source" et un sommet p appelé "super puits", ainsi que des arcs (sx) pour tout $x \in S$ et des arcs (yp) pour tout $y \in P$, de capacité infinie.

Exemple



Problème de flot (4)

Généralisation

Voici quelques transformations qui peuvent modifier un peu les problèmes de recherche de flot maximal.

- Dans le réseau, il peut exister plusieurs arcs entre 2 sommets. Dans ce cas, on remplace ces arcs par un seul arc, dont la capacité sera égale à la somme des capacités des arcs d'origine.
On peut donc supposer qu'il n'existe pas d'arcs multiples dans le réseau.
- De même on peut imposer que le graphe est symétrique, ie. $\forall (ij) \in A, (ji) \in A$. Si un des deux arcs n'existe pas, on peut toujours le rajouter en lui donnant une capacité nulle (ainsi nous sommes sûrs qu'il ne sera pas utilisé).
- Il est possible de définir des capacités sur les nœuds du graphe. Par exemple, lorsque un nœud représente une station de pompage de capacité limitée. Dans ce cas, on dédouble le nœud x en question et on relie les 2 sommets x' et x'' obtenus, par un arc de capacité, la capacité du nœud x .

Algorithme de Ford-Fulkerson

Permet de trouver un flot maximal Φ de s à p dans un réseau $R = (X, A, c)$.

Définition

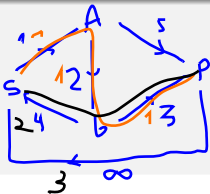
Un flot est dit **réalisable** lorsqu'il respecte les contraintes de capacité et de conservation du flot.

Un arc est dit **saturé** par le flot f si $f(a) = c(a)$.

Principe

Rechercher à partir d'une valeur donnée f du flot sur les arcs (flot réalisable), s'il existe un chemin de s à p le long duquel aucun arc n'est saturé. Si un tel chemin C existe, on peut améliorer la solution courante en augmentant sur les arcs du circuit Γ constitué de $C \cup \{ar\}$ la valeur du flot de $\delta = \min_{a \in \Gamma} [c(a) - f(a)]$

Exemple



On décide de passer 1 ds le premier chemin. Valeur du flot est donc de 1.

On décide de passer par le chemin noir (parce que $s \rightarrow a$ est saturé). De $b \rightarrow p$, on a déjà un flot de 1 donc on peut que rajouter 2. On augmente donc le flot à 2

Algorithme de Ford-Fulkerson

Préliminaires (1)

Définition

Soit $G = (X, A)$ un graphe orienté.

On appelle **graphe inverse** (ou graphe des prédécesseurs) de G , le graphe ayant pour sommets l'ensemble X et pour arcs, l'ensemble des arcs inversés de A . Pour obtenir la représentation graphique du graphe inversé de G , il suffit d'inverser le sens des flèches.

Soient G un graphe orienté et H son graphe inverse.

Si $(xy) \in G$ alors $\exists p$ tel que $y = G.\text{Succ}[p]$
et $(yx) \in H$ et $\exists k$ tel que $x = H.\text{Succ}[k]$.

On définit le tableau Inv tel que $\text{Inv}[k] = p$ qui permet de passer d'un arc à son inverse dans l'implémentation.

Algorithme de Ford-Fulkerson

Préliminaires (2)

Définition

Pour un réseau $R = (X, A, c)$ et un flot f réalisable dans ce réseau. Au couple (R, f) on associe un **graphe d'écart** $R^e(f) = (X, A^e, c^e)$ où :

- R^e a le même ensemble de sommets que R ,
- À tout arc $(xy) \in A$, de capacité $c(xy)$ et de flux $f(xy)$, on associe dans $R^e(f)$:
 - un arc (xy) de capacité $c^e(xy) = c(xy) - f(xy)$
si l'arc n'est pas saturé (ie. si $f(xy) < c(xy)$)
 - un arc (yx) de capacité $c^e(yx) = f(xy)$
si le flux passant par (xy) n'est pas nul (ie. si $f(xy) > 0$)



Exemple

Algorithme de Ford-Fulkerson

Préliminaires (3)

La **procédure de marquage** permet, étant donné un flot f sur le réseau R , de détecter si un cycle Γ non saturé existe. La marque $\delta(x)$ associe à un sommet x la quantité dont on pourrait augmenter le flot de s à x en suivant une chaîne. La notion de graphe d'écart est sous-jacente à cette procédure. Deux types de marquages sont possibles :

Marquage direct

Si un sommet x est marqué ($\delta(x)$) et
s'il existe y non marqué tel que $f(xy) < c(xy)$ (ie. arc (xy) est non saturé)

Alors :

$$\delta(y) \leftarrow \min[\delta(x), c(xy) - f(xy)] \quad (\text{et } y \text{ est marqué})$$

Marquage indirect

Si un sommet x est marqué ($\delta(x)$) et
s'il existe y non marqué tel que $f(yx) > 0$

Alors :

$$\delta(y) \leftarrow \min[\delta(x), f(yx)] \quad (\text{et } y \text{ est marqué})$$

La **procédure de changement de flot** est appelée lorsqu'un chemin non saturé a été mis en évidence par la procédure de marquage, ie. lorsque la procédure de marquage termine avec p marqué et $\delta > 0$.

Algorithme de Ford-Fulkerson

L'algorithme de Ford-Fulkerson est divisé en deux procédures :

- procédure de marquage
- procédure de changement de flot

Structure

Initialiser $\Phi := 0$, $f := 0$, $\delta := \infty$

Tant que $\delta > 0$

 {procédure de marquage}

 Chercher une chaîne améliorante μ de s à p dans R

 Si μ existe Alors Calculer δ , augmentation du flot possible sur μ

 Sinon $\delta := 0$

 {procédure de changement de flot}

 Augmenter Φ et les flux des arcs avant de μ de δ unités

 Diminuer les flux des arcs arrière de μ de δ unités

Fin Tant que

Algorithme de Ford-Fulkerson

Déroulement sur un exemple

Coupe de poids minimum

Définition

Dans un réseau $R = (X, A, c)$ avec $a_r = (ps)$, un ensemble d'arcs \mathcal{C} est appelé "**coupe séparant p de s** " si on peut trouver $Y \subset X$ avec $s \in Y$, $p \notin Y$ tel que

$$\mathcal{C} = \{a \in A \text{ tq } I(a) \in Y, T(a) \notin Y\}$$

En fait, \mathcal{C} est le cocycle sortant de Y ($\mathcal{C} = \omega^+(Y)$)

Remarques :

- Une coupe \mathcal{C} séparant p de s intersecte tout chemin de s à p dans R .
- On appelle capacité d'une coupe \mathcal{C} séparant p de s , la somme des capacités des arcs de \mathcal{C} .

$$C(\mathcal{C}) = \sum_{a \in \mathcal{C}} c(a)$$

Théorème de la coupe de poids minimum

Théorème 1

Pour tout flot réalisable f sur $R = (X, A, c)$ avec $c(a_r) = \infty$ et pour toute coupe \mathcal{C} séparant p de s , on a :

$$f(a_r) \leq C(\mathcal{C})$$

Théorème 2

Théorème de la coupe de poids minimum - Ford-Fulkerson

La valeur maximum $f(a_r)$, pour un flot réalisable sur $R = (X, A, c)$, est égale à la capacité d'une coupe de capacité minimum séparant p de s .

Exemple

Terminaison de l'algorithme de Ford-Fulkerson

Théorème

Lorsque l'algorithme se termine avec un flot f sans que l'on ait pu marquer p (dans la procédure de marquage), f est solution optimale du problème du flot maximum de s à p dans \mathbb{R} .

Démonstration :

Flots canalisés

Soit le réseau $R' = (X, A, b, c)$, une extension du réseau $R = (X, A, c)$ étudié précédemment où $b : A \rightarrow \mathbb{R} \cup \{-\infty\}$, $c : A \rightarrow \mathbb{R} \cup \{+\infty\}$ et $\forall a, b(a) \leq c(a)$.

Définition

Un flot dans le réseau R' est une application $f : A \rightarrow \mathbb{R}^+$ telle que :

- $\forall a, b(a) \leq f(a) \leq c(a)$
- $\forall w \in X, \sum_{u\vec{w} \in A} f(uw) = \sum_{w\vec{v} \in A} f(wv)$

Ce flot est dit **canalisé**.

Remarque : le problème du flot maximum correspond à celui-ci avec $b(a) = 0$, pour tout arc a .

Exemple

Flots canalisés : amélioration d'un flot

La recherche d'un flot canalisé maximum peut être effectuée par une extension très simple de l'algorithme de Ford-Fulkerson en modifiant la procédure de marquage inverse.

Marquage inverse modifié

Si un sommet x est marqué ($\delta(x)$) et
s'il existe y non marqué tel que $f(yx) > b(yx)$

Alors :

$$\delta(y) \leftarrow \min[\delta(x), f(yx) - b(yx)] \quad (\text{et } y \text{ est marqué})$$

Ainsi l'algorithme modifié de Ford-Fulkerson permet d'améliorer un flot existant. Reste le problème de la détermination d'un flot de départ, car contrairement au problème classique, le flot $\Phi = 0$ n'est pas toujours une solution réalisable.

Flots canalisés : recherche d'un flot initial

Théorème de Hoffman

Il existe un flot canalisé sur $R = (X, A, b, c)$
si et seulement si pour tout cocycle $\omega(Y)$ du graphe (X, A) ($Y \subset X$),
on a :

$$\sum_{a \in \omega^-(Y)} b(a) \leq \sum_{a \in \omega^+(Y)} c(a)$$

Exemple

La recherche d'un flot initial peut se faire en modifiant le graphe et en appliquant itérativement la procédure de recherche d'un flot canalisé maximum. Nous ne détaillerons pas ici cette méthode un peu complexe.

Application : Problème des cases admissibles (1)

On se donne un tableau rectangulaire $m \times n$ (m lignes, n colonnes) dont certaines cases sont interdites (les autres sont dites "admissibles"). On se donne également $m + n$ entiers positifs ou nuls d_1, d_2, \dots, d_m et b_1, b_2, \dots, b_n .

Le problème consiste à associer à chaque case admissible un nombre entier de telle sorte que :

- La somme des nombres affectés aux cases d'une ligne i soit inférieure ou égale à d_i ($i = 1, 2, \dots, m$).
- La somme des nombres affectés aux cases d'une colonne j soit inférieure ou égale à b_j ($j = 1, 2, \dots, n$).
- La somme des nombres affectés à l'ensemble des cases soit maximum.

Application : Problème des cases admissibles (2)

Ce problème revient à **rechercher le flot maximum** dans le réseau $R = (X, A, c)$ défini de la façon suivante :

- $X = L \cup C \cup \{s, p\}$ où
 $L = \{l_1, l_2, \dots, l_m\}$ et $C = \{c_1, c_2, \dots, c_n\}$.
- $A = A_1 \cup A_2 \cup A_3 \cup \{a_r\}$ où
 - $A_1 = \{(sl_i) \mid i = 1, 2, \dots, m\}$
 - $A_2 = \{(c_j p) \mid j = 1, 2, \dots, n\}$
 - $A_3 = \{(l_i c_j) \mid (i, j) \text{ est une case admissible} \}$
 - $a_r = (ps)$

- et $c : A \rightarrow \mathbb{R}$ telle que

$$c(a) = \begin{cases} +\infty & \text{si } a = a_r \text{ ou } a \in A_3 \\ d_i & \text{si } a = (sl_i) \in A_1 \\ b_j & \text{si } a = (c_j p) \in A_2 \end{cases}$$

Exemple

Recherche d'un flot maximum de coût minimal (1)

Soit le réseau $R = (X, A, c, k)$ une extension du réseau $R = (X, A, c)$, où $k : A \rightarrow \mathbb{R}^+$ tel que $k(a)$ représente le coût unitaire de l'arc a .

Objectif : rechercher dans ce réseau le flot maximal de coût minimal

Ce problème bi-objectif (flot maximal, coût minimal) est difficile à résoudre. En effet que privilégier ? Le flot ? Le coût ?

Ici on recherche **le flot de coût minimal parmi les flots de valeur maximale**.

Le coût (valeur) d'un flot est calculé par :

$$v(f) = \sum_{(xy) \in A} k(xy) \cdot f(xy)$$

Recherche d'un flot maximum de coût minimal (2)

Extension de la notion de graphe d'écart aux réseaux avec coûts

Pour un réseau $R = (X, A, c, k)$ et un flot f réalisable dans ce réseau. Au couple (R, f) on associe un **graphe d'écart** $R^e(f) = (X, A^e, c^e, k^e)$ où :

- R^e a le même ensemble de sommets que R ,
- À tout arc $(xy) \in A$, de capacité $c(xy)$ et de flux $f(xy)$, on associe dans $R^e(f)$:
 - un arc (xy)
 - de capacité $c^e(xy) = c(xy) - f(xy)$
 - de coût $k^e(xy) = k(xy)$
 - si l'arc n'est pas saturé
 - un arc (yx)
 - de capacité $c^e(yx) = f(xy)$
 - de coût $k^e(yx) = -k(xy)$
 - si le flux passant par (xy) n'est pas nul

Algorithme de Roy

Soit $R = (X, A, c, k)$. Cet algorithme recherche le flot maximum de coût minimal, en recherchant itérativement le chemin de coût minimal dans le graphe d'écart courant.

Structure

Initialiser f à 0

Répéter

 Construire $R^e(f) = (X, A^e, c^e, k^e)$

 Recherche Γ , le chemin de coût minimal de s à p dans $R^e(f)$

 Si Γ existe Alors

$\epsilon := \min_{(xy) \in \Gamma} (c^e(xy))$

$\forall (xy) \in \Gamma, f(xy) = f(xy) + \epsilon$

 FinSi

Jusqu'à Γ n'existe pas

Calculer le coût du flot

Algorithme de Roy

Déroulement sur un exemple

