- Présentation
- Les graphes
- Les arbres
- Représentation
- Complexité
- Chemin
- Ordonnancement
- Flot maximum
- Prog. linéaire
- Version PDF

Top Back Next

- Définitions
- Plus court chem.
- Dijkstra
- Dantzig

RECHERCHE DU PLUS COURT CHEMIN

DEFINITIONS

Réseau

Un réseau est un graphe G = (X;U) auquel on associe une fonction d: U → IR qui à chaque arc fait correspondre sa "longueur". On note R = (X;U;d) un tel réseau.

Longueur

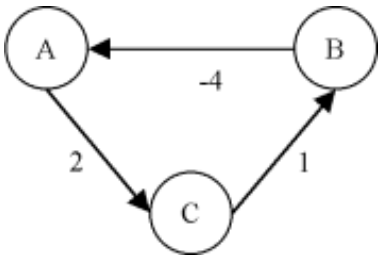
La longueur d'un chemin (d'une chaîne, d'un circuit ou d'un cycle) est la somme des longueurs de chaque arc qui le compose. Par convention, un chemin (une chaîne, un circuit ou un cycle) qui ne contient pas d'arc est de longueur nulle.

Circuit absorbant

Un circuit est absorbant si sa longueur est négative.

Exemple

La longueur du circuit suivant est -1. C'est donc un circuit absorbant.



PROBLEMES DE PLUS COURT CHEMIN

Différents problèmes peuvent être posés autour de la recherche de plus court chemin. Un premier problème A consiste à rechercher le plus court chemin entre deux points donnés. C'est-à-dire déterminer le chemin de plus petite longueur qui relie ces points. En fait, pour résoudre ce problème, on résout le problème B. Ce dernier consiste à déterminer, à partir d'un point donné le plus court chemin pour aller à tous les autres noeuds. Enfin, le problème C consiste à trouver, pour n'importe quelle paire de noeuds, le chemin le plus court entre eux. Le problème A étant résolu par le problème B, on s'intéressera

uniquement aux problèmes B et C.

Le problème B peut être résolu de nombreuses manières. Tout dépend des hypothèses émises sur la structure du réseau. Lorsque le réseau est sans circuit, on appliquera l'algorithme de Bellman. Lorsque le réseau n'a que des longueurs positives ou nulles (mais éventuellement avec des circuits), on utilisera la méthode de Dijkstra. Enfin, dans un réseau sans hypothèse particulière, on dispose d'un algorithme général. On s'intéressera ici à l'algorithme de Dijkstra.

Le problème C peut être résolu également de différentes manières. On s'intéressera ici à l'algorithme de Dantzig qui émet l'hypothèse qu'il n'y a pas de circuit absorbant dans le graphe. Il faut noter que le stockage de la solution du problème C peut prendre beaucoup de place. C'est pourquoi on se limitera ici à calculer les plus courtes distances et non pas à déterminer les plus courts chemins. Cependant, la démarche reste exactement la même.

ALGORITHME DE DIJKSTRA

Cet algorithme détermine **les plus courts chemins d'un point s à tous les autres points d'un réseau** R = (X;U;d). Il suppose que **les longueurs sur les arcs sont positives ou nulles.** L'idée de cet algorithme est de partager les noeuds en deux groupes: ceux dont on connaît la distance la plus courte au point s (ensemble S) et ceux dont on ne connaît pas cette distance (ensemble S'). On part avec tous les noeuds dans S'. Tous les noeuds ont une distance infinie (p(x) = +oo) avec le point s, excepté le point s lui-même qui a une distance nulle (p(s) = 0). A chaque itération, on choisit le noeud x qui a la plus petite distance au point s. Ce noeud est déplacé dans S. Ensuite, pour chaque successeur y de x, on regarde si la distance la plus courte connue jusqu'à lors entre s et y ne peut pas être améliorée en passant par x. Si c'est le cas, p(y) est modifiée. Ensuite, on recommence avec un autre noeud.

Algorithme

Titre: Dijkstra
Entrées: R = (X;U;d) un réseau, s un sommet.
Sorties: p() une fonction indiquant la plus courte distance qui sépare un point de s, pred() une fonction indiquant par quel arc on arrive à un point si on emprunte le plus court chemin.
Variables intermédiaires: S et S' deux sous-ensembles de sommets, x et y deux noeuds, sortie un booléen.

Début

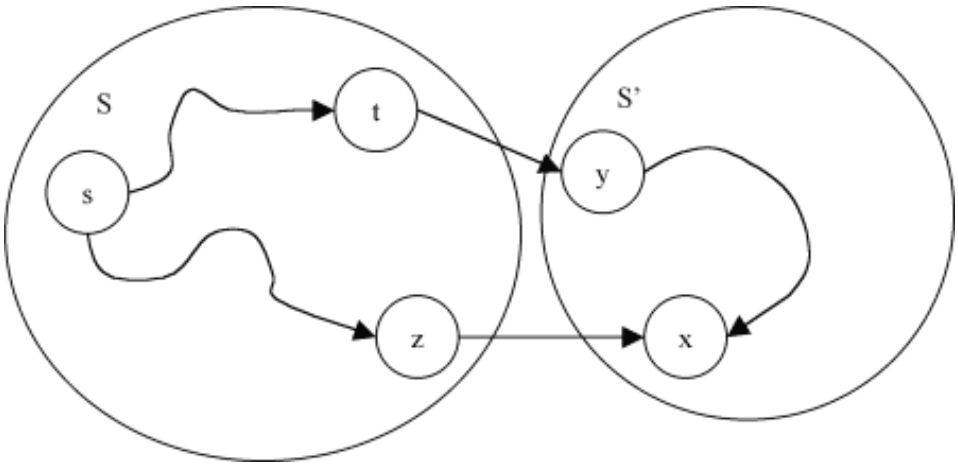
```
S ← ∅;
S' ← X;

pour tout x ∈ X faire
 p(x) ← +oo;
 pred(x) ← nil;
fin pour;

p(s) ← 0;
sortie ← faux;

tant que S' ≠ ∅ et non sortie faire
 choisir x ∈ S' tel que p(x) = min{p(y) | y ∈ S'};
 sortie ← (p(x) = +oo);
 S ← S ∪ {x};
 S' ← S' - {x};
```

```
  pour chaque arc u = (x;y) | y ∈ S' faire
   si p(y) > p(x) + d(u) alors
    p(y) ← p(x) + d(u);
    pred(y) ← u;
   fin si;
  fin pour;
 fin tant que;
```

Fin

Justification

Pour qu'un noeud x ait une distance p(x) différente de +oo , il faut qu'il ait un de ses prédécesseurs dans S. De plus, p(x) est la plus petite distance de s à x en empruntant uniquement les noeuds de S. Supposons que p(x) soit la plus petite distance de tout l'ensemble S'. Supposons également que le plus court chemin pour aller de s à x passe par t et y (cf. figure). Cela signifie que y est le prédécesseur de x. Il faudra donc que $p(y) \leq p(x)$. Cependant, si le chemin le plus court pour aller de s à x passe par t et y alors le chemin le plus court pour aller de s à y passe par t. Ce qui signifie que y connaît sa plus courte distance à s. Malheureusement, elle est supérieure à p(x) ce qui contredit l'hypothèse que p(x) est la plus courte distance de S'.



ALGORITHME DE DANTZIG

Cet algorithme détermine **la plus courte distance entre tous les couples de sommets d'un réseau** R = (X;U;d). On suppose que **le réseau ne contient pas de circuit absorbant**. On commence par numéroter les noeuds d'une manière quelconque. Ensuite, une matrice D de taille n x n est construite indiquant pour chaque noeud la plus courte distance qui le sépare d'un autre noeud. Chaque ligne et chaque colonne représentent donc un noeud. L'idée est de considérer un sous-réseau pour lequel on détermine les plus courtes distances entre ses noeuds. Ensuite, itérativement, on rajoute un noeud dans ce sous-réseau, on met les plus courtes distances à jour et on recommence jusqu'à ce que le sous-réseau soit le réseau lui-même.

A l'itération k, le sous-réseau a k noeuds et ce sont ceux numérotés de 1 à k. A l'itération k + 1, on ajoute le noeud k + 1 dans le sous-réseau. Les distances sont mises à jour de la manière suivante. Tout d'abord on détermine la plus courte distance entre le noeud k + 1 et n'importe quel autre noeud. Cela s'exprime par:

$$D(k+1,i) \leftarrow \min\{d(x_{k+1},x_j) + D(j,i)\} \text{ pour } i = 1...k.$$

Cela veut dire que l'on considère que pour aller de k + 1 à i, on passe par un des noeuds de 1 à k. On conserve la longueur du plus court chemin. De la même manière, on détermine la plus courte distance entre n'importe quel noeud et le noeud k + 1.

$$D(i,k+1) \leftarrow \min\{D(i,j) + d(x_j,x_{k+1})\} \text{ pour } i = 1...k.$$

Enfin, on met à jour la plus courte distance de chaque noeud de 1 à k vers chaque noeud de 1 à k car peut être que le plus court chemin entre deux de ces noeuds passent par k + 1.

$$D(i,j) \leftarrow \min\{D(i,j) , D(i,k+1) + D(k+1,j)\} \text{ pour } i, j = 1...k.$$

Algorithme

Titre: Dantzig
Entrées: R = (X;U;d) un réseau.
Sorties: D une matrice contenant les plus courtes distances.
Variables intermédiaires: i, j et k des entiers.

Début

```
 pour i ← 1 à n faire
  pour j ← 1 à n faire
   si i = j alors D(i,i) ← 0;
   sinon D(i,j) ← +oo;
  fin pour;
 fin pour;

 pour k ← 1 à n faire
  pour i ← 1 à k - 1 faire
   D(k,i) ← min{d(xk,xj) + D(j,i) | (xk;xj) ∈ U};
  fin pour;

  pour i ← 1 à k - 1 faire
   D(i,k) ← min{D(i,j) + d(xj,xk) | (xj;xk) ∈ U};
  fin pour;

  pour i ← 1 à k - 1 faire
   pour j ← 1 à k - 1 faire
    D(i,j) ← min{D(i,j) , D(i,k) + D(k,j)};
   fin pour;
  fin pour;
 fin pour;
```

Fin

Copyright (c) 1999-2000 - Bruno Bachelet - bachelet@ifrance.com - http://bruno.bachelet.net/

La permission est accordée de copier, distribuer et/ou modifier ce document sous les termes de la licence ***GNU Free Documentation License***, Version 1.1 ou toute version ultérieure publiée par la fondation *Free Software Foundation*. Voir cette licence pour plus de détails (http://www.gnu.org/).