PROBLEMES DE PLUS COURT CHEMIN

Amina ELJABRI

FSTM
Département Informatique

Définitions

Définition 1

Etant donné un graphe G = (X, U) et une application d qui à chaque arc associe sa longueur. Le triplet R = (X, U, d) s'appelle réseau. La longueur d'un chemin C dans R est

$$d(C) = \sum_{u \in C} d(u)$$

Si on considère le problème de plus court chemin de x à y, trois cas peuvent se présenter :

- Pas de chemin de x à y
- Il existe un chemin de x à y mais il n'existe pas de chemin de x à y de longueur minimum
- Il existe un chemin de x à y de longueur minimum auquel cas la longueur de ce chemin s'appelle plus courte distance de x à y.

Dans le contexte des problèmes de cheminement, six problèmes peuvent nous intéresser à savoir :

A. Plus court chemin élémentaire de s à p.

B. Plus court chemin élémentaire de s à tous les autres sommets.

C. Plus court chemin élémentaire entre tout couple de sommets.

A'. Plus court chemin de s à p.

B'. Plus court chemin de s à tous les autres sommets.

C'. Plus court chemin entre tout couple de sommets.

Définition 2

Un circuit est dit absorbant si sa longueur est négative

Theorem

Une condition nécessaire et suffisante pour que B' ait une solution est que :

1. *s soit une racine.*
2. *R ne contienne pas de circuit absorbant.*

théorème de caractérisation des plus courtes distances

Theorem

Une condition nécessaire et suffisante pour que les potentiels $\pi(x)$ soient les plus courtes distances de s à x est que :

- $\pi(s) = 0$
- $\pi(T(u)) - \pi(I(u)) \leq d(u)\ \forall u \in U$
- *Le graphe (X, U') tel que*
  $U' = \{\ u \in U\ /\ \pi(T(u)) - \pi(I(u)) = d(u)\ \}$
  admet s comme racine

Reconnaissance d'un graphe orienté sans circuit

Lemme

Si G = (X, U) est un graphe orienté sans circuit, il existe x $\in$ X tel que $d^-(x) = 0$.

Theorem

Un graphe orienté G = (X, U) est sans circuit si et seulement si :

- $\exists\, x \in X$ *tel que* $d^-(x) = 0$ *et*
- $\forall x \in X$ *tel que* $d^-(x) = 0, G \setminus \{x\}$ *est sans circuit.*

Algorithme de reconnaissance d'un graphe sans circuit

Données d'entrée : G = (X, U)
Début
Tant que G non vide faire
si tous les sommets ont un prédécesseur alors
  Ecrire(G contient un circuit) ;Arrêt;
sinon
  choisir x $\in X$ sans prédécesseur ;
  $G \leftarrow G \setminus \{x\}$; /* On élimine x et les arcs adjacents à x*/
Finsi
Fintantque
Fin.
Ci-dessous on détaille comment on choisit le sommet x pour obtenir un algorithme efficace avec une complexité : $O(\mid X \mid + \mid U \mid)$

Algorithme de reconnaissance d'un graphe sans circuit

$\forall x \in X$ faire $d^-(x) \leftarrow 0$;
$\forall x \in X$ faire $\forall y \in \Gamma^+(x)$ faire $d^-(y) + +$;
$Liste \leftarrow \emptyset$; $nbsommets \leftarrow 0$;
$\forall x \in X$ faire
si $(d^-(x) = 0)$alors $\{Liste \leftarrow Liste \cup \{x\}$; $nbsommets + +;\}$
Tant que $(Liste \neq \emptyset)faire$
$x \leftarrow tete(Liste)$;
$Liste \leftarrow Liste \setminus \{x\}$;
$\forall y \in \Gamma^+(x)$ faire
$d^-(y) - -$;
$si(d^-(y) = 0)alors\{$ $Liste \leftarrow Liste \cup \{y\}$; nbsommets++;$\}$
Fintanque
si (nbsommets = n) alors Ecrire (G sans circuit)
sinon Ecrire(G a un circuit)

Ordre (Tri) Topologique

Il est utile de déterminer un ordre sur les sommets d'un graphe orienté sans circuit. Cet ordre est appelé ordre topologique. Il s'agit d'un ordre de visite des sommets tel qu'un sommet soit toujours visité avant ses successeurs.

Définition (Tri topologique)

Soit G = (X,U) un graphe orienté sans circuit. Un tri topologique est une permutation $\sigma = (x_1, ..., x_n)$ des sommets de X telle que $(x_i \quad x_j) \in U \Rightarrow i < j$.

Function Tri-topologique (G = (X, U))

Le calcul d'un tri topologique peut s'effectuer à l'aide de l'algorithme de reconnaissance d'un graphe sans circuit.
Début
$F \leftarrow \emptyset$;
Tant que G non vide faire
si tous les sommets ont un prédécesseur alors
Ecrire(G contient un circuit) ;Stop;
sinon
choisir s $\in X$ sans prédécesseur ;
$G \leftarrow G \setminus \{s\}$;
Enfiler(F,s) ; Finsi
Fintantque
retour (F) ;
Fin.// L'ordre d'entrée dans la liste F est l'ordre topologique.

Différents tri topologiques peuvent être calculés pour un graphe donné, selon l'ordre dans lequel sont pris les successeurs d'un sommet donné et selon l'ordre dans lequel sont choisis les sommets sans prédécesseurs.
Le calcul d'un tri topologique peut s'effectuer aussi à l'aide d'un parcours en profondeur du graphe, au cours duquel on empile chaque sommet une fois tous ses successeurs visités. En dépilant, on obtient un tri topologique.
Dans le cas particulier d'un graphe orienté sans circuit de racine s, un ordre topologique peut être donné par la fonction rang obtenue en associant à chaque sommet i$\in$ X un nombre positif r(i) (appelé rang du sommet i) tel que:

- r(s) = 0;
- r(i) = le nombre d'arcs dans un chemin de cardinalité maximum entre s et i.

Détermination de la fonction rang d'un graphe orienté ou la détection d'un circuit

Algorithme (Entrée : G = (X,U), Sortie : r(i) $\forall i \in X$)
Début
/*Initialisation*/
$k \leftarrow 0$;
$R \leftarrow \emptyset$;
Pour tout $i \in X$ faire
  $d_i = d^-(i)$
  si $d_i = 0$ alors
    $R \leftarrow R \cup \{i\}$
    nbsommet ++;
  Finsi
Finpour

Détermination de la fonction rang d'un graphe orienté ou la détection d'un circuit

Tant que $(R \neq \emptyset)$ faire
Pour tout $i \in R$ faire
$r(i) \leftarrow k$; $R \leftarrow R \setminus \{i\}$;
Pour tout $j \in \Gamma^+(i)$ faire
$d_j - -$;
si $d_j = 0$ alors
$S \leftarrow S \cup \{j\}$;
nbsommet ++; Finsi
Finpour Finpour;
k ++; $R \leftarrow S$; $vider(S)$;
Fintantque
Si (nbsommet = n) alors Ecrire (G est sans circuit);
Sinon Ecrire (G a un circuit); Fin.

Tri topologique en se servant du Parcours en profondeur

DFSTopo($a \in X$, etat[]) {
$etat[a] \leftarrow 1$;
$\forall\ y \in \Gamma^+(a)$
    si (etat[y] $= 0$)
        DFSTopo(y,etat);
empiler(F, a);
}
Tri-topologique(G $=$ (X,U)){
$F \leftarrow \emptyset$; $\forall\ x \in X$ $etat[x] \leftarrow 0$;
$\forall x \in X$ faire
    si etat[x] $= 0$
      DFSTopo(x,etat);
retourner (F);
}

Algorithme de Bellman

Données d'entrée : R = (X, U, d) le réseau et s racine de R ;
$n =\mid X \mid et\ m =\mid U \mid$
Donnée de sortie : les plus courtes distances et l'arborescence des plus courts chemins
Hypothèse : Le réseau ne comporte pas de circuit
Idée : on ne calcule la plus courte distance à un sommet x que si on a calculé celle de tous ses prédécesseurs.

Algorithme de Bellman

Début
$S = \{s\}$;
$\pi(s) = 0$;
Tant que $(S \neq X)$ faire début
sélectionner $x \notin S$ *telque* $\Gamma^-(x) \subset S$;
$u*$: $d(u*) + \pi(I(u*)) = \min_{\{u \in U / T(u) = x\}} \{d(u) + \pi(I(u))\}$;
$\pi(x) = d(u*) + \pi(I(u*))$;
$S = S \cup \{x\}$;
$A(x) = u*$;
Fintantque
FIN.

Complexité : L'algorithme de Bellman se termine de façon correcte pourvu qu'il soit toujours possible de trouver le x.
L'algorithme a un temps d'exécution assez rapide de l'ordre de n^2.

Remarques

- Pratiquement, il existe une indétermination dans l'algorithme, c'est la façon de trouver le sommet x à une itération donnée de l'algorithme.
- Pour remédier à cela, les sommets peuvent être classés par niveaux de descendance par rapport à la racine puisque le graphe ne comporte pas de circuit. Le sommet x sera choisi selon l'ordre topologique donné par la fonction rang.
- Par hypothèse, le sommet s est le seul sommet de G tel que $d^-(s) = 0$.

Algorithme de Dijkstra

Hypothèse : les longueurs sont positives ou nulles
Idée : On maintient S l'ensemble des sommets traités et $\forall x \in X$, $\pi(x)$ longueur du plus court chemin de s à x qui ne passe que par des sommets de S.
A chaque itération, on choisit le sommet le plus proche de s, on l'ajoute à S et on met à jour les potentiels de ses voisins.

$S = \{s\}$;
$\pi(s) = 0; \quad \forall x \neq s : \ \pi(x) = \infty;$
k = 1 ;
$x_k = s;$

Algorithme de Dijkstra

DEBUT
Tant que $(k < n)$ faire
{ mise à jour des $\pi(x)$ pour les successeurs de x_k}
Pour tout u tq I(u) $= x_k$ et $T(u) \notin S$ faire
si $(\pi(T(u)) > \pi(x_k) + d(u))$ alors
$\pi(T(u)) = \pi(x_k) + d(u)$;
A(T(u)) = u ;
Finsi
Finpour
Choisir $x \notin S$ tq $\pi(x) = \min_{\{y \notin S\}} \{\pi(y)\}$
k++; $x_k = x$; $S = S \cup \{xk\}$;
Fintanque
FIN.

Complexité : L'implémentation utilisant les listes d'adjacence et une file de priorité pour l'ensemble S permet une complexité en $O(n^2)$: n itérations de la boucle extérieure avec une recherche du minimum en O(n).
On peut réduire la complexité en utilisant des structures de données plus performantes tels que les tas.

Cas des longueurs négatives

Il est possible d'adapter l'algorithme de Dijkstra pour le cas des longueurs négatives.
$\pi(y) = \pi(x) + d((xy))$ donne la valeur correcte de $\pi(y)$ si x est le sommet précédent y dans le plus court chemin de s à y et si la valeur de $\pi(x)$ est correcte.
Une idée pour assurer la correction de l'algorithme dans le cas des longueurs négatives est de tester sur tous les arcs ne faisant pas partie de l'arborescence le théorème de caractérisation des plus courtes distances.
Deux cas peuvent se présenter, les potentiels peuvent être améliorés pour donner les plus courtes distances ou alors un circuit absorbant est mis en évidence.

Algorithme de Ford-Fulkerson

Dijkstra(R, s, π, A);
$E = \{u / \exists x \in X : A(x) = u\}$;
Tant que $(\exists u \in U$ tq $\pi(T(u)) - \pi(I(u)) > d(u))$ faire
si $(X, E \cup \{u\})$ contient un circuit alors
arrêt ; /* c'est un circuit absorbant pas de solution */
sinon
$\delta = \pi(T(u)) - \pi(I(u)) - \mathrm{d(u)}$;
$\mathrm{x} = \mathrm{T(u)}$;
$E = E \cup \{u\} \setminus \{A(x)\}$;
$\mathrm{A(x)} = \mathrm{u}$;
$\forall y$ de la sous arborescence de (X, E) de racine x faire
$\pi(y) = \pi(y) - \delta$;
Finsi
Fintantque

Algorithme de Bellman-Ford

$\pi(s) = 0$; Pour tout $x \in X, x \neq s \quad \pi(x) = \infty$; Compteur $= 0$;
Répéter
  Fini $=$ vrai ;
  Compteur $=$ compteur $+ 1$;
  Pour tout $u = (x\ y) \in U$ faire
  si $(\pi(y) > \pi(x) + d(u))$ alors
    $\pi(y) = \pi(x) + d(u)$;
    Fini $=$ Faux ;
  finsi
Jusqu'à ($compteur \geq n - 1$ *ou* $Fini == vrai$)
Pour tout $u = (x\ y) \in U$ faire
  si $\pi(x) + d(u) < \pi(y)$ alors
    retourner faux /* présence d'un circuit*/
retourner vrai;

Algorithme de Floyd-Warshall

L'algorithme de Floyd-Warshall calcule deux matrices carrées d'ordre n:
D matrice des distances telle que D(i, j) = distance optimale de i à j.
P matrice des prédécesseurs telle que P(i, j) = prédécesseur de j dans le chemin optimal de i à j.

Algorithme Floyd-Warshall

Initialisation des matrices D et P :

$$D(i,j) = \begin{cases} c((i\ j))\ si\ (i\ j) \in U \\ 0\ si\ \ i = j \\ \infty\ si\ (i\ j) \notin U \end{cases}$$

$$P(i,j) = \begin{cases} i\ si\ (i\ j) \in U \\ 0\ ou -1\ sinon \end{cases}$$

Algorithme Floyd-Warshall

Début
Pour k allant de 1 à n faire
  Pour i allant de 1 à n faire
    Pour j allant de 1 à n faire
      si $(D(i,k) + D(k,j) < D(i,j))$ alors
        D(i, j) = D(i, k) + D(k, j) ;
        P(i, j) = P(k, j) ;
      Finsi
FIN.

Complexité : L'algorithme de Floyd-Warshall a un temps d'exécution très long de l'ordre de n^3 et consomme beaucoup d'espace mémoire. Son avantage c'est qu'il fonctionne sur tous les graphes.

Algorithme Floyd-Warshall

Début
Pour k allant de 1 à n faire
  Pour i allant de 1 à n faire
    Pour j allant de 1 à n faire
      si ($i \neq j$ et $D(i,k) + D(k,j) < D(i,j)$) alors
        D(i, j) = D(i, k) + D(k, j) ;
        P(i, j) = P(k, j) ;
      Finsi
      si (i = j et $D(i,k) + D(k,j) < 0$) alors
        Ecrire "Circuit absorbant";arrêt;
      Finsi
FIN.

Cet algorithme est valable quand on n'est pas sûr que le graphe ne comporte pas de circuit absorbant.

Notion de tas

La structure de données tas (appelé parfois "monceau")est généralement utilisée pour implémenter un arbre binaire complet ordonné selon la propriété suivante :
Pour tous les noeuds de l'arbre, étiquette(père) $>$ étiquette(fils).
Cette propriété implique que la plus grande étiquette est située à la racine du tas. ou selon la propriété :
Pour tous les noeuds de l'arbre, étiquette(père) $<$ étiquette(fils).
Cette propriété implique que la plus petite étiquette est située à la racine du tas.
Ils sont ainsi très utilisés pour implémenter les files à priorités car ils permettent des insertions en temps logarithmique et un accès direct au plus grand (ou petit)élément.

implémenter un tas

On peut représenter un tas d'une manière efficace par un tableau unidimensionnel indicé à partir de 0.
Le père d'un noeud en position i a pour position l'entier inférieur ou égal à $\frac{i-1}{2}$;
Les enfants d'un noeud en position i sont situés à $2i+1$ pour le fils gauche et $2i+2$ pour le fils droit.
Une caractéristique fondamentale de cette structure de données est que la propriété du tas peut être restaurée efficacement après la modification d'un noeud.

Modification de la valeur d'un noeud

Dans le cas où le tas vérifie la deuxième propriété :
Si la valeur du noeud est diminuée et devient inférieure à celle de son père, il suffit de l'échanger avec celle-ci, puis de continuer ce processus vers le haut jusqu'au rétablissement de la propriété du tas. C'est le processus de percolation : la valeur modifiée a été percolée jusqu'à sa nouvelle position.
Inversement, si la valeur du noeud est augmentée et devient supérieure à celle d'au moins un de ses fils, il suffit de l'échanger avec la plus petite des valeurs de ses fils, puis de continuer ce processus vers le bas jusqu'au rétablissement de la propriété du tas. Nous dirons que la valeur modifiée a été tamisée jusqu'à sa nouvelle position.

Percoler - Tamiser

fonction tamiser(T[0..n-1], i)

debut
$k \leftarrow i$;
repeter
$j \leftarrow k$;
(* recherche du plus petit fils du noeud j *)
si $(2j+1 \leq n-1$ et $T[2j+1] < T[k])$ alors
    $k \leftarrow 2j+1$;
si $(2j+1 < n-1$ et $T[2j+2] < T[k])$ alors
    $k \leftarrow 2j+2$;
echanger (T[j],T[k]);
jusqu'à (j = k);
fin.

Percoler - Tamiser

fonction percoler(T[0..n-1], i)

debut
$k \leftarrow i$;
repeter
$j \leftarrow k$;
si ($j > 0$ et $T[(j-1)div2] > T[k]$) alors
    $k \leftarrow (j-1)div2$;
finsi
echanger (T[j],T[k]);
jusqu'à (j = k);
fin.

Percoler - Tamiser

fonction modifiertas(T[0..n-1], i, v)

(* on veut changer la valeur v du noeud i en préservant la propriété du tas *)
debut
$x \leftarrow T[i]$;
$T[i] \leftarrow v$;
si $(v < x)$ alors
  percoler(T, i);
sinon
  tamiser(T, i);
fin.

utilité pour une file de priorité

Cette propriété du tas en fait une structure de données idéale pour trouver le minimum (ou maximum), éliminer la racine, ajouter un noeud et modifier un noeud. Ces opérations sont utiles pour l'implémentation efficace d'une file de priorité.

fonction ajouter(T[0..n], v)

$T[n] \leftarrow v$;
percoler(T,n);

fonction removemin(T[0..n-1])

$u \leftarrow T[0]$;
$T[0] \leftarrow T[n-1]$;
tamiser(T,0);
$n \leftarrow n-1$; return u;

cas particulier de l'algorithme de Dijkstra

Pour améliorer la performance de l'algorithme de Dijkstra, Le complémentaire de l'ensemble S mentionné dans l'algorithme (qu'on notera $\overline{S}$) est implémenté à l'aide de la structure de tas.
Au départ, $\overline{S}$ contient tous les sommets à part le sommet s.
$\overline{S}$[i] est composé de deux champs le sommet et son potentiel initialisé à ∞.
Nous avons besoin aussi d'un vecteur d'indices H dans le TAS.
Utilisant ce vecteur, étant donné un sommet, sa location dans le TAS $\overline{S}$ est déterminé en O(1).
H[v] = 0 si le sommet v n'est pas dans le TAS (au départ seul le sommet s n'est pas dans le tas H[s] = 0).
Si maintenant le sommet v est stocké dans la location $\overline{S}$[i], alors H[v] prendra la valeur i. Les opérations peuvent être implantées comme suit :

cas particulier de l'algorithme de Dijkstra : suite

A la modification du poids d'un sommet v, H[v] nous donne sa location dans le tas $\overline{S}$ puis le processus percoler est lancé pour maintenir la propriété du tas pour $\overline{S}$.
La fonction percoler sera modifiée de la sorte : la comparaison s'effectue sur le champs potentiel qui constitue la valeur de priorité et chaque fois qu'un élément de $\overline{S}$ est mis à jour (lors de la fonction echange) par exemple $\overline{S}$[i].sommet =j, on doit aussi mettre à jour le vecteur H par (H[j] = i).
On fait de même lors de la suppression de la racine.
De plus, si on note x le sommet supprimé du tas $\overline{S}$, son indice doit aussi être supprimé (H[x] = 0). Celà indique qu'il intègre l'ensemble S.

cas particulier de l'algorithme de Dijkstra

Algorithme Dijkstra(G, s):

Entrée: Un graphe pondéré G = (X, U) $n =\mid X \mid$
et un sommet s de G.
Sortie: Une étiquette $\pi[x]$, $\forall x \in X$, où $\pi[x]$ est la longueur d'un plus court chemin de s à x dans G.

Début
$H[s] \leftarrow 0$; initialisez $\pi(s)$ par 0 ; et $\forall x \neq s :\ \pi(x)$ par ∞;
$\overline{S}$ initialisé par les sommets de$X \setminus \{s\}$ avec leur étiquettes π comme clés et H initialisé en conséquence;
//si par ex $\overline{S}[i] = \{j, \pi(j)\}$ alors H[j] = i;
$k \leftarrow 1$;
$x \leftarrow s$;

cas particulier de l'algorithme de Dijkstra

Tant que $(k < n)$ faire
  Pour tout u tq(I(u) $= x$ et $H[T(u)] \neq 0$)faire
    si $(\pi(T(u)) > \pi(x) + d(u))$ alors
      $\pi(T(u)) = \pi(x) + d(u)$;
      A(T(u)) = u ;
      réorganiser $\overline{S}$; Finsi
  Finpour
  k++;
  $x \leftarrow$ removeMin($\overline{S}$);//x est tq $\pi(x)$ est le minimum ds $\overline{S}$
  $H[x] \leftarrow 0$;//x entre dans S
finTantque;
return π;
Fin.