**Chapitre 8 Recherches de plus courts chemins dans un graphe valué**

cout de chemin : cout(c) = la somme de poids de chaque arête

Définition :

- poids peuvent être négatifs

- circuit absorbant : un circuit de coût total négatif

- (donc tout passage par ce circuit fait diminuer le coût du chemin, il n’exsite pas de chemin de coût minimum

Théorème :

- G admet un plus court chemin s’il n’existe pas de circouuit absorbant

- s’il n’y a pas de circuit de coût nul, les plus courts chemins sont élémentaires

Hypothèse de l’algo : y a pas de circuit absorbant

Algo de Dijkstra :

Version 1 :

{ initialisations }

M := {s};

Pour i de 1 à n faire

Si (s,i) est une arête alors

Début

D[i] := cout(s,i);

P[i] := s;

ajouter(i,M);

Fin

sinon

D[i] := infini;

Tant que M<>{ } faire

Début

x := enleveTete(M);

Pour tout successeur y de x faire

Début

x := enleveTete(M);

Pour tout succeseur y de x faire

Début

d := D[x] + cout(x,y);

si d<D[y] alors

début

D[y] := d;

P[y] := x;

ajouter(y,M);

Fin

Fin

Fin

Fin

Version 2 :

{ initialisation }

M := { }

Pour i de 1 à n faire

Si (s,i) est une arête alors

D[i] := cout(s,i)

P[i] := s;

ajouter(i,M);

sinon

D[i] := infini;

Tant que M<>{ } faire

x := choisir\_min(M,d);

enlever(x,M);

Pour tout successeur y de x faire

si y est dans M alors

d := D[x] + count(x,y);

si d<D[y] alors

D[y] := d;

P[y] :=x;

ajouter(y,M);

Fin

Fin

Fin

Fin

Complexité de Dijkstra version 2

initialisation O(n)

boucle tant que sera parcouru au plus n fois, i-ème étape, choisirMin est O(n-i), parcours de sucesseur de x est O(degré entrant de sommet x)

boucle de traitement est sigma (n-i) + sigma(degré entrant de sommet x)

on considère la boucle tant que en disant d’une part qu’elle parcourra tous les sommets dans un certain ordre et d’autre part qu’elle traitera des ensembles M de cardinal n-i pour toute valeur de i

O(n^2) + O(m) = O(n^2)

Algorithme de Belleman :

{ Dist est initialisé à une valeur conventionnelle infinie, S est initialisé à X\S }

Dist[s] := 0;

P[s] := s;

fini := faux;

tant que non fini faire

début

fini := vrai;

pour tout élément y de S faire

{On traite tous les y de S dont tous les prédécesseurs sont dans M}

Oky := vrai;

d := infini;

pour tout prédécesseur x de y faire

si Dist[x] <> infini

alors

si Dist[x] + p(x,y) < d alors

Début

d := Dist[x] + p(x,y);

P[y] := x;

Fin

sinon

Oky := false

si Oky alors {On traite y}

début

fini := false;

enlever(y,S);

si S = [] alors

fini:=vrai;

Dist[y] :=d;

Fin

complexité : O(n^3)

Version 2 :

{Dist est initialisé à une valeur conventionnelle infinie}

Dist[s] := 0, P[s] := s;

{construction du tableau PS et de la liste L}

L := [ ];

pour y de 1 à n faire

Début

PS[y] := 0;

si y = s alors

PS[y]=-1;

pour chaque prédécesseur z de y faire

si z<>s alors

PS[y] := PS[y] + 1;

si PS[y] = 0 alors

ajouter(y,L);

Fin

tant que L<> [ ] faire

Début

y := enleveTete(L);

pour tout prédécesseur x de y faire

si Dist[x] + p(x,y) < Dist[y] alors

début

Dist[y] := Dist[x] + p(x,y);

P[y] := x;

fin

{Mise à jour de PS et L}

{tout successeur de y un prédécesseur de moins}

pour chaque sucesseur z de y faire

Début

PS[z] := PS[z] - 1;

si PS[z] = 0 alors ajouter(z,L)

Fin

Fin

Complexité : O(m)