esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

THEORIE DES GRAPHES

11) ALGORITHME DE FLOYD-WARSHALL (PAS DE CIRCUIT OU PAS DE CYCLE)

Contrairement aux algorithmes de Ford-Bellman et de Dijkstra qui recherchent les plus courts chemins d'un sommet s à tous les autres, l'algorithme de Floyd-Warshall recherche les plus courts chemins entre tous les couples de sommets.

Tout comme l'algorithme de Ford-Bellman, l'algorithme de Floyd ne peut s'appliquer que si le graphe ne comporte pas de circuit (ou cycle si le graphe n'est pas orienté).

Il convient de noter que la complexité en temps de l'algorithme de Floyd-Warshall, $\Theta(n^3)$ (n est le nombre de sommets), est supérieure à celle de l'algorithme de Ford-Bellman, $\Theta(nm)$ (n est le nombre de sommets et m est le nombre d'arcs ou d'arêtes), et que la complexité en temps de l'algorithme de Ford-Bellman est, en général, supérieure à celle de l'algorithme de Dijkstra, $\Theta(n^2)$.

Ainsi, quand les longueurs sont positives ou nulles, et face à un algorithme dense, l'algorithme le plus usité et que l'on doit utiliser est celui de Dijkstra.

L'algorithme de Floyd-Warshall consiste à chercher les plus courts chemins entre tous les couples de sommets et ce, en construisant une matrice $L=(L_{ij})$ où, L_{ij} dénote la longueur du plus court chemin de i à j.

Soient :

G = [X, U] un graphe valué donné et qui est sans circuit (ou cycle si le graphe n'est pas orienté);

X : L'ensemble des sommets. X = {1, 2, …, n} ;

U : L'ensemble des arcs ou des arêtes ;

L: La matrice des longueurs des plus courts chemins que l'on souhaite construire. $L=(L_{ij})$ où, L_{ij} dénote la longueur du plus court chemin de i à j ;

P: La matrice des prédécesseurs que l'on souhaite construire. $P=(P_{ij})$ où, P_{ij} dénote le prédécesseur de j sur le plus court chemin de i à j ;

Le principe de l'algorithme de Floyd-Warshall est le suivant.

- ✓ On initialise la matrice L comme suit.

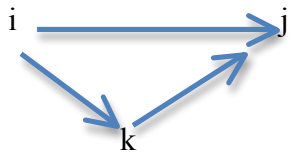
$$L_{ij} = \begin{cases} 0 \text{ si } i=j \\ \text{La longueur de l'arc ou de l'arête si i et j sont connectés} \\ \infty \text{ dans les autres cas.} \end{cases}$$

- ✓ On initialise la matrice P comme suit

$$P_{ii} = i$$

- ✓ Pour k allant du sommet 1 au sommet n,
  - on construit la matrice $L^k=(L_{ij})^k$ en considérant le minimum entre la dernière valeur déterminée de L_{ij}, L_{ij}^{k-1}, et la longueur du chemin de i à j passant par le sommet k.

$$\mathbf{L_{ij}^{k} = Min(L_{ij}^{k-1}, L_{ik}^{k-1} + L_{kj}^{k-1})}$$



Autrement dit, on cherche à savoir si en passant par le sommet k, on peut trouver un plus court chemin et ce pour tout couple de sommet (i, j).

  - Si L_{ij}^{k} devient $L_{ik}^{k-1} + L_{kj}^{k-1}$, alors k devient le prédécesseur de j sur le plus court chemin de i à j.

$$\mathbf{P_{ij} := P_{kj}}$$

Remarque

A l'issue de la $k^{ème}$ étape de l'algorithme, le coefficient L_{ij} est égal à la longueur du plus court chemin de i à j passant uniquement par les sommets 1,…,k.

L'algorithme de Floyd-Warshall est le suivant.

Procédure Floyd-Warshall (donnée G = [X, U] : graphe; donnée coût : longueur ; résultat L **: Matrice des longueurs des plus courts chemins ; résultat** P **: matrice des prédécesseurs) ;**

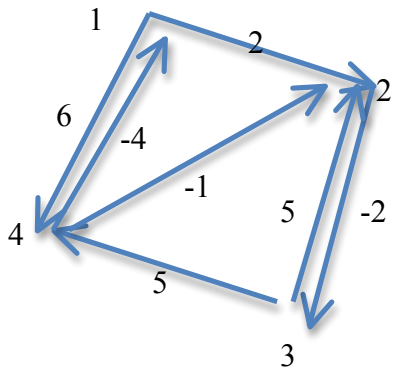
```
{On suppose que le graphe G ne comporte pas de circuit}
Var i, j, k : entier
Début
        {initialisation}
        pour i :=1 à n faire
                pour j :=1 à n faire
                        Si i=j alors
                                L[i, j] := 0
                        Sinon si G[i,j]=1 {j successeur de i}
                                        L[i,j] := coût[i,j]
                                Sinon
                                        L[i,j] := ∞
                                Finsi
                        Fin si
                        P[i, j] :=i
                Fin pour
        Fin pour

        pour k:=1 à n faire
                pour i :=1 à n faire
                        pour j :=1 à n faire
                                Si ((i ≠ j) et (L[i,k]+ L[k,j] < L[i,j])) alors
                                        L[i,j] := L[i,k]+ L[k,j]
                                        P[i, j] := P[k j]
                                Fin si
                                Si ((i = j) et (L[i,k]+ L[k,j] < 0)) alors
                                        Ecrire(« Circuit absorbant. Terminé »)
                                Fin si
                        Fin pour
                Fin pour
        Fin pour

Fin ;
```

Exemple

Considérons le graphe G ci-dessous.



Comme le graphe ne comporte pas de circuit absorbant, on peut alors appliquer l'algorithme de Floyd-Warshall.

La simulation de l'algorithme de Floyd-Warshall est la suivante.

Initialisation :

$$L = \begin{pmatrix} 0 & 2 & \infty & 6 \\ \infty & 0 & -2 & \infty \\ \infty & 5 & 0 & 5 \\ -4 & -1 & \infty & 0 \end{pmatrix}$$

$$P = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{pmatrix}$$

1ère étape, k = 1 :

$$L = \begin{pmatrix} 0 & 2 & \infty & 6 \\ \infty & 0 & -2 & \infty \\ \infty & 5 & 0 & 5 \\ -4 & -2 & \infty & 0 \end{pmatrix}$$

$$P = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 1 & 4 & 4 \end{pmatrix}$$

L[4, 2] = -1 > L[4, 1] + L[1, 2] = -4 + 2 = -2

Donc, L[4, 2] devient -2 et P[4, 2] devient P[1, 2] = 1.

P[4, 2] = 1 signifie que 1 est le prédécesseur de 2 sur le chemin allant de 4 à 2.

2ème étape, k = 2 :

$$L = \begin{pmatrix} 0 & 2 & 0 & 6 \\ \infty & 0 & -2 & \infty \\ \infty & 5 & 0 & 5 \\ -4 & -2 & -4 & 0 \end{pmatrix}$$

$$P = \begin{pmatrix} 1 & 1 & 2 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 1 & 2 & 4 \end{pmatrix}$$

L[1, 3] devient 0 et P[1, 3] devient P[2, 3] = 2.

L[4, 3] devient -4 et P[4, 3] devient P[2, 3] = 2.

3ème étape, k = 3 :

$$L = \begin{pmatrix} 0 & 2 & 0 & 5 \\ \infty & 0 & -2 & 3 \\ \infty & 5 & 0 & 5 \\ -4 & -2 & -4 & 0 \end{pmatrix}$$

$$P = \begin{pmatrix} 1 & 1 & 2 & 3 \\ 2 & 2 & 2 & 3 \\ 3 & 3 & 3 & 3 \\ 4 & 1 & 2 & 4 \end{pmatrix}$$

L[1, 4] devient 5 et P[1, 4] devient P[3, 4] = 3.

L[2, 4] devient 3 et P[2, 4] devient P[3, 4] = 3.

4ème étape, k = 4 :

$$L = \begin{pmatrix} 0 & 2 & 0 & 5 \\ -1 & 0 & -2 & 3 \\ 1 & 3 & 0 & 5 \\ -4 & -2 & -4 & 0 \end{pmatrix}$$

$$P = \begin{pmatrix} 1 & 1 & 2 & 3 \\ 4 & 2 & 2 & 3 \\ 4 & 1 & 3 & 3 \\ 4 & 1 & 2 & 4 \end{pmatrix}$$

L[2, 1] devient -1 et P[2, 1] devient P[4, 1] = 4.

L[3, 1] devient 1 et P[3, 1] devient P[4, 1] = 4.

L[3, 2] devient 3 et P[3, 2] devient P[4, 2] = 1.

Remarque

Les arborescences obtenues à partir de chaque sommet du graphe sont celles qui suivent.