CHAPITRE 3.

LE PROBLEME DU PLUS COURT CHEMIN.

Les problèmes de cheminement dans les graphes (en particulier la recherche d'un plus court chemin) comptent parmi les problèmes les plus anciens de la théorie des graphes et les plus importants par leurs applications.

3.1. DEFINITION.

Soit G = (X, U) un graphe valué; on associe à chaque arc u=(i, j) une longueur l(u) ou l_{ij}.

Le Problème du plus court chemin entre i et j est de trouver un chemin μ(i, j) de i à j tel que :

$$l(\mu) = \sum_{u} l(u)$$

soit minimal.

Interprétation de l(μ) : coût de transport, dépense de construction, temps nécessaire de parcours, …

Remarque. La recherche du plus court chemin est analogue à la recherche du plus long chemin.

Les algorithmes seront différents suivant les propriétés des graphes :

- l(u) ≥ 0, ∀ u ∈ U.
- Les longueurs l(u) égales ⇔ l(u) = 1, ∀ u ∈ U. (problème du plus court chemin en nombre d'arcs)
- G sans circuit.
- G et l(u) quelconques.

Et suivant le problème considéré :

- Recherche du plus court chemin d'un sommet à tous les autres,
- Recherche du plus court chemin entre tous les couples de sommets.

3.2. PRINCIPE D' OPTIMALITE.

Le principe d' optimalité énonce le fait que les sous-chemins des plus courts chemins sont des plus courts chemins (la programmation dynamique repose sur ce principe fondamental).

LEMME.

Soient un graphe G(X,U) et une fonction de pondération l : X x X $\rightarrow$ R, Soit C = « $X_1, X_2,\ldots,X_k$ » un plus court chemin de X_1 à X_k et pour tout (i, j) tel que $1 \leq i \leq j \leq k$, soit C_{ij} = « $X_i, X_{i+1},\ldots,X_j$ » un sous chemin de C allant de X_i à X_j. Alors C_{ij} est un plus court chemin de X_i à X_j.

Principe des algorithmes de recherche de chemins minimaux :

- Une distance d(i) est associée à x_i.
- En fin d'algorithme, cette distance représente la longueur d'un plus court chemin de l'origine au sommet considéré.

3.3. VARIANTES DU PROBLEME : D' UN SOMMET A TOUS LES AUTRES.

Ce problème est aussi appelé **le problème de recherche du plus court chemin à origine unique**. Beaucoup d'autres problèmes peuvent être résolus par l'algorithme avec **origine unique** :

- Plus court chemin à destination unique (inversion du sens de chaque arc du graphe).
- Plus court chemin pour un couple de sommets donné.
- Plus court chemin pour tout couple de sommets (algorithmes à origine unique à partir de chaque sommet).

3.3.1. ALGORITHME DE DIJKSTRA-MOORE (1959).

Supposons que les longueurs des arcs sont non négatives ($l(u) \geq 0$) and l'ensemble de n sommets est numéroté de 1 à n. Le problème posé est la recherche du plus court chemin entre 1 et tous les noeuds accessibles depuis 1.

Notations :

- M = L' ensemble de noeuds non marqués
- Pr(p) = Sommet précédant p sur le plus court chemin de l'origine à p.
- d = Plus courte distance de l'origine aux noeds restant. En convention ∞ dans le cas n'a pas de chemin de l'origin (1) à lui-même.
- Mark = L'ensemble des noeuds marqués.

PRINCIPE DE L'ALGORITHME.

1. Au départ du noeud 1. M = {2,…n}
2. À chaque itération, Choisir un noeud à marquer :c' est le noeud qui a la plus courte distance.
   - k = $\text{Argmin}_{x \in M}$ d[x].
   - Mises à jour d[i], Pr[i] avec $i \in M \setminus \{k\}$ à l'aide de la formule:
     - d[i] = d[k] + l[k,i] si d[i] > d[k] +l[k,i].
     - Pr[i] = k.
   - Remplacer M := M\{k}.

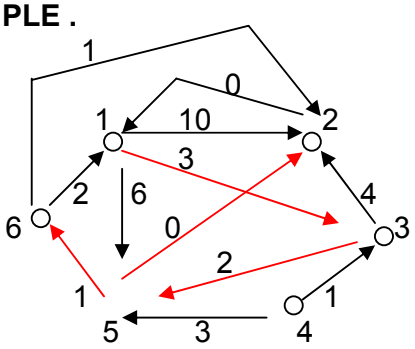
   Si M = ∅. L' algorithme se termine, sinon retourner à 2.

PROCEDURE DIJKSTRA – MOORE ;

```
//Suppose que l' on a la matrice de longuers l est Stocké sous la forme de
 matrice d'adjacence
//Initialisations de M, d, Pr, Mark
for (i= 1 ; i≤ n ;i++)
     {d[i] = l(1,i) ; pr[i] :=1 ; Mark[i] :=0 ;}
Mark[1] :=1 ; n0 :=n-1 ;
WHILE (n0 > 0)
{
    k:= Argmin {d[i] : i∈ M} ;
    //Remise à jour d, Pr, M et Mark
     Mark[k] :=1 ;
     ∀ i ∈ M { d[i] := d[k] +l[k,i] si d[i] > d[k] +l[k,i].
                  Pr[i] = k.}
     //Supprimer le noeud k
     M := M\{k} ;
}END WHILE ;
```

Complexité : O(n²) ou O(mlogn) avec une structure de tas, intéressante si le graphe est peu dense (i.e., m <<< n²)
L'algorithme de Dijktra-Moore utilise une stratégie **gloutonne** lorsqu'il choisit le sommet le moins coûteux à chaque étape. On démontre que dans le cas de cet algorithme, cette stratégie conduit à un résultat global optimal.

EXEMPLE .



Initialisation : M, d, Pr :
M = { 2, 3, 4, 5, 6}
d = [0, 10, 3, ∝, 6, ∝]
Pr = [1, 1, 1, 1, 1, 1]

FIG. 3.1. Graphe valué orienté.

- **1er étape.** Choisir s_3. Remise à jour M, d, Pr :
  M = { 2, , 4, 5, 6}
  d = [0, 7, 3, ∝, 5, ∝]
  Pr = [1, 3, 1, 1, 3, 1]
- **2è étape.** s_3 est le sommet actuel. Choisir s_5. Remise à jour M, d, Pr :
  M = { 2, , 4, , 6}
  d = [0, 5, 3, ∝, 5, 6]
  Pr = [1, 5, 1, 1, 3, 5]
- **3è étape.** s_5 est le sommet actuel. Choisir s_2. Remise à jour M, d, Pr :
  M = { , , 4, , 6}
  d = [0, 5, 3, ∝, 5, 6]
  Pr = [1, 5, 1, 1, 3, 5]
- **4è étape.** s_2 est le sommet actuel. Choisir s_6. Remise à jour M, d, Pr :
  M = { , , 4, , }
  d = [0, 5, 3, ∝, 5, 6]
  Pr = [1, 5, 1, 1, 3, 5]

Algorithme se termine car 4 = Argmin {d[i] : i∈ M}; et d[4] = ∝.

À l'issue de la procédure, on obtient le résultat suivant :

- Le plus court chemin de s_1 vers s_2 est: $s_1 \to s_3 \to s_5 \to s_2$ et son coût est égal à 5
- Le plus court chemin de s_1 vers s_3 est: $s_1 \to s_3$ et son coût est égal à 3
- Le plus court chemin de s_1 vers s_5 est: $s_1 \to s_3 \to s_5$ et son coût est égal à 5
- Le plus court chemin de s_1 vers s_6 est: $s_1 \to s_5 \to s_6$ et son coût est égal à 6

- On n'a pas trouvé de plus court chemin de s_1 vers s_4 (d[4] = ∞ à la fin), car s_4 est inaccessible depuis s_1.

REMARQUE.

L'hypothèse « Les coûts sont tous positifs ou nuls » est fondamental. L'utilisation de L'algorithme de Dijktra-Moore pour le graphe de la figure FIG.3.2., où les poids ne sont pas tous positifs ou nuls, conduit à un resultat incorrect si on choisit comme source le sommet s_1. En effet, d'abord on choisit le sommet s_2, ($s_1 \rightarrow s_2$) tandis que le chemin de s_1 vers s_2 passant s_3 est plus court.

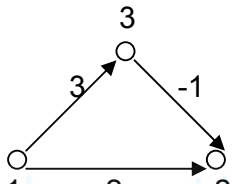


FIG. 3.2. Graphe valué orienté par des coûts quelconques.

3.3.2. ALGORITHME DE BELLMAN-FORD (1958-1962)

La présence de longueurs de signes différents (l(u) quelconques), permet par exemple de modéliser des coûts et des profits. L'algorithme de DIJKSTRA-MOORE ne permet pas de considérer les arcs négatifs, car une fois qu'un sommet est marqué on ne peut changer ce marquage lors des itérations suivantes. L'algorithme de DIJKSTRA-MOORE est ainsi dit à **fixation d'étiquettes**. On considère donc ici un algorithme qui permet un marquage qui n'est pas définitif tant que le programme n'est pas déterminé (le marquage est modifié itérativement). Ce type d'algorithme est appelé à **correction d'étiquettes.**

L' algorithme de BELLMAN-FORD est valable pour des graphes sans circuit, valués par des longueurs quelconques.

Notations :

- S = L'ensemble de n sommets est numéroté de 1 à n.
- C = L' ensemble de noeuds est déjà marqué.
- M = L' ensemble de noeuds non marqués (= S\C), pour lesquels les plus courtes distances ne sont pas encore connues. La plus coute distance de l'origin à un sommet v ne calcule que lorsque tous les prédécesseurs de v ($\Gamma^{-}(v)$) sont dans C.
- Pr(p) = Sommet précédant p sur le plus court chemin de l'origine à p.
- d = Plus courte distance de l'origine aux autre sommets.

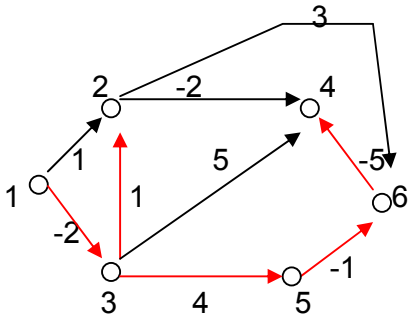
PRINCIPE DE L'ALGORITHME.

1. Initialisations.
   - Choisir le sommet s_1 pour l'origine.
   - C = $\{s_1\}$; M = $\{s_2,\ldots,s_n\}$.
   - d[1] = 0.
   - Pr[1] = 1.

2. À chaque itération :
   - Choisir un sommet $x \in M$ tel que tous les prédécesseurs de $x \in CC$, c'est à dire $\Gamma^{-}(v) \subset C$.
   - Mises à jour CC et M :
     $C := C \cup \{x\}$; $M = S\backslash C$.
   - Calculer $d[x] = \min \{ d[y] + l[y,x]: y \in \Gamma^{-}(x)\}$,
     et Pr[x] qui est l' indice que ce minimun est atteint.

Complexité : O(nm). $O(n^3)$ pour des graphes denses, i.e., des graphes tels que $m \approx n^2$.

EXEMPLE.



Initialisation : M, d, Pr :
M = { 2, 3, 4, 5, 6}, C={1}
d = [0, 10, 3, ∝, 6, ∝]
Pr = [1, 1, 1, 1, 1, 1]
$\Gamma^-(2)$ ={1,3}; $\Gamma^-(3)$={1} ; $\Gamma^-(4)$={2,3,6}
$\Gamma^-(5)$ ={3} ; $\Gamma^-(6)$ ={2,5}

FIG.3.1. Graphe valué orienté sans circuit de racine s_1.

- **1er étape.** Choisir s_3 car $\Gamma^-(3)$={1} . Remise à jour M, C, d, Pr :
  M = { 2, , 4, 5, 6} C= {1,3}
  d = [0, , -2, , ,]
  Pr = [1, , 1, , ,]
- **2è étape.** À cette étape,on aurait pu choisir de supprimer le sommet s_5 au lieu du sommet s_2. Remise à jour M, C, d, Pr :
  M = { 2, , 4, , 6} C= {1,3,5}
  d = [0, , -2, , 2 ,]
  Pr = [1, , 1, , 3,]
- **3è étape.** Choisir s_2. Remise à jour M, C, d, Pr :
  M = { , , , 4, , 6} C= {1,2,3,5}
  d = [0, -1, -2, , 2 ,]
  Pr = [1, 3, 1, , 3,]
- **4è étape.** Choisir s_6. Remise à jour M, C, d, Pr :
  M = { , , , 4, , } C= {1,2,3,5,6}
  d = [0, -1, -2, , 2 , 1]
  Pr = [1, 3, 1, , 3, 5]
- **5è étape.** Choisir s_4. Remise à jour M, C, d, Pr :
  M = { , , , , , } C= {1,2,3,4,5,6}
  d = [0, -1, -2, -4, 2 , 1]
  Pr = [1, 3, 1, 6, 3, 5]

Algorithme se termine car M = ∅.

À l'issue de la procédure, on obtient le résultat suivant :

- Le plus court chemin de s_1 vers s_2 est: $s_1 \to s_3 \to s_2$ et son coût est égal à -1
- Le plus court chemin de s_1 vers s_3 est: $s_1 \to s_3$ et son coût est égal à -2
- Le plus court chemin de s_1 vers s_4 est: $s_1 \to s_3 \to s_5 \to s_6 \to s_4$ et son coût est égal à –4.
- Le plus court chemin de s_1 vers s_5 est: $s_1 \to s_3 \to s_5$ et son coût est égal à 2
- Le plus court chemin de s_1 vers s_6 est: $s_1 \to s_3 \to s_5 \to s_6$ et son coût est égal à 1

3.4. ENTRE TOUS LES COUPLES DE SOMMETS : ALGORITHME DE FLOYD (algorithme matriciel) (1962).

On va ainsi calculer un distancier n x n. Si tous les arcs sont tous de longueur positive ou nulle ($l(u) \geq 0$), on peut appliquer n fois l'algorithme de Dijktra-Moore pour chaque sommet i. Si le graphe comporte des arcs de longueur strictement négative, on peut appliquer n fois l'algorithme de Bellman-Ford. L'algorithme de Floyd constitue une autre approche qui peut être avantageuse principalement par rapport à la seconde solution, qui nécessite un temps d'exécution en $O(n^4)$ pour des graphes denses. Contrairement aux algorithmes à origine unique qui supposent que le graphe est représenté par une liste d'adjacence, l'algorithme de Floyd (algorithme de programmation dynamique) utilise une représentation par matrice d'adjacence.

Soient les matrices : $L = [l_{ij}]$; $P = [p_{ij}]$.

$l_{ij} = l(i, j)$ si $(i, j) \in U$
$= \infty$ sinon
$l_{ii} = 0$.
$l_{ii} = 0$

$p_{ij} = 0$ si $l_{ii} = \infty$
$p_{ij} = i$ sinon.

En fin d'algorithme :

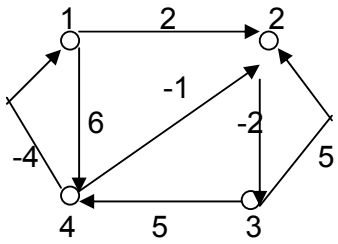
p_{ij} = prédécesseur de j sur le plus court chemin de i à j.
l_{ij} = longueur du plus court chemin entre i et j.

PROCEDURE FLOYD (L, P)

```
For (k=1 ; k≤ n ; k++)
    For (i=1 ; i≤ n ; i++)
        For (j=1 ; j≤ n ; j++)
            IF  (l[i,k] + l[k,j] < l[i,j])
                    {l[i,j] = l[i,k]  + l[k,j]  ; p[i,j] =p[k,j]}
```

Complexité : $O(n^3)$.

EXEMPLE .



Initialisation : les matrices L, P.

L_0 =

	1	2	3	4
1	0	2	∝	6
2	∝	0	-2	∝
3	∝	5	0	5
4	-4	-1	∝	0

P_0 =

1	1	0	1	
0	2	2	0	
0	3	3	3	
4	4	0	4	

Les étapes :

- **k =1.**

L_1 =

	1	2	3	4
1	0	2	∝	6
2	∝	0	-2	∝
3	∝	5	0	5
4	-4	-2	∝	0

P_1 =

1	1	0	1	
0	2	2	0	
0	3	3	3	
4	1	0	4	

- **k = 2**

L_2 =

	1	2	3	4
1	0	2	0	6
2	∝	0	-2	∝
3	∝	5	0	5
4	-4	-2	-4	0

P_2 =

1	1	2	1	
0	2	2	0	
0	3	3	3	
4	1	2	4	

- **k =3**

L_3 =

	1	2	3	4
1	0	2	0	5
2	∝	0	-2	3
3	∝	5	0	5
4	-4	-2	-4	0

P_3 =

1	1	2	3	
0	2	2	3	
0	3	3	3	
4	1	2	4	

- **k = 4**

L_4 =

	1	2	3	4
1	0	2	0	5
2	-1	0	-2	3
3	1	3	0	5
4	-4	-2	-4	0

P_4 =

1	1	2	3	
0	2	2	3	
4	1	3	3	
4	1	2	4	

Obtention des plus court chemin.

Pour obtenir un plus court chemin de s_l à s_j , il suffit d'utiliser la ligne numéro i de la matrice P. Par exemple, si on veut obtenir le plus court chemin μ de s_4 à s_3, on consulte la matrice P ainsi : P[4,3]=2 :s_2 est donc le prédécesseur de s_3 ; P[4,2]=1 : s_1 est donc le prédécesseur de s_2 ; P[4,1]=4 :s_4 est donc le prédécesseur de s_1 Finallement le chemin $\mu = s_4 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3$.

L'algorithme utilisé est celui de Floyd (dans une application de recherche de la fermeture transitive d'un graphe, cet algorithme a été développé par Warshall la même année (1962) ; cet algorithme est donc souvent appelé **« Floyd-WARSHALL ».**

PROCEDURE FLOYD-WARSHALL (L, P)

Soient les matrices : L = $[l_{ij}]$; P = $[p_{ij}]$.

l_{ij} = 1 si (i, j) $\in$ U
= 0 sinon

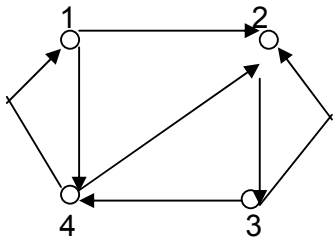
p_{ij} = 0 si l_{ii} = 0
p_{ij} = i sinon.

PROCEDURE FLOYD-WARSHALL (L, P)

```
For (k=1 ; k≤ n ; k++)
    For (i=1 ; i≤ n ; i++)
        For (j=1 ; j≤ n ; j++)
            IF  (l[i,j]  = = 0)
                    {l[i,j] = l[i,k] *l[k,j]  ; p[i,j] =p[k,j] ;}
```

Complexité : $O(n^3)$.

EXEMPLE .



Initialisation : les matrices L, P.

$L_0 =$

	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	0	1	0	1
4	1	1	0	0

$P_0 =$

	1	2	3	4
1	0	1	0	1
2	0	0	2	0
3	0	3	0	3
4	4	4	0	0

Les étapes :

- **k =1.**

$L_1 =$

	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	0	1	0	1
4	1	1	0	1

$P_1 =$

	1	2	3	4
1	0	1	0	1
2	0	0	2	0
3	0	3	0	3
4	4	4	0	1

- **k = 2**

$L_2 =$

	1	2	3	4
1	0	1	1	1
2	0	0	1	0
3	0	1	1	1
4	1	1	1	1

$P_2 =$

	1	2	3	4
1	0	1	2	1
2	0	0	2	0
3	0	3	2	3
4	4	4	2	1

- **k =3**

$L_3 =$

	1	2	3	4
1	0	1	1	1
2	0	1	1	1
3	0	1	1	1
4	1	1	1	1

$P_3 =$

	1	2	3	4
1	0	1	2	1
2	0	3	2	3
3	0	3	2	3
4	4	4	2	1

- **k = 4**

$L_4 =$

	1	2	3	4
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1

$P_4 =$

	1	2	3	4
1	4	1	2	1
2	4	3	2	3
3	4	3	2	3
4	4	4	2	1