PLANTIN Ludovic

MPSI
Lycée Jean Perrin

TIPE
Distance d’un point à l’origine

Année 2004-2005

Sommaire

Annexe

I. Introduction : Positionnement du problème

Sous le nom « distance à l'origine » qui peut au première abord paraître être le nom d'un problème sans utilité, ce cache en réalité un problème auquel nous sommes toujours confrontés. En effet il s'agit simplement de trouver quel est le plus court chemin pour aller d'un point à un autre, question fréquente pour un automobiliste par exemple, car si toutes les routes mènent à Rome, certaines y mènent plus rapidement.

La réponse à ce problème peut peut-être sembler enfantine, il suffit en effet de comparer les différents chemins pour trouver le plus court, mais pour être sur d'avoir comparé tout les chemins, il faut les parcourir avec une méthode rigoureuses, sans en oublier et pourtant en éliminant certains, sinon rien comment compter la distance ou le temps que prendrait un trajet si l'on tourne en rond indéfiniment.

C'est pourquoi certains scientifiques se sont penchés, au cours des dernières décennies, sur le problème du plus court chemin à l'origine, pour créer des algorithmes capables de résoudre de façon optimale ce problème. Ces algorithmes ont eux permis la mise en place des logiciels traçant un itinéraire, comme les GPS, mais sont utiles dans de nombreux autres domaines, en effet gérer le trafic sur le réseau de la SNCF serait une tâche incroyablement lourde sans eux, et ils sont appréciables pour déterminés les routeurs Internet les plus appropriés pour relier votre ordinateur à l'autre bout du monde.

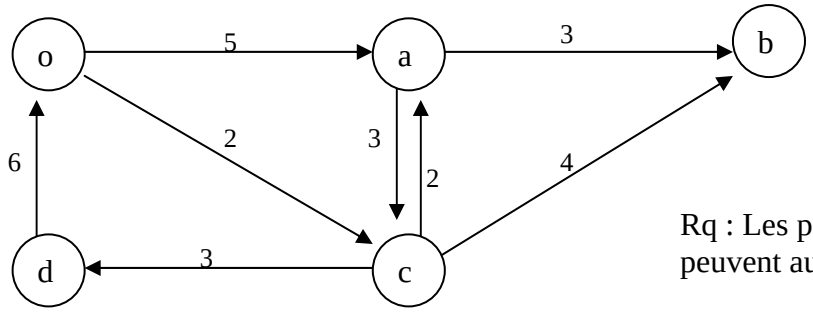
II. Les graphes pondérés

2.1 Définition mathématique

Un graphe pondéré est la donnée d'un quadruplé (S,A,X,f) avec:

- *S* un ensemble fini non vide des *sommets* du graphe (les *points*).
- *A* une partie de $S \times S$ dont les éléments forment les *arêtes* du graphe (les *liaisons* ou *arcs*).
- *X* un ensemble quelconque, fini ou non, qui détermine les *poids* possibles des arêtes.
- $f: A \to X$ *la fonction de valuation* du graphe qui à chaque arête associe son poids.

2.2 Représentation schématique



Rq : Les poids des liaisons peuvent aussi être négatif.

- S:={o,a,b,c,d}
- A:={(o,a),(o,c),(a,b),(a,c),(c,a),(c,b),(c,d),(d,o)}
- X:=$\mathbb{N}$
- f peut être représentée par la matrice :

u->v	o	a	b	c	d
o		5		2	
a			3	3	
b					
c		2	4		3
d	6				

2.3 Remarques

- On dit que le graphe est *symétrique* si chaque liaison du graphe peut être parcourue dans les deux sens dans le même temps.
- On dit que le graphe est *connexe* si pour tout couple de sommets distincts (*a*,*b*), il existe un chemin (voir 3.1) de *a* à *b*.

2.4 Application

Si l'on reprend notre exemple de l'itinéraire, chaque sommet (nœud) du graphe représente l'intersection de deux routes (un *carrefour*), chaque flèche une route qu'on ne peut prendre que dans le sens de la flèche (*route à sens unique*), son poids ou coefficient représentant alors *la longueur de cette route* (on peut aussi le voir comme le temps nécessaire pour la parcourir).

III. **Définitions et lemme fondamentaux**

3.1 Définition du chemin

Soit (S,A,X,f) un graphe pondéré, $(a,b)\in S^2$.
On appelle *chemin de a à b* toute suite finie $(u_0,u_1,...,u_n)\in S^{n+1}$ telle que $u_0=a$, $u_n=b$ et telle que pour tout i, $1<i\leq n$, (u_{i-1},u_i) appartient à A.
Exemple : (a,c,d,o,c,b)

3.2 Définition de la longueur d'un chemin

Soit (S,A,X,f) un graphe pondéré, $p:=(u_0,u_1,...,u_n)\in S^{n+1}$ un chemin de u_0 à u_n.
On définit la *longueur d'un chemin p* comme *la somme des poids* (longueurs) *des liaisons* qui le constituent :

$$f(p)=\sum_{i=1}^{k} f(u_{i-1},u_i)$$

Exemple : f((a,c,b))=f((a,c)) + f((c,b))=3+4=7

3.3 Définition d'un plus court chemin

Soit (S,A,X,f) un graphe pondéré, $(a,b)\in S^2$.
Soit E l'ensemble des chemins de a à b.
On définit la *longueur d'un plus court chemin* entre deux sommets a et b par :

$$\delta(a,b):=\begin{cases}\min\{f(p) : p\in E\} & \text{si } E\neq\varnothing\\ \infty & \text{sinon}\end{cases}$$

On définit alors un plus court chemin de a à b, comme un chemin p de a à b tel que $f(p)=\delta(a,b)$.

3.4 Lemme : Les sous chemins de plus courts chemins sont des plus courts chemins

Soit (S,A,X,f) un graphe pondéré, $p:=(u_0,u_1,...,u_n)\in S^{n+1}$ un plus court chemin de u_0 à u_n.
Soit $i\in[|0,n|]$, $j\in[|i,n|]$, $p_{ij}:=(u_i,u_{i+1},...,u_j)$.
Alors p_{ij} est un plus court chemin de u_i à u_j.

Démonstration

Soit $p_{0i}:=(u_0,u_1,...,u_i)$ et $p_{jn}:=(u_j,u_{j+1},...,u_n)$.
Donc $\delta(u_0,u_n)=f(p)=f(p_{0i})+f(p_{ij})+f(p_{jn})$.

On suppose qu'il existe q un chemin de u_i à u_j et que $f(q)<f(p_{ij})$.
Alors $f(p_{0i})+q+f(p_{jn})$ est un chemin de u_0 à u_n et $f(p_{0i})+q+f(p_{jn})<\delta(u_0,u_n)$,
ce qui est impossible par définition de δ.

Ce raisonnement par l'absurde montre donc que $\delta(u_i,u_j)=p_{ij}$.

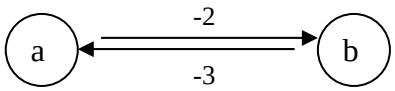
IV. Algorithme de Floyd

4.1 Présentation

L'algorithme de Floyd est un des algorithmes traitant du problèmes des plus courts chemins les plus simples à mettre en œuvre. Il définit une façon de parcourir les graphes et de comparés tous les différents chemins entre eux sans en oublier, et retourne la longueur des chemins les plus court entre les différents sommets.

4.2 Hypothèses

Pour utiliser cet algorithme on supposera disposer d'un graphe pondéré (S,A,X,f) valué positivement ($X \subset [|0 ; +\infty [$), sinon l'existence d'un chemin le plus court n'est pas assuré, comme dans l'exemple suivant :



On pose :

- n:=card(S) (le nombre de sommet du graphe).
- $\{u_1,...,u_n\}:=S$ (l'ensemble des sommets du graphe).
- δ^0, une matrice de taille $n \times n$ représentative de la fonction :

$$g : \begin{cases} S \times S \to \mathbb{R} \\ (u,v) \ \alpha \begin{cases} f(u,v) & \text{si } (u,v) \in A \\ 0 & \text{si } u=v \\ \infty & \text{sinon} \end{cases} \end{cases}$$

4.3 Principe

L'algorithme repose sur le lemme 3.4 et sur le fait que, comme les poids des arrêtes sont positifs, tout chemin contenant un cycle est nécessairement plus long que les autres.

Il suit une méthode ascendante, qui calcul successivement pour u_k, k croissant de 1 à n et pour tout couple de sommet (u,v) du graphe, les chemins minimaux de u à v parmi ceux dont les sommets intermédiaires sont dans $\{u_1,...,u_k\}$. On notes ces chemins les k-chemins.

Notons $\delta^k(u,v)$ la longueur du plus court k-chemin de u à v (ie : la longueur du plus court chemin de u à v dont les sommets intermédiaires sont dans $\{u_1,...,u_k\}$).

On a alors : $\boxed{\delta^{k}(u,v)=\min(\delta^{k-1}(u,v), \delta^{k-1}(u,u_k)+\delta^{k-1}(u_k,v))}$

et : $\boxed{\delta^{n}(u,v)=\delta(u,v)}$

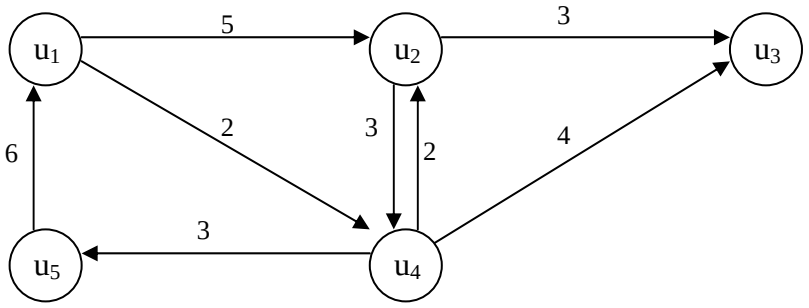
On construit ainsi par récurrence une suite $(\delta^k)_{k \in [|0,n|]}$ de matrice (tableau) de taille $n \times n$ représentant les longueurs des chemins minimaux liants deux à deux les sommets du graphes et dont les sommets intermédiaires sont dans $\{u_1,...,u_k\}$.

4.4 Application

Soit un graphe pondéré (S,A,X,f) valué positivement tel que :

- $S:=\{u_1,u_2,u_3,u_4,u_5\}$
- $A:=\{(u_1,u_2),(u_1,u_4),(u_2,u_3),(u_2,u_4),(u_4,u_2),(u_4,u_3),(u_4,u_5),(u_5,u_1)\}$
- $X:=\mathbb{N}$
- f :

u->v	u_1	u_2	u_3	u_4	u_5
u_1		5		2	
u_2			3	3	
u_3					
u_4		2	4		3
u_5	6				



On a donc :

- $\delta^0 = \begin{pmatrix} 0 & 5 & \infty & 2 & \infty \\ \infty & 0 & 3 & 3 & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & 2 & 4 & 0 & 3 \\ 6 & \infty & \infty & \infty & 0 \end{pmatrix}$ (Les arrêtes inexistantes sont de longueur infinie, et la distance d'un sommet à lui même est nulle.)
- $n:=card(S)=5$ (Il faudra donc calculer 5 matrices).

On calcul ensuite δ^1 en utilisant $\delta^1(u,v)=\min(\delta^0(u,v), \delta^0(u,u_k)+\delta^0(u_k,v))$, puis $\delta^2,\ldots,\delta^5$ en se basant sur le même procédé de récurrence :

$$\delta^1 = \begin{pmatrix} 0 & 5 & \infty & 2 & \infty \\ \infty & 0 & 3 & 3 & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & 2 & 4 & 0 & 3 \\ 6 & 11 & \infty & 8 & 0 \end{pmatrix} \qquad \delta^2 = \begin{pmatrix} 0 & 5 & 8 & 2 & \infty \\ \infty & 0 & 3 & 3 & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & 2 & 4 & 0 & 3 \\ 6 & 11 & 14 & 8 & 0 \end{pmatrix}$$

$$\delta^3 = \begin{pmatrix} 0 & 5 & 8 & 2 & \infty \\ \infty & 0 & 3 & 3 & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & 2 & 4 & 0 & 3 \\ 6 & 11 & 14 & 8 & 0 \end{pmatrix} \qquad \delta^4 = \begin{pmatrix} 0 & 4 & 6 & 2 & 5 \\ \infty & 0 & 3 & 3 & 6 \\ \infty & \infty & 0 & \infty & \infty \\ \infty & 2 & 4 & 0 & 3 \\ 6 & 10 & 12 & 8 & 0 \end{pmatrix}$$

$$\delta^5 = \begin{pmatrix} 0 & 4 & 6 & 2 & 5 \\ 12 & 0 & 3 & 3 & 6 \\ \infty & \infty & 0 & \infty & \infty \\ 9 & 2 & 4 & 0 & 3 \\ 6 & 10 & 12 & 8 & 0 \end{pmatrix}$$

4.5 Programmes

Programme MAPLE

```
>floyd:=proc(mat) local d,n,dik,dkj,s,k,i,j;
  d:=mat;
  n:=nops(mat);
  for k to n do
    for i to n do
      for j to n do
        s:= d[i][k]+d[k][j];
        if s<d[i][j] then d[i][j]:=s fi;
      od;
    od;
  od;
  RETURN(d);
end proc;
```

```
>mat:=[[0,5,infinity,2,infinity],
       [infinity,0,3,3,infinity],
       [infinity,infinity,0,infinity,infinity],
       [infinity,2,4,0,3],
       [6,infinity,infinity,infinity,0]];
```

```
>floyd(mat);
```

[[0,4,6,2,5],[12,0,3,3,6],[∞,∞,0,∞,∞],[9,2,4,0,3],[6,10,12,8,0]]

Programme JAVA

```
public class floyd {
   static final int M=Integer.MAX_VALUE;
   static int[][] g={
     {0, 5, M, 2, M},
     {M, 0, 3, 3, M},
     {M, M, 0, M, M},
     {M, 2, 4, 0, 3},
     {6, M, M, M, 0}
   };
   public static void main(String[] args) throws Exception {
      int n=g.length;
      int[][] d=new int[n][n];
      for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
          d[i][j]=g[i][j];
        }
      }
      for (int k=0; k<n; k++) {
        for (int i=0; i<n; i++) {
          for (int j=0; j<n; j++) {
            if (d[i][k] == M || d[k][j] == M) continue;
            int s=d[i][k]+d[k][j];
            if (s<d[i][j]) d[i][j]=s;
          }
        }
      }
      for (int i=0; i<n; i++) {
        String result="";
        for (int j=0; j<n; j++) {
          if (d[i][j]==M) result=result+"     M";
          else result=result+"     "+d[i][j];
        }
        System.out.println(result);
      }
   }
}
```

4.6 Défauts

Le défaut de cet algorithme est qu'il est très coûteux, il demande beaucoup de calculs (n^3) et de mémoire et en plus il ne renvoie pas un plus court chemin mais la longueur d'un plus court chemin, il n'est donc pas réellement adapté au problème du plus court chemin.

Pour qu'il retourne un plus court chemin il faut ajouter une matrice C (type String) dans laquelle, à chaque fois que le test « s<d[i][j] » est vérifié, on ajoute le nom du k-ième sommet en C[i][j].

V. Algorithme de Dijkstra

5.1 Présentation

L'algorithme de Dijkstra, publié en 1959, est une bonne alternative à celui de Floyd. Il est certes plus difficile à mettre en place, mais nécessite beaucoup moins d'opérations et de mémoire. Il permet de calculer le plus court chemin entre un sommet particulier et tous les autres. C'est un algorithme de type « glouton » car à chaque itération il traite un nouveau sommet.

5.2 Hypothèses

Comme dans l'algorithme précédent, on supposera disposer d'un graphe pondéré (S,A,X,f) valué positivement (pour les mêmes raisons).

On pose :

- $n:=card(S)$ (le nombre de sommet du graphe).
- $\{u_1,...,u_n\}:=S$ (l'ensemble des sommets du graphe).

On appelle u_1 l'origine, c'est le sommet pour lequel on va chercher les chemins les plus court le séparant de chacun des autres sommets de S.

On pose un ensembles E de sommets dont les longueurs finales de plus court chemin à partir de l'origine u_1 ont déjà été calculée. Au départ $E=\{u_1\}$.

5.3 Principe

On va construire un premier vecteur $\lambda=(\lambda_1,\lambda_2,...,\lambda_n)\in\mathbb{R}^n$ ayant n composantes tel que λ_i soit la longueur d'un plus court chemin allant de u_1 à u_i dont les sommets intermédiaires sont dans E.

On va construire un second vecteur $p=(p_1,p_2,\ldots,p_n)\in S^n$ ayant n composantes pour mémoriser le chemin, chaque valeur p_i donnant le sommet qui précède u_i dans un chemin le plus court de u_1 à u_i dont les sommets intermédiaires sont dans E.

Au début de l'algorithme, le chemin le plus court connu entre la source et chacun des sommets est le chemin direct, avec une arête de poids infini s'il n'y a pas de liaison entre les deux sommets et une arrête de poids nul si le sommet considéré est l'origine. On initialise donc le vecteur λ par les poids des arêtes reliant la source à chacun des sommets, et les valeurs p_i du vecteur p, par l'origine u_1, si il existe un arc allant de u_1 au sommet et par NULL sinon.

A chaque itération de l'algorithme, on choisit un sommet u_i du graphe parmi ceux qui n'ont pas encore été traité (et différent du sommet origine u_1) (ie : dans S\E), tel que la longueur connue (pour l'instant) du plus court chemin allant de u_1 à u_i (ie : λ_i) soit la plus courte possible.

On ajoute ensuite u_i dans E et pour tout sommet $u_j\in S\backslash E$ tel que $(u_i,u_j)\in A$ (ie : il existe une liaison allant de u_i à u_j) on effectue : Si $\lambda_j>\lambda_i+f(i,j)$ alors $\lambda_j=\lambda_i+f(i,j)$ et $p_j=i$.

On reprend ensuite l'itération.

On arrête quand S\E est vide.

5.4 Validité

Il reste à prouver que cet algorithme fonctionne bel et bien, ce qui à première vue n'est pas franchement une évidence. La récurrence à effectuer repose sur les deux hypothèses suivantes :

- Les vecteurs λ et p permettent d'obtenir pour tous les sommets u_i de E un plus court chemin qui mène de la source à u_i.
- Les vecteurs λ et p permettent d'obtenir pour tous les sommets u_i de S\E un plus court chemin qui mène de la source à u_i en ne passant que par des sommets de E.

- Ces deux hypothèses sont bien vérifiées à l'initialisation des tableaux par construction.

- Supposons donc ces hypothèses vérifiées à une étape donnée de l'algorithme.

Soit u_i le sommet de S\E tel que λ_i minimise le vecteur λ.
Montrons que les deux hypothèses sont encore valables pour $E \cup \{u_i\}$.

D'après la première hypothèse, il existe un plus court chemin c allant de la source à p_i et ne passant que par des points de E.

-- Montrons que $c \cup (p_i,u_i)$ est un chemin minimal de la source à u_i.

Par l'absurde, si ce n'était pas le cas, on pourrait trouver un autre chemin plus court qui entre nécessairement dans l'ensemble E par un certain sommet u_j (sinon on contredit la deuxième hypothèse) différent de u_i (en pointillés épais sur le dessin). Or, d'après le choix de u_i, $\lambda_i < \lambda_j$ (distance à la source), donc comme toutes les arêtes sont de poids positifs, ce nouveau chemin passant par u_j aura un poids supérieur ou égal au poids de $c \cup (p_i,u_i)$.

Ainsi, $c \cup (p_i,u_i)$ est bien un chemin minimal de la source à u_i, la première hypothèse sera bien vérifiée au début de l'étape suivante.

-- Pour démontrer la seconde hypothèse appliquée à $E \cup \{u_i\}$, considérons un sommet u_j de S\E distinct de u_i. De deux choses l'une : soit on peut trouver un plus court chemin de la source à u_j en ne passant que par des points de $E \cup \{u_i\}$, soit il n'y a rien à modifier et c'est fini.

Si on peut effectivement trouver un plus court chemin, il passe par u_j d'après la deuxième hypothèse appliquée à E. Il reste à voir que $c \cup (p_i,u_i) \cup (u_i,u_j)$ est bien un plus court chemin de la source à u_j en ne passant que par des sommets de $E \cup \{u_i\}$.

Considérons donc un tel chemin, passant par u_i.

S'il retourne dans E après avoir franchi u_i en un certain sommet u_k, comme u_k a été incorporé à E avant u_i, il existe un chemin de la source à u_k plus court que $c \cup (p_i,u_i)$, et donc a fortiori plus court que $c \cup (p_i,u_i) \cup (u_i,u_j)$, on obtient donc une contradiction.

Ainsi, ce chemin ne retourne pas dans E et est donc bien de la forme escomptée. La mise à jour des vecteurs λ et p réalise les modifications nécessaires pour les seuls sommets où il peut y avoir lieu de modifier quelque chose, c'est-à-dire ceux directement reliés à u_i.

Ce raisonnement par récurrence montre la véracité des hypothèses à chaque étape.

A la fin de l'algorithme E est alors égal à S, donc la deuxième hypothèse fournit la preuve de la validité de l'algorithme.

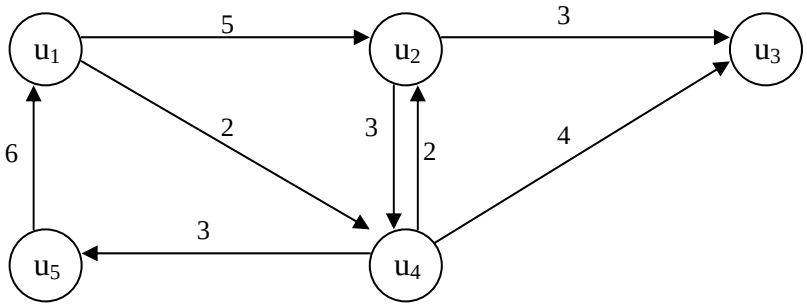
On remarque aussi que la dernière étape (la $(n-1)^{ème}$)) est inutile, puisque, comme il ne reste qu'un seul sommet u dans S\E, un plus court chemin ne passant que par des sommets de E\{u} est un plus court chemin de la source à u.

5.5 Application

Soit un graphe pondéré (S,A,X,f) valué positivement tel que :

- $S:=\{u_1,u_2,u_3,u_4,u_5\}$
- $A:=\{(u_1,u_2),(u_1,u_4),(u_2,u_3),(u_2,u_4),(u_4,u_2),(u_4,u_3),(u_4,u_5),(u_5,u_1)\}$
- $X:=\mathbb{N}$
- f :

u->v	u_1	u_2	u_3	u_4	u_5
u_1		5		2	
u_2			3	3	
u_3					
u_4		2	4		3
u_5	6				



On a donc :

- n :=card(S)=5
- $\{u_1,u_2,u_3,u_4,u_5\}:=S$

On prend le sommet u_1 comme origine, on pose $E:=\{u_1\}$.
On pose $\lambda:=(\lambda_1,\lambda_2,\lambda_3,\lambda_4,\lambda_5)\in\mathbb{R}^5$ et $p:=(p_1,p_2,p_3,p_4,p_5)\in S^5$.

On initialise ces deux vecteurs :

- $\lambda=(0,5,\infty,2,\infty)$
- p=(NULL,u_1,NULL,u_1,NULL) (Seuls (u_1,u_2) et (u_1,u_4) appartiennent à A)

1ère itération : On choisit $u_4\in S\backslash E$ car $\lambda_4=\min(\lambda_2,\lambda_3,\lambda_4,\lambda_5)$ ($2=\min(5,\infty,2,\infty)$)
On ajoute u_4 dans E, donc $E=\{u_1,u_4\}$
Les sommets v tel que $v\in S\backslash E=\{u_2,u_3,u_5\}$ et tel que l'arc (u_4,v) existe (ie : $(u_4,v)\in A$) sont u_2, u_3, et u_5 donc :

- λ_2 prend la nouvelle valeur $\min(\lambda_2;\lambda_4+f(u_4,u_2))=\min(5;2+2)=4$.
  - $p_2=4$ (On vient de remarquer que le chemin était plus court en passant par u_4 quand y allant directement).
- λ_3 prend la nouvelle valeur $\min(\lambda_3;\lambda_4+f(u_4,u_3))=\min(\infty;2+4)=6$.
  - $p_3=4$.
- λ_5 prend la nouvelle valeur $\min(\lambda_2;\lambda_4+f(u_4,u_5))=\min(\infty;2+3)=5$.
  - $p_5=4$.

On a donc à la fin de cette itération : $E=\{u_1,u_4\}$; $\lambda=(0,4,6,2,5)$; p=(NULL,u_4,u_4,u_1,u_4)

2ème itération : On choisit $u_2 \in S\backslash E=\{u_2,u_3,u_5\}$ car $\lambda_4=\min(\lambda_2,\lambda_3,\lambda_5)$ (4=min(4,6,5))
On ajoute u_2 dans E, donc $E=\{u_1,u_2,u_4\}$
Le seul sommet v tel que $v \in S\backslash E=\{u_3,u_5\}$ et tel que l'arc (u_2,v) existe est u_3 (en effet $(u_2,u_5) \notin A$) donc :

- λ_3 prend la nouvelle valeur $\min(\lambda_3;\lambda_2+f(u_2,u_3))=\min(6;4+3)=6$.
- $P_3=4$ (On vient de remarquer passer par u_2 rallonge le chemin).

On a donc à la fin de cette itération : $E=\{u_1,u_2,u_4\}$; $\lambda=(0,4,6,2,5)$; $p=(NULL,u_4,u_4,u_1,u_4)$

3ème itération : On choisit $u_5 \in S\backslash E=\{u_3,u_5\}$ car $\lambda_4=\min(\lambda_3,\lambda_5)$ (5=min(6,5))
On ajoute u_5 dans E, donc $E=\{u_1,u_2,u_4,u_5\}$
Il n'existe pas de sommets v tel que $v \in S\backslash E=\{u_3\}$ et tel que l'arc (u_5,v) existe (en effet $(u_3,u_5) \notin A$).

On a donc à la fin de cette itération : $E=\{u_1,u_2,u_3,u_4\}$; $\lambda=(0,4,6,2,5)$; $p=(NULL,u_4,u_4,u_1,u_4)$

$S\backslash E=\{u_5\}$, et on a vu dans la démonstration qu'une nouvelle étape serait inutile, on peut donc arrêter.

Rq : Si l'on s'intéresse à un chemin le plus court allant de u_1à u_2, λ_2 nous donne sa longueur, et il suffit de parcourir le vecteur p pour avoir le chemin, ainsi $p_2=u_4$ et $p_2=u_1$ donc un chemin le plus court est (u_2,u_4,u_1).

5.6 Programmes

Programme MAPLE

```
>dijkstra:=proc(mat) local f,n,S,d,p,di,i,j,k;
  f:=mat;
  S:={NULL};
  n:=nops(mat);
  d:=[];
  p:=[];
  for k to n do
    if k<>1 then S:=S union {k} fi;
    d:=[op(d),f[1][k]];
    if (k<>1 and d[k]<>infinity) then p:=[op(p),1];
    else p:=[op(p),Aucun] fi;
  od;
  while S<>{NULL} do
    di:=infinity;
    i:=S[1];
    for j to n do
      if member(j,S) then
        if d[j]<di then di:=d[j] : i:=j fi;
      fi;
    od;
    S:=S minus {i};
    for j to n do
      if member(j,S) and f[i][j]<>infinity then
        if d[j]>di+f[i][j] then d[j]:=di+f[i][j] : p[j]:=i fi;
      fi;
    od;
  od;
  RETURN(d,p);
end proc;



>mat:=[[0,5,infinity,2,infinity],
       [infinity,0,3,3,infinity],
       [infinity,infinity,0,infinity,infinity],
       [infinity,2,4,0,3],
       [6,infinity,infinity,infinity,0]];



>dijkstra(mat);
                                [0,4,6,2,5],[Aucun,4,4,1,4]
```

Programme JAVA

```
import java.util.Vector;

public class dijkstra {
   static final int M=Integer.MAX_VALUE;
   static int[][] f={
     {0, 5, M, 2, M},
     {M, 0, 3, 3, M},
     {M, M, 0, M, M},
     {M, 2, 4, 0, 3},
     {6, M, M, M, 0}
   };
   public static void main(String[] args) throws Exception {
      int n=f.length;
      Vector S=new Vector();
      int[] d=new int[n],p=new int[n];
      for (int k=0; k<n; k++) {
        if (k!=0) S.addElement(String.valueOf(k));
        d[k]=f[0][k];
        if (k!=0 && d[k]!=M) p[k]=1;
        else p[k]=0;
      }
      while (S.size()>1) {
        int di=M,i=Integer.parseInt((String) S.elementAt(0));
        for (int j=0; j<n; j++) {
          if (S.indexOf(String.valueOf(j))>=0) {
            if (d[j]<di) {
              di=d[j];
              i=j;
            }
          }
        }
        S.removeElementAt(S.indexOf(String.valueOf(i)));
        for (int j=0; j<n; j++) {
          if (S.indexOf(String.valueOf(j))>=0 && f[i][j]!=M) {
            if (d[j]>di+f[i][j] && di!=M) {
              d[j]=di+f[i][j];
              p[j]=i+1;
            }
          }
        }
      }
      String sd="d=(0",sp="p=(0";
      for (int k=1; k<n; k++) {
        if (d[k]==M) sd=sd+",M";
        else sd=sd+","+d[k];
        sp=sp+","+p[k];
      }
      System.out.println(sd+")");
      System.out.println(sp+")");
   }
}
```

VI. Conclusion

Ces deux algorithme permette donc de résoudre de façon certaine de problème du plus court chemin à l'origine, et servent tous les jours pour toutes les structures basées sur un réseau (routier , SNCF, Internet).

Néanmoins, ces structures n'utilisent pas forcément l'algorithme de Dijkstra, par exemple le réseau Internet utilise pour le moment un autre type d'algorithme pour déterminer le chemin à suivre pour transmettre des données entre deux serveurs : Routing Information Protocol (RIP).

Ceci étant dit, Dijkstra commence à s'implanter là aussi et à remplacer progressivement ce protocole. Pourquoi ? Pour la bonne et simple raison qu'il est pour l'heure le plus rapide à proposer une solution au problème des plus courts chemins.

On remarque également que pour utiliser ces deux algorithmes on a été amené à supposer que le graphe était valué positivement, un graphe possédant des poids négatifs ne pourrait pas être traiter par l'un de ces deux algorithmes, on utiliserait sans doute plutôt celui de Bellman-Ford, qui est juste un peu plus lent que celui de Dijkstra.

Enfin certains scientifiques tentent de créer des systèmes analogues au modèle de la fourmilière. En effet on a pu observer que les fourmis empruntaient toujours le chemin le plus court pour déplacer la nourriture de sa source à la fourmilière. En simplifiant, chaque fourmi dépose des hormones tout le long du chemin qu'elle suit, plus un chemin va être rapide, plus il va y avoir d'hormone dessus puisque la fourmi fera plus vite les aller et retours en le suivant, et les autres fourmis vont ensuite suivrent la trace où il y a le plus d'hormone.

Annexes

Biographie de Edsger Dijkstra

(Un article de Wikipédia, l'encyclopédie libre)

Edsger Dijkstra (Rotterdam, 11 mai 1930 - 6 août 2002) est un mathématicien et informaticien néerlandais du XX^{e} siècle

Après des études de physique théorique, il s'engagea dès 1955 dans le domaine de l'informatique naissante dont il fut l'un des pionniers les plus éclairés.

Parmi ses contributions se trouve un algorithme de plus court chemin dans les graphes, connu sous le nom d'algorithme de Dijkstra. Enseignant à l'université d'Eindhoven, il commença à se faire connaître en matière de systèmes avec *The THE operating system*, un système construit en couches d'abstractions successives, et idéal pour l'enseignement (« THE » signifiant *Technische Hochschule Eindhoven*). Fort de l'expérience d'écriture de ce système, il formalisa une notion avant lui diffuse, celle de sémaphore, et introduisit la notion de *section critique* avec deux exemples devenus des classiques :

- Le problème des lecteurs et des rédacteurs
- Le problème des *philosophes aux spaghettis*

Constatant les dégâts provoqués par l'usage incontrôlé de l'instruction GOTO en programmation, il rédigea en 1968 dans les *Communcations of the ACM* l'article fondateur *On the Go To Statement Considered Harmful*, considéré comme l'article qui décida nombre de programmeurs à passer à la programmation structurée (contenant des structures de contrôle plus policées comme *if ... then ... else ... fi, do...while, repeat...until*).
Il avait eu un rôle important dans le développement du langage ALGOL à la fin des années 1950, et développa ensuite *« la science et l'art des langages de programmation en général, contribuant grandement à notre compréhension de leur structure, de leur représentation et de leur implémentation »* (citation de l'ACM, *Association for Computer Machinery*). Toutefois Niklaus Wirth (créateur d'Algol W et de Pascal) alla plus loin que lui dans le domaine en considérant de pair la structuration des *programmes* et celles des *données*.
Le discours qu'il prononça en 1972 lorsqu'il reçut le prix Turing, *The Humble Programer*, est resté célèbre. Il est disponible en traduction
française (*http://www.adrahon.org/classiques/le_programmeur_modeste.php*) (*lien externe*).

Anecdotes

Il est né dans le même village que Vincent Van Gogh et son ambition était de devenir plus célèbre que lui. Parmi les informaticiens c'est réussi mais pour le grand public ce n'est pas certain.
Il adorait Martin Luther King

Citations

L'informatique est autant la science de l'ordinateur que l'astronomie est celle du téléscope.
A propos de l'intelligence artificielle :
Se demander si un ordinateur sait penser c'est comme se demander si un sous-marin sait nager.

Sources

- Introduction à l'algorithmique, écrit par Cormen, Leiserson, Rivest et Stein, édition DUNOD :
  Livre de second cycle et école d'ingénieur sur l'algorithmique
- http://www.nimbustier.net/publications/djikstra/index.html :
  Dossier sur l' « Implémentation de l'algorithme de Dijkstra »
- http://cermics.enpc.fr/polys/oap/node77.html :
  L'algorithme de Floyd
- http://www.aromath.net/Page.php?IDP=624&IDD=0 :
  L'algorithme de Dijkstra par Michael Atiyah
- http://www.apprendre-en-ligne.net/graphes/dijkstra/algorithme.html :
  Description de l'algorithme de Dijkstra
- http://wims.auto.u-psud.fr/wims/wims.cgi?lang=fr&+module=U1/graph/dijkstra.fr :
  Site proposant une application réalisant l'algorithme de Dijkstra
- http://www.cs.utexas.edu/users/EWD/welcome.html :
  Site consacré à Dijstra (anglophone)
- http://brassens.upmf-grenoble.fr/IMSS/mamass/graphecomp/bellmannFord.htm :
  L'algorithme de Bellmann-Ford
- http://fr.wikipedia.org :
  Encyclopédie libre et gratuite en ligne, permettant à chacun de s'exprimer.