

TP N° 8

Algorithmes sur les graphes

Buts : L'objectif de cette séance est de programmer l'algorithme de plus court chemin de Dijkstra.

1 Description de l'algorithme de Dijkstra

On considère un graphe $G=(V, E)$ non orienté dont chaque arête e est de longueur l_e . L'algorithme de Dijkstra permet de calculer un plus court chemin d'un sommet source s à tous les autres sommets du graphe. Il maintient à jour, pour chaque sommet v une étiquette $Vlabel[v]$ qui est une borne supérieure sur la plus courte distance entre la source s et v . À chaque itération, l'algorithme divise les sommets en deux groupes :

- les sommets marqués, l'étiquette (permanente) $Vlabel[v]$ représente la longueur d'un plus court chemin entre la source et le sommet v ,
- les sommets non-marqués, l'étiquette (temporaire) $Vlabel[v]$ est une borne supérieure de la longueur d'un plus court chemin entre la source et le sommet v .

L'idée de base de l'algorithme est d'explorer le graphe à partir de la source et des sommets marqués pris par ordre croissant de leur distance à s . Au départ la source s reçoit comme étiquette (permanente) 0 et les autres sommets du graphe une étiquette (temporaire) ∞ .

À chaque itération, l'étiquette d'un sommet v est sa plus courte distance depuis la source le long d'un chemin dont tous les sommets internes (interne = les sommets du chemin sauf ses deux extrémités, ici s et v) sont marqués. L'algorithme choisit le sommet x non-marqué de plus petite étiquette, le marque et met à jour les étiquettes de tous les voisins i non-marqués de x :

$$Vlabel[i] \leftarrow \min\{Vlabel[i], Vlabel[x]+l_{ix}\}$$

Afin de connaître les chemins calculés, si $Vlabel[i] \leftarrow Vlabel[x]+l_{ix}$, on note que le prédécesseur de i sur le plus court chemin calculé est x . L'algorithme se termine lorsque tous les sommets sont marqués.

La figure 1 donne un exemple d'exécution de l'algorithme. Les sommets entourés représentent les sommets marqués, les étiquettes et les prédécesseurs sont également indiqués.

2 Programmation de l'algorithme

Les déclarations utilisées pour programmer l'algorithme de Dijkstra sont les suivantes:

```

const
  MAX_INT = ... {représente  $\infty$ }
  GraphSizeMax = 50;
type
  EdgeLengthType = -1 .. MAX_INT;
  NodeType = 1 .. GraphSizeMax;
  GraphType = array [NodeType, NodeType] of EdgeLengthType;
  LabelType = array [NodeType] of EdgeLengthType
  MarkedType = array [NodeType] of Boolean;
  PredType = array [NodeType] of NodeType;
var

```

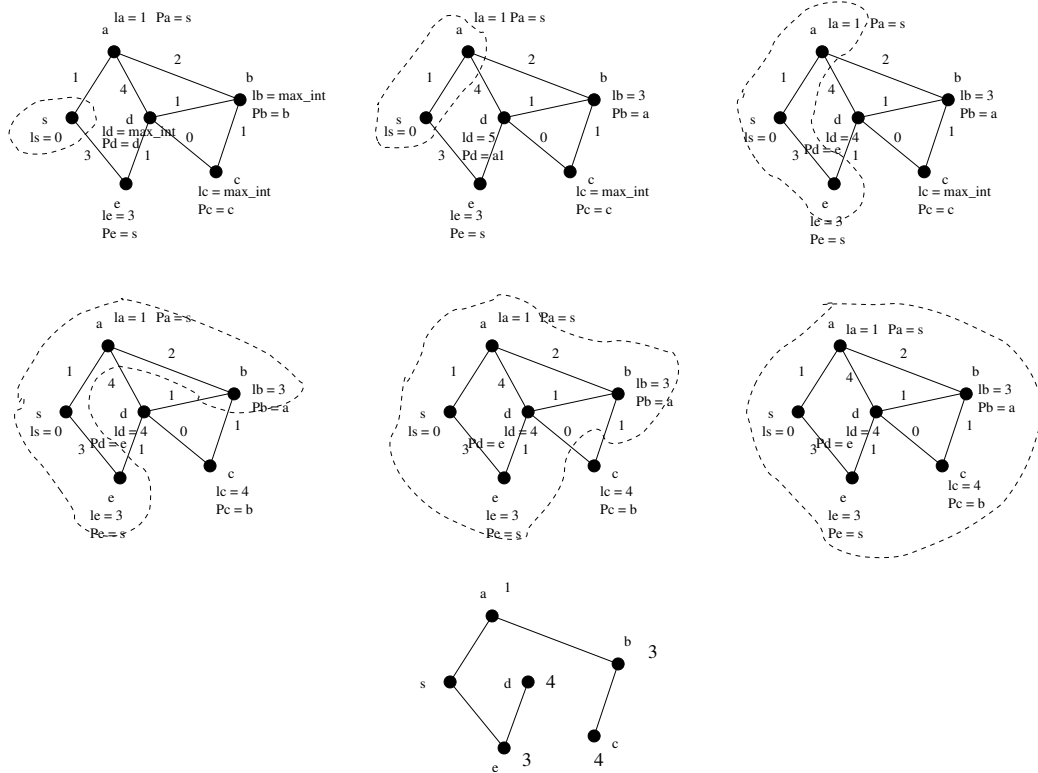


FIG. 1 – Exemple d'exécution de l'algorithme de Dijkstra

```

Graph : GraphType;      {matrice d'adjacence}
GraphSize : NodeType;   {nombre de sommets du graphe}
Pred : PredType;        {tableau indiquant le prédécesseur d'un sommet}
Marked : MarkedType;    {tableau indiquant si un sommet est marqué}
Vlabel : LabelType;     {tableau contenant l'étiquette d'un sommet}
Source : NodeType;
    
```

a. Les déclarations

Représenter par des schémas sur papier les variables déclarées précédemment, en tenant bien compte de leurs types.

b. Lire et construire un graphe

Écrire la procédure

```

procedure ReadGraph(var G : GraphType; var gs : NodeType);
    
```

qui initialise la taille gs du graphe traité et sa matrice d'adjacence avec les longueurs d'arêtes données au clavier par l'utilisateur. Si deux sommets x et y ne sont pas reliés par une arête, on considère que la longueur l_{xy} est égale à -1 . C'est la valeur qui sera placée dans la matrice d'adjacence.

c. Afficher un graphe

Écrire la procédure

```

procedure WriteGraph(var G : GraphType; var gs : NodeType);
    
```

pour afficher la matrice d'adjacence du graphe rentré par l'utilisateur.

d. Initialisation des tables de l'Algorithme

Écrire une fonction qui renvoie la valeur initiale de l'étiquette du sommet x (0 si $x = s, \infty$ sinon). Le sommet s est la source à partir de laquelle on calcule les plus courts chemins.

```
function InitLabelSP(s : NodeType; x : NodeType) : EdgeLengthType;
```

Écrire la procédure `InitDijkstra` qui prend en particulier une fonction ayant deux paramètres de type `NodeType`.

```
procedure InitDijkstra(var P : PredType; var M : MarkedType; var L :
LabelType; var s : NodeType; gs : NodeType; function
    InitLabelOp(s : NodeType; x :
    NodeType) : EdgeLengthType);
```

Cette procédure demande à l'utilisateur la source à utiliser dans l'algorithme de Dijkstra, initialise les tableaux des prédécesseurs (prédécesseur de $x = x$) et des sommets marqués et utilise le résultat de `InitLabelOp` pour initialiser les étiquettes.

e. Recherche du prochain sommet à marquer

Écrire les fonctions

```
function InitBestLabSP : EdgeLengthType;
function SelectionSP(lx : EdgeLengthType; li : EdgeLengthType) : Boolean;
function NodeSelection(M : MarkedType; L : LabelType; gs : NodeType; s : NodeType;
    function SelectionOp(lx : EdgeLengthType;
        Li : EdgeLengthType) :
        Boolean;
    function InitBestLabOp : EdgeLengthType)
    : NodeType;
```

La fonction `InitBestLabSP` renvoie toujours `MAX_INT`. Etant donné une matrice d'adjacence, un tableau des sommets marqués et un tableau d'étiquettes, `NodeSelection` renvoie le prochain sommet à marquer parmi les non-marqués, en utilisant `SelectionOp`, qui renvoie `True` si $lx < li$ et `False` sinon. S'il n'est possible de marquer aucun nouveau sommet, `NodeSelection` renvoie `s`.

f. Mise à jour des voisins du dernier sommet marqué

Écrire la fonction

```
function UpdateSP(lbx : EdgeLengthType; lbi : EdgeLengthType; lxi : EdgeLengthType);
```

qui calcule à partir de l'étiquette `lbx` du sommet marqué en dernier `x`, l'étiquette `lbi` d'un de ses voisins non-marqués `i` et de la longueur de l'arête entre ces deux sommets `lxi`, la valeur de la nouvelle étiquette du sommet `i` lorsqu'on calcule des plus courts chemins avec Dijkstra.

Écrire la procédure

```
procedure Update(G : GraphType; var M : MarkedType; var P : PredType;
    x : NodeType; gs : NodeType; var L : LabelType;
    function UpdateOp(lbx : EdgeLengthType; lbi : EdgeLengthType;
        lxi : EdgeLengthType) : EdgeLengthType);
```

qui marque le sommet `x`, et met à jour les étiquettes et les prédécesseurs de tous ses voisins non-marqués suivant la valeur rendue par la fonction `UpdateOp`.

g. Dijkstra

Écrire la procédure

```
procedure DijkstraSP(var G : GraphType; var M : MarkedType; var P :
    PredType; var L : LabelType; gs : NodeType; s : NodeType);
```

qui calcule les plus courts chemins sur le graphe `G` à partir de la source `s`.

Écrire la procédure

```
procedure WriteSol(P : PredType; L : LabelType; gs : NodeType);
```

pour afficher les chemins calculés dans `P` et pouvoir ainsi tester la procédure précédente.

3 Pour aller plus loin : Chemin de capacité maximum

En considérant maintenant que les longueurs données aux arêtes sont en fait des capacités de canalisations installées entre les sommets du graphe, on souhaite trouver un canal de capacité maximale entre un sommet source et tous les autres sommets du graphe. Pour cela quelques modifications par rapport au plus court chemin sont nécessaires :

- Initialisation : les étiquettes utilisées dans les deux versions de l'algorithme ne représentent pas la même grandeur, et ne sont pas utilisées de la même façon, leurs initialisations sont donc différentes. Si, pour le plus court chemin, l'étiquette initiale de la source est nulle, elle doit être de `MAX_INT` pour le chemin de capacité maximale, et inversement pour les autres sommets.
- Choix du sommet à marquer : pour calculer le chemin de capacité maximale on doit trouver le sommet qui a la plus grande étiquette et qui n'est pas encore marqué.
- Mise à jour :

$$Vlabel[i] = \max \{ \min\{Vlabel[x], l_{xi}\}, Vlabel[i] \}$$

Le sommet x devient le prédécesseur de i si la capacité du canal arrivant en i depuis x est plus grande que $Vlabel[i]$.

Pour procéder à ces changements il suffit de réécrire les fonctions passées en paramètre aux procédures du premier exercice et d'exécuter le même algorithme.

h. Initialisation pour LP

Écrire la fonction suivante qui renvoie la valeur initiale de l'étiquette pour le sommet x .

```
function InitLabelLP(s : NodeType; x : NodeType) : EdgeLengthType;
```

i. Recherche du prochain sommet à marquer

Écrire les fonctions

```
function InitBestLabLP : EdgeLengthType;
function SelectionLP(lx : EdgeLengthType; li : EdgeLengthType) : Boolean;
```

`InitBestLabLP` renvoie toujours 0. `SelectionLP` renvoie **True** si $lx > li$ et **False** sinon.

j. Mise à jour : Largest Path

Écrire la fonction

```
function UpdateLP(lbx : EdgeLengthType; lbi : EdgeLengthType;
                 lxi : EdgeLengthType);
```

qui calcule à partir de l'étiquette lb_x du sommet marqué en dernier x , l'étiquette lb_i d'un de ses voisins non-marqués i et de la capacité de l'arête entre ces deux sommets l_{xi} , la valeur de la nouvelle étiquette du sommet i .

k. Dijkstra

Écrire la procédure

```
procedure DijkstraLP(var G : GraphType; var M : MarkedType; var P :
                    PredType; gs : NodeType; s : NodeType);
```

qui calcule les chemins de capacité maximale sur le graphe G à partir de la source s . Tester cette procédure à l'aide de `WriteSol`.