

Représentation de graphe orienté valué – Algorithme de Dijkstra

A effectuer en binôme.

But du programme

- Lire un graphe orienté et valué dans un fichier ; le stocker dans une structure de données de votre choix.
- Dérouler l'algorithme de Dijkstra pour rechercher les chemins les plus courts partant d'un sommet donné par l'utilisateur de votre programme.
A chaque itération de l'algorithme, votre programme doit afficher, pour chaque sommet du graphe, la valeur provisoire de π (valeur du chemin) et celle de son prédécesseur dans le chemin ayant cette valeur.
- Afficher les chemins les plus courts pour chaque sommet du graphe.

Exemple de fichier de données contenant le graphe et lu par votre programme :

Notez bien que ceci n'est qu'un exemple. Vous n'avez pas obligation de faire la même chose...

7	Dans cet exemple :
10	
0 1 14	
0 4 5	- La première ligne contient le nombre de sommets.
1 3 6	- La seconde le nombre d'arcs.
2 3 8	
4 5 3	
3 0 2	- Chacune des lignes suivantes ont la forme :
5 6 2	extrémité_initiale extrémité_terminale valeur_de_l_arc
5 2 11	
6 1 9	- Le fichier est terminé par -1.
6 4 7	
-1	

Exemple de trace d'exécution :

```
Recherche des chemins
Graphe : G01
Sommet de départ : 0
Colonne de gauche : sommets transférés de M dans CC
('###,##' représente une longueur infinie et un predecesseur inconnus)
    0      1      2      3      4      5      6
    0 ,0 | 14 ,0 | ###,## | ###,## | 5 ,0 | ###,## | ###,## |
4 : 0 ,0 | 14 ,0 | ###,## | ###,## | 5 ,0 | 8 ,4 | ###,## |
5 : 0 ,0 | 14 ,0 | 19 ,5 | ###,## | 5 ,0 | 8 ,4 | 10 ,5 |
6 : 0 ,0 | 14 ,0 | 19 ,5 | ###,## | 5 ,0 | 8 ,4 | 10 ,5 |
1 : 0 ,0 | 14 ,0 | 19 ,5 | 20 ,1 | 5 ,0 | 8 ,4 | 10 ,5 |
2 : 0 ,0 | 14 ,0 | 19 ,5 | 20 ,1 | 5 ,0 | 8 ,4 | 10 ,5 |
3 : 0 ,0 | 14 ,0 | 19 ,5 | 20 ,1 | 5 ,0 | 8 ,4 | 10 ,5 |
    0 ,0 | 14 ,0 | 19 ,5 | 20 ,1 | 5 ,0 | 8 ,4 | 10 ,5 |
Plus courts chemins : longueur
0 : 0
0 - 1 : 14
0 - 4 - 5 - 2 : 19
0 - 1 - 3 : 20
0 - 4 : 5
0 - 4 - 5 : 8
0 - 4 - 5 - 6 : 10
```

Graphes à prendre en compte

Graphes orientés valeurs.

Valeurs : nombres entiers quelconques.

Sommets : numérotés par des nombres entiers positifs, à partir de 0, sans rupture de séquence.

Langage de programmation

C

C++ sans utilisation des constructeurs de type class, map, list, ...

Rendu du travail

A rendre en fin de séance, quelque soit le résultat obtenu.

Envoi par email à l'adresse fournie en séance de TP.

Graphes de test et sommets de départ fournis en séance de TP.

Sujet du mail : EFREI – L3 – TG – TP1 – *nom1 nom2*

Corps du message : vide

Pièces jointes :

Un fichier au format .txt contenant les traces d'exécution de votre programme.

Autant de fichier de code source (.c, .cpp, .h) que nécessaire.

Pas d'archive. Pas de fichier binaire. Pas de « fichier projet » géré par votre outil de développement.

Tous les fichiers doivent avoir un nom préfixé par les noms des deux binômes.

Tout manquement à ces consignes sera pénalisé.

Éléments de notation

Seront pris en compte :

- Votre code lui-même : choix des structures de données (représentation du graphe et déroulement de l'algorithme de Bellman) et mise en œuvre de l'algorithme.
- Son adaptabilité à d'autres graphes que ceux demandés en traces d'exécution.
- Les commentaires, et plus généralement la clarté de votre code.
- Les traces fournies.

Toute récupération évidente, même partielle, de code fait par un autre groupe se verra sanctionnée par un 0/20 à tous les étudiants concernés.

Annexe – Représentation de graphe valué

Les indications ci-dessous ne sont que des suggestions. Vous pouvez bien entendu adopter d'autres structures de données avec lesquelles vous vous sentiriez plus à l'aise.

Si votre SdD est dérivée d'une matrice d'adjacence « sommet x sommet » :

```
bool ** matAdj ; // ou 'int ** matAdj' en C car pas de type bool
```

le plus simple est alors d'avoir un second tableau de même taille contenant les valeurs :

```
int ** valAdj ;
```

Lorsque `matAdj[x][y]` est 'vrai', alors la valeur de l'arc est contenu en `valAdj[x][y]`.

Si votre SdD est dérivée d'une matrice d'incidence « sommet x arc » :

```
int ** matInc ;
```

le plus simple est alors d'avoir un second tableau où chaque case contient la valeur d'un arc :

```
int * valInc ;
```

Lorsque `matInc[x][a]` et `matInc[y][a]` valent respectivement '1' et '-1', alors l'arc (x,y) a sa valeur contenue dans `valInc[a]`.

Si votre SdD représente des listes (chaînées) de successeurs, alors le plus simple est d'ajouter la valeur de l'arc au moyen d'un champ `int val` dans la structure ('struct' élément de liste des successeurs) associée à chaque successeur (extrémité terminale d'un arc).

Attention aux limites associées aux matrices d'adjacence (cas de «graphe ayant plus d'un arc entre 2 même sommets») et d'incidence (cas de boucle)...

Annexe – Lecture de fichier et sauvegarde dans une structure de données / C++

Le code ci-dessous est essentiellement un exemple de lecture dans un fichier en utilisant le type « ifstream » (input file stream) ainsi que l'instruction « >> » dont l'utilisation est ici similaire à ce qui est fait avec l'instruction « cin » de saisie au clavier.

Vous pouvez bien entendu faire différemment !

Le code nécessite tout d'abord la déclaration de la bibliothèque suivante :

```
#include <fstream>
```

Compte-tenu de la structure de fichier proposée précédemment, le code suivant (C++) vous propose une trame de mécanisme de lecture. A vous de l'intégrer dans votre programme, et de le compléter.

Bien entendu, si vous utilisez une structure de fichier différente, ce code devra être adapté.

Ce code ne fait pas de vérification sur la structure du fichier, ni même sur son existence. Ne vous attardez pas à cela car ce n'est pas l'objet du TP : supposez que le fichier a une structure correcte !

```
// Variables locales
ifstream sourceGraphe ( "G24.txt" ) ;
    // Déclaration d'une variable de type "flux de données"
    // associée au fichier « G24.txt »
int nbSommets ;                // Nombre de sommets du graphe
int nbArcs ;                   // Nombre d'arcs du graphe
int extremiteInitiale ;        // Arcs du graphe
int extremiteTerminale ;
int valeurArc ;

// Lecture nombre de sommets et nombre d'arcs
sourceGraphe >> nbSommets ;
cout << "Nombre de sommets : " << nbSommets << "\n" ;
sourceGraphe >> nbArcs ;
cout << " Nombre d'arcs : " << nbArcs << "\n" ;
    // Utilisation similaire à un « cin » :
    // lecture dans le fichier de la première donnée de type int
    // (cf déclaration de nbSommets) et stockage de la valeur
    // dans la variable nbSommets ; puis lecture de la seconde
    // et stockage dans nbArcs
```

*... ajoutez ici votre code pour stocker cette valeur, si nécessaire,
...et éventuellement pour créer et initialiser vos SdD représentant le graphe...*

```
// Lecture des arcs du graphe
cout << "Arcs du graphe:\n" ;
sourceGraphe >> extremiteInitiale ;
while ( extremiteInitiale != -1 ) {
    sourceGraphe >> extremiteTerminale ;
    sourceGraphe >> valeurArc ;
    cout << extremiteInitiale << " -> " << extremiteTerminale
        << " = " << valeurArc << "\n" ;
```

... ajoutez ici votre code pour stocker cet arc dans vos SdD...

```
    sourceGraphe >> extremiteInitiale ;
}
```