INFO-F-203 - Algorithmique 2
CPS - GPS pour les étudiants

Maxime Desclefs et Simon Picard

21 Décembre 2012

Table des matières

Chapitre 1

Introduction

Dans le cadre du cours d'algorithmique de 2e bachelier informatique, nous avons été amené à réaliser un projet visant à mettre en application les concepts étudiés. Le programme est intégralment codé en langage Java et est exclusivement accessible en ligne de commande.

Cette application est en somme un GPS qui permet, à partir d'un Cercle donné, de lister les trajets les plus rapides vers les autres Cercles ainsi qu'un chemin passant par chacun des Cercles. Ce programme est prévu pour fonctionner sur un smartphone.

Dans la suite de ce document, seront expliqué les différents algorithmes, problèmes rencontrés et leurs résolutions avec quelque illustrations en pseudo-code.

Chapitre 2

Utilisation

2.1 Exécution

Pour fonctionner, cette application reçoit en paramètre des informations permetant la création d'un graphe et un Cercle de départ. Le graphe doit être représenté par un fichier txt où :

- La première ligne contient le nombre de point du graphe, dans notre cas, de Cercle
- Les suivantes indiquent le nom des différents Cercles
- Et enfin, les relations entre chacun d'eux sous la forme "CercleDeDépart" "CercleDArrivé" "Temps"

Il n'est pas nécéssaire de fournir toutes les relations pour lancer le programme mais les résultats seront manquant ou moins performant.

```
7
CI
CP
CM
CD
CK
...
CI CM 4
CI CK 16
CI CP 4
CP ISEP 2
CP CM 1
...
```

Ci-dessus, un exemple de fichier de description d'un graphe

2.2 Résultat

Voici un exemple d'exécution de l'application :

```
>./main graph.txt CI

Plus courts chemins :
CI => CP : 4.0 min
CI => CM : 4.0 min
CI => CD : 1.0 min
CI => CM => CK : 8.0 min
CI => CP => ISEP : 6.0 min
CI => CIG : 8.0 min
Chemin passant par tous les cercles :
CI => CD => CM => CK => CIG => CP => ISEP : 20.0 min
```

Chapitre 3

Parser

3.1 Fonctionnement

Pour pouvoir récupérer les informations se trouvant dans le fichier texte, il a fallut implémenter une classe. Celle-ci fonctionne de la manière suivante :

- L'ajout du contenu du fichier dans le buffer
- La lecture du buffer
- La fermeture du buffer

3.1.1 Création du buffer

Par soucis de rapidité, l'utilisation d'un *buffer* était la solution la plus adéquate car la mise en mémoire cache permet de ne plus devoir accéder à la mémoire de masse et donc d'éviter des trajets longs et inutiles vers celle-ci.

```
FileInputStream input = new FileInputStream(new File(fileName));
InputStreamReader inputReader = new InputStreamReader(input);
m_buffer = new BufferedReader(inputReader);
```

Le fichier est d'abord ouvert et lu à partir des fonctions **File** et **FileInputStream**. La fonction **FileInputStream** représente le fichier par *octet* et non pas par *caractères*. Il faut donc utiliser la fonction **InputStreamReader** pour effectuer la traduction. Et enfin introduire cette traduction dans le *buffer*.

3.1.2 Lecture du buffer

Puisque le *buffer* fourni la méthode **readLine**, nous pouvons parcourir le contenu du fichier facilement. Etant donné que la méthode **readLine** renvoit **null** lors ce qu'on arrive à la fin du fichier, on peut simplement l'utiliser dans une boucle **while**.

3.1.3 Fermeture du buffer

Pour pouvoir libérer la mémoire, il est impératif de fermer le buffer à l'aide de la méthode **close**.

3.2 Utilisation

3.2.1 Création d'une liste

Pour pouvoir identifier chaque Cercle qui se retrouveront sous forme d'indice dans le *graphe*, leurs noms ont été placé dans une *liste* **Vector** où l'indice sera le même aussi bien dans la *liste* que dans le *graphe*.

3.2.2 Création du graphe

La réalisation du *graphe* sous forme de *matrice* se révelait être la solution la plus adéquate dans notre cas. Premièrement, il fallait initialiser une *matrice* carré vide de la taille des éléments du *graphe* où la valeur [i][j] correspond au temps qu'il faut pour aller du Cercle i au Cercle j. Si cette information est inexistante, sa valeur vaudra 0. Ensuite, une mise à jour de la *matrice* est effectuer pour placer les valeurs correctes selon le fichier texte que la fonction a reçue.

Chapitre 4

Path

La classe Path, après avoir reçue les informations parsées en paramètres, recherche les différents trajets possible entre chaque Cercle à partir du Cercle de départ ainsi que le chemin le plus court aproximativement parcourant exactement une fois chaque cercle.
Pour résoudre ces deux problèmes, la mise en place de l'algorithme de Dijkstra fut nécéssaire.

4.1 Dijsktra

Pour réaliser le calcul des plus courts chemins à partir d'un sommet initial, nous pouvons modifier l'algorithme de Prim en stockant dans un vecteur, que nous appellerons dist, la distance de chaque sommet avec le sommet initial x. Quand on rajoute un sommet k dans l'arbre en construction, on met à jour la frange en parcourant la liste de successeurs de ce sommet k. Pour chaque sommet t de cette liste de successeurs, la distance du sommet x aux sommets de cette liste est égale à dist[k] + distance de k à t.
La propriété fondamentale sur laquelle se base cet algorithme est que, dans un digraphe, le chemin le plus

court entre un sommet a et un sommet c est composé du plus court chemin entre le sommet a et un sommet b et du plus court chemin entre ce sommet b et le sommet c, et ce pour tout sommet b du digraphe se trouvant sur le plus court chemin entre les sommets a et c. L'algorithme de Dijkstra réalise cette recherche des plus courts chemins depuis un sommet initial s jusqu'à tous les autres sommets du digraphe. Pour réaliser cela, nous manipulons un ensemble M de sommets contenant initialement tous les sommets du graphe à l'exception du sommet initial s.

Nous manipulons aussi deux vecteurs, dist et prec, tels que pour chaque sommet i distinct du sommet s, la valeur contenue en dist[i] indique la plus courte distance connue entre le sommet s et le sommet i, et prec[i] reprend le numéro du sommet précédant le sommet i sur le plus court chemin en construction entre les sommets s et i. Initialement, dist[i] contient le poids présent sur l'arc entre le sommet s et le sommet i. Si cet arc n'existe pas, le poids est supposé égal à l'infini. Les entrées correspondantes du vecteur prec valent toutes initialement s, puisque nous ne considérons dans un premier temps que les chemins ne comportant qu'un seul arc entre le sommet s et tous les autres sommets. Par la suite, en découvrant de nouveaux arcs, nous essayons d'allonger, en nombre d'arcs, les chemins, en constatant éventuellement que nous améliorons, en terme de distance, la situation existante.

Pour ce faire, nous réalisons une boucle qui retire de l'ensemble M le sommet m dont la valeur correspondante dans le vecteur dist, à savoir dist[m], est la plus petite. Pour chaque sommet y successeur de ce sommet m, nous regardons s'il appartient à l'ensemble M. Si le sommet y n'y appartient pas, cela signifie qu'il est déjà traité. Sinon, nous mettons à jour les vecteurs dist et prec si, en passant par le sommet m, le sommet y est plus proche du sommet initial s. Si dist[m] valait l'infini lors du retrait du minimum de l'ensemble M, cela signifie que le sommet m appartient à une autre composante que le sommet s. Cela sera alors aussi le cas pour tous les autres sommets potentiellement encore présents dans M. Dans cette situation, la recherche des plus courts chemins depuis le sommet s peut s'arrêter. Ces sommets encore présents dans l'ensemble M sont à une distance infinie du sommet s, valeur déjà présente pour ces sommets dans le vecteur dist. [1]

4.2 Solution récurssive

L'algorithme utilise la méthode du backtracking. En partant d'un point de départ, la fonction établie une liste de Cercle vers lesquelles le Cercle de départ peut se rendre. Pour chacun de ces Cercles, par ordre de distance croissante, nous rappelons la fonction en indiquant que nous somme passé par ce cercle. Lorsque la fonction aura été appelé récursivement autant de fois qu'il n'y a de cercle en tout, une solution a été trouvée. Par soucis d'optimisation, la solution recherchée n'est pas forcément la meilleur, mais la première trouvée.

Chapitre 5

Conclusion

Les algorithmes permettant de rechercher les trajets aussi bien, vers chaque Cercle, que celui passant par tous les Cercles ont été réalisé avec succès.

Pour satisfaire l'énoncé le plus possible, un fichier exécutable bash nommé projet pourra être lancé via la ligne de commande ./projet suivi du fichier représentant le graph et le Cercle de départ.

1. Source : Bernard Fortz, INFO-F-203 : Algorithmique 2, sylabus de cours, page 73