CSI2510 DEVOIR #4

Ivor Benderavage (8171462)
Hussein Hegazy (7934471)

Introduction

Ce devoir porte sur la théorie des graphes et les algorithmes de parcours des graphes; en particulier, sur l'algorithme de recherche par profondeur, l'algorithme de recherche par grandeur, et l'algorithme de Dijkstra.

Le devoir consiste en la création d'un programme en Java qui, étant donné un graphe comportant 376 sommets et 933 arêtes, peut effectuer plusieurs différent parcours sur le graphe. Plus spécifiquement, le graphe représente une carte du métro Parisien, et l'énoncé du problème est d'écrire un programme qui comporte les fonctions suivantes :

i) Étant donné un entier comme paramètre, imprimer tous les stations qui se trouvent sur la même ligne que la station représentée par le paramètre. Les stations doivent êtres imprimées dans le bon ordre, d'une extrémité à l'autre.
ii) Étant donné deux entiers comme paramètres, trouver le parcours le plus rapide entre ces deux stations.
iii) Étant donné trois entiers comme paramètres, trouver le parcours le plus rapide entre les stations représentées par les premiers deux paramètres, en considérant que la ligne sur lequel se trouve la station représentée par le troisième paramètre est fermé.

Notre implémentation des fonctions

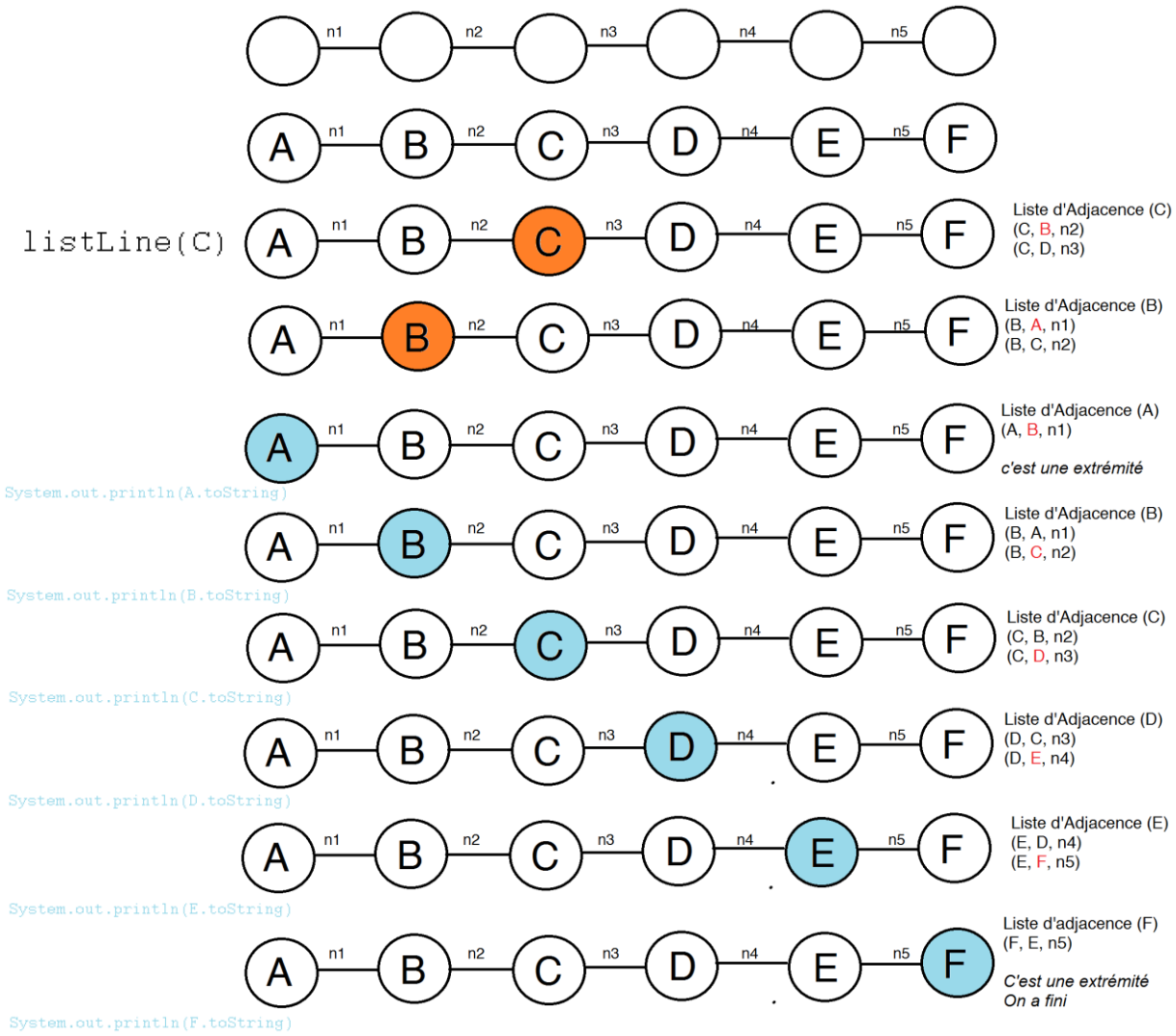
D'abord, nous avons choisi de créer nos propres classes pour chaque sommet et chaque arête. Nous avons implémentés l'objet sommet avec la classe **Station**, et l'objet arête avec la classe **Connection.** Ensuite, nous avons choisi de représenter la liste d'adjacence du graphe à l'aide d'un tableau contenant des LinkedList<Connection>. Ceci nous a permis de faire référence aux stations dans la liste d'adjacence simplement en regardant les indices du tableau. En conséquence de ceci, nous avons aussi implémenté un tableau contenant tous les Stations, afin de pouvoir faire référence aux noms des stations.

Partie I

Dans cette partie, nous avons trouvés que la meilleure manière d'imprimer tous les stations sur une ligne était de se rendre à l'extrémité de la ligne, et simplement itérer à travers la ligne. Puisque chaque objet *Connection* comporte les attributs *station1, station2,* et *traversalTime*, ceci était simplement le travail de visiter chaque *Station* indiquée par la valeur *station2* des arêtes d'une station donnée, à moins que cette station ne soit déjà visitée, ou que son attribut *TraversalTime* n'était pas égal à la valeur -1 (indiquant une station sur une autre ligne, ce que nous désirions éviter à cette étape).

Si la station donnée comme paramètre n'est *pas* une station d'extrémité, le programme itère à travers les stations dans une direction aléatoire sans imprimer les stations jusqu'au point que le programme aboutit à une station d'extrémité (c'est-à-dire, une station dont la liste d'adjacence ne comporte qu'une seule arête). Ceci fait, le programme commence à itérer à travers la ligne, en utilisant la méthode décrite plus tôt, tout en imprimant les stations traversées.

Le schéma suivant représente l'algorithme utilisé par nous dans la partie I. Ce schéma représente le cas où nous avons une ligne consistant des stations A-B-C-D-E-F, et que l'utilisateur passe comme paramètre la station C.



Le programme imprime:

```
A
B
C
D
E
F
```

Partie II

Cette partie nécessite une implémentation de l'algorithme de Dijkstra afin de pouvoir trouver le chemin le plus rapide entre deux stations données, ainsi que le temps requis pour faire ceci. Nous supposons dans cette partie que le temps requis pour changer d'une ligne vers l'autre (symbolisée dans le graphe comme un arête avec un poids de -1) est de 90 secondes.

Partie III

Cette partie consiste en une combinaison des parties I et II. Il est encore nécessaire d'utiliser l'algorithme de Dijkstra ici afin de trouver le chemin le plus court entre deux stations, mais d'abord, il est nécessaire d'utiliser l'algorithme de la partie I pour identifier la ligne sur lequel se trouve la troisième station, et de la « fermer ». « Fermer » une ligne consiste à faire une itération à travers la ligne, et de changer l'attribut `disabled` de chaque station à `true`.

Sources

Tout le code utilisé dans le cadre de ce devoir a été écrit par nous, Ivor Benderavage et Hussein Hegazy. Le devoir n'utilise en aucune partie du code obtenu des sources externes, sauf pour notre utilisation des libraires `java.util` et `java.io`.