Mission 6 : Rapport Final

Groupe 2.2

Vendredi 12 Décembre 2014

Introduction

Pour cette mission, il nous a été demandé de pouvoir établir la liste de connections aériennes à inclure dans l'offre d'une compagnie low cost. L'objectif était donc de pouvoir desservir un nombre maximum de destinations à moindre coût à partir d'une liste de coûts d'ores et déjà définis entre deux villes.

Description de l'implémentation

Nous avons divisé le programme en deux classes : Graphe et SpanningTree. La première classe va nous servir à créer le Graphe à partir d'une liste de coûts entre deux villes présentée dans le format suivant :

0	37	398
1	39	929
	...	

Nous avons décidé de définir deux constructeurs pour la classe Graphe. Ceux-ci fonctionnent exactement de la même manière à la différence que le premier calcule la taille de la liste alors que le second suppose que cette information lui est déjà fournie. Le fait d'avoir connaissance ou non de la taille de la liste peut réduire considérablement la complexité temporelle (de n fois précisément) ce qui justifie l'existence de deux constructeurs. Ces derniers vont donc prendre en argument un String filename représentant un fichier qui contient une liste comme expliquée ci-dessus et en fonction du constructeur, il y aura un second argument de type int qui représente la taille de cette liste.

Après avoir reçu ou calculé la taille de la liste, l'algorithme va se poursuivre en créant une *ArrayList* de taille n contenant à chaque indice une liste d'arêtes *Edge*. Il est important de préciser que *Edge* est une sous classe de Graphe. Celle-ci va nous permettre de représenter une arête composée d'un coût, d'une origine et d'une destination. Cette sous classe est accompagnée d'une méthode toString et d'une méthode compareTo pour pouvoir comparer deux arêtes sur base de leur coût respectif.

La seconde classe, SpanningTree, ne contient qu'une méthode main qui va réaliser tous les calculs de plus courts chemins ; c'est elle qui va nous fournir la liste de connections aériennes à inclure dans l'offre de la compagnie low cost. Pour ce faire, la méthode va tout d'abord se charger de créer un Graphe par le biais d'un nom de fichier indiqué en argument. Ensuite, elle va créer une *PriorityQueue* qui contiendra toutes les arêtes du graphe classé en fonction de leurs coûts respectifs. De plus nous allons utiliser un tableau de *boolean* pour savoir quel noeud a déjà été visité lors de l'application de l'algorithme.

Dans un premier temps l'agorithme va ajouter à la *PriorityQueue* toutes les arêtes du premier noeud du graphe. Ensuite nous allons récupérer la première arêtes de la *PriorityQueue*, c'est-à-dire celle avec le plus petit coûts, et vérifier si soit l'origine ou la destination de cette arrêtes n'est pas encore été visité, si ce n'est pas le cas on ajoute l'arrêtes à la liste de solution, on note que l'origine ou la destination de cette arêtes a été visité et on rajoute dans la *PriorityQueue* toutes les arrêtes du noeud d'origine ou de destination de cette arêtes. Et ainsi de suite jusqu'à ce que la *PriorityQueue* soit vide.

Question 1 : Solution

Question 2 : Complexité

Question 3 : Adaptation

1. On prend un nœud au hasard comme racine.
2. On prend un autre nœud au hasard.
3. On calcule un Dijkstra modifié entre ces deux nœuds et on sélectionne le meilleur chemin.
4. On ajoute tous les nœuds sur ce chemin à l'ensemble des nœuds connectés.
5. On prend un autre noeud au hasard, on fait Dijkstra pour calculer sa distance à tous les nœuds.
6. On ajoute les nœuds et arête sur le chemin le plus court vers un nœud déjà relié au graphe des nœuds connectés.

Tant que tous les nœuds ne sont pas reliés, on reprend à l'étape 5.

N.B. : La solution exacte du problème semble très compliquée et en tous cas ne peut sans doute pas être trouvé avec une *greedymethod*.

Conclusion