Projet de modélisation du métropolitain

Aubier Jules
10821

Rossi Alessandro
10896

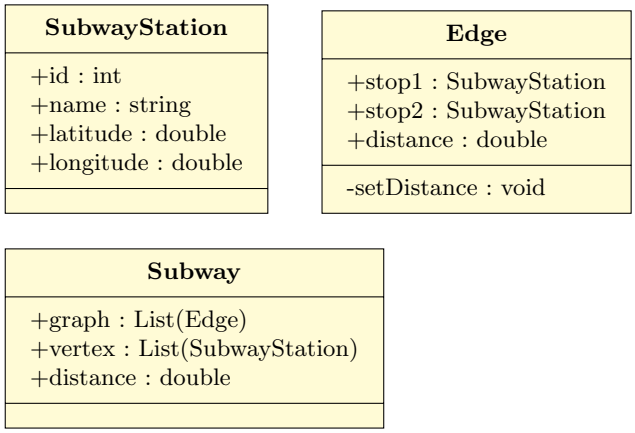
9 juin 2019

Table des matières

Introduction

L'objectif de ce projet est de réaliser une application capable de donner le plus court ou le plus long chemin entre deux stations de métropolitain.Le second objectif est d'identifier les différents clusters du graphe pondéré.

Pour réaliser cet objectif nous allons modéliser le réseau parisien sous la forme d'un graphe non pondéré en premier lieu puis pondéré avec les distances entre les stations. Pour cela nous utilisons les données fournies par la RATP(`https://data.ratp.fr/explore/`) sur le réseau souterrain uniquement.
Ci-dessous la structure du graphe.



Dans la réalisation de ce projet, nous utiliserons java 8 et particulièrement l'API stream. Un stream est une séquence d'éléments sur laquelle on peut exécuter des opérations de manière séquentielle ou parallèle. Cette API permet d'une part de simplifier la syntaxe du code et de le rendre plus lisible. La grande force de cette API reste tout de même la parallélisation des streams permettant une exécution plus rapide du code mais à utiliser avec caution.

Graphe non pondéré

Dans cette partie, nous avons modélisé le métropolitain parisien comme un graphe non pondéré ou les stations sont les nœuds et les liens entre les stations les arêtes.

0.1 Chemin le plus court

Pour obtenir le chemin le plus court dans un graphe non pondéré nous allons utiliser le BFS (Breadth First Search). Il est évident que le graphe ne contient pas de boucle. Il n'y a pas de métro qui partent d'une station pour arriver à la même station. De plus, nous sommes dans un graphe non pondéré. Toutes les conditions d'utilisation du BFS sont remplies.

0.1.1 Algorithme pour le plus court chemin

L'objectif est de construire un arbre simple ayant pour racine une station de métro. Ainsi pour obtenir le chemin le plus rapide entre cette station racine et une autre station, il ne reste qu'à remonter l'arbre jusqu'à la source. Ci-dessous la description de l'algorithme.

L'algorithme implémente plusieurs méthodes qui sont :

- La méthode **getNeighbours(SubwayStation x)** qui permet de récupérer la liste des voisins directes d'une station et les lignes associées sous la forme d'une HashMap() `<SubwayStation,String>`.
- La méthode **BFS(SubwayStation x)** qui construit l'arbre à partir de la station racine x. Cette méthode renvoie une List `<Edge>` correspondant à l'arbre.
- La méthode **ShortestPath(SubwayStation a,SubwayStation b)** qui ne renvoie rien mais affiche le plus court chemin entre les stations a et b.

0.1.2 Description de BFS(SubwayStation x)

On initialise trois collections que sont
- List<Edge> **bfsTree** qui contiendra les arêtes de l'arbre construit après le BFS. Cette liste est initialisée comme vide.
- Map<SubwayStation,Boolean> **visited**. Cette map prends pour clé une station et pour valeur un booléen qui indique si ce nœud a déjà été visité.
- Map<SubwayStation,String> **voisins** qui contient les voisins obtenus par la fonction **getNeighbours(SubwayStation x)**.

On initialise également une queue<SubwayStation> qui contient initialement la station source.

Tant que cette queue n'est pas vide :
On initialise une station S en retirant la tête de la queue.
On update les voisins comme voisins de S.
Pour chaque voisin, on vérifie si il a été visité. Si le voisin a déjà été visité on passe au suivant.
Autrement on marque le voisin comme visité et on ajoute à **bfsTree** une nouvelle arête ayant pour source S et pour destination ce voisin.

0.2 Diamètre et rayon

Graphe pondéré

Dans cette partie, nous avons modélisé le métropolitain parisien comme un graphe pondéré.

0.3 Chemin le plus court

0.3.1 Algorithme pour le plus court chemin

Dans le cas d'un graphe pondéré ou les poids des arêtes ne sont pas tous équivalents le BFS n'est pas utilisable. Nous allons davantage envisager l'utilisation de l'algorithme de Djikistra.

Les conditions d'utilisation de Djikistra sont comme suit :
— Un graphe pondéré et acyclique
— Aucun poids d'arête négatif ou nul
— Les distances entre deux stations sont des réels

Dans notre étude de cas, les poids des arêtes sera la distance séparant deux stations de métro. Cette distance **strictement positive** est la distance euclidienne entre deux stations représentées comme des points dans un plan en 2D ayant pour coordonnées (longitude,latitude). Les arêtes ne prennent donc pas en compte la route des rails de métro.

Une amélioration possible de cette modélisation est de pondérer le graphe avec le temps entre chaque station (fourni par l'API RATP) plutôt qu'avec la distance euclidienne qui n'est pas représentative de la réalité.

0.3.2 Algorithme de Djikistra

Ce qui suit est un rappel du fonctionnement de l'algorithme de Djikistra. L'algorithme de Djikistra permet d'obtenir un arbre simple à partir d'un nœud source de telle manière que le chemin de ce nœud source à un autre nœud dest soit la distance la plus courte possible.

Cet algorithme est de complexité temporelle polynomiale.Pour n noeuds et a arcs, la complexité est en O(a+nlog(n)).

0.3.3 Implémentation

0.4 Diamètre et rayon

Clusters

Conclusion