Rapport du Projet
Metro

HADJ MAHFOUD Kenza et DJAROUN Mohamed

Le 06 mai 2020

SOMMAIRE :

1/ Présentation du projet :

Metro est un projet qui a pour but de permettre à un utilisateur de déterminer le chemin le plus court entre deux stations du métro parisien. Il est programmé avec java comme langage ainsi que Eclipse comme IDE.

2/ Structure du projet :

2.1/ src

2.1.1 package fastmetro :

2.1.1.1 classe Station :

Elle contient tout ce qui concerne une station ; c'est-à-dire le numéro de la Ligne à qui elle appartient, l'id qui l'identifie ainsi que le nom de la gare à qui elle appartient : on a une liste de couple des voisins des stations.

2.1.1.2 classe Gare :

Cette classe est une liste de stations qui peut contenir soit une ou plusieurs stations. La gare contient un nom, des coordonnées qui vont nous permettre de la dessiner sur la carte ainsi que l'id.

2.1.1.3 classe ReadFile :

Cette classe permet de lire les fichiers en utilisant la bibliothèque gson ainsi que de les stocker dans une liste de gares et de stations pour faciliter les manipulations des données (stations, id, time). Ces fichiers sont les suivants :

- Le fichier « parisGraphe.json » où nous avons récupéré « t » ; le temps entre deux stations en fonction de leurs id ; écrit sous cette forme [station1, station2, time].

- Le fichier « parisStation.json » où nous avons récupéré la liste des gares (liste de stations) ainsi que leurs id, leurs numéros de ligne ainsi que les coordonnées (x,y) qui vont nous permettre de dessiner ces gares sur notre carte.

2.1.1.4 classe StationPereTime :

Cette classe nous permet d'avoir le couple père-temps qu'on va utiliser pour Djikstra. Elle contient l'id du père ainsi que le temps.

2.1.1.5 classe DjikstraPath :

Cette classe implémente l'algorithme de djikstra qui permet de rechercher le plus court chemin entre deux stations. Pour cela nous avons utilisé des matrices : l'une traite la stations et le couple père/temps en utilisant la classe StationPereTime ce qui nous permet de trouver ces stations père par ordre successif (à cela s'ajoute des petites méthodes qui affichent l'itinéraire à prendre : station, ligne). La seconde représente le plus court chemin (il y aura une explication plus profonde en bas de ce rapport).

2.1.1.6 Machine :

Cette classe contient le nom qui sera le titre de notre interface ainsi que le chemin pour accéder à l'image « paris.gif ». Elle permet d'importer toutes les stations, d'initialiser la carte sur laquelle en affichera ces stations, et enfin d'utiliser djikstra.

2.1.1.7 Main :

C'est la classe principale de ce programme. Elle contient une instanciation de la classe machine ainsi que l'utilisation de toutes ces méthodes désignées dans sa description.

2.1.2 Package gui :

Ce package contient toutes les classes qui nous permettent d'avoir une interface graphique. Pour cela nous avons utilisé JFrame (bibliothèque graphique) ainsi que Maven (outil de gestion et d'automatisation de production des projets logiciels en java) et JPanel (conteneur élémentaire).

2.1.2.1 classe Circle :

Elle contient tout ce qui est nécessaire pour dessiner des cercles : la couleur, les caractéristiques d'un cercle ainsi que ses coordonnées.

- Cercle jaune : désigne les stations sélectionnées (départ, arrivée)
- Cercle rouge : désigne toutes les gares de notre carte
- Cercle noir : désigne le trajet à prendre et donc toutes les stations parcourues.

2.1.2.2 classe Fenêtre :

Comme son nom l'indique cette classe représente la fenêtre de notre interface graphique : elle hérite de JFrame .

2.1.2.3 classe ClickSelectStation :

Cette classe hérite de l'interface MouseListener qui va lui permettre de répondre aux évènements souris (gestion de la souris) et de gérer les cliques de l'utilisateur.

2.1.2.4 classe StationPanel :

Cette classe permet de dessiner les stations sur la carte et de changer leurs couleurs. Elle hérite de JPanel.

3/Algorithme de Djikstra :

3.1 Description :

Nous avons choisi Djikstra comme algorithme de calcul du plus court chemin, il utilise deux ensemble

- C L'ensemble des stations qui restent à visiter
- D Les stations pour lesquelles on connait déjà la valeur du plus court chemin
- L'algorithme se termine quand la station d'arrivée est dans D
- Algorithme :
    - ❖ Initialisation :
        - ➢ T = {s} ; d = 0 ;
        - ➢ ∀ i ≠ s, si l'arc (s,i) excite alors
            - ❖ di = w(s,i)(∞ sinon),
            - ❖ père(s) = i;

    - ❖ Boucle principale : Tant que T ≠ V faire
        - ➢ Trouver un nœud t de V-T tq
          dt = Min(di , i € V-T),
        - ➢ T = T U {t}
        - ➢ ∀ k € ┌+
            - ❖ Si (dk > dt + w(t,k)) alors
            - ❖ dk = dt + w(t,k), père(k) = t

3.2 Structure de données :

L'idée de l'algorithme est de se baser sur l'utilisation de Djikstra. La structure de données utilisée est une table d'hachage avec une clé qui correspond au sommet (id). On trouve d'abord les distances (valeurs) des plus courts chemins qu'on stocke dans l'une des matrices qu'on appellera matrice1. Quand on ajoute la station de départ (aussi appelée la station père) on la stocke dans la matrice2 où elle sera traitée : c'est-à-dire qu'on va chercher les voisins de la station père. Pour chaque voisin, s'il est déjà traité et qu'on avait trouvé un plus court chemin on le

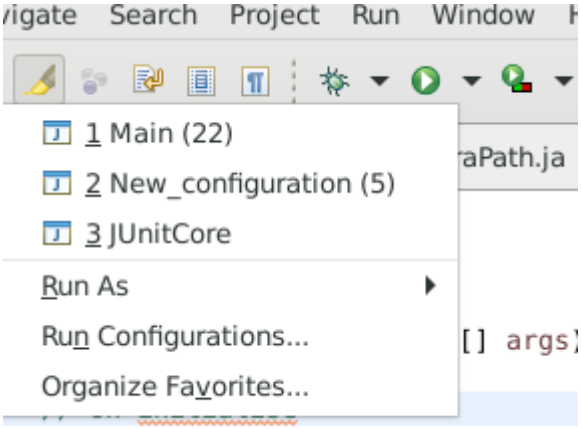
rajoute dans notre matrice. Dans le cas contraire, on ne fait rien et on passe au suivant.

L'autre étape consiste à remonter les stations en partant de la station d'arrivée (par ordre successif)

4/ Fonctionnement de la carte :

4.1 Exécution du projet :

Comme nous avons travaillé sur Eclipse nous avons compilé avec la console



4.2 Remarques à propos de la manipulation :

- Le mode d'emploi est juste à l'ouverture de l'interface du logiciel

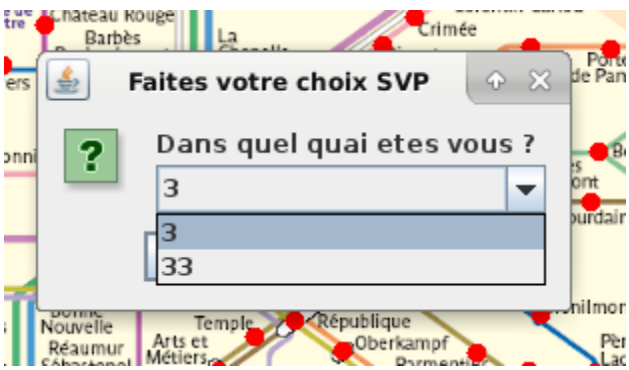
- Les lignes 3bis, 7bis, 13 bis sont désignées ainsi dans notre programme :

  3bis → 33

  7bis → 77

  13 → 1313

Exemple :



- Pour ce qui est de la direction de nos lignes de métro, le sens est indiqué par l'affichage des cercles noirs qui permettent d'orienter l'utilisateur.

- Le plus court chemin qu'on obtient dépend de la ligne de départ qu'on choisit

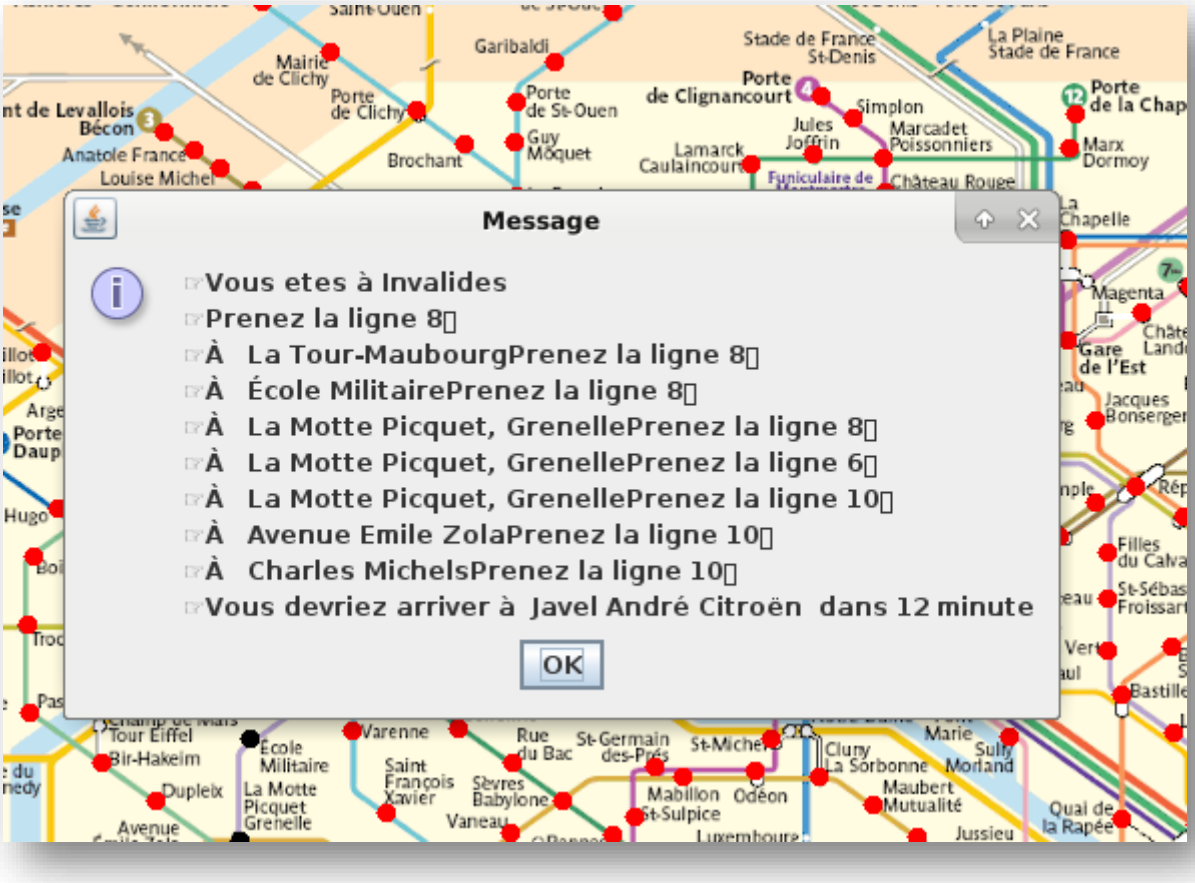
Exemple :

On prend l'exemple du trajet de :

Invalide ⟶ *javel André Citroën*

❖ **En prenant la ligne 13 :**

❖ En prenant la ligne 8 :



On constate qu'en prenant la ligne 13 le trajet dure 15 minutes par contre si on emprunte la ligne 8 la durée n'est que de 12 min : cela est dû au temps qu'on a pris en allant de la ligne 13 à la ligne 8.

Donc il y a plusieurs plus courts chemins différents, tous dépendent de la ligne que l'utilisateur a sélectionnée au départ.

Conclusion :

Ce projet nous a appris à utiliser l’algorithme de djikstra sur des problèmes complexes.

Il nous a également permis d’utiliser la théorie des graphes.