Compte rendu numéro 2 Joachim Veyrie

Nils Dufourmantelle

Partie Graphe : Plan du métro Parisien

Ce programme consistait à :

* Modéliser le métro parisien sous forme de graphe
* D’y associer des stations (nœuds) et des connexions (liens pondérés)
* D’effectuer des calculs de plus court chemin (en distance ou temps)
* De visualiser dynamiquement ce graphe avec les vraies positions GPS
* Et d’afficher le chemin optimal entre deux stations sélectionnées

Pour cela nous avons créé 5 classes :

La classe Noeud : elle représente une station de métro, identifiée par un ID, un nom, une liste de lignes, et sa position GPS (latitude, longitude).

La classe Lien : Elle représente une connexion entre deux stations, avec une pondération (distance ou temps de trajet).

La classe Graphe : C’est la classe qui permet de structurer l’ensemble du réseau contenant les nœuds, les liens, et une liste d’adjacence.

La classe Visualisation : C ‘est la classe qui va permettre la visualisation de ce graphe, ici elle s’appuie sur la bibliothèque graphique SkiaSharp afin dessiner le graphe sous forme d’image png.

Le programme.cs : Il y’a le Main du projet mais également la méthode créationgraphe qui vont permettre de charger les données des fichiers Excel afin d’instancier le graphe, qui va exécuter les algorithmes de plus courts chemins et générer la visualisation.

Sources de données :

Pour les données extraites dans ce programme j’ai décidé de diviser l’excel fourni à la base en deux Excel différents :

* Liens\_corrige.xlsx
* MetroParis.xlsx

Dans MetroParis.xlsx j’ai renseigné pour chaque station leur longitude ainsi que leur latitude afin de pouvoir ensuite les placées sur le graphe, j’ai également renseigné les différentes lignes de métro qui passent par ces stations, ainsi dans le main lorsque que l’on voudra construire les coordonnées pour une station définie (id,nom,latitude,longitude) le programme ira reconnaître en fonction de l’id de la station sa latitude et sa longitude.

Dans Liens\_corrige j’ai renseigné l’id de chaque station, (unique même si une station peut apparaître sur plusieurs lignes), son nom, sa ligne et également la distance en km l’éloignant de la station prochaine. Pour cela j’ai utilisé la formule de Haversine et les différentes latitudes et longitudes de chaque station.

Ainsi ces distances permettront la pondération de mon graphe

C’est également dans ce fichier que l’algorithme ira chercher l’ensembles des noms et identifiants des stations permettant l’implémentation de l’ensemble des stations.

Algorithmes de plus courts chemins :

Pour les algorithmes de plus courts chemins j’ai implémenté :

* Un algorithme de Djikstra : L'algorithme de Dijkstra a été implémenté pour trouver le plus court chemin entre deux stations du métro parisien, en utilisant comme pondération soit la distance, soit un temps de trajet arbitraire
* Affichage : le chemin trouvé est affiché dans la console, avec les vrais noms de stations, et visualisé sur l’image reseau\_metro.png en rouge.
* Un algorithme de BellmanFord: L’algorithme de Bellman-Ford a également été intégré comme alternative à Dijkstra, notamment pour sa capacité à gérer des poids négatifs, même si ce n’est pas nécessaire dans le contexte du métro. Avec cet algorithme on effectue plusieurs passes pour relâcher tous les arcs du graphe, et on détecte les cycles de poids négatifs si besoin.

Quel que soit l’algorithme utilisé :

* Le chemin optimal est surligné en rouge sur le graphe.
* Le nom réel des stations est affiché pour chaque nœud.
* Le temps total estimé est indiqué en bas de l’image.

Visualisation :

Nous avons d’abord re téléchargé la bibliothèque de données SkiaSharp et ensuite nous avons émis à chat Gpt une première requête qui était de pouvoir afficher les différents nœuds en fonction de leurs latitudes et reliées par des connexions :

* Nous voulions que : Les nœuds soient représentés par des cercles remplis et les connexions par des lignes
* Nous avons ensuite voulu pondérer les différentes connexions par la distance en km séparant les stations, nous avons donc demandés à chat Gpt de pondérer en fonction de cela les différentes connexions et d’en ajuster les longueurs
* J’ai également demandé que chaque nœuds et liens ne puissent pas se superposer, et que le graphe soit agrémenté de couleurs
* Après que cela est été fait nous avons demandé que le plus court chemin (via Dijkstra ou Bellman-Ford) soit représenté par une ligne rouge traversant l’ensemble du chemin et passant par les stations composant ce chemin.
* Ainsi le chemin le plus court est surligné en rouge.
* Pour finir nous avons également demandé à ce que l’image s’ouvre directement après création ce qui est réalisé par l’intermédiaire de System.Diagnostics.Process.Start.