Rapport Mathématiques Pour L'informatique (ALIF51)

Métro, boulot, dodo

EFREI 2022-23

Christine LI - Sophie MINOS -
Le Dan NGUYEN - Fandresena RAKOTOMAHEFA

SOMMAIRE

I/ INTRODUCTION

Pour ce projet, il nous a été demandé de développer un programme permettant de trouver le trajet le plus court afin d'aller d'une station à une autre.

Le concept est simple, on indique la station de départ et d'arrivée afin que le programme puisse présenter le chemin le plus rapide avec le temps le plus optimisé. Il y sera présenté les changements de ligne, les différents noms d'arrêt et le temps total du trajet.

Afin de récupérer le code du projet, nous vous invitons à vous rendre sur le GitHub du projet :
https://github.com/Fandresena02/Theorie-des-graphes-Metro

II/ RÉALISATION DU PROGRAMME

II/1) Structures de donnée principale

Pour ce qui est de la récupération des différentes données fournies dans les fichiers .txt, nous avons décidé de les ordonner de deux manières différentes.

Il nous a été fourni deux fichiers, un regroupant l'ensemble des sommets et un autre contenant toutes les arêtes reliant les sommets.

Afin d'accéder aux différents sommets, nous avons décidé d'utiliser une liste de dictionnaires:

```
[ { nom : "nom_station", ligne : numero_ligne, terminus : true/false, branchement : 0/1} ]
```

Une liste dont chaque élément est un dictionnaire contenant les informations de chaque station.

L'index de chaque élément sera l'identifiant de chaque station.

Afin d'accéder aux différents arêtes, nous avons décidé d'utiliser un dictionnaire de dictionnaires:

```
{ n° du sommet : { n° sommet de voisin : temps } }
```

Un dictionnaire avec pour clé, le numéro de chaque sommet et en valeur un dictionnaire contenant les voisins du sommets en clé, et en valeur, le temps de trajet entre le sommet et son voisin.

Cependant, pour ce qui est de l'algorithme de kruskal, nous avons décidé l'extrait les données des fichiers sous forme de tableau. Nous avons jugé plus simple pour cet algorithme de procéder ainsi.

II/2) Choix d'implémentation (dijkstra/kruskal)

Afin d'extraire l'arbre couvrant de poids minimum (ACPM) on a utilisé l'algorithme de Kruskal. Quant au plus court chemin entre deux stations, l'utilisateur rentre, on a utilisé l'algorithme de Dijkstra pour chercher ce plus court chemin.

II/3) Organisation des modules

Nous retrouvons dans l'ensemble du projet, 5 modules/fichiers différents. Chacun d'eux abrite une fonction.

- lire_fichier_sommets.py : contient la fonction fichier_sommet(nom_fichier) qui permet de lire un fichier donné et d'en extraire les informations des sommets.

- lire_fichier_aretes.py : contient la fonction fichier_arete(nom_fichier) qui permet de lire un fichier donné et d'en extraire les informations des arêtes.

- connexe.py : contient la fonction existe_chemin(matrice, fichier, u, v) qui permet de déterminer si un chemin entre deux sommets existe (renvoie True si le chemin existe, False sinon). Abrite également la fonction principale connexe(fichier), qui permet de déterminer si un graphe est connexe ou non (renvoie True si le graphe est connexe, False sinon)

- dijkstra.py : fichier contient des fonctions utiles à l'implémentation de l'algorithme de dijkstra, telles que :

- station_dans_chemin(chemin, num_station)
- station_duree_min(duree_min, stations_vues, chemin)
- algo_duree_min(depart)
- get_station(num_station)
- get_station_de_ligne(nom_station, ligne)
- get_ligne_lien(station1, station2)

- get_direction(station1, station2, ligne)
- parcours_chemin(duree_min, peres, depart, arrivee)

- kruskal.py : fichier contenant l'algorithme de kruskal. On y retrouve deux classes principales, UnionFind ainsi que Graphe (qui contient elle-même la fonction kruskal). En effet, afin d'éviter les potentiels cycles lors du choix des arêtes, l'implementation de la classe UnionFind était le plus adapté.

- main.py : qui contient le programme principal. Celui-ci demandera à l'utilisateur, la destination voulue en partant de la station choisie.

Nous avons également décidé de scinder le fichiers contenant les données (sommets / arêtes) en deux fichiers distincts afin de faciliter l'exploitation de ces derniers. On retrouve donc, un fichier **sommets.txt** et **arêtes.txt.**

III/ MODE D'EMPLOI

III/1) Commande d'exécution

✤ Assurez-vous de bien avoir la dernière version de python.

- Afin de lancer le programme principal:
`$ python3 main.py`
Il sera demandé à l'utilisateur, le nom de l'arrêt de départ ainsi que le nom de l'arrêt d'arrivée

- Pour lancer le programme pour savoir si le graphe est connexe ou non:
`$python3 connexe.py`
(PS: Si ce programme met du temps à s'exécuter c'est normal, car sur chaque sommet, on essaie de voir s'il existe un chemin depuis un sommet vers les autres sommets)
Le programme met à peu près 4 min de temps d'exécution

- Pour lancer le programme qui permet d'extraire l'arbre couvrant de poids minimum
(ACPM):
`$python kruskal.py`

IV/ GESTION DU DEVOIR

IV/1) Les outils utilisées

En ce qui concerne l'éditeur de texte utilisé, ***Visual Studio Code*** ainsi que ***Jupiter*** a été préféré pour ce devoir.
En ce qui concerne la communication entre les membres du groupe, Discord, SMS et GitHub ont été les canaux de communication préférés. Nous avons également échangé par téléphone afin de nous mettre d'accord sur la bonne voie à suivre, ainsi que sur l'évolution du devoir.

IV/2) Difficultés rencontrées

Nous avons rencontré plusieurs difficultés tout au long de ce projet, certain plus complexes que d'autres.

Pour la recherche du plus court chemin, la difficulté majeure a été de coder l'algorithme en s'adaptant aux structures de données choisies. Les stations pouvaient être dans plusieurs lignes de métro.

Il fallait donc vérifier la ligne prise lors d'un changement en cherchant la ligne commune avec la station précédente. Pour trouver les terminus, sur le même principe, il fallait avancer de stations en stations jusqu'à la station ayant l'attribut "terminus" à True. Il était donc nécessaire de coder des fonctions spécialisées, car elles sont utilisées de nombreuses fois.

CONCLUSION

Ainsi, la réalisation du programme nous a permis d'en apprendre davantage sur la manipulation des graphes et leurs utilités. Nous sommes plutôt satisfaits de la réalisation de ce projet, de notre organisation et de la structure finale de celui-ci.

Néanmoins, nous sommes plutôt déçus de ne pas avoir réussi (par manque d'organisation) à implémenter une interface graphique.
Mis à part cela, nous espérons que notre programme vous plaira et que vous prendrez du plaisir à le lire.