Exercices sur les graphes

1 Introduction

Lors de ce TP, nous travaillerons sur le jeu de données du réseau interurbain Fluo, disponible ici : `https://transport.data.gouv.fr/datasets/fr-200052264-t0051-0000-1`. Ce jeu de données contient diverses informations concernant les arrêts, les lignes et les horaires des transports publics dans le département de Meurthe-et-Moselle.

Ce jeu de données est au format GTFS (`https://fr.wikipedia.org/wiki/General_Transit_Feed_Specification`), qui est une archive ZIP au format standard pour l'échange d'information sur les transports publics.

1.1 Fichiers fournis

Trois fichiers vous sont fournis :

`stations.py` : Ce fichier fourni une classe `Stations` qui représente l'ensemble des arrêts de transport en commun sur le département. Cette classe utilise (en interne) la classe `Station`, qui représente un arrêt unique. Chaque arrêt est constitué d'un nom, d'un identifiant, et d'un couple de coordonnées (latitude et longitude)

`routes.py` : Ce fichier fourni une classe `Routes` qui représente l'ensemble des lignes de transport en commun du département. En interne, cette classe utilise la classe `Route`, qui représente une unique ligne. Chaque route est constituée d'une liste de segments, où un segment est une paire de deux arrêts consécutifs. Par exemple, une route qui passe par les arrêts dont les identifiants sont `A`, puis `B`, puis `C` aura les segments `(A, B)` et `(B, C)`.

`fluo.py` : Ceci est le fichier principal du programme sur lequel vous allez travailler. Il contient d'office une fonction `nx_graph_to_map` qui permettra plus tard d'afficher le graphe du réseau sur une carte.

1. Dans le fichier `routes.py`, regardez la classe `Routes`, si besoin au débogueur. Quel(s) champs ont les objets de cette classe ?

   Solution:

2. Dans le fichier `stations.py`, trouvez la méthode `get_coords_of_station`. Quelle classe défini cette méthode ? Que fait cette méthode ? Quel est le type de la valeur renvoyée ?

Solution:

Remarque (Identifiants des arrêts) : Les arrêts sont caractérisés par un identifiant unique, un nom, une latitude et une longitude. La classe `Station` groupe ces quatre éléments pour chaque arrêt. Dans les autres classes, les arrêts sont identifiés via leur identifiant unique. On peut naturellement obtenir les autres éléments à partir de cet identifiant via les méthodes `get_name_of_stop` et `get_coords_of_station` de la classe `Stations`.

2 Création de graphe

3. Écrivez un algorithme (sur papier, en pseudo-code) pour créer un graphe dont les nœuds sont des arrêts de transport en commun (identifiés par l'identifiant de l'arrêt) et dont les arrêtes sont les segments qui forment les lignes du réseau. Vous pouvez utiliser une variable `arrêts` qui est une liste d'arrêts et `routes` qui est une liste de route, où chaque route est une liste de paires `(id_départ, id_arrivée)`, où `id_départ` et `id_arrivée` sont les identifiants des arrêts de départ et d'arrivée de chaque segment de la ligne.

Solution:

4. Dans le fichier `routes.py`, complétez la fonction `get_nx_graph` afin qu'elle renvoie le graphe du réseau de transport en commun. Pour chaque arrête du graphe, ajoutez la distance entre les deux arrêts d'origine et de destination.

Remarque : `geopy.distance.geodesic(coord1, coord2)` permet de calculer la distance entre deux coordonnées. Utilisez la méthode `get_coords_of_station(id)` de la classe `Stations` pour obtenir les coordonnées d'un arrêt à partir de son identifiant.

Remarque : La fonction fournie `nx_graph_to_map` permet de visualiser, si besoin, le graphe du réseau sur une carte.

3 Manipulation de graphes

3.1 Composantes connexes

5. Combien y a-t-il de composantes connexes dans le réseau ? Pour chaque composante connexe, trouvez le nombre de nœuds de la composante. Trouver la plus grande composante connexe du réseau.

   Remarque : `https://fr.wikipedia.org/wiki/Composante_connexe_(th%C3%A9orie_des_graphes)`

   Remarque : Dans ce qui suit, on ne considérera que la plus grande composante connexe.

Solution:

3.2 Degrés des nœuds

6. Quel est l'arrêt le mieux desservi (i.e. celui avec le plus de voisins) ?

Solution:

3.3 Plus court chemin

7. Explorez la documentation de NetworkX afin de trouver une fonction qui calcule le chemin le plus court entre deux nœuds d'un graphe. Quelle fonction avez-vous trouvé ?

Solution:

Admettons qu'on souhaite faire de l'escalade aux falaises de Maron. Quel est le chemin le plus court depuis le campus de l'école (prendre l'arrêt Vélodrome) vers les falaises (prendre l'arrêt Maron Charles de Gaulle) ? Écrivez un programme qui affiche tous les arrêts intermédiaires.

8. Admettons maintenant qu'on souhaite organiser une compétition sportive départementale. Pour des raisons logistiques, on cherche à trouver l'arrêt de transport en commun qui est le plus proche des autres; c'est-à-dire trouver l'arrêt A qui minimise :

$$\sum_{s \in \mathbb{S} \setminus \{A\}} d(A, s)$$

où $\mathbb{S}$ est l'ensemble des arrêts et $d(s_1, s_2)$ est la distance entre s_1 et s_2[1].

Remarque : Cette question est bien entendu simplifiée par rapport à un cas réel. En pratique, on aimerait minimiser la distance parcourue, en prenant donc en compte que certains arrêts sont plus utilisés que d'autres. Il faudrait aussi prendre en compte d'autres aspects (comme la disponibilité d'infrastructures, etc.).

Dans un premier temps, admettons qu'on ait un dictionnaire de dictionnaires de distances entre tous les arrêts (e.g. `distance[source][destination]` contient la distance entre `source` et `destination`). Écrivez un algorithme qui trouve l'arrêt le plus proche des autres.

Remarque : On suppose que le graphe du réseau est connexe.

[1] Dans le graphe qu'on a construit, les arrêtes ont un attribut *distance* qui représente la distance à vol d'oiseau entre deux nœuds. On utilisera cette distance comme une approximation de la distance réellement parcouru par les véhicules de transports en commun.

Solution:

9. Cherchez, dans la documentation, une fonction qui calcule un tel dictionnaire. Comment s'appelle cette fonction ?

Solution:

10. Implémentez maintenant votre algorithme, en utilisant la fonction que vous avez trouvé.

    Remarque : Il est important de considérer la plus grande composante connexe. Si vous utilisez le graphe complet, vous trouverez des distances infinies.