TD 21 : Graphe, liste d'adjacence

Le fichier `metro_paris.txt` de 1311 lignes, encodé en `utf-8`, contient la description naïve du graphe orienté du métro parisien :

- les 376 stations correspondent aux sommets du graphe et sont dans la partie `[Vertices]` sous la forme `nnnn x...x` où `nnnn` est le numéro de la station `x...x`.
- les 933 arcs orientés du graphe sont dans la partie `[Edges]` sous la forme `nx ny ttt` où `nx` est le numéro de la station de départ, `ny` le numéro de la station d'arrivée et `ttt` la durée en seconde du trajet entre les 2 stations. Les correspondances sont placées à la fin, avec un temps conventionnel de 2 minutes. On ne fait aucune hypothèse sur l'ordre d'apparition des arcs dans le fichier.
- chaque ligne contient un retour à la ligne qu'il faudra enlever.

Dans les prochains TD, on utilisera la représentation obtenue par liste d'adjacence pour mettre en oeuvre les algorithmes de parcours de graphe. Il est donc impératif de finir cet exercice sous peine de ne pas pouvoir commencer les prochains TD.

Exercice 1 (Graphe orienté du métro parisien, représentation par liste d'adjacence)

On s'intéresse au **graphe orienté non pondéré** du métro parisien et l'on ignore donc les données temporelles du fichier `metro_paris.txt`.

1. Copier dans le même répertoire le fichier `metro_paris.txt` et le fichier `TD21_metro_fichier_a_completer.py` (que l'on retrouvera aussi à la fin de ce document).
2. Ouvrir le fichier `metro_paris.txt` dans votre éditeur et parcourir sa structure. Observer que la plupart des liaisons sont bidirectionnelles.
3. Compléter le fichier `TD21_metro_fichier_a_completer.py` afin de construire une représentation du graphe `G` sous la forme basique `graphe_metro = (S,A)` où `S` est l'ensemble des noms de stations (sommets) et `A` l'ensemble des arcs.
4. Ecrire une fonction `liste_adjacence(G)` qui prend en argument un graphe `G=(S,A)` défini à la question précédente et qui retourne un tableau contenant la représentation du graphe `G` par liste d'adjacence. **La fonction doit trier chaque voisinage par indices de stations croissants**.
5. Appliquer cette fonction au graphe `graphe_metro` pour définir la variable globale `metro_la`. On doit obtenir la représentation par liste d'adjacence suivante des 376 sommets triés : `[[159, 238], [12, 235], [110, 139], [210, 262], ... , [53, 186], [196], [29, 134], [165, 310]]`. Le premier sommet (station Abbesses) est relié aux stations 159 et 238, ce que vous pouvez vérifier sur le plan fourni `carte-metro-paris.pdf` (cette carte contient davantage de stations que le graphe et montre aussi les lignes RER). Vous pouvez utiliser la fonction `CRTL-f` dans un lecteur pdf pour rechercher la station Abbesses.

Exercice 2 (Utilisation de la représentation sous forme de liste d'adjacence)

1. Ecrire une fonction `stations_voisines(la, noms, station)` qui renvoie la liste des noms des stations voisines de la station `station` dans le graphe représenté par la liste d'adjacence `la`, dont le nom des sommets est contenu dans `noms`.
2. Améliorer cette fonction pour qu'elle fonctionne si le nom de la station est présent plusieurs fois dans l'ensemble des sommets. `stations_voisines(metro_la, S, 'Opéra'` doit renvoyer

   ```
   [ ['Havre Caumartin', 'Quatre Septembre'],
     ["Chaussée d'Antin, La Fayette", 'Pyramides'],
     ['Madeleine', 'Richelieu Drouot'] ]
   ```

   car la station `Opéra` est à l'intersection des lignes 3, 7 et 8. On a pris soin de supprimer la station `Opéra` elle-même de la réponse.
3. Ecrire une fonction `ligne(la, noms, station)` qui reçoit en argument le nom de la station `station` en bout d'une ligne de métro et qui renvoie la liste des noms des stations de toute la ligne. Attention lors des tests : la carte du métro donnée est plus récente et des stations ont parfois été récemment ajoutées en bout de ligne.

   Exemple : `ligne(metro_la, S, 'Mairie des Lilas'` renvoie `['Mairie des Lilas', 'Porte des Lilas', 'Télégraphe', 'Place des Fêtes', 'Jourdain', 'Pyrénées', 'Belleville', 'Goncourt', 'République', 'Arts et Métiers', 'Rambuteau', 'Hôtel de Ville', 'Châtelet']`.

   Indication : parcourir le voisinage de la station actuelle et choisir le premier voisin dont le nom est différent et qui n'est pas la station précédente sur la ligne : c'est forcément la suivante.

```
''' Lycée Vaugelas MPSI
    Graphe du métro parisien
    Représentation par liste d'adjacence
'''
# lecture du fichier du métro parisien
import os as os
repertoire_courant = os.getcwd()
nom_fichier = '/metro_paris.txt'
fichier = open( , 'rt', encoding='utf-8')

Ordre =
Taille =
ligne = fichier.readline() # ligne 1 à éliminer

S = []
for i in range(Ordre):
    ligne = fichier.readline().rstrip('\n')
    S.append(   )

ligne = fichier.readline() # ligne 378 à éliminer

A = []
for i in range(Taille):
    ligne = fichier.readline().split(' ')
    arc =  ,
    A.append( )

fichier.close()
graphe_metro = (S,A)

# création de la représentation par liste d'adjacences
def liste_adjacence(G):
    LA = [] # la liste d'adjacence à constituer



    return LA

metro_la = liste_adjacence(graphe_metro)

print(metro_la[:4],'...', metro_la[-4:])

# stations voisines d'une station donnée par son nom
def stations_voisines(g, Noms, station):
    Ordre = len(Noms)
```

```
print(stations_voisines(metro_la, S, 'Opéra'))


# liste des stations d'une ligne de métro
# on suppose que la station en bout de ligne n'appartient qu'à une seule ligne
def ligne(g, Noms, station):
    for i, nom in enumerate(Noms):
        if nom == station:
            k = i # k est l'indice de la station en bout de ligne
            break
    ligne = [station]



    return ligne

print(ligne(metro_la, S, 'Galliéni'))
print(ligne(metro_la, S, 'Château de Vincennes'))
print(ligne(metro_la, S, "Mairie d'Issy"))
print(ligne(metro_la, S, 'Mairie des Lilas'))
print(ligne(metro_la, S, 'Porte de Clignancourt'))
```