TD 30 : Graphes : A*

Récupérer le fichier `metro_A_star.txt` de 1688 lignes, au format utf-8, qui contient la description du graphe orienté du métro parisien, de manière naïve :

- une ligne d'entête ([Vertices])
- une liste de 376 stations (sommets) : chaque ligne du fichier est constituée d'un numéro de station, d'une espace, du nom de la station, d'un caractère de fin de ligne.
- une ligne d'entête ([Coordinates])
- une liste des coordonnées géographiques des 376 stations (les abscisses et ordonnées des stations sur une grille entière) : numéro de station, espace, abscisse, espace, ordonnée.
- une ligne d'entête ([Edges])
- une liste de 933 arcs du graphe, c'est-à dire des couples de stations voisines, reliées par une ligne de métro ou par une correspondance. Ainsi lorsqu'une station appartient à plusieurs lignes de métro, elle apparaît plusieurs fois. Chaque ligne est constituée du numéro d'une station, d'une espace, du numéro d'une autre station, d'une espace, du temps en seconde pour aller de la première à la seconde station, d'un caractère de fin de ligne. Les correspondances sont placées à la fin, avec un temps conventionnel de deux minutes. **On ne fait aucune hypothèse sur l'ordre d'apparition des arcs dans le fichier.**

Objectif du TD

Comparer les algorithmes de Dijkstra et A* pour rechercher un plus court chemin dans un graphe orienté et pondéré.

Exercice 1 (Représentation du graphe par liste d'adjacences)

On s'intéresse au **graphe orienté pondéré** du métro parisien décrit dans le fichier `metro_A_star.txt`.
Pour vous laisser le temps d'aborder le coeur du TD qui se situe dans l'exercice 2, vous trouverez un fichier nommé astar_exercice_1.py qui créé une représentation naïve du graphe orienté sous la forme basique `G = (S,A)`, où S est l'ensemble des **noms** des sommets (stations) et A l'ensemble des **arcs valués**, sous la forme de triplets (s,s',w). Les sommets s et s' sont des entiers et le poids w de l'arc liant s à s' est un flottant.
Puis il créé un tableau global `Coords` contenant les coordonnées (sous forme de couples) des différentes stations.
Enfin, il contient une fonction `liste_adjacence(G)` qui prend en argument un graphe $G = (S, A)$ naïf (du type de celui qui vient d'être créé) et qui retourne un tableau (liste Python) contenant la représentation du graphe G par liste d'adjacence.

Appliquer cette fonction au graphe obtenu pour définir la variable globale `graphe_metro`.
On doit obtenir une représentation par liste d'adjacence du type :
[[(238, 41.0), (159, 46.0)], [(12, 36.0), (235, 44.0)], [(110, 69.0), (139, 50.0)], [(262, 33.0), (210, 41.0)],$\cdots$]
Le premier voisinage [[(238, 41.0), (159, 46.0)] indique avec le couple (238, 41.0) que la station Abbesses (sommet 0) est reliée à la station Pigalle (238) en 41 secondes.
Rappel : le plan du métro fourni dans le fichier `carte-metro-paris.pdf` contient davantage de stations que le graphe et montre aussi les lignes de RER.

Exercice 2 (Algorithme A*)

On utilise le module `queue` et la structure de file de priorité `PriorityQueue` qu'il contient. Une file de priorité vide est créée par `F = PriorityQueue()`, on enfile par `F.put((p,s))`, on défile par `F.get()` et on teste par `F.empty()`.
Les résultats pour les couples de stations suivantes : (0, 1) ; (89, 253) ; (12, 362) , (257, 175) ; (35, 80) sont à observer.

1. **Plus court chemin par l'algorithme A***
   Rappeler l'algorithme A*, pour une heuristique.
   Écrire une fonction `A_star(g,h,s1,s2)` qui retourne le tableau des distances (poids total des chemins) et le tableau des prédécesseurs pour le plus court chemin trouvé.
2. **Heuristique**
   Écrire une fonction `h(s1,s2)` qui retourne le temps de parcours en ligne droite entre les stations `s1` et `s2` à la vitesse de 10 mètres par seconde, en utilisant les coordonnées des stations, sachant qu'une unité sur la grille correspond à $25,7\ m$.
   Pour info : les poids des arêtes ont justement été calculés comme si le métro allait en ligne droite à cette vitesse moyenne entre deux stations : cette heuristique est admissible et consistante.
3. **Reconstitution du plus court chemin**
   Écrire une fonction `plus_court_chemin(g,s1,s2)` qui retourne le plus court chemin de `s1` à `s2` donné par la fonction `A_star` précédente, et le poids de ce chemin, sous forme d'une liste de sommets, de `s1` à `s2` et d'un flottant.
4. Tester la fonction `plus_court_chemin(g,s1,s2)` sur le graphe du métro. Par exemple pour aller de la station Avron à la station Victor Hugo, qui sont sur la même ligne 2, il est avantageux de changer à Nation et à Charles de Gaulle, Étoile.
5. Reprendre la fonction `Dijkstra` écrite dans un TD précédent pour trouver un plus court chemin de `s1` à `s2`. Comparer les chemins obtenus (ils doivent être identiques !)
6. **Comparaison A* - Dijkstra**
   Afin de comparer l'efficacité des algorithmes de Dijkstra et A*, représenter graphiquement, pour ces deux algorithmes, la carte du métro parisien en marquant les stations par de petits cercles :
   — cercle vert pour une station non explorée
   — cercle rouge pour une station explorée
   — cercle bleu pour les stations du plus court chemin trouvé