Algorithme de Dijkstra et métro de Paris

Exercice 1 (𝄞⋆). Soit le graphe (orienté et pondéré) `G=(D,M)` où `D={'A':0,'B':1,'C':2,'D':3,'E':4,'F':5}` et

$$M = \begin{pmatrix} 0 & 1 & 2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 5 & 10 & 0 \\ 1 & 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 2 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 50 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

1. Représenter le graphe `G`.
2. On rappelle le fonctionnement de l'algorithme de Dijkstra : remplir deux dictionnaires `Dis` et `P`, `Dis[but]` va représenter la plus courte distance entre une source, ici `A`, et le sommet `but`, `P[but]` va contenir le dernier point par lequel il faut passer pour aller de `A` vers `but` (de même pour tous les autres sommets) [1].
3. Compléter alors le tableau jusqu'à ce que tous les points soient traités. Gardez en tête que l'on actualise `P` et `Dis` que si on trouve un itinéraire plus court. À chaque étape, on choisit le sommet le plus proche de `A` parmi les points non traités, puis pour ce sommet, regarder ces voisins, si vous trouvez un chemin plus court alors actualisez `Dis` et `P`.

Étape	T	Dis[A]	Dis[B]	Dis[C]	Dis[D]	Dis[E]	Dis[F]	P[A]	P[B]	P[C]	P[D]	P[E]	P[F]
0	`[]`	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	A					
1	`[A]`												
2													
3													
4													
5													
6													

4. À l'aide du tableau retrouver le plus court chemin de `A` vers `F`.

Exercice 2 (𝄞⋆⋆). 1. Téléchargez les fichiers `TP15.py`, `TP15D.npy` et `TP15M.npy` et mettez les dans un même dossier [2]. Une fois que vous avez exécutez le fichier `TP15.py`, `D` est un dictionnaire dont les clés sont les noms des arrêts de métro et la valeur associée à une clé est le numéro de l'arrêt. `M` est la matrice d'adjacence du graphe (modélisée comme une liste de listes) : `M[i][j]` représente le temps du trajet partant de l'arrêt dont le numéro est `i` vers l'arrêt dont le numéro est `j` s'il existe une ligne qui rejoint ces arrêts (sans aucun arrêt intermédiaire) ou s'il s'agit d'un changement de quai d'une station avec plusieurs lignes, 0 sinon.

2. Dans le fichier `TP15.py`, compléter la fonction `Dijkstra(D:dict,M:list,source:str)` suivant le principe expliqué lors de l'exercice 1. Cette fonction renverra les dictionnaires des prédécesseurs et des distances entre le sommet `source` (c'était `A` dans l'exemple précédent) et tous les autres sommets. On pourra au besoin utiliser la fonction `ListeVoisins` déjà écrite :
   - Initialiser d'abord les dictionnaires `Dis` et `P` et la liste `T` comme à l'étape 0 : `P` est un dictionnaire dont la seule clé est `source` et la valeur est `source`, `Dis` est un dictionnaire dont les clés sont les sommets et dont les valeurs sont toutes l'infini sauf celle de `source` qui vaudra 0. On rappelle que les sommets sont les clés du dictionnaire contenu dans `G` et que $+\infty$ peut se code avec `np.inf` (si `numpy` est chargé avec l'alias `numpy`).
   - Écrire une fonction `SommetNonTraitePlusProche(Dis,T)` qui renvoie un sommet qui est non traité dont la valeur dans `Dis` est minimal.
   - Dans une boucle `while`, tant que tous les sommets ne sont pas dans `T` faire :
     — À l'aide de la fonction créée à la question précédente, noter `smin` un sommet non traité dont la valeur dans `Dis` est minimal
     — Rajouter `smin` à `T`
     — Pour chaque voisin `v` de `smin`, si la distance entre `source` et `smin` additionné à celle entre `smin` et `v` est strictement inférieur à `Dis[v]`, alors actualiser en conséquence `Dis[v]` et `P[v]`.
   - Renvoyer `Dis` et `P`.

1. Cherchez le plus court chemin vers but finalement c'est : «*droit au but !*»
2. Les données du métro de Paris qui servent à ce TP ont été récupérés sur https://perso.esiee.fr/~coupriem/Graphestp3/graphestp3.html et https://github.com/BTajini/Paris-Metro-Project. Un grand merci à leurs auteurs.

3. Créer une variante de la fonction `Dijkstra`, nommée `Dijkstra2`, où cette fois-ci, `T` sera un dictionnaire, on rajoutera les sommets traités comme des clés de `T` avec une valeur arbitraire. Comparer les temps de calculs avec les deux fonctions.
4. Quel est le plus court chemin partant de `"Gare de l'Est"` et allant à `'Mirabeau'` et quel temps cela prend-il ? Pour cela, créer une fonction `Itineraire(G,source,but)` qui renverra la distance et l'itinéraire pour aller du point `source` au point `but`. Pour trouver la distance, il suffit d'utiliser `Dis` fourni par la fonction `Dijkstra`, pour trouver l'itinéraire, il faut partir du point d'arrivé, considérer son prédécesseur, puis le prédécesseur du prédécesseur etc. jusqu'à retomber sur `source` en utilisant le dictionnaire `P` créé par `Dijkstra`.

Exercice 3 (⋆⋆)**.** Trouver les deux stations du métro de Paris les plus éloignés en temps de transport.

Exercice 4 (𝄞⋆)**.** Écrire une fonction `Parcours(G,source)` qui parcourt le graphe soit en largeur partant du sommet `source`.

Exercice 5 (⋆⋆)**.** Modifier l'algorithme du parcours en largeur pour que la fonction renvoie le nombre sommets qu'il a fallu traverser pour aller de `source` à `but`. Pour cela, il suffit de considérer que si on visite un nouveau sommet à partir d'un sommet relié en n points de la source, alors ce nouveau sommet se visite en $n+1$ points.

Remarque 1. Des expériences en sciences sociales suggèrent ainsi qu'entre deux personnes dans le monde il y a au plus six degrés de séparation : https://en.wikipedia.org/wiki/Six_degrees_of_separation

Exercice 6 (⋆⋆)**.** Actuellement le métro est connexe : entre deux arrêts de métro il existe toujours un chemin. Soit $n \in \mathbb{N}^*$, supposons que l'on supprime de façon aléatoire n arcs du graphe (en mettant les coefficients de la matrice à zéro), tester si le graphe est encore connexe, pour cela il suffit de vérifier que pour tout sommet `source` on peut aller à n'importe quel autre sommet (autrement dit le dictionnaire `dis` ne contient pas $+\infty$). Vous pouvez aussi tracer le temps de parcours entre deux stations données en fonction du nombres d'arêtes enlevés (cela permet de tester la résilience du métro en cas de panne).

Remarque 2. Savoir si un graphe est encore connexe après qu'on lui ait retiré des arêtes et savoir combien il y a en moyenne de composantes connexes est tout un champ d'étude en mathématiques et informatique : https://fr.wikipedia.org/wiki/Th%C3%A9orie_de_la_percolation.

Exercice 7 (⋆)**.**
1. Dans la matrice `M`, comptez le nombre de coefficients qui sont nuls et ceux qui ne sont pas nuls.
2. La question précédente, montre qu'il y a beaucoup de zéros dans la matrice, ainsi le stockage de cette matrice prend inutilement de la place. Créer le graphe sous forme de dictionnaire d'adjacence. Chaque arrêt est une clé du dictionnaire et la valeur associée à cette clé est elle-même un dictionnaire dont les clés sont les voisins et les valeurs les temps de parcours.