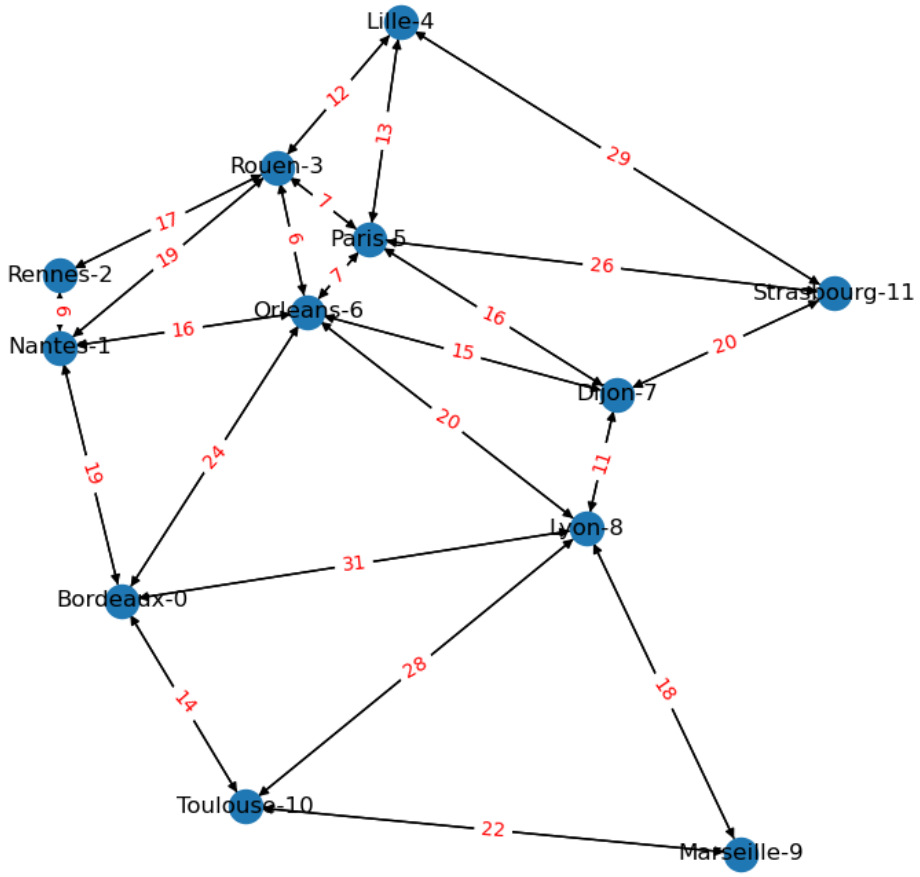
TP N°07 : GRAPHES PONDERES ET ALGORITHMES DE PLUS COURT CHEMIN

OBJECTIFS DU TP

- Recherche d'un plus court chemin dans un graphe pondéré avec des poids positifs avec l'algorithme de Dijkstra
- Mettre en évidence qu'il est possible d'améliorer sa rapidité avec une heuristique.

Document : Bordeaux – Strasbourg en vélo ?

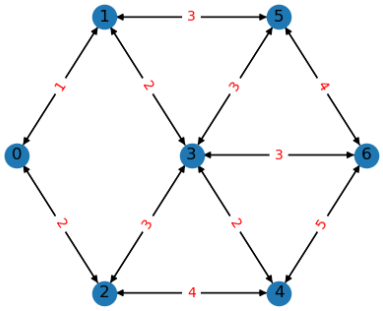
Toujours à la recherche de nouveaux exploits à accomplir, vous décidez de partir de Bordeaux pour rejoindre Strasbourg en empruntant les chemins décrits sur cette carte. Une connexion est faite entre les chefs-lieux des régions contiguës. Le chiffre associé représente le **nombre d'heures à vélo** (sans les pauses...).



Ouvrir avec pyzo le fichier `ITP07-donnees.py` qui se trouve sur le réseau dans le dossier partagé de votre classe. **Enregistrer le dans vos documents**. Vous pouvez écrire vos fonctions à la suite de ce script en utilisant les données.

Q1. Trouver « à la main » sans stratégie particulière, le plus court chemin entre Bordeaux et Strasbourg.

I. Recherche du plus court chemin avec l'algorithme de Dijkstra



Pour (re)prendre en main l'algorithme de Dijkstra, on va dans un premier temps se concentrer sur un graphe d'ordre inférieur, ce qui va nous permettre d'appliquer l'algorithme « à la main » dans un temps raisonnable.

Q2. Sans l'implémenter, trouver le plus court chemin en utilisant manuellement l'algorithme de Dijkstra entre les sommets 0 et 6 sur le graphe ci-dessus. Noter chaque étape de raisonnement dans le tableau ci-dessous pour vous familiariser avec l'algorithme. L'algorithme est rappelé ci-dessous.

Étape	Listes E des sommets à parcourir	S	Voisins de S	Distance à D							Prédécesseurs						
				0	1	2	3	4	5	6	0	1	2	3	4	5	6
Ini																	
1																	
2																	
3																	
4																	
5																	
6																	
7																	

Initialisation : On définit :

- n, l'ordre du graphe
- Une liste `E` contenant les sommets de G (liste permettant de voir quels sommets il reste à parcourir)
- Une liste `dist` contenant n fois ∞ (`dist[i]` correspond à la plus courte distance trouvée entre D et i)
- Une liste `pred` contenant n valeurs non affectées.

Le sommet que l'on est en train d'étudier est noté S.

- Par définition $dist[D] \leftarrow 0$ et S = D et on enlève D des sommets à parcourir.

Tant que la liste E n'est pas vide :

- On parcourt les sommets v_i accessibles depuis S
  - Si `dist[S]` + poids de l'arc (S,v_i) est la valeur de distance minimale entre D et v_i rencontrée jusqu'à maintenant, alors cette valeur est placée dans la liste `dist`.
  - S devient alors le prédécesseur de v_i sur ce chemin
- On repart ensuite du sommet qui, parmi ceux qui restent à explorer, présentant la plus faible distance au sommet de départ D.

Pour obtenir le chemin, on reprend la liste des prédécesseurs en partant du sommet d'arrivée.
On retourne la plus faible distance D-A et la liste des sommets décrivant ce chemin.

Q3. Ecrire une fonction `voisins` qui prend en argument une matrice d'adjacence M d'un graphe pondéré et un entier S puis renvoie la liste des sommets adjacents au sommet S.

Q4. Ecrire une fonction `Dijkstra` prenant en argument une matrice M d'adjacence du graphe pondéré G(S,A) et deux entiers D et A représentant respectivement les sommets de départ et d'arrivée. Cette fonction retourne la liste des sommets correspondant au plus court chemin pour aller du départ à l'arrivée et un entier correspondant à la distance de ce chemin. On pourra utiliser la fonction `voisins`.

Q5. Utiliser la fonction `Dijkstra` pour trouver le plus court chemin ainsi que sa durée entre Bordeaux et Strasbourg.

Q6. Modifier la fonction précédente pour qu'elle renvoie également la liste de tous les sommets explorés pour trouver le plus court chemin.

Dijkstra garantit la meilleure solution possible. Mais la question 4 vous permet de remarquer qu'il s'aventure dans des recoins peu pertinents, par exemple, tester **Rouen** dans un trajet **Bordeaux → Strasbourg.**

Or, il est parfois indispensable d'aller vite, quitte à trouver une solution "presque" optimale : c'est par exemple le cas du jeu vidéo.
Il est possible d'améliorer la rapidité de cet algorithme avec l'algorithme A* (a star) qui utilise une heuristique : on va pénaliser les sommets du graphe dont il nous semble peu probable qu'ils appartiennent au chemin recherché. En pratique, une **heuristique** est une fonction de coût qui, étant donné un sommet, nous indique s'il a des chances de se trouver sur le chemin optimal.

Comme nous souhaitons minimiser la distance parcourue, l'heuristique que nous allons utiliser est la durée à vol d'oiseau entre les sommets et la ville s'arrivée, Strasbourg.

Ville	Bordeaux	Nantes	Rennes	Rouen	Lille	Paris	Orleans	Dijon	Lyon	Marseille	Toulouse	Strasbourg
Sommet	0	1	2	3	4	5	6	7	8	9	10	11
Durée à vol d'oiseau de Strasbourg	47	44	43	31	25	25	27	15	24	39	46	0

Le fonctionnement de l'algorithme de A* est identique à celui de Dijkstra. La seule étape qui diffère est l'ordonnancement des sommets qui restent à visiter :

- Dans un algorithme de Dijkstra, on choisit comme prochain sommet à visiter celui qui est actuellement le plus proche du sommet de départ ;
- Dans l'algorithme d'A*, on choisit le sommet dont la somme entre la distance actuelle au point de départ et l'heuristique est la plus faible. On défavorise ainsi les points qui s'éloignent de l'heuristique.

Q7. Ecrire une instruction permettant de créer une liste `H` contenant à l'indice i l'heuristique proposée pour le sommet i.

Q8. Ecrire une fonction `A_star` prenant en argument une matrice M d'adjacence du graphe pondéré G(S,A), deux entiers D et A représentant respectivement les sommets de départ et d'arrivée et une liste H représentant l'heuristique retenue. Cette fonction retourne la liste des sommets correspondant à un des plus courts chemins pour aller du départ à l'arrivée. On pourra utiliser la fonction `voisins`.

Remarque : l'algorithme s'arrête dès qu'une solution est trouvée. Ce n'est pas nécessairement la meilleure mais souvent l'une des meilleures.

Q9. Modifier la fonction précédente pour qu'elle retourne également la liste de tous les sommets visités.

Q10. Commenter en comparant les deux algorithmes.

Pour les plus rapides : utiliser les fonctions de la bibliothèque `networkx` *pour afficher en couleurs les sommets visités pour trouver le plus court chemin et montrer visuellement que A* permet d'en explorer moins.*