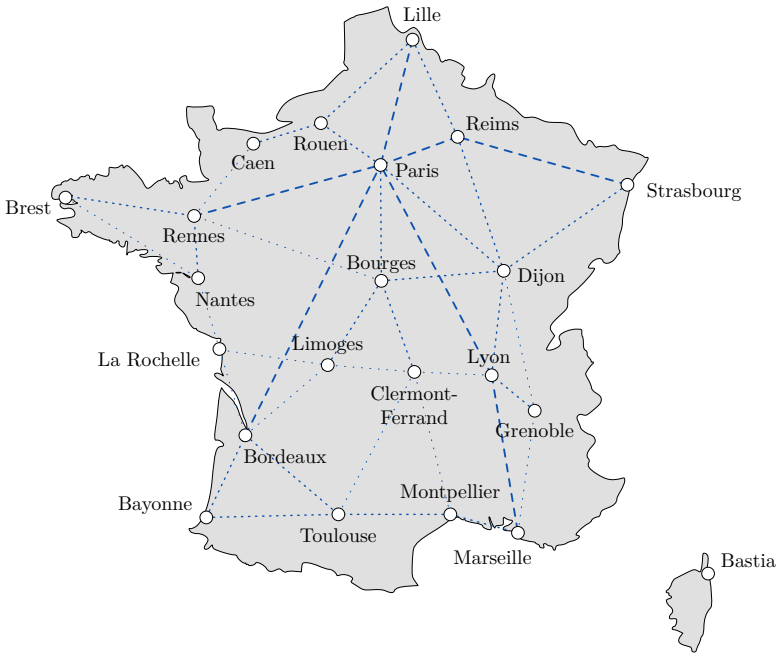
? À rendre mercredi 4 décembre 2024

Devoir Maison n°4

- Ce DM traite de divers thèmes
  - graphes (1re année),
  - parcours des graphes (1re année)
  - programmation dynamique.
- Le DM devra être rendu en **manuscrit**. Je vous encourage à y réfléchir dessus avec un crayon et un papier, sans ordinateur.
- Les fonctions des questions sont à traiter **sur le document réponse** page 5.
- Pour toute question, n'hésitez pas à me contacter par mail : nvalade.pcsi@gmail.com ou par cahier-de-prepa.

L'an prochain, vous serez tous disséminés aux quatre coins de la France dans vos écoles de rêve respectives. M. et G., délégués de qualité décident de venir vous rendre visite durant l'année.



I Modélisation d'un graphe

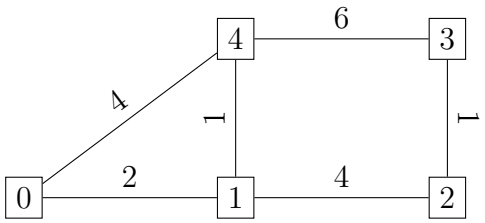
On représente le graphe pondéré par une matrice d'adjacence.

On rappelle que la matrice d'adjacence d'un graphe pondéré sans boucle dont les sommets sont étiquetés par les entiers $0, 1, \ldots, n-1$ est une matrice $M = (m_{i,j})_{i,j\in\{0,\ldots,n-1\}}$ telle que, pour tout sommets i et j, on a

$$m_{i,j} = \begin{cases} 0 \text{ si } i = j \\ \text{le poids de l'arc } i \to j \text{ s'il en existe une} \\ +\infty \text{ sinon} \end{cases}$$

On représente les graphes par leur matrice d'adjacence, l'élément $+\infty$ pouvant être obtenu avec `float('inf')`. Une matrice est ici une liste de listes. Ainsi, pour i un sommet du graphe de matrice d'adjacence `M`, la liste `M[i]` est celle des poids des arcs de i vers un autre sommet, où plus précisément `M[i][j]` est le poids de l'arc $i \to j$ ($+\infty$ s'il n'en existe pas, 0 si $i = j$).

On considère l'exemple suivant :



Q1. Donner la matrice d'adjacence de ce graphe.

On rappelle qu'un chemin dans un graphe est une suite $(s_0, s_1, \ldots, s_p)$ de sommets du graphe tels qu'il existe toujours un arc $s_i \to s_j$. Le poids total d'un tel chemin est la somme des poids des arcs $s_0 \to s_1, s_1 \to s_2, \ldots, s_{p-1} \to s_p$.

Q2. Écrire une fonction `poidsTotal(M:list,c:list)->float` qui prend comme argument la matrice `M` d'adjacence d'un graphe et un chemin dans ce graphe donné par une liste Python `c`, et renvoie comme résultat le poids total de ce chemin dans ce graphe ($+\infty$ si le chemin comporte deux sommets consécutifs entre lesquels il n'y a pas d'arc).

II Algorithme de Dijkstra

On souhaite déterminer le poids minimal depuis un sommet i donné jusqu'à n'importe quel sommet j donné.

Cela pourra permettre à M. (ou à G.) de connaître le chemin minimal pour aller rendre visite à chaque autre étudiant.e de la classe.

Pour cette partie, le graphe est représenté par un dictionnaire d'adjacence. Les clés sont les sommets, et les valeurs associées sont des listes de pondération : ce sont des listes de listes, dont le premier élément est un sommet avec lequel la clé est liée, et le deuxième élément est la distance correspondante.

Par exemple :

```
{s0 : [[s1,3],[s2,4]],s1:[[s0,3],[s3,5]]}
```

Q3. Écrire le dictionnaire du graphe présenté précédemment.

Edsger Dijkstra a publié son algorithme en 1959. Il utilise un parcours en largeur et calcule le plus court chemin entre un sommet et chacun de ses autres sommets. On suppose le graphe connexe et orienté.

On part du sommet 0 (par exemple), et on cherche le plus court chemin entre ce sommet et chacun des autres sommets (ou un sommet en particulier). On commence par affecter une valeur très grande à chacun des autres sommets, disons infinie, et la valeur 0 au sommet de départ (0 dans notre exemple).

À chaque étape, on effectue le meilleur choix possible : c'est un algorithme glouton. Tout au long de l'algorithme on va garder en mémoire le chemin le plus court depuis 0 pour chacun des autres points dans un tableau.

On répète toujours le même processus :

1. On choisit le sommet accessible de distance minimale comme sommet à explorer.
2. À partir de ce sommet, on explore ses voisins et on met à jour les distances pour chacun. On ne met à jour la distance que si elle est inférieure à celle que l'on avait auparavant.
3. On répète jusqu'à ce qu'on arrive au point d'arrivée ou jusqu'à ce que tous les sommets aient été explorés.

Q4. Appliquer l'algorithme de Dijkstra au graphe précédent, décrire les différentes étapes. Pour cela, recopier et remplir le tableau ci-dessous.
En déduire les chemins les plus courts depuis 0 vers chaque sommet.

Étape	0	1	2	3	4
0	**0**	∞	∞	∞	∞
1	$\times$				
2	$\times$				
3	$\times$				
4	$\times$				

Pour écrire le programme, nous allons utiliser un dictionnaire dont les clés sont les sommets et les valeurs sont les distances respectives entre chaque sommet et le sommet de départ, comme dans le tableau complété précédemment.

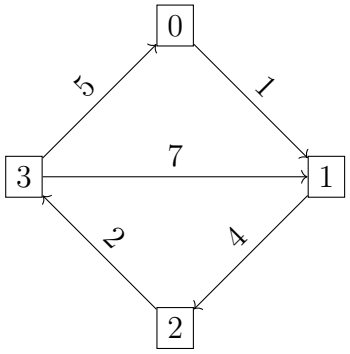
Q5. Nous avons pour cela besoin d'une fonction qui calcule le minimum des valeurs contenues dans un dictionnaire et renvoie la clé correspondante. On utilise la valeur `inf` (infini) de type `float`.
Compléter la fonction `minimum` sur le **document réponse**.

Q6. Compléter le programme `dijkstra` du **document réponse** mettant en œuvre l'algorithme décrit précédemment.

III Algorithme de Floyd-Warshall

Dans la suite, on s'intéresse au problème suivant : déterminer tous les chemins de poids total minimal du sommet i vers le sommet j pour tout couple (i, j) de sommets.
Cela pourra permettre à tout.e étudiant.e de connaître le chemin minimal pour aller rendre visite à chaque autre étudiant.e de la classe.

Q7. On considère le petit graphe ci-contre.
(a) Quels sont les deux chemins envisageables reliant 3 à 1 ? Indiquer pour chacun d'eux la distance associée ?
(b) Même question pour le trajet de 2 à 1.
(c) En quoi ces deux questions font apparaître un chevauchement de sous-problèmes ?



Q8. Pour résoudre le problème, on construit une matrice `D` initialisée par la matrice d'adjacence du graphe.
(a) Remplir la matrice `D` pour l'exemple du petit graphe étudié précédemment.

	0	1	2	3
0				
1				
2				
3				

(b) On considère à présent les trajets de `i` à `j` passant par le nœud (0). Sur que trajet `i` → `j`, la distance est-elle raccourcie ?
(c) Modifier alors une case de la matrice `D`, de sorte à ce que chaque cellule `(i,j)` indique toujours la valeur du chemin le plus court de `i` à `j`.

	0	1	2	3
0				
1				
2				
3				

Comment cette cellule est-elle calculée à partir des autres coefficients de la matrice ?

Pour tout $k \in \{0, \dots, n\}$, on note $D^{(k)}$ la matrice telle que $D^{(k)}_{i,j}$ est le poids d'un chemin de poids minimal de i vers j avec comme éventuels sommets intermédiaires uniquement des sommets entre 0 et $k-1$.
En particulier on a :

- $D^{(1)}_{i,j}$ est le minimum entre le poids de l'arc $i \to j$ (s'il existe) et le poids du chemin $i \to 0 \to j$ (s'il existe) ;
- $D^{(2)}_{i,j}$ est le poids d'un chemin de poids minimal qui part de i, arrive sur j, et ne peut avoir comme sommets intermédiaires que 0 et 1 (éventuellement un seul des deux ou aucun des deux).
- ...
- $D^{(n)}_{i,j}$ est le poids d'un chemin de poids minimal de i vers j, pouvant passer par n'importe quel sommet intermédiaire.

La matrice $D^{(n)}$ est donc la solution de notre problème, et les autres $D^{(k)}$ sont nos sous-problèmes.

Q9. Que vaut $D^{(0)}$ la matrice des distances minimales quand on passe par aucun sommet intermédiaire ?

Q10. On cherche la relation de récurrence entre $D_{i,j}^{(k+1)}$ en fonction des valeurs de $D_{p,q}^{(k)}$.
Il y a deux possibilités pour ce chemin minimal qui part du sommet `i` arrive en `j` et passe par des sommets de numéros inférieurs ou égaux à `k` :

(a) Soit il passe par le sommet k avec le parcours entre i et k puis k et j tous les deux minimaux.
Comment s'exprime la distance de ce parcours en fonction de $D_{i,k}^{(k)}$ et $D_{k,j}^{(k)}$?

(b) Soit il ne passe pas par le sommet k mais seulement par les précédents. Comment s'exprime $D_{i,j}^{(k+1)}$ en fonction de $D_{i,j}^{(k)}$?

(c) Comment choisir entre les deux valeurs précédentes ?

(d) En déduire la relation de récurrence écrite sous la forme :

$$D_{i,j}^{(k+1)} = \ldots\ldots\left(\ldots\ldots\ldots\ ,\ \ldots\ldots\ldots\ldots\ldots\ldots\right)$$

Q11. Compléter la fonction `matriceSuivante(D:list[list],k:int)->list[list]` **sur le document réponse** qui, si on lui donne comme argument une matrice `D` (liste de liste) correspondant à $D^{(k)}$, ainsi que `k`, et renvoie comme résultat la matrice (liste de liste) correspondant à $D^{(k+1)}$.

Q12. Écrire une fonction `FloydWarshall(M:list[list])->list[list]` qui prend comme argument la matrice `M` d'adjacence d'un graphe et renvoie la matrice `P` telle que $P_{i,j}$ donne le poids minimal d'un chemin de i vers j. On utilisera la fonction `matriceSuivante(D,k)`.

IV Base de données *(Pour le 11/12/2024 au plus tard)*

On considère la base de données suivante :

- La table `communes` contient les informations sur les communes en 2019 : l'identifiant de la commune `com` (clé primaire), l'identifiant de son département `dep` (clé étrangère) et le nom de la commune `nom`.
- La table `departements` contient les informations sur les départements en 2019 : l'identifiant du département `dep` (clé primaire), l'identifiant de sa région `reg` (clé étrangère), l'identifiant `com` de la commune chef-lieu du département (clé étrangère) et le nom du département `nom`.
- La table `regions` contient les informations sur les régions en 2019 : l'identifiant de la région `reg` (clé primaire), l'identifiant de la commune chef-lieu de région (clé étrangère) `com` et le nom de la région `nom`.
- La table `demographie` contient des informations sur la démographie. L'identifiant de la commune `com` (clé étrangère), la répartition de la population en 3 classes d'âge en 2016 (`pop24` pour les moins de 25 ans, `pop2564` pour ceux entre 25 et 64 ans et `pop65` pour les plus de 65 ans), le nombre de naissances `naissances` en 2018 et le nombre de décès `deces` en 2018.

Dans toute la suite on demande d'écrire une requête en SQL qui permet de répondre à la question posée.

Q13. Combien de communes s'appellent 'Saint-Paul' ?

Q14. Quels sont les noms des communes qui portent le même nom que leur département ?

Q15. Dans combien de régions différentes existe-t-il une commune s'appelant 'Sainte-Marie' ?

Q16. Quelles sont les communes (tous les attributs) qui ne sont pas des chefs-lieux de département ?

Q17. Quelle est la population de moins de 25 ans par département ?

Q18. Quels sont, dans l'ordre lexicographique croissant, les codes des communes avec strictement plus de naissances que la moyenne des naissances par commune ?

Q19. Quelles sont les communes sans aucun habitant ?

Q20. Quelles sont les 19 communes les moins peuplées, parmi celles qui comportent au moins un habitant renseigné ? On donnera le nom de la commune, le nom de son département et sa population.

Q21. Quelle est la proportion des moins de 25 ans en France, en pourcentage ?

Q22. Donnez les noms des communes qui sont utilisés par au moins deux communes, ainsi que le nombre de communes utilisant chacun de ces noms. La table sera triée par ordre décroissant suivant le nombre de communes, puis par ordre croissant des noms des communes.

DOCUMENT RÉPONSE À RENDRE

NOM : _____ Prénom : _____

Q5

```
def minimum(dico):
    """
    Argument :
    dico={clé:valeur,...} dico qui contient au moins une valeur différente de inf
    Retour :
    clé qui correspond de valeur minimale
    """
    mini=float('inf') # initialisation
    # suite à compléter :




    return
```

Q6

```
def dijkstra(G,s0):
    """
    arguments :
    G : dictionnaire qui représente le graphe
    s0 : sommet de départ
    retour :
    dictionnaire des distances minimales entre s0 et les différents sommets
    """
    sol={} # dictionnaire des sommets visités et de leurs distances minimales depuis s
    d={k:float('inf') for k in G} # initialisation des distances à l'infini des sommets restants à visiter
    d[s0]= ..... # distance 0 au  sommet s de départ
    while .....: # fini quand d est vide
        k= .....     # choix du sommet restant à visiter situé à la distance minimale
        for j ..... # visite tous les voisins de k
            # v un voisin de k, et c la distance entre k et v :
            v,c=.....
            if ..... # si v n'a pas déjà été atteint
                # distance minimale entre la précédente, et celle pour arriver à v depuis k (utiliser la fonction min(a,b)) :
                d[v]=.....
        sol[k]= ..... # copie le sommet et la distance dans distances
        del d[k]                  # supprime le sommet de d
    return sol
```

Q11

```
from copy import deepcopy
def matriceSuivante(D,k):
    Dsuiv=deepcopy(D) # copie en profondeur de D, permet de modifier Dsuiv sans modifier D
    n=len(D)
    for  i ..... : # parcours des lignes
        for j ..... : # parcours des colonnes
            dk = ..... # distance du parcours si pour aller de i à j on passe par k
            if ..... :  # si c'est plus court de passer par k que de ne pas y passer, pour aller de i à j
                Dsuiv[i][j] = ..... # modification de la distance minimale pour aller de i à j
    return Dsuiv
```