TP de Python numéro 11

*Semaine du **5 juin**.*

I. Graphes pondérés, distances pondérées

1. Définition, matrice d'adjacence pondérée

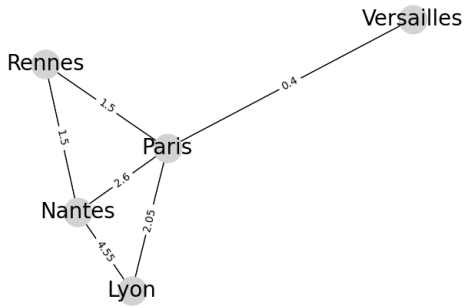
Un graphe pondéré est un graphe auquel on affecte un nombre réel à chaque arête, appelé poids de l'arête.

Définition 1. Soit $G = (S, A)$ un graphe (orienté ou non). On, appelle pondération sur G tout application

$$p : A \to \mathbb{R}.$$

On appelle graphe pondéré la donnée (G, p) d'un graphe G et d'une pondération p sur G. Si a est une arête d'un graphe pondéré (G, p), $p(a)$ est appelé le poids de l'arête a.

Exemple 2. La représentation graphique ci-dessous représente un graphe pondéré (G, p).



Le graphe $G = (S, A)$ a pour sommets "Paris", "Lyon", "Nantes", "Rennes" et "Versailles".

La pondération p vérifie par exemple :

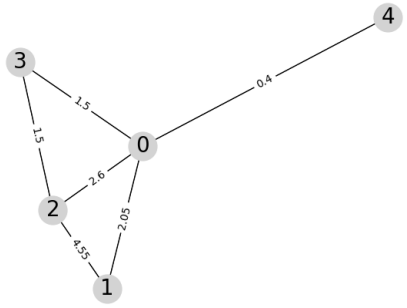
$$p(\{\text{“}Paris\text{”}, \text{“}Versailles\text{”}\}) = 0,4 \text{ et } p(\{\text{“}Rennes\text{”}, \text{“}Nantes\text{”}\}) = 1,5.$$

Dans la suite de ce TP, on ne considérera que des graphes **simples et non orientés** pour simplifier l'exposé.

Définition 3. Soit (G, p) un graphe pondéré (non orienté). Notons n l'ordre de G, $s_1, \dots s_n$ les sommets numérotés de G et A l'ensemble des arêtes de G.
On appelle matrice d'adjacence pondérée du graphe pondéré (G, p) la matrice $M = (m_{i,j})_{1 \leq i,j \leq n}$ donnée par :

$$\forall (i, j) \in [\![1, n]\!]^2, m_{i,j} = \begin{cases} p(\{s_i, s_j\}) \text{ si } \{s_i, s_j\} \in A \\ 0 \text{ sinon} \end{cases}.$$

Exemple 4. La matrice d'adjacence du graphe pondéré (G, p) représenté ci-dessous, où le sommets sont numérotés dans l'ordre des entiers naturels,



est :

$$\begin{pmatrix} 0 & 2,05 & 2,6 & 1,5 & 0,4 \\ 2,05 & 0 & 4,55 & 0 & 0 \\ 2,6 & 4,55 & 0 & 1,5 & 0 \\ 1,5 & 0 & 1,5 & 0 & 0 \\ 0,4 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Remarque.

- Lorsque la pondération p s'annule, cette matrice ne résume pas toute l'information du graphe pondéré considéré (un 0 doit-il être interprété comme une absence d'arête ou comme une pondération nulle?)
- Dans ce TP, on s'intéressera aux graphes pondérés dont la pondération représente une "distance", en un certain sens.
- Pour ces raisons, **dans toute la suite de ce TP**, on considérera des graphes pondérés (G, p) dont la pondération p est à valeurs dans $\mathbb{R}_+^*$. La matrice d'adjacence pondérée d'un tel graphe permet alors de retrouver entièrement le graphe pondéré considéré, et sera donc **la modélisation informatique qu'on gardera** des graphes pondérés.

Exercice 5.

1. Définir en Python la matrice d'adjacence du graphe considéré dans l'exemple 4.
2. Écrire une fonction Python `oubliePonderation` prenant en entrée la matrice d'adjacence pondérée d'un graphe pondéré et renvoyant en sortie la matrice d'adjacence du graphe (non pondéré) sous-jacent. La tester avec la matrice de la question précédente.
3. Écrire une fonction `poidsMax` prenant en entrée la matrice d'adjacence pondérée d'un graphe et renvoyant en sortie le maximum des poids des arêtes de ce graphe (et 0 si ce graphe n'a pas d'arête). Tester cette fonction sur le graphe de l'exemple 4.

Convention

Dans la suite, les conventions adoptées sont les suivantes :

- Les graphes considérés sont simples et non orientés,
- les pondérations considérées sont à valeurs strictement positives,
- pour les modélisations informatiques, les graphes considérés ont pour sommets les entiers de 0 à $n-1$ (où n étant l'ordre du graphe considéré),
- les graphes pondérés considérés sont représentés en Python par leur matrice d'adjacence pondérée, avec la numérotation des sommets induite par la convention précédente.

2. Distance de graphe

Définition 6. Soit G un graphe, soient s et s' deux sommets de G. On appelle distance de graphe de s à s', et on note $d(s,s')$, la longueur de la plus courte chaîne reliant s à s' si s et s' sont reliés par une chaîne, et on pose $d(s,s') = +\infty$ si s et s' ne sont pas reliés par une chaîne.

Exemple 7. Dans le graphe G de l'exemple 4, en oubliant la pondération pour ne considérer que le graphe sous-jacent :

- la distance de graphe du sommet 1 au sommet 3 est de 2,
- la distance de graphe du sommet 1 au sommet 0 est de 1.

L'infini en Python

Le module numpy permet de manipuler un symbole représentant l'infini.
Suite à l'import `import numpy as np`, la commande `np.inf` renvoie une quantité représentant $+\infty$, de type `float`, se comportant de la manière suivante :

- Pour tout nombre `a`, ou si `a` est `np.inf`, `a+ np.inf` renvoie `np.inf`.
- `a < np.inf` renvoie `True` pour tout nombre `a` et `False` si `a` est `np.inf`.
- `a <= np.inf` renvoie `True` dès que `a` est un nombre ou `np.inf`.

Exemple 8. On peut utiliser la matrice d'adjacence d'un graphe pour déterminer la distance entre deux de ses sommets (voir TP9). L'idée est que pour un graphe d'ordre n, deux sommets reliés par une chaîne sont reliés par une chaîne de longueur au plus $n-1$. De plus, les puissances de la matrice d'adjacence d'un graphe permettent de connaitre le nombre de chaîne d'une longueur voulue.

Voici le code d'une fonction `distance` prenant en entrée la matrice d'adjacence `M` d'un graphe ainsi que les numéros `i` et `j` de deux de ses sommets, et renvoyant en sortie la distance de graphe du sommet `i` au sommet `j`.

```
import numpy as np
def distance(M,i,j):
    n,_=np.shape(M)
    A=np.eye(n)
    for k in range(n):
        if A[i,j]!=0:
            return(k)
        A=np.dot(A,M)
    return(np.inf)
```

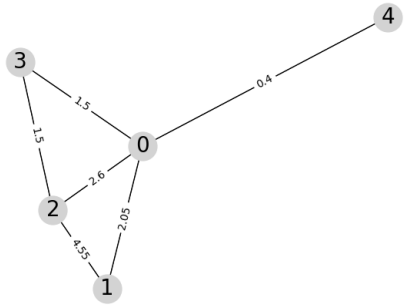
Remarque. Dans ce code,

- `n` est l'ordre du graphe,
- `A` contient, une à une, les puissances $M^0, M^1, \ldots, M^{n-1}$ de la matrice d'adjacence donnée,
- la boucle renvoie le plus petit entier k pour lequel on détecte une chaîne de longueur k reliant les deux sommets donnés,
- la fonction renvoie `np.inf` si aucun chemin de longueur au plus `n-1` n'a été trouvé.

3. Distance pondérée

Lorsque la pondération d'un graphe pondéré représente par exemple une distance ou un coût, il devient plus intéressant de rechercher, entre deux sommets donnés, une chaîne dont la somme des poids des arêtes parcourues est minimale.

Exemple 9. Reprenons le graphe pondéré (G, p) de l'exemple 4.



Si l'on estime que ces pondérations représentent un temps de trajet, il est strictement plus intéressant, pour aller du sommet 1 au sommet 3, de passer par le sommet 0 (temps total : 2,05+1,5=3,55) que de passer par le sommet 2 (temps : 6,05).

Définition 10. Soit (G, p) un graphe pondéré. Soit c une chaîne du graphe G. On appelle poids total de la chaine c la somme des poids des arêtes parcourues par c.
Plus précisément, si $c = (s_0, \ldots, s_n)$ est une chaîne de G, alors le poids total de cette chaîne c est :

$$\sum_{i=0}^{n-1} p(\{s_i, s_{i+1}\}).$$

En particulier, le poids total d'une chaîne de longueur 0 est 0.

Exemple 11. La chaine $1 - 0 - 4 - 0 - 2$ a pour poids total :

$$2,05 + 0,4 + 0,4 + 2,6 = 5,45.$$

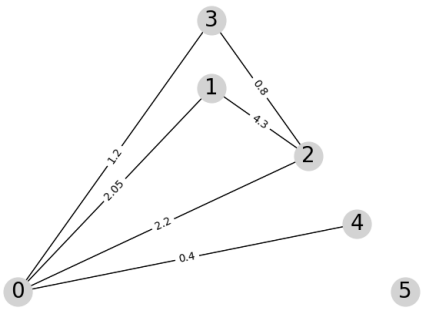
Chaque parcours d'une arête est pris en compte.

Définition 12. Soit (G, p) un graphe pondéré. Soient s et s' deux sommets de G.
On appelle distance pondérée de s à s' :

- Le plus petit poids total d'une chaîne reliant s et s', si s et s' sont reliés par une chaîne,
- $+\infty$ si s et s' ne sont pas reliés par une chaîne.

En particulier, la distance pondérée d'un sommet à lui même est nulle.

Exemple 13. Considérons le graphe pondéré ci-contre. Déterminons, pour les deux notions de distances introduites, la distances des sommets au sommet 2.



Pour la **distance de graphe :**

Sommet :	0	1	2	3	4	5
Distance à 2 :						

Pour la **distance pondérée :**

Sommet :	0	1	2	3	4	5
Distance à 2 :						

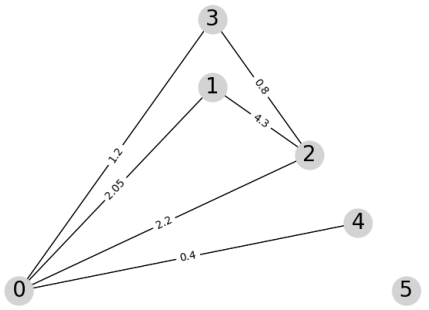
II. L'algorithme de Dijsktra

Pour déterminer les distances pondérées entre les sommets d'un graphe pondéré, on ne peut plus simplement utiliser la matrice d'adjacence, comme pour la distance de graphe.

À la place, on peut utiliser l'algorithme de Dijkstra.

Étant donné un graphe pondéré (G, p) et un sommet s de G, l'algorithme de Dijsktra construit la liste des distances pondérées des sommets de G au sommet s.

Exercice 14. Reprenons le graphe pondéré précédent. Remplir à la main le tableau ci-contre des distances pondérées des sommets de ce graphe au sommet 0.



Sommet	0	1	2	3	4	5
Distance à 0						

1. L'algorithme sur un exemple

Prenons l'exemple du graphe G ci-dessus et du sommet $s = 0$, pour comprendre cet algorithme.

Le principe est le suivant :

- On construit le **tableau des distances** au sommet s choisi, dont le remplissage va évoluer au cours de l'algorithme jusqu'à être correctement rempli à la fin. Pour cette raison, on parlera des ***distances connues*** pour désigner, au cours de l'algorithme, les valeurs présentes dans ce tableau.
- L'algorithme de Disjktra est constitué d'**explorations** successives de sommets :
  - Les sommets ne seront explorés qu'une fois, donc à la fin de l'exploration d'un sommet, on va le ***marquer*** pour ne pas y revenir.
  - Si, à la fin d'une exploration, il reste des sommets non marqués dont la distance connue au sommet s est finie, on procède à une exploration supplémentaire.
  - Sinon, l'algorithme est terminé et le tableau des distances connues est renvoyé par l'algorithme.
- Lors des étapes d'exploration d'un sommet :
  - On commence par sélectionner le sommet à explorer. Il s'agit d'un sommet non marqué dont la distance connue au sommet s est minimale.
  - Notons v le sommet en cours d'exploration. Cette exploration consiste à :
    - Établir la liste des voisins non marqués de v, puis
    - pour chacun de ces voisins v' de v, comparer la distance connue de s à v', et la distance qu'on obtiendrait entre ces sommets en allant de s à v, puis de v à v' par l'arête entre v et v'. Si la distance obtenue de s à v' "en passant par v" est inférieure à la distance connue de s à v', on met à jour notre tableau avec cette distance.

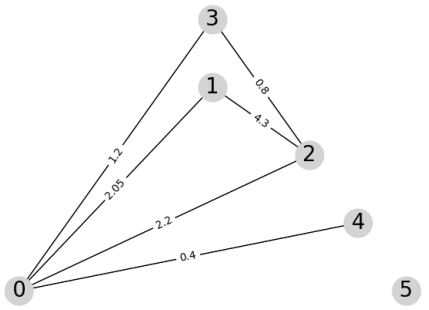
Reprenons notre exemple précédent, toujours avec $s = 0$.

Dans l'illustration de l'algorithme ci-dessous :

- **Les sommets en cours d'exploration** seront notés par le caractère `>`.
- **Les sommets marqués** seront notés par le caractère `x`.

État initial :

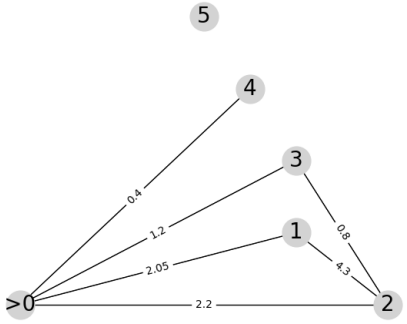
Aucun sommet n'a été exploré, et la liste des distances connues est pauvre : le sommet 0 est à distance 0 de lui-même, et, le graphe n'ayant pas encore été exploré, il est à distance à priori infinie des autres (aucune chaîne n'a été trouvée à ce moment). Aucun sommet n'est marqué initialement.



Sommet	0	1	2	3	4	5
Distance connue à 0	0	∞	∞	∞	∞	∞
État						

Exploration d'un sommet :

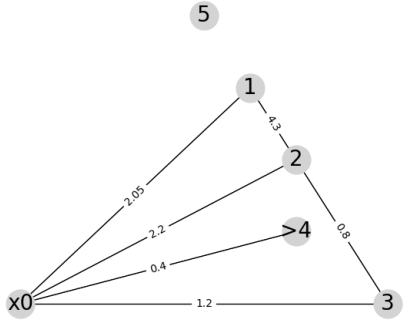
- On choisit le sommet non marqué dont la distance connue au sommet choisi est minimale et non infinie. Ici, ce sommet est 0.
- On explore ce sommet :
  - Les voisins non marqués de 0 sont $1, 2, 3$ et 4.
  - Pour le sommet 1, la distance connue est infinie, alors qu'en allant de 0 à 0 (distance connue : 0) puis de 0 à 1 par l'arête $\{0, 1\}$ (poids $2,05$), on obtient une chaîne de poids total $2,05$ inférieure. On inscrit donc $2,05$ comme distance connue de 0 à 1.
  - On fait de même pour les sommets $2, 3$ et 4.
  - En fin d'exploration, on marque le sommet 0.



Sommet	0	1	2	3	4	5
Distance connue à 0	0	2,05	2,2	1,2	0,4	∞
État	$>$					

Exploration suivante: Il reste des sommets non marqués à distance connue finie de 0.

Le sommet non marqué à distance minimale et finie de 0 est le sommet 4, on l'explore. 4 n'ayant pas d'autres voisins que 0, qui est déjà marqué, il ne se passe rien à part qu'on le marque.

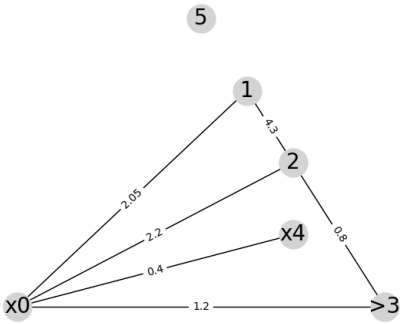


Sommet	0	1	2	3	4	5
Distance connue à 0	0	2,05	2,2	1,2	0,4	∞
État	$\times$				$>$	

Exploration suivante : Le sommet non marqué à distance minimale et finie de 0 est le sommet 3, on l'explore. Ici, $3-2$ est une arête de poids $0,8$. Pour la prendre en compte dans nos distances connues, on compare :

- La distance connue de 0 à 2,
- La distance qu'on obtiendrai pour aller de 0 à 2 en allant au sommet 3 (par la distance connue), puis en empruntant cette arête.

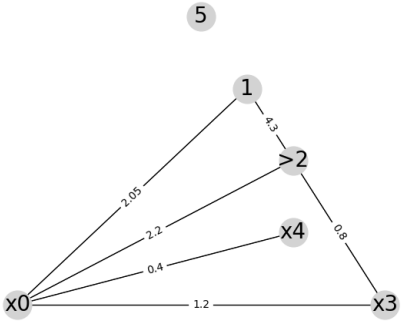
Ici, la distance connue 2,2 est supérieure à la somme 1,2+0,8 : ce chemin est meilleur.



Sommet	0	1	2	3	4	5
Distance connue à 0	0	2,05	2	1,2	0,4	∞
État	$\times$			$>$	$\times$	

Exploration : Le sommet non marqué à distance minimale et finie de 0 est le sommet 2, on l'explore.

- 2+4.3 est supérieur à la distance connue de 0 à 1, donc la distance connue de 0 à 1 ne change pas.
- Le sommet 3 étant marqué, on ignore l'arête reliant 2 à 3.

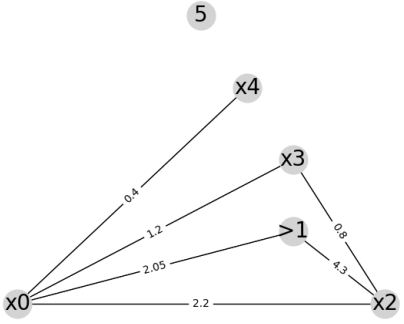


Sommet	0	1	2	3	4	5
Distance connue à 0	0	2,05	2	1,2	0,4	∞
État	$\times$		$>$	$\times$	$\times$	

Exploration : Le sommet non marqué à distance minimale et finie de 0 est le sommet 1, on l'explore.

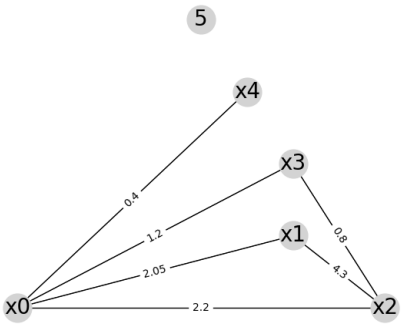
- 0 est marqué, donc on ignore l'arête reliant 1 à 0.
- 2 est marqué, donc on ignore l'arête de 1 à 2.

Les distances connues ne changent pas, on marque simplement le sommet 1.



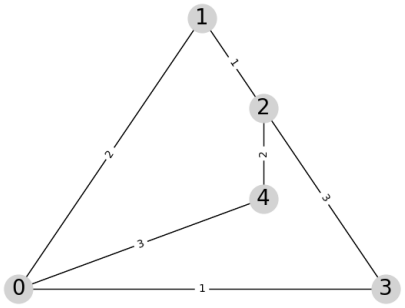
Sommet	0	1	2	3	4	5
Distance connue à 0	0	2,05	2	1,2	0,4	∞
État	$\times$	$>$	$\times$	$\times$	$\times$	

Fin de l'algorithme : Tous les sommets dont la distance connue à 0 est finie sont marqués : c'est la fin de l'algorithme, et la liste des distances connues est la liste des distances pondérées (théorème admis : cet algorithme est correcte).



Sommet	0	1	2	3	4	5
Distance connue à 0	0	2,05	2	1,2	0,4	∞
État	×	×	×	×	×	

Exemple 15. Appliquons à la main l'algorithme de Dijkstra pour le graphe pondéré ci-dessous afin de déterminer la liste des distances pondérées des sommets au sommet 2.



2. L'algorithme de Dijkstra en Python

Pour mener à bien l'algorithme de Dijsktra, on utilise deux variables intermédiaires :

- Une liste `DistC` comportant la liste des distances connues au sommet choisi.
- Une liste `Marques` comportant la liste des sommets déjà explorés.

Le code Python aura la structure suivante.

```
import numpy as np
def algoDijkstra(A,s):
    n,_=np.shape(A)
    DistC=[np.inf]*n
    DistC[s]=0
    Marques=[]

    while fonction1(DistC,Marques):
        #Sommet à explorer
        explore=function2(DistC,Marques)

        #Exploration
        DistC=fonction3(A,DistC,Marques,explore)

        #Marquage du sommet exploré
        Marques.append(explore)
    #fin de boucle :
    return(DistC)
```

Exercice 16. 1. Que doivent renvoyer les fonctions `fonction1, fonction2, fonction3` afin que le code ci-dessus réalise l'algorithme de Dijsktra à partir de la matrice d'adjacence pondérée `A` d'un graphe pondéré et le numéro `s` de l'un de ses sommets ?

2. Coder ces fonctions, puis tester l'algorithme de Disjktra pour le graphe de l'exemple précédent.

Finalement, voici une implémentation de cet algorithme en Python.

- Entrée : La matrice d'adjacence pondérée `A` d'un graphe pondéré (G, p), et le numéro s d'un sommet.
- Sortie : la liste `D` des distances pondérées de s aux sommets de G : `D[i]` est la distance pondérée du sommet numéro `i` au sommet numéro `s`.

```python
import numpy as np
def algoDijkstra(A,s):

    #ordre du graphe
    n,_=np.shape(A)

    #Etat initial
    DistC=[np.inf]*n
    DistC[s]=0
    Marques=[]

    #Condition d'exploration : il reste des sommets non marqués à distance finie
    while [i in range(n) if not(i in Marques) and not(DistC[i]==np.inf)] !=[]:

        #Détermination du sommet à explorer : distance minimale parmi les non marqués
        dmin=np.inf
        for k in range(n):
            if DistC[k]<dmin and not(k in Marques):
                dmin=DistC[k]
                explore=k
        #Maintenant, explore contient le sommet à explorer.

        #Liste des voisins du sommet exploré à prendre en compte
        Voisins = [v for v in range(n) if not(v in Marques) and A[explore,v]!=0]

        #Mise à jour des distances connues
        for v in Voisins :
            DistC[v]=min(DistC[v],DistC[explore]+A[explore,v])

        #Marquage du sommet exploré
        Marques.append(explore)

    #fin de boucle :
    return(DistC)
```