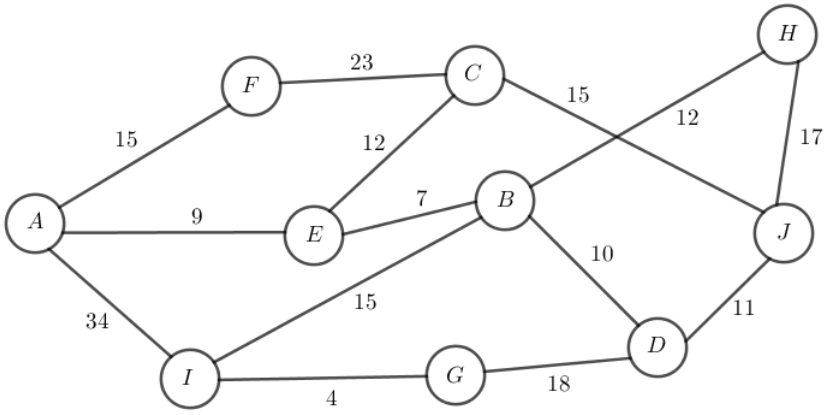
TP 10 - Graphes - Partie 2
Algorithme de Dijkstra

I) Présentation du problème et de l'algorithme

On considère un graphe pondéré, c'est à dire un graphe dans lequel des poids sont placés sur chaque arête. Il s'agit d'aller d'un sommet à un autre en utilisant le plus court chemin (au sens des poids).

Par exemple, considérons le graphe suivant, dont les villes sont représentées par les sommets de A à J. Les poids représentent les distances en kilomètres, et on souhaite déterminer le plus court chemin menant de A à J.



Voici le principe de l'algorithme :

- Au départ, les distance entre le sommet de départ et les autres sommets sont considérées comme infinies, et la distance entre le sommet de départ et lui-même vaut 0.
- A chaque étape, on sélectionne le sommet dont la distance au sommet de départ est la plus petite, et on met à jour les distances des sommets qui lui sont adjacents. Pour ce faire, on calcule la somme de la distance du sommet sélectionné et du poids de l'arête entre ce sommet et le voisin considéré :
  - ☐ si le total est inférieur à la distance existante on le garde ;
  - ☐ sinon on garde la distance existante.
- On s'arrête lorsque le sommet de sortie a été atteint.

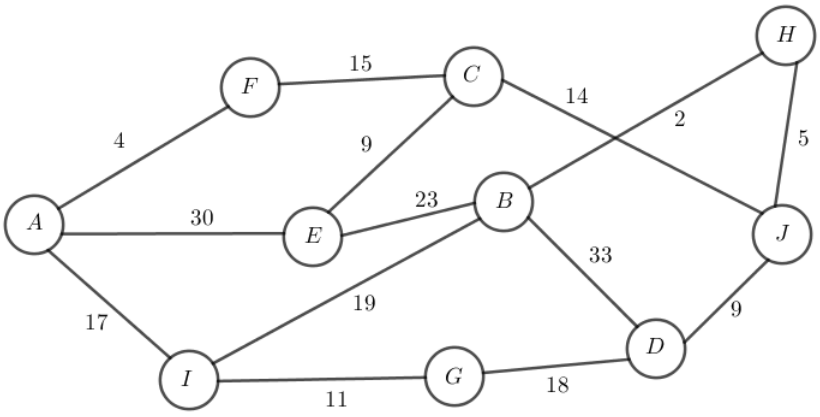
Pour mettre en œuvre l'algorithme, utilisons un tableau dans lequel une ligne donne les distances en cours des sommets depuis le sommet de départ, et une colonne donne l'évolution de la distance d'un sommet au sommet de départ. On pourra éventuellement barrer les distances trop grandes.

A	B	C	D	E	F	G	H	I	J	Choix

En conclusion :

- Le plus court chemin de A à J est donc :
- La distance de ce plus court chemin est :

Exercice. Appliquer l'algorithme de Dijkstra pour déterminer, dans le graphe pondéré suivant, le chemin le plus court pour aller du point C au point D.



III) Implémentation en Python

Commençons par décrire l'algorithme. Un graphe pondéré sera représenté par une matrice d'adjacence M, dans lequel le coefficient $M_{i,j}$ vaudra le poids de l'arête reliant i à j (et 0 s'il n'y a pas d'arête reliant i à j). On supposera donc les sommets numérotés à partir de 0.

On utilisera de plus :

- Deux variables So et Sf contenant respectivement les sommets initiaux et finaux.
- Une variable Sd contenant la liste des sommets disponibles.
- Une variable St permettant de savoir à quelle étape le sommet final a été trouvé.
- Une variable représentant l'infini, `np.inf` (en ayant importé `Numpy` avec le raccourci `np`). Cette variable renvoie `True` à tout test du type `x<np.inf`, où x est une variable de type `int` ou `float`.
- Une matrice T de type `Numpy`, de même taille que M. Cette matrice servira à représenter les distances dans le tableau présenté dans la partie I). Chaque coefficient $T[i,j]$ contiendra la distance de So à j à la $i-$ème étape.
- Une matrice U de type `Numpy`, de même taille que M. Cette matrice servira à représenter les sommets dans le tableau présenté dans la partie I). Chaque coefficient $U[i,j]$ contiendra le numéro du sommet qui précède j dans le chemin de So à Sf.
- Une variable $Dmin$ qui servira à stocker la distance minimale courante, et une variable $Smin$ pour le sommet correspondant.
- Une variable *chemin* qui contient une liste initialisée à Sf ; une variable s qui va contenir les sommets successifs du plus court chemin.

Voici une façon de procéder :

- Construire Sd, T et U. On initialisera T avec des nombres de type `float` (par exemple avec `np.zeros`, que l'on pourra aussi utiliser pour U).
- Initialiser une variable i à 0 (indice de l'étape courante).
- Remplacer $T[0,So]$ par 0 et $T[0,k]$ par `np.inf` pour $k \neq Si$. Remplacer $U[0,So]$ par So.
- Enlever So de Sd.
- Tant qu'il reste des sommets disponibles :
  - ▷ Chercher le minimum de la liste $T[i]$, stocker la distance dans une variable $Dmin$ et le numéro du sommet dans une variable $Smin$.
  - ▷ Supprimer $Smin$ de Sd.
  - ▷ Incrémenter i de 1.
  - ▷ Dans la colonne $Smin$ de T et de U, remplacer les valeurs après la ligne i par `np.inf`.
  - ▷ Chercher les sommets adjacents de $Smin$ à l'aide de la matrice M, et pour chaque tel sommet j :
    - ☐ Si la longueur du chemin le plus court chemin reliant So à $Smin$ à laquelle on ajoute la distance de $Smin$ à j est inférieure à la longueur du plus court chemin de So à j : placer la somme des distances dans $T[i,j]$ et $Smin$ dans $U[i,j]$.
    - ☐ Sinon, mettre $T[i-1,j]$ dans $T[i,j]$ et $U[i-1,j]$ dans $U[i,j]$.
- Afficher la longueur du plus court chemin de So à Sf, qui se trouve dans la ligne $St-1$ et la colonne Sf de T.
- Initialiser la variable s au sommet précédant Sf dans le plus court chemin, il se trouve dans la ligne $St-1$ et la colonne Sf de U.
- Initialiser la variable `chemin` avec une liste contenant Sf puis s.
- Tant que s n'est pas So :
  - ▷ Remonter la colonne de s tant qu'on trouve la valeur `np.inf`.
  - ▷ Mettre à jour s et l'ajouter à la variable *chemin*.
- Donner la liste des chemins dans l'ordre.

Recopier et compléter le code ci-dessous, puis l'appliquer sur les deux graphes précédents pour retrouver ce qui a été fait en classe, et faire ensuite des essais supplémentaires.

```
def Dij(M,So,Sf):
    St = 0
    n = len(M)
    T = np.zeros((n,n))
    U = np.zeros((n,n))
    Sd = list(range(n))
    i = 0

    for k in range(n): #Remplissage de la première ligne de T de U
        T[0,k] = ..............
    T[0,So] = ..............
    U[0,So] = ..............

    while Sd!=[] and i<n-1: #Remplissage des autres lignes de T et U
        Dmin = np.min(T[i])
        Smin = np.argmin(T[i])  #Position du minimum
        Sd.remove(Smin)
        i = i+1
        if Smin == Sf : St = i
        for k in ..............:
            T[k,Smin] = ..............
            U[k,Smin] = np.inf
        for j in Sd:
            if M[Smin,j]!=0 and T[i-1,Smin]+M[Smin,j]<T[i-1,j]:
                T[i,j] = ..............
                U[i,j] = ..............
            else:
                T[i,j] = ..............
                U[i,j] = ..............

    i = St-1
    lg = T[i,Sf]
    print('Le plus court chemin est de longueur : ',lg)
    s = int(U[i,Sf])
    chemin = ..............
    while ..............: #Parcours de T pour chercher le plus court chemin
        while T[i,int(s)]==np.inf:
            i = i-1
        s = int(U[i,int(s)])
        chemin.append(..............)
    chemin = list(reversed(chemin))
    print('Le plus court chemin est : ',chemin)
    return (lg,chemin)
```

Source : Mathématiques appliquées et informatique - ECG 1ère année - 4ème édition - Ellipses (modifié).