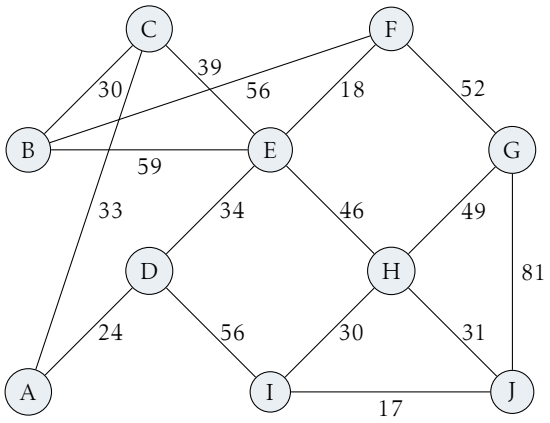
Python - TP 9
Graphes : algorithme de Dijkstra

Objectifs de ce TP :

- utiliser l'algorithme de Dijkstra pour déterminer les plus courts chemins entre un sommet et tous les autres sommets du graphe,
- puis utiliser ce qui précède pour déterminer le plus court chemin entre un sommet de départ et un sommet d'arrivée bien définis.

On considère le graphe suivant :



1. Récupérer la matrice des poids associée à ce graphe.
2. Écrire la liste des sommets dans une chaîne de caractères nommée `S`.
3. Définir une fonction `initialisation(s)` qui prend en argument d'entrée un sommet $s \in S$ et renvoie trois listes à n éléments (où n est l'ordre du graphe) :
   - la liste `LDM` (liste des distances minimales) contenant les distances de s aux autres sommets, initialisée à 0 pour s et ∞ pour les autres sommets,
   - la liste `LSC` (liste des sommets choisis) contenant `True` si le sommet est choisi, `False` sinon ; elle est initialisée à `False` partout, et `True` pour le sommet s,
   - la liste `LP` (liste des prédécesseurs) ne contenant initialement que `None`.
4. Compléter le programme suivant de sorte que l'appel de `dijkstra(M,s)` renvoie les listes `LDM,LSC,LP` à la fin de l'exécution de l'algorithme de Dijkstra sur le graphe dont M est la matrice des poids en partant du sommet `s`.

```
import numpy as np
inf=np.inf
S='ABCDEFGHIJ' #liste des sommets

def initialisation(s):
    depart=S.index(s)
    ......
    ......
    ......
    ......
    ......
    return LDM,LSC,LP

def dijkstra(M,s):
    n=len(S) #n=ordre du graphe
    LDM,LSC,LP=initialisation(s)
    rang_choisi=......
    dist_choisi=0 #dist entre s et le sommet choisi
    for nb in range(1,n):
        minimum=...
        for k in ...... #k=rang du sommet exploré
            if ......
                dist_prov=......
                if ......
                    LDM[k]=......
                    LP[k]=......
                if ......
                    minimum=......
                    rang_prochain_choisi=......
        rang_choisi=......
        LSC[rang_choisi]=......
        dist_choisi=......
    return LDM,LSC,LP
```

Petite remarque

On note au passage la structure d'une matrice en `Python` : une liste de listes dans un environnement `np.array()` pour que chaque liste soit écrite sur une nouvelle ligne...

5. Écrire une fonction `dijkstra_pcc(M,s_depart,s_arrivee)` qui renvoie le plus court chemin pour aller de `s_depart` à `s_arrivee` ainsi que sa longueur.

En guise de vérification des programmes :

- `initialisation('A')` renvoie : `([0, inf, inf, inf, inf, inf, inf, inf, inf, inf], [True, False, False, False, False, False, False, False, False, False], [None, None, None, None, None, None, None, None, None, None])`
- `dijkstra(Mpoids,'A')` renvoie `([0, 63.0, 33.0, 24.0, 58.0, 76.0, 128.0, 104.0, 80.0, 97.0], [True, True, True, True, True, True, True, True, True, True], [None, 'C', 'A', 'A', 'D', 'E', 'F', 'E', 'D', 'I'])`
- `dijkstra_pcc(Mpoids,'A','G')` renvoie `(['A', 'D', 'E', 'F', 'G'], 128.0)`

```python
#Il est indispensable d'avoir parfaitement compris le principe
#de l'algorithme sur papier pour espérer comprendre le programme...

#L47->L49 : on regarde la distance minimale sur la ligne que l'on vient de compléter
# puis on choisit le sommet de + petite distance comme nouveau sommet pour la suite


import numpy as np

inf=np.inf
Mpoids=np.array([
    [0,inf,33,24,inf,inf,inf,inf,inf,inf],
    [inf,0,30,inf,59,inf,inf,inf,inf,inf],
    [33,30,0,inf,39,56,inf, inf,inf,inf],
    [24,inf,inf,0,34,inf,inf,inf,56,inf],
    [inf,59,39,34,0,18,inf,46,inf,inf],
    [inf,inf,56,inf,18,0,52,inf,inf,inf],
    [inf,inf,inf,inf,inf,52,0,49,inf,81],
    [inf,inf,inf,inf,46,inf,49,0,30,31],
    [inf,inf,inf,56,inf,inf,inf,30,0,17],
    [inf,inf,inf,inf,inf,inf,81,31,17,0]])

S='ABCDEFGHIJ' #liste des sommets

def initialisation(s):
    depart=S.index(s) #rang du sommet initial
    LDM=[inf for x  in S] #liste des distances minimales
    LDM[depart]=0
    LSC=[False for x in S] #liste de sommets choisis : False si sommet pas choisi, True sinon
    LSC[depart]=True
    LP=[None for x in S] #liste des prédecesseurs
    return LDM,LSC,LP

def dijkstra(M,s):
    n=len(S) #n=ordre du graphe
    LDM,LSC,LP=initialisation(s)
    rang_choisi=S.index(s) #rang du sommet choisi
    dist_choisi=0 #dist entre s et le sommet choisi
    for nb in range(1,n):
        minimum=inf
        for k in range(n): #k=rang du sommet exploré
            if LSC[k]==False:
                dist_prov=dist_choisi+M[rang_choisi,k]
                if dist_prov<LDM[k]:
                    LDM[k]=dist_prov
                    LP[k]=S[rang_choisi]
                if LDM[k]<minimum:
                    minimum=LDM[k]
                    rang_prochain_choisi=k
        rang_choisi=rang_prochain_choisi
        LSC[rang_choisi]=True
        dist_choisi=LDM[rang_choisi]
    return LDM,LSC,LP
```