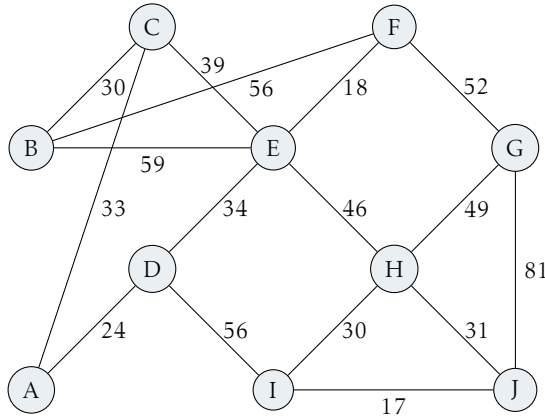


Objectifs de ce TP :

- 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  $\infty$  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.

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...

```

1 import numpy as np
2 inf=np.inf
3 S='ABCDEFGHJIJ' #liste des sommets
4
5 def initialisation(s):
6     depart=S.index(s)
7     .....
8     .....
9     .....
10    .....
11    .....
12    return LDM,LSC,LP
13
14 def dijkstra(M,s):
15     n=len(S) #n=ordre du graphe
16     LDM,LSC,LP=initialisation(s)
17     rang_choisi=.....
18     dist_choisi=0 #dist entre s et le sommet choisi
19     for nb in range(1,n):
20         minimum=...
21         for k in ..... #k=rang du sommet exploré
22             if .....
23                 dist_prov=.....
24                 if .....
25                     LDM[k]=.....
26                     LP[k]=.....
27                 if .....
28                     minimum=.....
29                     rang_prochain_choisi=.....
30     rang_choisi=.....
31     LSC[rang_choisi]=.....
32     dist_choisi=.....
33     return LDM,LSC,LP

```

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], [True, False, False, False, False, False, False, False, False], [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], [None, 'C', 'A', 'A', 'D', 'E', 'F', 'E', 'D', 'I'])`
- `dijkstra_pcc(Mpoids,'A','G')` renvoie `(['A', 'D', 'E', 'F', 'G'], 128.0)`

```

1 #Il est indispensable d'avoir parfaitement compris le principe
2 #de l'algorithme sur papier pour espérer comprendre le programme...
3
4 #L47->L49 : on regarde la distance minimale sur la ligne que l'on vient de compléter
5 # puis on choisit le sommet de + petite distance comme nouveau sommet pour la suite
6
7
8 import numpy as np
9
10 inf=np.inf
11 Mpoids=np.array([
12     [0,inf,33,24,inf,inf,inf,inf,inf,inf],
13     [inf,0,30,inf,59,inf,inf,inf,inf,inf],
14     [33,30,0,inf,39,56,inf,inf,inf,inf],
15     [24,inf,inf,0,34,inf,inf,inf,56,inf],
16     [inf,59,39,34,0,18,inf,46,inf,inf],
17     [inf,inf,56,inf,18,0,52,inf,inf,inf],
18     [inf,inf,inf,inf,inf,52,0,49,inf,81],
19     [inf,inf,inf,inf,46,inf,49,0,30,31],
20     [inf,inf,inf,56,inf,inf,inf,30,0,17],
21     [inf,inf,inf,inf,inf,inf,81,31,17,0]])
22
23 S='ABCDEFGHJI' #liste des sommets
24
25 def initialisation(s):
26     depart=S.index(s) #rang du sommet initial
27     LDM=[inf for x in S] #liste des distances minimales
28     LDM[depart]=0
29     LSC=[False for x in S] #liste de sommets choisis : False si sommet pas choisi, True sinon
30     LSC[depart]=True
31     LP=[None for x in S] #liste des predecesseurs
32     return LDM,LSC,LP
33
34 def dijkstra(M,s):
35     n=len(S) #n=ordre du graphe
36     LDM,LSC,LP=initialisation(s)
37     rang_choisi=S.index(s) #rang du sommet choisi
38     dist_choisi=0 #dist entre s et le sommet choisi
39     for nb in range(1,n):
40         minimum=inf
41         for k in range(n): #k=rang du sommet exploré
42             if LSC[k]==False:
43                 dist_prov=dist_choisi+M[rang_choisi,k]
44                 if dist_prov<LDM[k]:
45                     LDM[k]=dist_prov
46                     LP[k]=S[rang_choisi]
47                 if LDM[k]<minimum:
48                     minimum=LDM[k]
49                     rang_prochain_choisi=k
50         rang_choisi=rang_prochain_choisi
51         LSC[rang_choisi]=True
52         dist_choisi=LDM[rang_choisi]
53     return LDM,LSC,LP

```