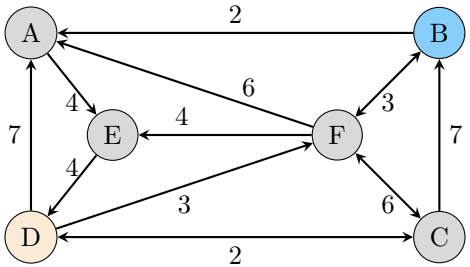
Informatique - TD 5 | Algorithme de Dijkstra

Le but de ce TD est d'étudier l'agorithme de Dijkstra qui permet de trouver le chemin le plus court entre deux points dans un graphe de chemins pondérés.

Description de l'algorithme :

Par exemple, considérons le graphe de sommets A, B, C, D, E, F ci-dessous :



On cherche par exemple le plus court chemin pour aller de B à D. L'algorithme de Dijkstra consiste à construire étape par étape le tableau "des chemins", constitué ici de 7 colonnes. (Une pour chaque sommet +1)

A	B	C	D	E	F	Départ
⋮	⋮	⋮	⋮	⋮	⋮	⋮

À chaque étape, les points dans la colonne "Départ" sont les points desquels on part et les cases en en-tête (grises) seront les points sur lesquels on arrive.
On commence à l'emplacement de départ, appelé "Entrée" (dans notre exemple, c'est "B"). Nous avons comme but d'aller jusqu'à l'emplacement d'arrivée, appelé "Sortie" (dans notre exemple : "D").

Étape 1 : *Première ligne du tableau*

Représentation du point d'entrée dans le tableau :

- On note "B" dans la première ligne de la case "Départ"
- On met 0 dans la case "B". *(signifira qu'on s'interdit de repasser par le point "B" pour éviter les cycles.)*

A	B	C	D	E	F	Départ
	0					B

Remplissage du reste de la ligne :

On représente ensuite tous les chemins possibles partant de "B" et allant aux différents points "A,C,D,E,F" de la manière suivante :
- Une flèche relie B à A dans la direction B→A de longueur 2. On note donc "2B" dans la case "A". (*signifiera qu'on est arrivé de B par un chemin de longueur totale* 2.)
- On peut remplir de même la case F.
- On ne peut arriver sur aucun autre sommet depuis B *(soit parce qu'il n'y pas de liaison ou parce que les flèches sont dans le mauvais sens)*. On met alors ∞ dans les autres cases.

On obtient :

A	B	C	D	E	F	Départ
2B	0	∞	∞	∞	3B	B

Le point de Sortie (dans notre exemple : "D") n'est pas encore atteint, donc on continue.

Étape 2 : *2ème ligne et suivantes*

Quand une ligne est entièrement remplie, on repère le chemin de longueur minimale fait jusque là (ici, en colonne A) et on recommence avec les consignes suivantes. Par exemple, pour la deuxième ligne :
- Le nouveau point de départ (ici : "A") est à nouveau mis dans la colonne "Départ" et noté 0 dans sa colonne.
- La (ou les) colonne(s) contenant déjà 0 (ici : "B") restent vides. (On ne repasse pas deux fois par le même point ; *inutile de tourner en rond ...*)
- Pour un point atteint pour la première fois (2ème ligne : "E"), on additionne les longueurs depuis le sommet d'entrée (B) jusqu'au sommet en question (E) et on remplit la case comme dans l'étape 1. (ici : 2+4=6, point de départ : A, d'où la notation "6A").
- Pour un sommet atteint les fois précédentes qui ne l'est plus (ici, "F"), on recopie la case du dessus *(le chemin allant jusqu'à F est toujours le plus court)*
- On continue à noter ∞ pour les points qui ne sont toujours pas atteints. (ici : "C,D")

On obtient :

A	B	C	D	E	F	Départ
2B	0	∞	∞	∞	3B	B
0		∞	∞	6A	3B	A

Pour l'instant, on a donc explicité les chemins les plus courts et leur longueur partant de B et arrivant aux divers points donnés en en-tête. Le point de sortie (D) n'est toujours pas atteint. Répétons alors l'étape 2 sur la ligne 3. Pour le sommet E, on est confronté

à un dilemne. Essayez de remplir la ligne 3 (sauf E) sans regarder la suite pour voir si vous avez compris.
On obtient (avant E)

A	B	C	D	E	F	Départ
2B	0	∞	∞	∞	3B	B
0		∞	∞	6A	3B	A
		9F	∞		0	F

Pour E, deux chemins permettent d'y arriver : 6A (chemin précédent) et 7F (3+4=7 ; départ de F). On choisit donc encore le chemin le plus court. Ici, c'est 6A.

A	B	C	D	E	F	Départ
2B	0	∞	∞	∞	3B	B
0		∞	∞	6A	3B	A
		9F	∞	6A	0	F

En continuant jusqu'à arriver au point de sortie, on obtient encore une ligne :

A	B	C	D	E	F	Départ
2B	0	∞	∞	∞	3B	B
0		∞	∞	6A	3B	A
		9F	∞	6A	0	F
		9F	10E	0		E

Le sommet de sortie est atteint. Si le chemin correspondant est de longueur minimale (ici, 10, non minimal), on s'arrête. Sinon, on continue pour vérifier s'il n'y a pas un autre moyen d'y arriver de manière plus courte :

A	B	C	D	E	F	Départ
2B	0	∞	∞	∞	3B	B
0		∞	∞	6A	3B	A
		9F	∞	6A	0	F
		9F	10E	0		E
		0	10E			C

Il ne reste plus qu'une seule colonne. On ne peut pas continuer, le chemin minimal est donc atteint. On obtient immédiatement sa longueur : 10. Attention, le chemin effectué n'est pas forcément constitué des lettres dans la colonne de droite. En réalité, ici, c'est

BAED

(à méditer !)

Remarque : Il est possible, suivant les cas, d'obtenir deux chemins possibles avec la même longueur.
⋆ *Premier cas :* on arrive au même point de deux manières différentes. Par exemple, dans la ligne 3, si la longueur de F à E était de 3 et non 4, on aurait eu :

A	B	C	D	E	F	Départ
2B	0	∞	∞	∞	3B	B
0		∞	∞	6A	3B	A
		9F	∞	6A ou 6F	0	F
		9F	10E	0		E

dans ce cas, on peut naturellement choisir n'importe quel sommet entre 6A ou 6F.
Dans certains cas, on peut éventuellement opter pour celui qui possède le moins de sommets intermédiaires. *(Par exemple, dans des problèmes de minimisation des feux ou de carrefours dans une ville pour éviter la perte de temps.)*
⋆ *Deuxième cas :* On n'arrive pas au même point. Par exemple (ce cas ne se produit pas dans notre exemple) :

A	B	C	D	E	...
⋮	⋮	⋮	⋮		
		5B	∞	5B	...

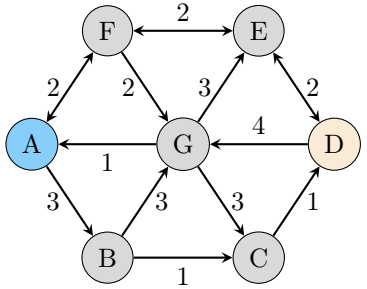
Dans ce cas encore, on choisit un des deux sommets (peu importe lequel) et on continue jusqu'au bout.

Mise en pratique :

Exercice 1:

1. Refaire entièrement le cas de l'exemple vous même.
2. Sur le même graphe, construire le tableau et déterminer le chemin le plus court entre A et F

Exercice 2:
On considère le graphe de chemins pondérés ci-dessous



Établir à l'aide de l'agorithme de Dijkstra un chemin de longueur minimale pour aller de A à D.

On cherche maintenant à faire faire le travail à Python.

Exercice 3:

Commandes éventuellement utiles :
si `C` est une chaine de caractère (ou une liste), `C.index('mot')` donnera l'indice d'apparition du mot 'mot' dans la chaine `C`.

On se donne un ensemble de sommets noté de la manière dont vous préférez (par ex : `Sommets=['A','B','C','D','E',...]`) muni de chemins pondérés symbolisés par un tableau de distances. On souhaite proposer une programmation de l'algorithme de Dijkstra donnant le plus court chemin d'un point d'entrée à un point de sortie. On s'appuiera sur l'exemple intégralement décrit dans ce TD.

1. On donne un tableau `Dist` constitué des distances allant d'un point à un autre de la manière suivante :

- On numérote les sommets par leurs indices.
- `Dist[i,j]` représente la distance du chemin allant **du point n° i au point n° j.** *(Attention au sens des flèches ! Dans les cas où elles vont dans les deux sens, on pensera à mettre le même nombre dans `Dist[i,j]` et `Dist[j,i]`)*
- On pourra par exemple mettre `-1` si les sommets ne sont pas reliés (ex : $A \to B$ ou $A \to C$) et 0 si on ne change pas de sommet (ex : $A \to A$).

Construire ce tableau pour l'algorithme de l'exemple principal décrit au début de la feuille.

On va maintenant modéliser la situation.
*(**Attention !** un élément de type "array" ne permet pas de mélanger les chaines de caractères et les nombres, alors qu'une liste le peut. Choisissez bien votre modélisation suivant vos souhaits.)*

2. Constuire une fonction `EtapeSupp` qui réalise une ligne supplémentaire au tableau sachant qu'il existe déjà une ligne.

3. Construire une fonction `Rechermin` permettant de repérer, pour une ligne fixée, les informations relatives au chemin de longueur minimal permettant de continuer l'agorithme à la ligne suivante. Par exemple, si le tableau correspond à

A	B	C	D	E	F	Départ
2B	0	∞	∞	∞	3B	B
0		∞	∞	6A	3B	A

`Rechermin` pourrait rendre 5, 3 ou encore F car le plus petit chemin (3B) est dans la 5ème case à partir de la gauche, que sa longueur est 3, et que le sommet d'arrivée et F.

Remarque : Si votre modélisation est différente, n'hésitez pas à modifier les résultats du programme en conséquence.

4. En déduire un algorithme général `Dijkstra` qui permet d'établir le tableau de DIjkstra complet allant d'un sommet à un autre.

5. Construire une fonction `chemin` qui affiche le chemin le plus court (s'il existe.)

6. Appliquer l'algorithme à l'exemple de l'exercice 1 et vérifier vos résultats.

FE 5 - Algorithme de Dijkstra — Solutions

Exercice 3

1. Notons que

$$\texttt{Dist} = \begin{pmatrix} 0 & -1 & -1 & -1 & 4 & -1 \\ 2 & 0 & -1 & -1 & -1 & 3 \\ -1 & 7 & 0 & 2 & -1 & 6 \\ 7 & -1 & 2 & 0 & -1 & 3 \\ -1 & -1 & -1 & 4 & 0 & -1 \\ 6 & 3 & 6 & -1 & 4 & 0 \end{pmatrix}$$

On en déduit alors le script

```
Dist=array([
[0,-1,-1,-1,4,-1],
[2,0,-1,-1,-1,3],
[-1,7,0,2,-1,6],
[7,-1,2,0,-1,3],
[-1,-1,-1,4,0,-1],
[6,3,6,-1,4,0]
])
```

2.

```
def EtapeSuppTab(Dist,IndDepart,Dij):
    '''réalise une ligne supplémentaire dans le tableau Dij en partant du sommet de numéro IndDepart avec le tableau des distances Dist .'''
    L=cp.deepcopy(Dij[-1]) # On initialise la nouvelle ligne du tableau Dij (On reprend l'ancienne et on modifiera après si besoin)
    L[IndDepart]=[0,-2] # On ne veut pas faire de cycle. (remplacement par -2 au lieu de 0 pour ne pas confondre avec le numéro du sommet.)
    n=shape(Dist)[0]

    for i in range(n): # On veut remplir la case de la colonne nř i
        CaseDessus=Dij[-1][i] # Case juste au dessus
        if CaseDessus[1]!=-2: # Si 0, on est déjà passé par cette case, on ne change rien, L vaut déjà "0".
            d=Dist[IndDepart,i]
            if d>0: # Il y a effectivement un chemin allant du sommet "Départ" au sommet i
                long_totale=Dij[-1,IndDepart][0]+Dist[IndDepart,i] # chemin jusqu'au point précédent + longueur flèche.
                long_precedente=Dij[-1,i][1] # longueur pour aller au point i s'il existait déjà un chemin
                if long_precedente==-1 or long_totale<long_precedente: # il se peut qu'il y ait un chemin plus court pour arriver à i
                    L[i][0]=long_totale
                    L[i][1]=IndDepart
                # sinon, on ne touche à rien et on recopie le résultat précédent
```

```
18
19      return(append(Dij,[L],axis=0)) # rend le
            tableau avec une nouvelle ligne
```

3.

```
1   def RecherminTab(Ligne,n):
2   '''On cherche le chemin de longeur minimale dans
        Ligne qui est de longeur n. Rend l'indice du
        minimum'''
3       # initialiser l'indice minimum
4       min=-1
5       indicemin=-1
6       i=0
7       Min_Non_Initialise=True
8       while Min_Non_Initialise and i<n:
9           if i<n and Ligne[i][0]>0:
10              Min_Non_Initialise=False
11              min=Ligne[i][0]
12              indicemin=i
13          i+=1
14      # Chercher l'indice du min
15      for i in range(n):
16          if Ligne[i][0]>0 and Ligne[i][0]<min:
17              indicemin=i
18      return(indicemin) # l'indice du min vaut -1 si
            toutes les longueurs valent 0, ce qui peut
            donner un critère de fin si le sommet de
            sortie ne peut être atteint.
```

4.

```
1   def Dijkstra(Dist,IndEntree,IndSortie):
2       '''Rend le tableau de Dijkstra complet. Dist
            est le tableau des distances et IndEntree et
             IndSortie sont les indices des lettres d'
            entrée et sortie'''
3       n=shape(Dist)[0]
4       Dij=array([n*[[0,-1]]]) # Initialisation.
5       SortieNonAtteinte=True
6       IndDepart=IndEntree
7       while IndDepart!=-1 and SortieNonAtteinte: # s'
            arrête quand l'indice minimum vaut -1 (point
             de sortie ne pouvant être atteint) ou quand
             on a atteint la sortie.
8           Dij=EtapeSuppTab(Dist,IndDepart,Dij)
9           # recherche du sommet contenant le chemin
                de longueur mini :
```

5.

```
1   def Chemin(Dist,Sommets):
2       '''Dist est le tableau des distances ; Sommet
            la liste des sommets ; Entree la lettre du
            sommet d'entree ; Sortie la lettre du sommet
             de sortie'''
3       Entree=input("Quel est le sommet d'entrée ?")
4       Sortie= input('Sommet de sortie ?')
5       # Construction du tableau
6       IndEntree=Sommets.index(Entree)
7       IndSortie=Sommets.index(Sortie)
8       Dij=Dijkstra(Dist,IndEntree,IndSortie)
9       # Construction du chemin
10      long_totale=Dij[-1,IndSortie][0]
11      if long_totale==0:
12          print("Il n'y a aucun chemin permettant d'
                aller de ",Entree,' à ',Sortie)
13      else:
14          Chemin=[Sortie]
15          indice=IndSortie
16          i=1
17          while indice!=IndEntree:
18              indicePrec=Dij[-i,indice][1]
19              if indicePrec!=indice and indicePrec
                    >-1:
20                  indice=indicePrec
21                  Chemin=[Sommets[indice]]+Chemin
22              i+=1
23          print("Le chemin le plus court est ",Chemin
                ,' avec une longeur totale de ',
                long_totale)
```