"Graphes" : l'algorithme de Dijkstra

Table des matières

I Description de l'algorithme **2**

II En autonomie **3**

III Implémentation en Python **3**

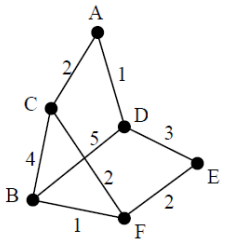
III.1 Plus court chemin entre un sommet et chacun des autres sommets 3

III.2 Plus court chemin entre deux sommets 4

I Description de l'algorithme

L'*algorithme de Dijkstra*, publié par Edsger Dijkstra en 1959 et utilisant un parcours en largeur, permet de calculer le plus court chemin entre un sommet d'un graphe (connexe, non orienté et pondéré) et chacun des autres sommets.

Exemple
On considère le graphe ci-dessous.
L'objectif est de trouver un plus court chemin reliant A et B.



↪ Détaillons l'*algorithme de Dijkstra* sur cet exemple.

Le tableau donné ci-après est un moyen de construire son raisonnement et de présenter le résultat.

Etape 1.
Le sommet de départ est affecté du poids provisoire 0 et les autres sommets du poids provisoire ∞.
On sélectionne le sommet A et on fixe son poids provisoire.

Etapes suivantes.
Tant que tous les sommets n'ont pas un poids définitif, ou que le sommet d'arrivée B n'est pas affecté du plus petit poids provisoire, on poursuit.

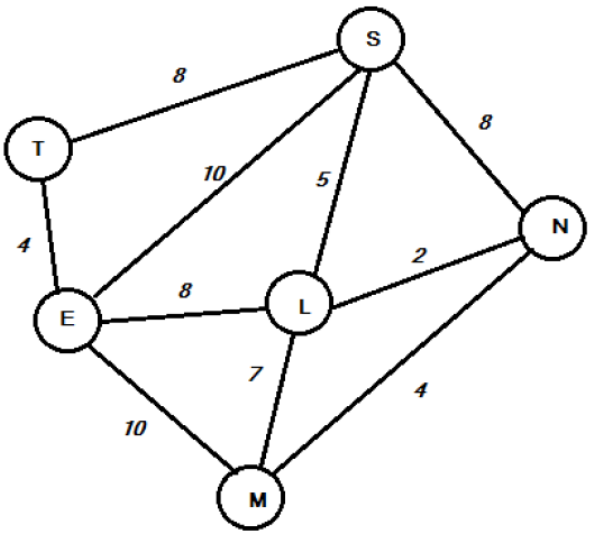
Pour tout sommet T' *non sélectionné et adjacent au dernier sommet sélectionné* T, on calcule la somme s du poids définitif affecté à T et du poids de l'arête reliant T à T'.
Si cette somme s est inférieure au poids provisoire affecté à T', on affecte T' du poids provisoire s que l'on note s_T pour indiquer que T est le sommet qui précède T'. Parmi les sommets de poids provisoire, on sélectionne un sommet de poids provisoire minimum et on fixe son poids provisoire.

Conclusion.
On obtient un plus court chemin entre A et B en l'écrivant de la droite vers la gauche ainsi :
on écrit le sommet d'arrivée B, puis le sommet inscrit au plus bas de la colonne B (c'est-a-dire F), puis le sommet inscrit au plus bas de la colonne F (c'est-a-dire C), et ainsi de suite jusqu'au sommet de départ A.
On obtient ainsi un plus court chemin entre A et B : $A - C - F - B$.

A	C	D	E	F	B	Sommet Sélectionné Poids
0	∞	∞	∞	∞	∞	A_0
||	2_A	1_A	∞	∞	∞	D_1
||	2_A	||	4_D	∞	6_D	C_2
||	||	||	4_D	4_C	6_D	E_4
||	||	||	||	4_C	6_D	F_4
||	||	||	||	||	5_F	B_5

II En autonomie

On considère le graphe pondéré non orienté ci-dessous.



En utilisant l'algorithme de Dijkstra ...

1. Déterminer un plus court chemin reliant M et S.
2. Déterminer un plus court chemin reliant T et N.
3. Déterminer un plus court chemin reliant T et M.

On construira des tableaux comme ci-dessous pour répondre à la question posée.

E	L	M	N	S	T	Sommet Sélectionné Poids

III Implémentation en Python

Pour représenter l'ensemble des sommets et leur poids (distance à un autre sommet), nous allons utiliser une matrice d'adjacence.

À chaque étape, il faut comparer la distance à laquelle se trouve chaque sommet.

III.1 Plus court chemin entre un sommet et chacun des autres sommets

```
def dijkstraDist(G, depart):
    # G matrice des poids du graphe
    # On récupère le nombre de sommets du graphe
    N = np.size(G,0)
    # Initialisation du tableau des plus courts chemins
    # Le booléen pour savoir si le sommet a déjà été sélectionné
    pcc = list()
    for i in range(N):
        pcc.append([10**4, False])
    sommet_u = depart
    dist_u = 0
    pcc[depart][0] = 0
    # Le premier sommet sélectionné est le sommet depart
    pcc[depart][1] = True
    # On compte le nombre de sommets sélectionnés
    cpt = 0
    while cpt != N-1:
        # À chaque étape, la solution optimale doit être conservée
        # (pour sélection du sommet correspondant à l'étape suivante)
        minimum = 10**4
        #float('inf')
        # Étape de relâchement
        for k in range(N):
        # Si le sommet k n'a pas encore été sélectionné
            if pcc[k][1] == False:
                dist_uv = G[sommet_u][k]
                # Distance totale du chemin s -> ... -> u -> v
                dist_totale = dist_u + dist_uv
                # Mise à jour du tableau des plus courts chemins
                if dist_totale < pcc[k][0]:
                    pcc[k][0] = dist_totale

                    # Mise à jour de la solution minimale à cette étape
                if pcc[k][0] < minimum:
                    minimum = pcc[k][0]
                    prochain_sommet_select = k

        # On a traité complétement un sommet
        cpt = cpt + 1

        # Le sommet à traiter est sélectionné et d[u] est mis à jour
        sommet_u = prochain_sommet_select
        pcc[sommet_u][1] = True
        dist_u = pcc[sommet_u][0]

        #print(dist_u)

    return pcc
```

III.2 Plus court chemin entre deux sommets

```
def dijkstraPCC(G, depart, arrivee):
    pcc = dijkstraDistChemin(G, depart)
    # Reconstitution du plus court chemin
    chemin = list()
    # On reconstitue le plus court chemin d'arrivee vers depart
    c = arrivee
    chemin.append(c)
    while c != depart:
        c = pcc[c][2]
        chemin.append(c)
    # On demande le miroir de la liste obtenue pour
   # que les sommets apparaissent dans l'ordre
    return(list(reversed(chemin)))
```