TP Informatique n° 9/10
Algorithme de Dijkstra

I. Quelques notions de graphes

L'algorithme de Dijkstra est un algorithme permettant de déterminer les plus courts chemins entre certains points d'un graphe. Avant de détailler le fonctionnement de cet algorithme, commençons par introduire le vocabulaire nécessaire sur les graphes.

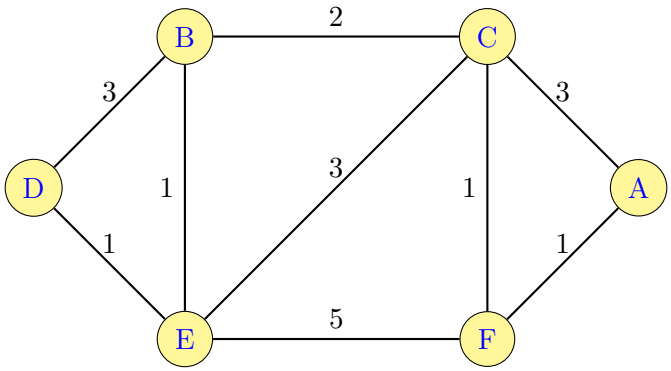
Définition
On appelle graphe $\mathcal{G} = (S, \mathcal{A})$ un couple où :

× S est un ensemble, appelé ensemble des **sommets**,

× $\mathcal{A}$ est une partie de $S \times S$, appelée ensemble des **arêtes**.

Dans la suite, on supposera $\mathcal{S}$ fini.

Exemple
On considère le graphe suivant, donné par sa représentation graphique.



Le graphe $\mathcal{G} = (S, A)$ est ici défini par :

× l'ensemble des sommets $S = \{A, B, C, D, E, F\}$,

× l'ensemble des arêtes $\mathcal{A} = \{(A, B), \ldots, (E, F), \ldots, (F, A)\}$.

Remarque
Le graphe présenté ici possède les caractéristiques suivantes.

- Il n'est pas **orienté** : si $(u, v) \in \mathcal{A}$ alors on a $(v, u) \in \mathcal{A}$.
  Ainsi, si on peut aller de u à v, alors on peut aussi aller de v à u.
- Il est **simple** : il y a au plus une arête entre deux sommets.
- Il est **pondéré** : à chaque arête on associe une valeur positive appelée poids.
  Ce poids représente la distance entre les deux sommets.

Définition
On appelle **matrice des poids** d'un graphe la matrice $G = (g_{i,j})_{(i,j)\in S^2}$ telle que pour tout couple de sommets (i, j) :

× si $(i, j) \in \mathcal{A}$ alors $g_{i,j}$ est égal au poids de l'arête (i, j),

× si $(i, j) \notin \mathcal{A}$ alors $g_{i,j} = +\infty$.

Exemple
Le graphe précédent possède 6 sommets.
Après renommage des sommets, la matrice des poids de ce graphe est :

$$\begin{array}{c} \\ \mathrm{D} \\ \mathrm{B} \\ \mathrm{E} \\ \mathrm{C} \\ \mathrm{F} \\ \mathrm{A} \end{array} \begin{array}{c} \begin{array}{cccccc} \mathrm{D} & \mathrm{B} & \mathrm{E} & \mathrm{C} & \mathrm{F} & \mathrm{A} \end{array} \\ \begin{pmatrix} 0 & 3 & 1 & +\infty & +\infty & +\infty \\ 3 & 0 & 1 & 2 & +\infty & +\infty \\ 1 & 1 & 0 & 3 & 5 & +\infty \\ +\infty & 2 & 3 & 0 & 1 & 3 \\ +\infty & +\infty & 5 & 1 & 0 & 1 \\ +\infty & +\infty & +\infty & 3 & 1 & 0 \end{pmatrix} \end{array}$$

On peut représenter cette matrice en **Python** sous la forme d'un tableau de type `array`.

```
import numpy as np

Inf = np.inf
G = np.array([ [0,3,1,Inf,Inf,Inf], [3,0,1,2,Inf,Inf], [1,1,0,3,5,Inf], \
[Inf,2,3,0,1,3], [Inf,Inf,5,1,0,1], [Inf,Inf,Inf,3,1,0] ])
```

▶ Quelle remarque peut-on faire sur la matrice de poids de ce graphe.

C'est une matrice symétrique.

▶ De quelle propriété provient cette caractéristique ?

Le graphe précédent n'est pas orienté.
Ainsi chaque arête (u, v) fournit une arête (v, u) de même poids.

II. Algorithme de Dijkstra

II.1. Brève présentation

L'algorithme de Dijkstra consiste en la recherche des plus courts chemins menant d'un sommet unique $s \in S$ à chaque autre sommet d'un graphe pondéré $\mathcal{G} = (S, \mathcal{A})$.
Tous les arcs de $\mathcal{G}$ sont supposés de poids positif ce qui permet d'empêcher la présence de cycles de poids strictement négatifs.

Notation
Pour tout $t \in S$, on nomme $\delta(t)$ la longueur du plus court chemin menant de s à t.

II.2. Calcul de la longueur des plus courts chemins

Avant de déterminer les plus courts chemins entre s et tout autre sommet t, on commence par déterminer la longueur de chacun de ces plus courts chemins *i.e.* la fonction δ.

Pour ce faire, on utilise un attribut $d[v]$ du sommet $v \in S$ qui contient le poids du supposé plus court chemin menant de s à v. Cet attribut est corrigé au fur et à mesure de l'algorithme de sorte à contenir $\delta(v)$ à la fin de l'exécution. Évidemment, cet attribut est, à chaque étape de l'algorithme, un majorant du poids d'un plus court chemin menant de s à v. Autrement dit :

$$\boxed{\forall v \in S, \quad \delta(v) \leqslant d[v]}$$

Détaillons la fonctionnement de cet attribut.

Mise à jour de l'attribut $d[v]$

1) **Initialisation**

Au début de l'algorithme, on considère :

- $d[s] \leftarrow 0$,
- $d[v] \leftarrow +\infty$ pour tout sommet $s \in S$.

Exemple

Sur l'exemple précédent, et si on choisit $s = D$, l'étape d'initialisation s'écrit :

D	B	E	C	F	A
0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

2) **Principe de relâchement**

L'opération de relâchement d'un arc $(u, v) \in S \times S$ consiste en un test permettant de savoir s'il est possible, en passant par u, d'améliorer le plus court chemin jusqu'à v.

Si oui, il faut mettre à jour $d[v]$.

Plus précisément, on effectue le test suivant :

$$\boxed{d[u] + g_{u,v} \overset{?}{<} d[v]}$$

- Si le test est négatif, on n'effectue pas de mise à jour.
- Si le test est positif, cela signifie qu'un chemin de plus petite taille est exhibé pour passer du sommet s au sommet v :
  - on passe de s à u (ce chemin à le poids $d[u]$),
  - on termine ce chemin en utilisant l'arc (u, v) de poids $g_{u,v}$.
    Il convient alors d'effectuer la mise à jour :

$$d[v] \leftarrow d[u] + g_{u,v}$$

Exemple

1) Sur l'exemple précédent, on prend $u = D$

On cherche alors à déterminer, pour tout sommet $v \in S$ si on peut mettre à jours $d[v]$ à l'aide d'un chemin dont le dernier arc est (u, v).

Étant donnée l'étape d'initialisation, cela consiste à considérer les chemins de s à v de taille 1 :

D	B	E	C	F	A
0	3	1	$+\infty$	$+\infty$	$+\infty$

2) On prend alors $u = E$
On cherche alors à déterminer, pour tout sommet $v \in S$ si on peut mettre à jours $d[v]$ à l'aide d'un chemin dont le dernier arc est (u, v).

Étant donnée l'étape précédente, cela consiste à considérer les chemins de s à v de taille 2 (autrement dit les chemins $s \to u \to v$) :

D	B	E	C	F	A
0	2	1	4	6	$+\infty$

3) On prend alors $u = B$
On cherche alors à déterminer, pour tout sommet $v \in S$ si on peut mettre à jours $d[v]$ à l'aide d'un chemin dont le dernier arc est (u, v).

Étant donnée l'étape précédente, cela consiste à considérer les chemins de s à v de taille 3 (autrement dit les chemins $s \to \ldots \to u \to v$) :

D	B	E	C	F	A
0	2	1	4	6	$+\infty$

4) On prend alors $u = C$
On cherche alors à déterminer, pour tout sommet $v \in S$ si on peut mettre à jours $d[v]$ à l'aide d'un chemin dont le dernier arc est (u, v).

Étant donnée l'étape précédente, cela consiste à considérer les chemins de s à v de taille 4 (autrement dit les chemins $s \to \ldots \to \ldots \to u \to v$) :

D	B	E	C	F	A
0	2	1	4	5	7

5) On prend alors $u = F$
On cherche alors à déterminer, pour tout sommet $v \in S$ si on peut mettre à jours $d[v]$ à l'aide d'un chemin dont le dernier arc est (u, v).

Étant donnée l'étape précédente, cela consiste à considérer les chemins de s à v de taille 5 (autrement dit les chemins $s \to \ldots \to \ldots \to \ldots \to u \to v$) :

D	B	E	C	F	A
0	2	1	4	5	6

6) On prend alors $u = A$
On cherche alors à déterminer, pour tout sommet $v \in S$ si on peut mettre à jours $d[v]$ à l'aide d'un chemin dont le dernier arc est (u, v).

Étant donnée l'étape précédente, cela consiste à considérer les chemins de s à v de taille 6 (autrement dit les chemins $s \to \ldots \to \ldots \to \ldots \to \ldots \to u \to v$) :

D	B	E	C	F	A
0	2	1	4	5	6

Sélection du sommet u à chaque étape

À chaque étape de l'algorithme, on sélectionne le sommet u qui vérifie les propriétés suivantes :

- u n'a pas encore été sélectionné,
- l'attribut $d[u]$ est le plus faible (parmi tous les sommets non encore sélectionnés).

On parle d'algorithme **glouton** : on sélectionne à chaque étape la sous-solution optimale *i.e.* le sommet réalisant la meilleure distance.

Fin de l'algorithme

Une fois tous les sommets visités, on a trouvé tous les plus courts chemins (de s à tout v) de taille inférieure ou égale au nombre de sommets en tout. On a donc trouvé tous les plus courts de chemins (de s à tout v).

II.3. Implémentation

▶ Appliquer l'algorithme de Dijkstra avec origine D.

D	B	E	C	F	A
0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
0	3	1	$+\infty$	$+\infty$	$+\infty$
0	2	1	4	6	$+\infty$
0	2	1	4	6	$+\infty$
0	2	1	4	5	7
0	2	1	4	5	6
0	2	1	4	5	6

▶ Appliquer l'algorithme de Dijkstra avec origine E.

D	B	E	C	F	A

▶ Appliquer l'algorithme de Dijkstra avec origine C.

D	B	E	C	F	A

- Implémenter `DijkstraDist(G, depart)` qui prend en paramètre la matrice des poids `G` et le sommet `depart` et renvoie la taille des plus courts chemins de `depart` à tout sommet v.

```
def dijkstraDist(G, depart):
    # 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([Inf, 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 = 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]

    return(pcc)
```

II.4. Exhiber les plus courts chemins

▶ En reprenant les exemples précédents (avec origine D puis avec origine E), expliquer comment on peut obtenir les plus courts chemins à l'aide des calculs de taille précédents.

- Si à une étape, $d[v]$ a été modifié, il faut se rappeler de quel sommet u provient cette modification.
- L'idée étant alors que l'arc (u, v) sera un arc du plus court chemin.

▶ Modifier le premier tableau (avec origine en D) pour qu'il prenne en compte cette nouvelle information.

D	B	E	C	F	A
0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
0	3	1	$+\infty$	$+\infty$	$+\infty$
0	2	1	4^{E}	6^{E}	$+\infty$
0	2	1	4	6	$+\infty$
0	2	1	4	5	7
0	2	1	4	5	6
0	2	1	4	5	6

▶ Reconstituer alors le plus court chemin de D vers A, celui de D vers E et celui de D vers B.

- Le plus court chemin de D vers A est réalisé par $D \to E \to C \to F \to A$.
- Le plus court chemin de D vers E est réalisé par $D \to E$.
- Le plus court chemin de D vers B est réalisé par $D \to E \to B$.

▶ Implémenter la fonction `DijsktraDistChemin(G, depart)` qui prend en paramètre la matrice des poids `G` et le sommet origine `depart` et renvoie la taille des plus courts chemins de `depart` à tout sommet v ainsi que le dernier sommet utilisé pour calculer cette taille.

On écrira seulement les deux lignes qui diffèrent de la fonction précédente.

- La ligne 9 doit être modifiée : on doit se souvenir d'une information supplémentaire.

```
pcc.append([Inf, False, None])
```

- On doit ajouter une ligne 34 pour se souvenir de quel sommet provient la modification.

```
pcc[k][2] = sommet_u
```

- Implémenter la fonction `dijkstraPCC(G, depart, arrivee)` qui permet d'obtenir le plus court chemin de `depart` à `arrivee`.

```
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
    ville = arrivee
    chemin.append(ville)
    while ville != depart:
        ville = pcc[ville][2]
        chemin.append(ville)
    # On demande le miroir de la liste obtenue pour
    # que les sommets apparaissent dans l'ordre
    return(list(reversed(chemin)))
```

- Tester votre fonction en prenant pour origine D puis E.