Plus courts chemins issus d'un sommet par l'algorithme de Dijkstra

On considère un graphe orienté $G = (S, A)$ valué **par des réels positifs**, c'est-à-dire muni d'une fonction $\omega : A \to \mathbb{R}^+$, où A est l'ensemble des arêtes du graphe G.

On définit le poids d'un chemin par la somme des poids de ses arêtes. On fixe un sommet $s \in S$ et on cherche à calculer pour chaque sommet x un chemin de poids minimum reliant s à x.

A chaque étape de l'algorithme, on dispose :

- d'une liste L de sommets pour lesquels on connaît le poids minimum d'un chemin reliant s à x

- de la bordure B de L dans S,

c'est-à-dire des sommets admettant un prédecesseur dans L mais qui ne sont pas dans L

- d'un tableau d de sorte que :

(a) : $\forall x \in L$, $d[x]$ est **le poids minimal** d'un chemin de s à x dans le graphe

(b) : $\forall x \in B$, $d[x]$ est le poids minimal d'un chemin de s à x **dont les sommets intermédiaires sont dans** L

1) Principe de l'algorithme :

On utilise un tableau de marquage M de sorte que x est marqué 2, 1 ou 0 selon l'appartenance de x à L, à B ou au reste (reste = ensemble des sommets non encore visités).

Etape 1 :

- On crée un tableau M de longueur n initialisé à 0
- On crée un tableau d de longueur n initialisé à -1 (par exemple)
- On initialise B à $\{s\}$ en prenant $M[s] = 1$

Etape 2 : Tant que B n'est pas vide :

- On détermine parmi les éléments de B l'élément x minimisant d.
- On l'ajoute à L. On met à jour le tableau d et la bordure :

Pour toute arête valuée (x, y) de poids $v = \omega(x, y)$,

si $y \in L$: on ne fait rien

si $y \in B$: on met à jour : $d[y] = \min(d[y], d[x] + v)$

sinon, on ajoute y à B (en le marquant à 1) et on prend $d[y] = d[x] + v$

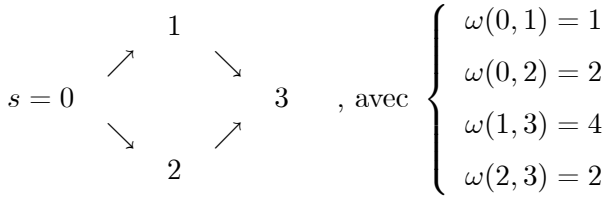
Etape 3 : On renvoie le tableau d

Remarque : A la fin, si $d[x] \neq -1$, $d[x]$ représente le poids minimal d'un chemin de s à x.

Lorsque $d[x] = -1$, il n'existe pas de chemin de s à x.

2) Exemple

On considère le graphe valué suivant, avec ici $S = \{0, 1, 2, 3\}$:



, avec $\begin{cases} \omega(0,1) = 1 \\ \omega(0,2) = 2 \\ \omega(1,3) = 4 \\ \omega(2,3) = 2 \end{cases}$

Valeurs successives de L et de d au cours des étapes de l'algorithme :

étape boucle while	Liste L	Liste B (l'élément avec d min est en gras)	Tableau d
0	$\emptyset$	$[\mathbf{0}]$	$[0, -1, -1, -1]$
1	$\{0\}$	$[\mathbf{1}, 2]$	$[0, 1, 2, -1]$
2	$\{0, 1\}$	$[\mathbf{2}, 3]$	$[0, 1, 2, 5]$
3	$\{0, 1, 2\}$	$[\mathbf{3}]$	$[0, 1, 2, 4]$
4	$\{0, 1, 3, 2\}$	[ ]	$[0, 1, 2, 4]$

Noter la mise-à-jour de $d[3]$ lors de la prise en compte du chemin $0 \to 2 \to 3$ à l'étape 3

3) Code Python :

On suppose $S = [\![0, n-1]\!]$ et que G est codé par la liste des listes de sommets adjacents pondérés :

Pour tout $x \in S$, $G[x]$ est la liste des couples (y, v) où $(x, y) \in A$ arête et $v = w(x, y)$ poids de l'arête.

```
01.  def disjktra(G,s) :
02.      n = len(G) ; d = [-1]*n ; M = [0]*n
03.      d[s] = 0 ; M[s] = 1
04.      for k in range(n) :
          # on sélectionne x minimisant d parmi les sommets vérifiant M[y] = 1 :
05.          x = select_min(d,M) ; M[x] = 2
          # on met à jour la bordure et le tableau d
06.          for (y,v) in G[x] :
07.              if M[y] == 0 :
08.                  M[y] = 1 ; d[y] = d[x]+v
08.              else :
10.                  if d[x]+v < d[y] :  d[y] = d[x]+v
11.      return d
```

4) Utilisation des distances infinies :

Une variante consiste à considérer la possibilité de poids égal à $+\infty$: autrement dit, le graphe est complet, mais les arêtes ajoutées par rapport au graphe initial ont $+\infty$ comme poids.

En PYTHON, l'infini est représenté par `math.inf`

De ce fait, la bordure B est alors égale au complémentaire de L ;

on n'utilise alors que les marques 0 (pour B) et 1 (pour L). D'où le code :

```
def dijkstra(G,s) :
    n = len(G) ; d = [math.inf]*n ; M = [0]*n ; d[s] = 0
    for k in range(n) :
        x = select_min(d,M) ; M[x] = 1
        # sélectionne un sommet x minimisant d parmi les sommets vérifiant M[y] = 0
        for (y,v) in G[x] :  d[y] = min(d[y],d[x]+v)
    return d
```

5) Obtention des chemins optimaux

On remplit au cours de l'algorithme un tableau `pere` de sorte que pour tout $x \in S$, `pere[x]` est le sommet précédent x dans le chemin optimal (parmi les chemins déjà considérés) reliant s à x.

```
01.     def disjktra(G,s) :
02.         n = len(G) ; d = [-1]*n ; M = [0]*n ; pere = [-1]*n
03.         d[s] = 0 ; M[s] = 1 ; pere[s] = s
04.         for k in range(n) :
05.             x = select_min(d,M) ; M[x] = 2
             # sélectionne un sommet x minimisant d parmi les sommets vérifiant M[y] = 1
06.             for (y,v) in G[x] :
07.                 if M[y] == 0 :  M[y] = 1 ; d[y] = d[x]+v ; pere[y]=x
08.                 else :
09.                     if d[x]+v < d[y] :  d[y] = d[x]+v ; pere[y]=x
10.         return d
```

Ensuite, pour récupérer le chemin (obtenu d'abord dans le sens inverse) reliant s à t, on écrit :

```
chemin = [t] ; x = t
while x != s :  chemin.append(x) ; x = pere[x]
chemin.append(s) :  chemin.reverse()
```

6) Utilisation d'un tas pour optimiser la recherche du minimum :

Durant l'algorithme de Dijjstra, chaque sommet est ajouté et supprimé une seule fois dans la bordure B.

Chaque arête (x, y) est traitée au plus une seule fois (ligne 06).

La recherche du minimum par l'algorithme naïf demande $O(n)$ opérations (même si on choisissait une structure permettant de limiter la recherche aux éléments de la bordure B).

La complexité de l'algorithme de Dijkstra admet donc une complexité totale en $O(n(n+m))$.

Mais on peut utiliser une structure de tas (file de priorité), stockant les éléments de la bordure B avec d comme fonction de priorité. La recherche du minimum (et mise à jour du tas) se fait alors en $O(\log n)$ opérations ; il faut aussi mettre à jour le tas lorsque d est modifié. Autrement dit :

- suppression de la racine du tas en ligne 05
- ajout au tas d'un élément en ligne 07
- modification de la priorité d'un élément en ligne 08.

Chacune de ces opérations peuvent être effectuée en $O(\log n)$.

La structure de tas permet donc d'obtenir une complexité en $O((\log n)(n+m))$.

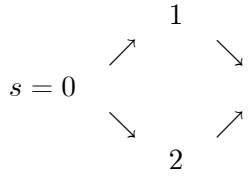
Pour le code, on suppose que la structure de tas a été implémentée, avec les opérations :

- `creerTasVide()` : cette fonction renvoie un tas ne contenant aucun élément
- `inserer(x,d,tas)` : cette procédure ajoute x avec $d[x]$ comme priorité dans le tas
- `supprimer(tas)` : cette procédure supprime la racine x du tas, et renvoie la valeur x
- `modifier(x,d,v,tas)` : cette procédure modifie $d[x]$ en prenant v comme nouvelle valeur et modifie le tas pour tenir compte de cette modification de priorité.

7) Cas des poids négatifs

Considérons le cas des graphes valués par des poids réels de signes arbitraires. La notion de chemin de poids minimal est bien défini ssi le graphe n'admet pas de cycle absorbant (c'est-à-dire de cycle de poids < 0). Dans ce cas, on peut donc se limiter aux chemins réduits (= élémentaires), qui sont en sûr en nombre fini. D'où l'existence de tels chemins.

Mais l'algorithme (glouton) de Dijkstra peut ne pas fonctionner si certains poids sont négatifs :



, avec $\begin{cases} \omega(0,1) = 1 \\ \omega(0,2) = 5 \\ \omega(1,3) = 2 \\ \omega(2,3) = -4 \end{cases}$

Valeurs successives de L et de d au cours des étapes :

Liste L	Liste B	Tableau d
$\{0\}$	$[\mathbf{1}, 2]$	$[0, 1, 5, *]$
$\{0, 1\}$	$[2, \mathbf{3}]$	$[0, 1, 5, 3]$
$\{0, 1, 3\}$	$[\mathbf{2}]$	$[0, 1, 5, 3]$
$\{0, 1, 3, 2\}$	[ ]	$[0, 1, 5, 3]$

On obtient ainsi $d[3] = 1 + 2 = 3$, alors que le chemin de poids minimal est $0 \to 2 \to 3$ (de poids 1).

8) Terminaison et correction de l'algorithme de Dijkstra

Chaque sommet est ajouté au plus une fois dans la bordure B_L.

A chaque itération de la boucle while, un sommet passe de la bordure à L.

Donc il y a au plus n itérations. D'où la terminaison de l'algorithme.

Pour prouver la correction, on met en évidence **des invariants de boucle** :

Prop : A chaque étape de l'algorithme, on a les invariants :

(a) si $x \in L$, $d[x]$ est le poids minimum d'un chemin reliant s à x **dans le graphe**

(b) si $x \in B$, $d[x]$ est le poids minimum d'un chemin **dont les sommets intermédiaires sont dans** L.

Preuve : On procède par récurrence (par exemple) sur le nombre de sommets de L :

Supposons les propriétés vraies avant d'ajouter un sommet x vérifiant $d[x] = \min\{d[y],\, y \in B\}$.

(a) : Considérons un chemin Γ de s à x qui ne soit pas inclus dans L.

Alors il existe un premier sommet $y \notin L$. Son prédécesseur appartient à L (car $s \in L$), donc $y \in B$.

Comme les poids sont tous positifs, le poids du chemin Γ est $\geq d[y] \geq d[x]$.

Par hypothèse (a), $d[x]$ est le poids du plus court chemin de s à x inclus dans L.

On en déduit que $d[x]$ est bien le poids du plus court chemin reliant s à x.

(b) : Considérons $y \in B$ (nouvelle bordure qui tient compte des successeurs de x). Les chemins (réduits) reliant s à y et dont les sommets intermédiaires sont inclus dans L sont de deux types :

- ceux qui ne contiennent pas x : le poids minimal est alors la valeur ancienne de $d[y]$.

- ceux qui contiennent x comme dernier sommet intermédiaire : le poids minimal est $d[x] + \omega(x, y)$

- ceux qui contiennent x et dont le dernier sommet intermédiaire est un sommet $z \neq x$.

Le sommet z a été ajouté à L avant x : donc il existe un chemin de coût minimal reliant s à z sans passer par x : on peut donc se ramener au premier cas (chemin qui ne contient pas x).

Ainsi, la propriété (b) est bien vérifiée en mettant à jour $d[y]$ par $d[y] = \min(d[x] + \omega(x, y), d[y])$.