TD03 : Programmation dynamique 2

Algorithme de Bellman-Ford

- Source : https://fr.wikipedia.org/wiki/Algorithme_de_Bellman-Ford

L'algorithme de Bellman-Ford est un algorithme qui résout, comme le fait l'algorithme de Dijkstra, pour un graphe orienté pondéré, le problème suivant :

> déterminer, à partir d'un sommet-source, un plus court chemin, jusqu'à chaque sommet du graphe accessible depuis le sommet-source.

L'algorithme, appelé aussi algorithme de Bellman-Ford-Moore, porte le nom de ses inventeurs, Richard Bellman et Lester Randolf Ford junior (publication en 1956 et 1958), et d'Edward Forrest Moore qui le redécouvrit en 1959.

Contrairement à l'algorithme de Dijkstra, l'algorithme de Belmann-Ford autorise la présence d'arcs de poids négatifs et permet de détecter l'existence d'un circuit absorbant, c'est-à-dire de poids total strictement négatif, accessible depuis le sommet-source[1].

La complexité de l'algorithme est en $O(nm)$ où n est l'ordre du graphe et m la taille du graphe dans le pire cas, et en $O(m)$ dans le meilleur.
Sa complexité en espace est en $O(n)$ dans le pire cas.

- Description de l'algorithme :

> L'algorithme de Bellman-Ford calcule, pour un graphe $G = (S, A)$ (S l'ensemble des sommets et V l'ensemble des arcs, sous-ensemble de $S \times S$) - sans cycle de poids négatif, et à partir d'un sommet s_0 donné, un plus court chemin vers chacun des sommets du graphe.
> Il permet en plus de trouver un tel chemin en retournant les prédécesseurs de chaque sommet le long d'un tel chemin.
> En outre, il permet de détecter les cas où il existe un cycle de poids négatif et donc dans lesquels il n'existe pas nécessairement de plus court chemin entre deux sommets.

L'algorithme utilise le principe de la programmation dynamique, en ceci qu'il décompose le problème à résoudre en sous-problèmes « plus petits » dont les solutions permettent de résoudre le problème « plus grand » dont ils sont issus.
Ici un sous-problème est de déterminer

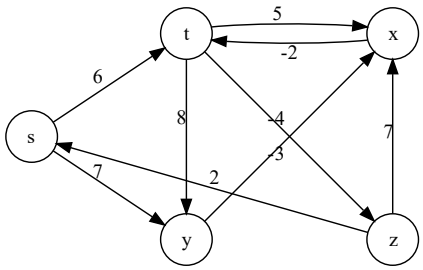
> $d[t,k]$, la distance du sommet-source s_0 à tout sommet t en empruntant un chemin comportant au plus k arcs, i.e. le poids minimal pour un chemin de s_0 à t comportant au plus k arcs.

- pour $k = 0$:
$$d[t,0] = \infty \text{ pour } t \neq s_0 \text{ et } d[s_0,0] = 0 \text{ ;}$$
- pour tout $k \geq 1$, la résolution du problème d'indice k est liée à la résolution du problème d'indice $k-1$ par la formule de récurrence suivante :
$$d[t,k] = \min\left(d[t,k], \min_{arc\ (u,t)\in A} d[u,k-1] + poids\big((u,v)\big)\right).$$

L'algorithme calcule les valeurs $d[t,k]$ par valeurs de k croissantes.
La valeur de s_0 à un sommet t quelconque du graphe est donnée par la valeur de $d[t,n-1]$ où $n = |S|$ (ordre de graphe).

[1] auquel cas tout chemin vers un sommet qui passe par un sommet du cycle a un poids qui peut tendre vers $-\infty$ dès lors que l'on commence à parcourir ce cycle un nombre croissant de fois.

On considère à titre d'exemple le graphe suivant :



On note `LS` la liste des noms des sommets du graphe, et on pose `LS = ['s', 't', 'x', 'y', 'z']`.

Question 1. Définir une représentation du graphe en exemple par une matrice `M` dont chaque coefficient, $m_{i,j}$, est égal au poids de l'arc $i \to j$ si l'arc $i \to j$ existe dans le graphe, où à la valeur « infinie » `float('inf')` si l'arc $i \to j$ n'existe pas. Les sommets sont désignés ici par leur position dans la liste `LS`.

Question 2. Définir, à partir de la liste `LS` des sommets du graphe et de la matrice `M`, un dictionnaire, `dArcs`, dont les clés sont les couples des noms de sommets entre lesquels un arc existe, avec pour valeur associée le poids de l'arc.

On considère l'algorithme en pseudo-code donné ici :

Pseudo-code [modifier | modifier le code]

L'algorithme de Bellman-Ford s'écrit donc en utilisant directement la relation de récurrence donnée dans la section précédente[5].

```
fonction Bellman-Ford(G = (S, A), poids, s)
  pour u dans S faire
  |       d[u] = +∞
  |       pred[u] = null
  d[s] = 0
  //Boucle principale
  pour k = 1 jusqu'à taille(S) - 1 faire
   |      pour chaque arc (u, v) du graphe faire
   |      |    si d[u] + poids(u, v) < d[v] alors
   |      |    |    d[v] := d[u] + poids(u, v)
   |      |    |    pred[v]:= u

  retourner d, pred
```

À l'issue de l'exécution de cet algorithme, $d[u]$ représente la longueur d'un plus court chemin de s à u dans G, alors que $pred[u]$ représente le prédécesseur de u dans un plus court chemin de s à u. La valeur null signifie qu'aucun prédécesseur n'a pour l'instant été assigné à $pred[u]$.

Question 3. Justifier qu'il n'est besoin que d'examiner les chemins comportant au plus $n-1$ arcs, où n est l'ordre du graphe, pour trouver des chemins de poids minimal dans un graphe ne comportant pas de cycle de poids négatif.

Question 4. Donner une implémentation en Python de l'algorithme sous la forme d'une fonction `bellman_ford(LS, dA, s0)` prenant en entrée la liste `LS` des sommets du graphe, un dictionnaire `dA`, décrivant le graphe, tel que défini à la **question 2**, et un sommet `s0`, désigné par son nom, utilisé comme sommet-origine, et qui renvoie deux dictionnaires, dont les clés sont les noms des sommets du graphe, associées, pour le premier, au poids d'un plus court chemin de `s0` vers le sommet-clé, et, pour le second, l'avant-dernier sommet le long d'un plus court chemin de `s0` vers le sommet-clé (*i.e.* le prédécesseur du sommet-clé le long d'un tel chemin).

Question 5. Définir une fonction `plus_court_chemin(LS, dA, s1, s2)`, faisant appel à la fonction `bellman_ford`, et renvoyant, sous la forme d'un tuple, la longueur d'un plus court chemin depuis un sommet `s1` jusqu'à un sommet `s2`, ainsi que la liste des sommets le long d'un tel plus court chemin.

L'algorithme de Bellman-Ford n'est correct que dans un graphe sans cycle de poids négatif. On peut détecter la présence d'un tel cycle (à condition qu'il soit accessible depuis le sommet s) de la façon suivante : il y a un cycle de poids négatif si et seulement si un nouveau tour de boucle fait diminuer une distance. Ainsi, à la fin de l'algorithme, on fait :

```
pour chaque arc (u, v) du graphe faire
 |    si d[v] > d[u] + poids(u, v) alors
 |         afficher "il existe un cycle absorbant"
```

Question 6. Compléter votre implémentation de la **question 4**, comme prescrit ci-dessus, mais en utilisant une instruction `assert`.

Question 7. Vérifier que le graphe proposé ne comporte pas de cycle absorbant. Puis modifier le poids d'un arc du graphe de sorte à créer un cycle absorbant et vérifier qu'il est détecté.

On donne la complexité de cet algorithme

La complexité de l'algorithme est en $O(|S||A|)$ où $|S|$ est le nombre de sommets et $|A|$ est le nombre d'arcs. Cela correspond à une complexité en $O(|S|^3)$ pour un graphe simple dense[5].

Question 8. Justifier la complexité annoncée à partir de votre implémentation.

On donne à titre d'information la preuve de correction de l'algorithme :

La démonstration de la correction de l'algorithme peut se faire par récurrence :

Lemme. Après i répétitions de la boucle principale :

- Si $d[u] \neq +\infty$, alors $d[u]$ est le poids d'un chemin de s à u ;
- S'il existe un chemin de s à u d'au plus i arêtes, alors $d[u]$ est au plus la longueur du plus court chemin de s à u comprenant au plus i arêtes.

Preuve. Pour le cas $i = 0$, correspondant à la première exécution de la boucle `for`, alors, d'une part, $d[s] = 0$ et, pour tout sommet $u \neq s$, $d[u] = +\infty$, prouvant le premier point et, d'autre part, s est le seul sommet tel qu'il existe un chemin d'au plus 0 arêtes le reliant à s.

Pour le cas i non nul quelconque, alors :

- D'une part, supposons $d[v]$ soit mis à jour avec $d[v] := d[u] + \text{poids}(u, v)$. Alors, comme $d[u] \neq +\infty$ (car sinon, la mise à jour n'aurait pas lieu), $d[u]$ représente le poids d'un chemin de s à u. Donc, par construction, $d[v]$ est le poids d'un chemin de s à v ;
- D'autre part, soit C, un plus court chemin de s à v d'au plus i arêtes. Soit u le dernier sommet avant v sur ce chemin. Soit C', le sous-chemin de C allant de s à u. Alors, par induction, C' est un plus court chemin de s à u d'au plus $i - 1$ arêtes (en effet, sinon, il existe un chemin strictement plus court de s à u. En y ajoutant l'arête de u à v, on trouve un chemin strictement plus court que C allant de s à v, ce qui est contradictoire avec la définition de C). Alors, par hypothèse de récurrence, à l'issue de l'itération $i - 1$, $d[u]$ était au plus la longueur d'un plus court chemin de s à u d'au plus $i - 1$ arêtes. Ainsi, à la i-ème itération, $d[v] \leq d[u] + \text{poids}(u, v)$, en raison de la structure de l'algorithme. Or, cette dernière longueur est au plus la longueur d'un plus court chemin de s à v.

Le lemme est donc démontré. Il s'ensuit que la longueur $d[u]$ est la longueur d'un plus court chemin de s à u, ce qui prouve la correction de l'algorithme de Bellman-Ford dans le cas où G n'a pas de cycle de poids négatif[5].