T.P. d'informatique n° 6-2

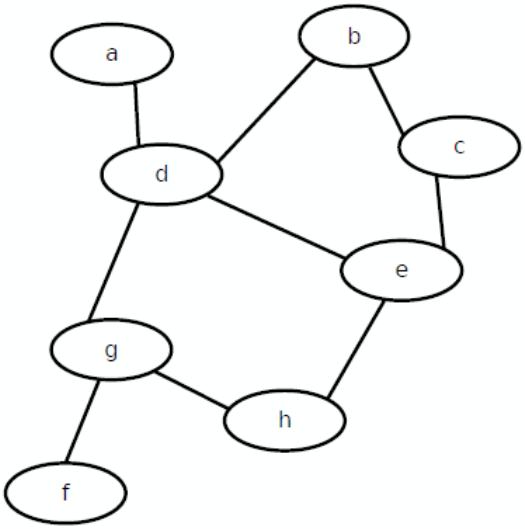
Algorithmique des graphes - Dijkstra

Généralités

On appelle **graphe** tout couple (V, E) tel que :

- V est un ensemble fini
- E est un ensemble de paires d'éléments de V

Les éléments de V sont appelés **sommets** (vertices en anglais) et ceux de E **arêtes** (edges en anglais).
Une arête e est donc une paire $\{x, y\}$ de sommets. Dans ce cas, on dit que x et y sont les **extrémités** de e et si $x = y$, on dit que e est une boucle.
Si le graphe (V, E) n'a pas de boucle, on dit qu'il est **simple**.
Exemple : Si $V = \{a, b, c, d, e, f, g, h\}$ et $E = \{\{a, d\}, \{b, c\}, \{b, d\}, \{c, e\}, \{e, d\}, \{e, h\}, \{g, d\}, \{f, g\}, \{h, g\}\}$, on peut représenter le graphe (V, E) par :



Soient (V, E) un graphe et x et y dans V.

- Le cardinal de V (i.e. le nombre de sommets) est appelé **ordre** du graphe.
- Les sommets x et y sont dits **adjacents** ou **voisins** sssi $\{x, y\}$ est dans E.
- On appelle **degré** de x et on note $d(x)$, le nombre de voisins de x.

Dans l'exemple précédent, l'ordre du graphe est : *notez votre réponse*
Notez sur le graphe, le degré de chacun de ses sommets

Soit (V, E) un graphe d'ordre $n \in \mathbb{N}^*$. On note $V = \{x_1, \ldots, x_n\}$.
On appelle matrice d'adjacence de ce graphe la matrice $(a_{i,j})_{1 \leqslant i,j \leqslant n} \in \mathcal{M}_n(\mathbb{R})$ vérifiant, pour tout (i, j), $a_{i,j}$ vaut 1 si les sommets x_i et x_j sont adjacents et 0 sinon.

Dans l'exemple précédent, on pose $a = x_1$, $b = x_2$, …, $h = x_8$. La matrice d'adjacence est alors $\begin{pmatrix} - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \end{pmatrix}$

On appelle **graphe pondéré** tout graphe dont les arêtes sont dotés d'un poids ;
Un graphe pondéré d'ordre $n \in \mathbb{N}^*$ pour lequel on note $V = \{x_1, \ldots, x_n\}$, est entièrement connu par sa **matrice d'adjacence** généralisation de la matrice d'adjacence vue précédemment :

la matrice d'adjacence de ce graphe pondéré est la matrice $(a_{i,j})_{1\leqslant i,j\leqslant n} \in \mathcal{M}_n(\mathbb{R})$ vérifiant, pour tout (i,j), $a_{i,j}$ vaut p où p est le poids de l'arête reliant x_i et x_j si les sommets x_i et x_j sont adjacents et 0 sinon.
On appelle **chemin** toute liste $(x_0, \ldots, x_k)$ de sommets telle que , pour tout $i \in [\![0, k-1]\!]$, x_i et x_{i+1} sont adjacents.
On appelle **poids** ou **longueur** d'un chemin, la somme des poids des arêtes qui le constituent.

Algorithme de Dijkstra

L'algorithme de Dijkstra est un algorithme de recherche de plus court chemin dans un graphe pondéré à poids positifs. Pour fixer les idées, le poids de l'arête (i,j) correspond à la distance nécessaire pour parcourir l'arête du graphe allant du $i^{\text{ème}}$ au $j^{\text{ème}}$ sommet (ou noeud) du graphe. Le graphe est entièrement représenté par sa matrice d'adjacence qui contient les poids des arêtes : c'est une matrice symétrique.
En conformité avec la numérotation des listes chez Python, n étant l'ordre de notre graphe, on num̀erote les sommets de 0 à $n-1$.

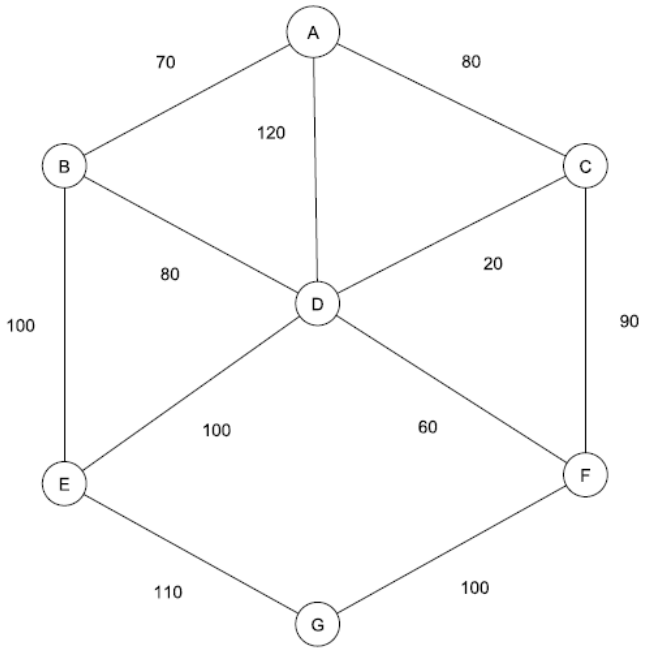
Principe de l'algorithme de Dijkstra

Il s'agit de construire progressivement, à partir des données initiales, un sous-graphe des sommets parcourus en gardant en mémoire (par exemple dans une liste) leur distance minimale aus sommet de départ. La distance correspond à la somme des poids des arêtes empruntées.
Au départ, on considère que les distances de chaque sommet au sommet de départ sont infinies.
Au court de chaque itération, on va mettre à jour les distances des sommets voisins (ou adjacents) du dernier sommet du sous-graphe des sommets parcourus (en ajoutant le poids de l'arc à la distance séparant ce dernier sommet du sommet de départ ; si la distance obtenue ainsi est supérieure à celle qui précédait, la distance n'est cependant pas modifiée).
Après cette mise à jour,on examine l'ensemble des sommets n,on parcourus (autrement dit des sommets qui ne font pas partie du sous-graphe des sommets parcourus), et on choisit celui dont la distance est minimale pour l'ajouter au sous-graphe des sommets parcourus.

La première étape consiste à ne considérer comme parcouru que le sommet de départ et à lui attribuer une distance de 0. Les sommets qui lui sont voisins sont mis à jour avec une valeur égale au poids de l'arc qui les relie au sommet de départ (ou à celui de poids le plus faible si plusieurs arcs les relient) et les autres sommets conservent leur distance infinie.
Le plus proche des sommets voisins est alors ajouté au sous-graphe des ommets parcourus.

La seconde étape consiste à mettre à jour les distances des sommets adjacents à ce dernier. Encore une fois, on recherche alors le sommet non parcouru doté de la distance (au sommet de départ) la plus faible Comme tous les sommets n'avaient plus une valeur infinie, il est possible que le sommet choisi ne soit pas un des derniers mis à jour.
On l'ajoute au sous-graphe des sommets parcourus, puis on continue ainsi à partir du dernier sommet ajouté, jusqu'à épuisement des sommets (ou jusquà ce que le sommet d'arrivée soit parcouru).

Appliquer l'algorithme "à la main" sur l'exemple suivant



Question 1 : Ecrire une fonction **listeSommetsVoisins** prenant en argument un **graphe** (en fait une matrice d'adjacence) et un entier **sommet** (correspondant à l'indice du sommet dont on cherche la liste des voisins) et retournant la liste des indices des sommets voisins (ou adjacents)

Question 2 : Ecrire une fonction **plusCourtChemin** prenant en argument un **graphe** (en fait une matrice d'adjacence) et deux entiers (correspondant aux indices des sommets de départ et d'arrivée) et retournant le chemin obtenu par l'algorithme de Dijkstra.
Pour traduire l'algorithme, on conseille d'utiliser les variables locales suivantes dans la fonction :

- **nbSommets** est un entier égal à l'ordre du graphe
- **distanceParcourue** est une liste dont le $i-$ème terme est la distance du sommet i au sommet de départ (et les distances sont toutes initialisées à -1, la valeur -1 signifiant ici "∞")
- **sommetPrecedent** est une liste dont le $i-$ème terme est le sommet précédent le sommet i (tous les termes sont initialisés à -1 et **sommetPrecedent[i]** est actualisé en même temps que **distanceParcourue[i]**)
- **nonParcourus** est la liste des indices des sommets du graphe non encore parcourus.

Tester avec l'exemple ci-dessus dont la matrice d'adjacence est

$$
\begin{array}{lrrrrrrrl}
graphe = & [& [0, & 70, & 80, & 120, & 0, & 0, & 0 &], \\
 & & [70, & 0, & 0, & 80, & 100, & 0, & 0 &], \\
 & & [80, & 0, & 0, & 20, & 0, & 90, & 0 &], \\
 & & [120, & 80, & 20, & 0, & 100, & 60, & 0 &], \\
 & & [0, & 100, & 0, & 100, & 0, & 0, & 110 &], \\
 & & [0, & 0, & 90, & 60, & 0, & 0, & 100 &], \\
 & & [0, & 0, & 0, & 0, & 110, & 100, & 0 &] &]
\end{array}
$$