TD : Algorithme de Prim et de Dijkstra

4 novembre 2022

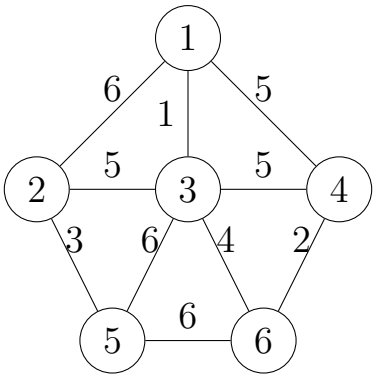
1 Algorithme de Prim.

L'algorithme de Prim est un algorithme glouton qui calcule un arbre couvrant minimal dans un graphe connexe pondéré et non orienté. On rappelle qu'un arbre est un graphe connexe sans cycle.

Définition 1

Soit $G = (\mathcal{A}, S)$ un graphe. Alors $T = (\mathcal{A}', S)$ est un arbre couvrant de G si $\mathcal{A}' \subseteq A$.

Dans toute la suite, G est un un graphe connexe pondéré et non orienté. On le codera en Python à l'aide d'un dictionnaire. Nous allons travaillé avec le graphe suivant :



Donc par exemple ici on aura en python :

```
1 Graphe = {1:{2:6 ,3:1 ,4:5} ,2:{..... } ,..... }.
```

1 Combien d'arêtes le graphe cherché aura-t-il ?

Le principe de l'algorithme est le suivant.

- Supposons T partiellement construit alors on a $S_G = S_T \cup S_{G\setminus T}$ où $S_{G\setminus T}$ est l'ensemble des sommets qui sont dans G mais qui ne sont pas dans T. On

cherche l'arête du type $\{s, s'\}$ avec $(s, s') \in S_T \times S_{G\setminus T}$ de poids minimum. Lorsqu'on l'a trouvé, on met à jour le poids de T, on ajoute s' à T et on ajoute l'arête $\{s, s'\}$ à T. On réitère jusqu'à obtenir la condition de la question précédente.

- On initialise T au sommet de départ. On peut montrer que le sommet choisit au départ n'a pas d'importance.

2 Ecrire une fonction *ordre* qui prend un dictionnaire en entrée et qui renvoie le nombre de clefs dans ce dictionnaire.

3 Ecrire une fonction *liste_sommets_complementaires* qui prend en entrée deux dictionnaires *dico* et *dico*2 et qui renvoie une liste contenant les clefs qui sont dans *dico* mais qui ne sont pas dans dico2.

4 Donner la complexité de la fonction *liste_sommet_complementaire* si *dico* a n_1 clefs et *dico*2 a n_2 clefs.

5 Ecrire une fonction *Prim* qui prend en entrée un dictionnaire *dico* qui est le dictionnaire associé au graphe G et *depart* un des sommets de G. La fonction renvoie l'arbre couvrant de poids minimal dans un dictionnaire noté *Arbre_couvrant* et le poids de la somme de toutes les arêtes de cet arbre.

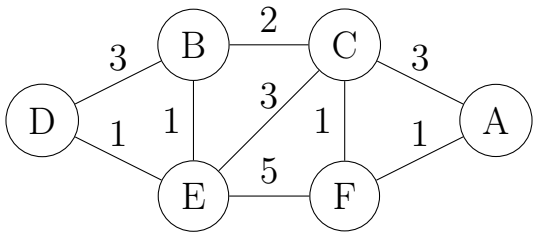
6 Que doit retourner *Prim* dans notre exemple ?

2 Algorithme de Dijkstra.

L'algorithme de Dijkstra sert à résoudre le problème du plus court chemin. Il permet, par exemple, de déterminer un plus court chemin pour se rendre d'une ville à une autre connaissant le réseau routier d'une région. Plus précisément, il calcule des plus courts chemins à partir d'une source $s \in \mathcal{S}$ vers tous les autres sommets dans un graphe pondéré par des réels positifs. On peut aussi l'utiliser pour calculer un plus court chemin entre un sommet de départ et un sommet d'arrivée. Dans toute la suite $\mathcal{G} = (\mathcal{A}, S)$ est un graphe non orienté. On définit la matrice d'adjacence $G = (g_{i,j})_{(i,j)\in\mathcal{S}^2}$ de la manière suivante dans ce cas

- Si $(i, j) \in \mathcal{A}$ alors $g_{i,j}$ est égal au poids de l'arête $\{i, j\}$.
- Pour $i \in S$, $g_{i,i} = 0$.
- Si $(i, j) \notin \mathcal{A}$ alors $g_{i,j} = +\infty$.

On considère le graphe pondéré suivant :



7 Donner la matrice d'adjacence de notre exemple. On numérotera dans l'ordre croissant les sommets : D,B,E,C,F et A.

Le principe est le suivant :

- Notation : Pour tout sommet v du graphe on note $\delta(v)$ la longueur du plus court chemin menant de s à v.

Avant de déterminer le plus court chemin entre s et tout autre sommet t on commence par déterminer la longueur de chacun de ces plus court chemins. Autrement dit pour déterminer $\delta(t)$ il nous faut nécessairement connaitre toutes les valeurs prisent par la fonction δ.

Pour ce faire nous allons introduire la variable $d[v]$ qui contient le poids du supposé plus court chemin menant de s à v. Nous allons corriger cette valeur au fur et à mesure de l'algorithme si nous trouvons un chemin plus court de tel sorte qu'à la dernière étape on aura $d[v] = \delta(v)$. Ainsi on aura tout au long de l'algorithme :

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

Nous allons corriger notre valeur $d[v]$ de la manière suivante :

- **Initialisation** : Au début de l'algorithme on considère :
  - $d[s] = 0$
  - $\forall v \in \mathcal{S} \setminus \{s\}, d[v] = +\infty$.

  Dans notre exemple notre ville de départ est D. On a l'habitude de donner les valeurs de $d[v]$ sous forme d'un tableau que l'on construit au fur et à mesure en suivant le principe de l'algorithme. Ici on a

D	B	E	C	F	A

- **Sélection du sommet** u : A chaque étape de l'algorithme on sélectionne le sommet u qui vérifie les propriétés suivantes :
  - u n'a pas encore été sélectionné.
  - $d[u]$ est le plus petit parmi tous les sommets non encore sélectionnés.
- **Mise à jour de** $d[v]$: S'il est possible en passant par u d'obtenir un plus court chemin jusqu'à v alors il faut mettre $d[v]$ à jour. Le chemin pour aller de s à v en passant par u à pour longueur . On doit donc tester l'inégalité

.

- Si le test est négatif alors on ne modifie pas $d[v]$.
- Si le test est positif alors $d[v] =$ et il faut se souvenir qu'on vient du sommet u.

1. **L'algorithme prend fin** lorsque tous les sommets ont été visités. On a donc trouvé tous les plus courts chemins de s à v pour $v \in \mathcal{S}$.

Remarque 1

A chaque étape on sélectionne la sous-solution optimale c'est-à-dire le sommet réalisant la plus petite distance. C'est donc un algorithme glouton !

7 Compléter le tableau précédent en suivant le fonctionnement de l'algorithme de Dijkstra en prenant comme sommet de départ le sommet D.

8 Déterminer en appliquant l'algorithme le plus court chemin de D vers A puis de D vers F puis de D vers B.

Dans un premier temps nous voulons récupérer un dictionnaire noté *distance* qui a chaque clef $v \in S$ associe un dictionnaire qui comprendra trois clefs : "distance" qui associera la distance au sommet de départ, "visite" qui associera un booléen qui précise si le sommet a été visité ou non et "*sommet_precedent*" qui associe le sommet d'où l'on venait pour choisir ce sommet.

9 Ecrire une fonction *initialisation_distance* qui prend en entrée un dictionnaire *dico* associé au graphe et *depart* le sommet de départ et qui initialise le dictionnaire *distance*.

10 Ecrire une fonction *rech_sommet_min* qui prend en entrée le dictionnaire distance et qui renvoie le sommet à distance minimale parmi les sommets non visités.

11 Ecrire une fonction *Dijkstra_distance* qui prend en entrée un dictionnaire *dico* associé au graphe et le sommet de départ et qui renvoie un dictionnaire *distance* après avoir appliqué l'algorithme de Dijkstra.

12 Ecrire une fonction *Dijkstra_plus_court_chemin* qui prend en un dictionnaire *dico* associé au graphe, le sommet de départ et le sommet d'arrivé et qui renvoie une liste notée *chemin* contenant les sommets dans l'ordre du chemin le plus court pour aller de *depart* à *arrive*.