Algorithme de Dijkstra

Introduction

L'objectif de ce TP de graphe est de réaliser un programme en Python permettant de trouver le chemin le plus court d'un graphe à l'aide de l'algorithme de Dijsktra. Dans la suite de ce document, je présente dans un premier temps (en partie 1) l'algorithme de Dijkstra par la théorie. J'expose ensuite (en section 2) le programme que j'ai réalisé. Enfin j'explique (en section 3) le jeux de tests effectué pour confirmer le bon fonctionnement de mon programme.

1. L'algorithme de Dijsktra par la théorie

En théorie des graphes, l'algorithme de Dijkstra sert à résoudre le problème du plus court chemin (à partir du moment où les coûts sont positifs). Il permet par exemple, de déterminer un plus court chemin pour se rendre d'une ville à une autre connaissant l réseau routier d'une région. Plus précisément, il calcule des plus courts chemins à partir d'une source dans un graphe orienté et 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'arrivé.

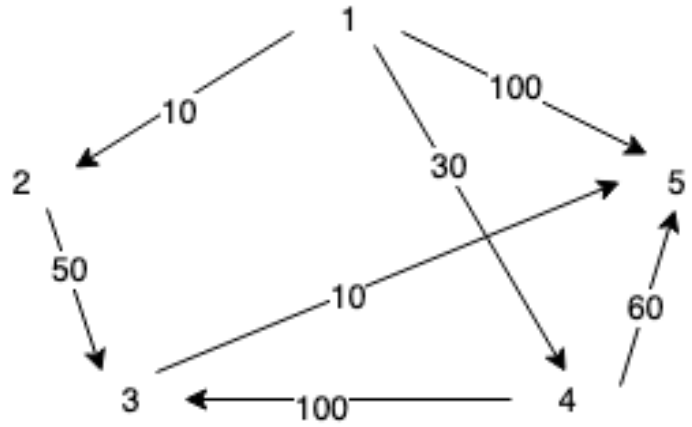

Figure 1 : Exemple de graphe pour l'algorithme de Dijkstra

Comme on peut le voir à l'aide de la figure 1 ci-dessus, ce graphe pour être pris en charge par Dijkstra car il ne comporte pas de poids inférieur ou égal à zéro.

E	**Nœuds accessibles**	**2**	**3**	**4**	**5**	**Nœud le plus proche**
{1}	{2,4,5}	10	Inf.	30	100	2
{1,2}	{3,4,5}		60	30	100	4
{1,2,4}	{3,5}		50		90	3
{1,2,3,4}	{5}				60	5

Tableau 1 : Processus de l'algorithme de Dijkstra

Le tableau 1 ci-dessus présente le déroulement de l'algorithme sur le graphe de la figure 1. Tout d'abord, la première colonne correspond au sommets dont on connaît la plus courte distance à la source. Les colonnes 3 à 6 correspondent à la valeur courante du chemin le plus court de i à la source s.

Tout d'abord, on choisit comme source le point 1. On regarde les nœuds accessibles par rapport à cette source et on les note comme accessibles dans la colonne 2. Ensuite, pour chacun de ces points, on note la valeur courante entre la source et le point. Les nœuds inaccessibles sont notés comme étant à l'infinie. Une fois ce travail fait, on en déduit le nœuds

le plus proche (étant caractérisé par la valeur courante la plus petite). On répète ensuite, ce processus avec les autres points jusqu'à couvrir l'ensemble du graphe.

2. L'algorithme de Dijsktra par la programmation

Ce programme se déroule de la manière suivante. On sélectionne le point de départ et d'arrivé du graphe. Si un des points n'est pas présent dans le graphe, alors on lève une exception et on arrête le programme. Ensuite, on sélectionne le point de départ et s'il n'est pas marqué comme étant visité, alors on met sa distance à 0. Après ça, On visite les successeurs de ce point et on calcule leur poids par rapport au point de départ. Une fois cela fait, on marque le point de départ comme étant visité et on met les points non visités avec une distance à l'infini. On exécute ensuite une fonction récursive. Cette fonction prend en paramètre le graphe, le point départ correspondant au point non visité avec la plus petite distance, le point d'arrivé, le tableau des points visités, le tableau des distances et le tableau de prédécesseurs. Une fois l'ensemble des points du graphe traité, on liste le meilleur chemin de routage et on l'affiche.

3. Les jeux de tests

L'objectif de cette partie est de montrer le bon fonctionnement de mon programme. Pour cela, j'ai d'abord réalisé un premier test avec le graphe défini dans la partie 1 de ce rapport. Voici, dans le tableau 2, le graphe sous la forme d'un dictionnaire.

```
graph = {
    '1':{
        '2':10,
        '4': 30,
        '5':100
    },
    '2':{
        '3':50
    },
    '3':{
        '5':10,
    },
    '4':{
        '3':20,
        '5':60
    },
    '5':{
        # Pas de successeurs
    }
}
```

Tableau 2 : Dictionnaire de données comportant les points du graphe et leurs successeurs

Ce dictionnaire comporte deux niveaux. Le premier niveau correspond aux différents sommets du graphe. Le second niveau correspond aux successeurs avec leurs poids, du point courant.

En exécutant le programme avec comme source le point « 1 » et comme arrivé le point « 5 », on obtient le chemin le plus court suivant :

Le chemin le plus court : ['1', '4', '3', '5'] avec une distance de 60.

Cela correspond donc bien aux valeurs obtenue dans le tableau de la partie 1.

Dans le cas où l'on choisit un point de départ ou de sortie qui n'est pas dans le graphe, nous obtenons ce résultat :

```
Point de départ :
Point d'arrivé :
Traceback (most recent call last):
  File "/Users/sueur/dev/enssat/dijkstra_project/dijkstra.py", line 60, in <module>
    dijkstra(graph,startPoint,destinationPoint)
  File "/Users/sueur/dev/enssat/dijkstra_project/dijkstra.py", line 4, in dijkstra
    raise Exception('Le point de départ n\'existe pas.')
Exception: Le point de départ n'existe pas.
```

Le seul test qui n'a pas été réalisé dans ce programme est le cas où l'on utilise un graphe avec un sommet comportant un poids négatif.

Conclusion

Lors de ce TP de graphe, j'ai effectué dans un premier temps une présentation par la théorie de l'algorithme de Dijkstra. Ensuite, j'ai expliqué le programme réalisé. Enfin j'ai réalisé différents jeux de tests afin de montrer le bon fonctionnement de mon programme. Ce TP m'a permis dans un premier temps de comprendre comment fonctionnait l'algorithme de Dijkstra. Il m'a aussi permis de découvrir le langage Python qui était pour moi, jusqu'à maintenant, inconnu.