INSA – MS
ROUFFIAC Jean-Eudes

Année 2019-2020

Algorithmique des graphes

Rapport de TP n°3

Titre : *Algorithme de Dijkstra*

1 Le problème

Dans ce TP, nous voulons programmer l'algorithme de Dijkstra, qui est un algorithme de complexité polynomial permettant de trouver les plus courts chemins dans un graphe orienté pondéré (positivement) à partir d'un sommet.

2 Résolution du problème

Rappelons l'algorithme de Dijkstra.

début
$\pi(s) = 0$
pour $i \in S(i)$ **faire**
$\pi(i) = l(si)$
p(i) = s
fin
pour $i \notin S(i)$ **faire**
$\pi(i) = +\infty$
fin
S = s
$\bar{S}$ = X - s
tant que $\bar{S} \neq \emptyset$ **faire**
$i = arg \min_{j \in \bar{S}} \pi(j)$
$S = S \bigcup i$
$\bar{S}$ = S - i
pour $j \in S(i) \bigcap \bar{S}$ **faire**
si $\pi(i) + l(ij) < \pi(j)$ **alors**
$\pi(j) = \pi(i) + l(ij)$
$p(j) = i$
fin
fin
fin
retourner (π, p)**fin**

Algorithme 1 : Dijkstra(graph, s)

Le code complet en python est le suivant :

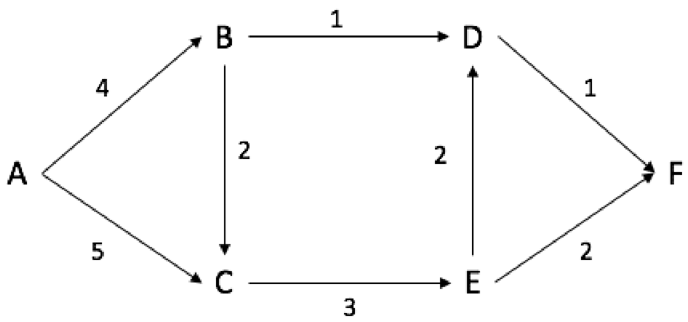
```python
import numpy as np

def dijkstra(graph, point_depart):

#Initialisation
    pi, p = dict(), dict()
    S, S_bar = [], []
```

```python

    for sommet in graph:
        if sommet != point_depart:
            pi[sommet], p[sommet] = float('inf'), None
            S_bar.append(sommet)

    for voisin in graph[point_depart]:
        pi[voisin] = graph[point_depart][voisin]
        p[voisin] = point_depart

    S.append(point_depart)

    pi[point_depart] = 0

#Iteration
    while (len(S_bar) != 0 ) :

        temp = [pi[j] for j in S_bar]
        i = S_bar[np.argmin(temp)]

        S.append(i)
        S_bar.remove(i)

        for voisin in graph[i] :
            if voisin in S_bar :
                if pi[voisin] > pi[i] + graph[i][voisin]:
                    pi[voisin] = pi[i] + graph[i][voisin]
                    p[voisin] = i


    return (pi, p)
```

3 Résultats

3.1 Exemple 1

Prenons le graphe suivant :



En entrée de l'algorithme, on donne la liste des successeurs avec la longueur de chaque arc. Ainsi, en python, on représente le graphe de la manière suivante :

```
graph1 = {
    'a': {'b': 4, 'c':  5},
    'b': {'c': 2, 'd':  1},
    'c': {'e': 3},
    'd': {'f':  1},
    'e': {'d': 2, 'f': 2},
    'f': {}
}
```

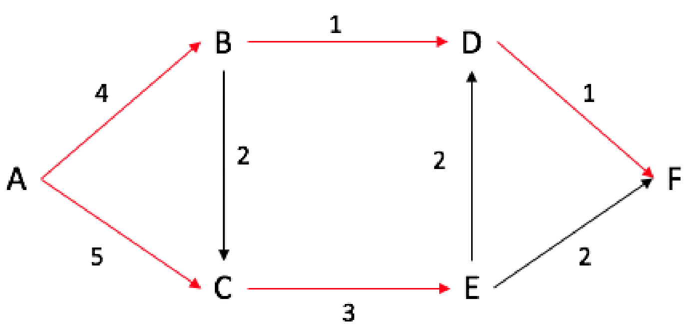
On appelle alors notre fonction avec entrée l'objet graph, ainsi que le 1er élément à savoir le sommet 'A'.

```
pi, p = dijkstra(graph1, point_depart='a')
```

Le résultat donné par le programme est alors le suivant :

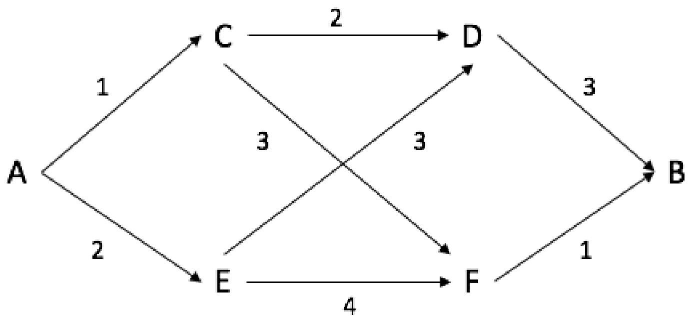
```
{'b': 4, 'c': 5, 'd': 5, 'e': 8, 'f': 6, 'a': 0}
{'b': 'a', 'c': 'a', 'd': 'b', 'e': 'c', 'f': 'd'}
```

La première ligne correspond au coût pour arriver au sommet x. La deuxième liste donne les sommets avec leur prédécesseur. On peut ainsi construire les plus courts chemins à partir du sommet de départ.



3.2 Exemple 2

Prenons le graphe suivant :



En entrée de l'algorithme, on donne la liste des successeurs avec la longueur de chaque arc. Ainsi, en python, on représente le graphe de la manière suivante :

```python
graph2 = {
    'a': {'c': 1, 'e': 2},
    'b': {},
    'c': {'d': 2, 'f': 3},
    'd': {'b': 3},
    'e': {'d': 3, 'f': 4},
    'f': {'b': 1}
}
```

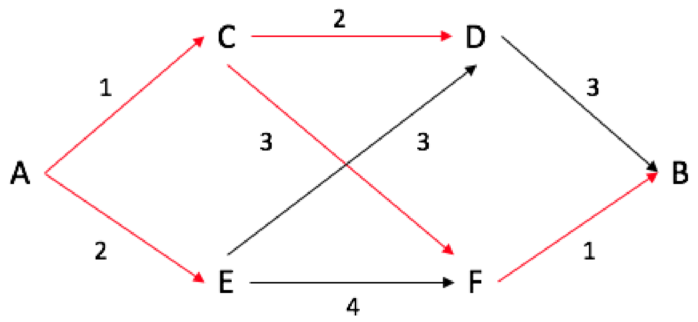
On appelle alors notre fonction avec entrée l'objet graph, ainsi que le 1er élément à savoir le sommet 'A'.

```python
pi, p = dijkstra(graph2, point_depart='a')
```

Le résultat donné par le programme est alors le suivant :

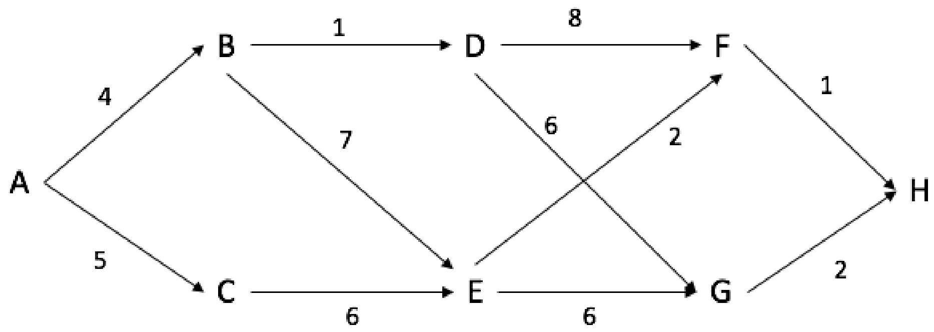
```
{'b': 5, 'c': 1, 'd': 3, 'e': 2, 'f': 4, 'a': 0}
{'b': 'f', 'c': 'a', 'd': 'c', 'e': 'a', 'f': 'c'}
```

La première ligne correspond au coût pour arriver au sommet x. La deuxième liste donne les sommets avec leur prédécesseur. On peut ainsi construire les plus courts chemins à partir du sommet de départ.



3.3 Exemple 3

Prenons le graphe suivant :



En entrée de l'algorithme, on donne la liste des successeurs avec la longueur de chaque arc. Ainsi, en python, on représente le graphe de la manière suivante :

```python
graph3 = {
    'a': {'b': 4, 'c': 5},
    'b': {'d': 1, 'e': 7},
    'c': {'e': 6},
    'd': {'f': 8, 'g': 6},
    'e': {'f': 2, 'g': 6},
    'f': {'h': 1},
    'g': {'h': 2},
    'h': {}
}
```

On appelle alors notre fonction avec entrée l'objet graph, ainsi que le 1er élément à savoir le sommet 'A'.

```python
pi, p = dijkstra(graph3, point_depart='a')
```

Le résultat donné par le programme est alors le suivant :

```
{'b': 4, 'c': 5, 'd': 5, 'e': 11, 'f': 13, 'g': 11, 'h': 13, 'a': 0}
{'b': 'a', 'c': 'a', 'd': 'b', 'e': 'b', 'f': 'd', 'g': 'd', 'h': 'g'}
```

La première ligne correspond au coût pour arriver au sommet x. La deuxième liste donne les sommets avec leur prédécesseur. On peut ainsi construire les plus courts chemins à partir du sommet de départ.