INSA – MS Année 2019-2020
ROUFFIAC Jean-Eudes

Algorithmique des graphes

Rapport de TP n°2

Titre : *Algorithme de Ford*

1 Le problème

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

2 Résolution du problème

Rappelons l'algorithme de Ford.

début
    $\pi(sommet\_depart) = 0$
    p (sommet_depart) = vide
    k = 0
    B = Vrai
    tant que B *and* $k < n$ **faire**
        B = Faux
        pour *arc* $ij \in U$ **faire**
            si $\pi(i) + l(ij) < \pi(j)$ **alors**
                $\pi(j) = \pi(i) + l(ij)$
                p(j) = i
                B = Vrai
            fin
        fin
        k = k + 1
    fin
    retourner (π, p)**fin**

Algorithme 1 : Ford(graph, sommet_depart)

Le code complet en python est le suivant :

```python
import sys

def ford(graph, point_depart):

    pi, p = dict(), dict()

    for sommet in graph:
        pi[sommet] = float('inf')
        p[sommet] = None

    pi[point_depart] = 0
    k = 0
    b = True
```

```
    while (b and k < len(graph)) :
        b = False
        for sommet in graph :
            for voisin in graph[sommet]:
                if pi[voisin] > pi[sommet] + graph[sommet][voisin]:
                    pi[voisin] = pi[sommet] + graph[sommet][voisin]
                    p[voisin] = sommet
                    b = True
        k = k + 1

    for sommet in graph:
        for voisin in graph[sommet]:
            if (pi[voisin] > pi[sommet] + graph[sommet][voisin]):
                sys.exit("Circuit absorbant")


    return (pi, p)
```

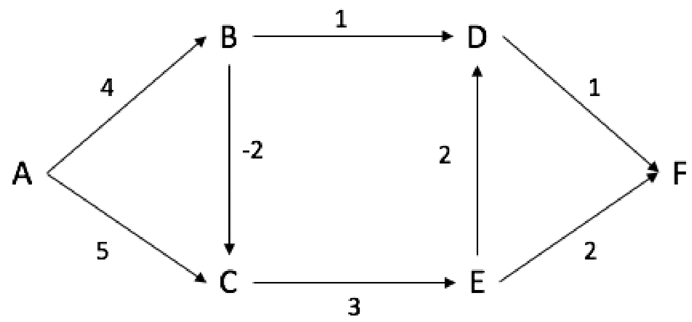
Les lignes suivantes dans la fonction permettent de chercher s'il y a un circuit absorbant dans le graphe. En effet, lorsque les itérations sont finies, si la condition ”pi[voisin] ¿ pi[sommet] + graph[sommet][voisin]” est encore vérifiée, alors nécessairement il y a un circuit absorbant dans le graphe, et donc il n'y a pas de plus courts chemins.

```
for sommet in graph:
    for voisin in graph[sommet]:
        if (pi[voisin] > pi[sommet] + graph[sommet][voisin]):
            sys.exit("Circuit absorbant")
```

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': {}
}
```

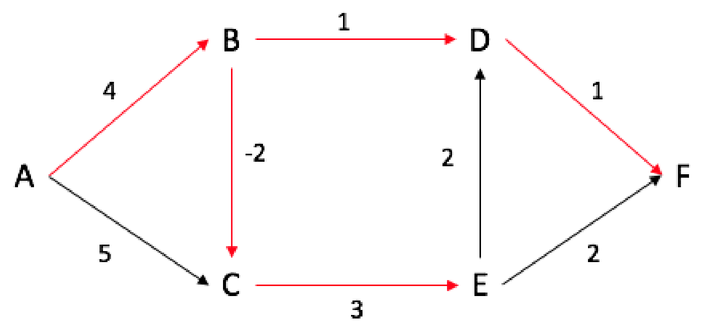
On appelle alors notre fonction avec entrée l'objet graph, ainsi que le 1er élément à savoir le sommet 'A'.

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

Le résultat donné par le programme est alors le suivant :

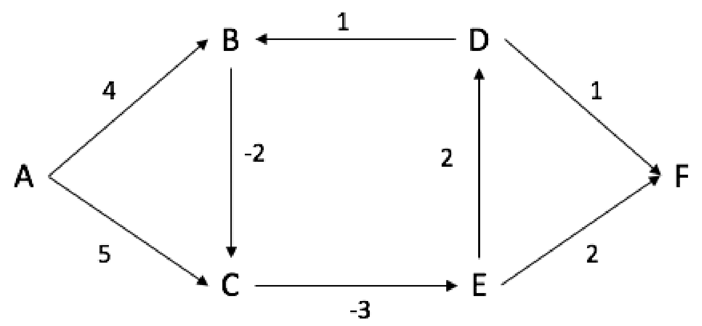
```
{'a': 0, 'b': 4, 'c': 2, 'd': 5, 'e': 5, 'f': 6}
{'a': None, 'b': 'a', 'c': 'b', 'd': 'b', 'e': 'c', 'f': 'd'}
```

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 :

```
graph2 = {
    'a': {'b': 4, 'c': 5},
    'b': {'c': -2},
    'c': {'e': -3},
    'd': {'b': 1, '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 = ford(graph2, point_depart='a')
```

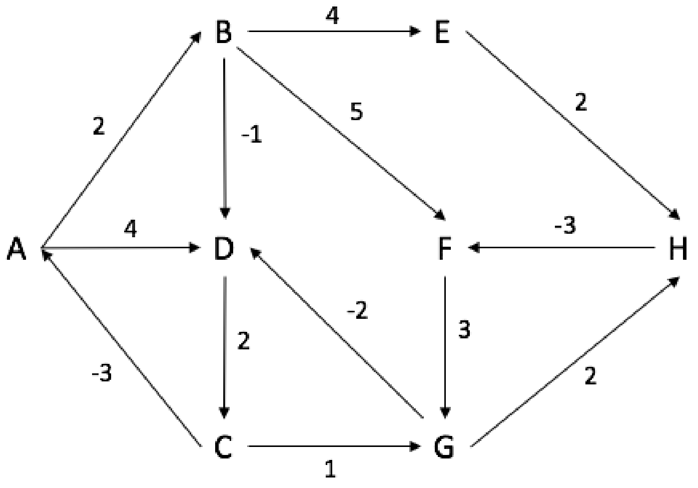
Le résultat donné par le programme est alors le suivant :

```
SystemExit: Circuit absorbant
```

En effet, dans le graphe, il y a un circuit absorbant (formée par les sommets BCDE). Le programme a donc renvoyé un bon résultat.

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 :

```
graph3 = {
    'a': {'b': 2, 'd': 4},
    'b': {'d': -1, 'e': 4, 'f': 5},
    'c': {'a': -3, 'g': 1},
    'd': {'c': 2},
    'e': {'h': 2},
    'f': {'g': 3},
    'g': {'d': -2, 'h': 2},
    'h': {'f': -3}
}
```

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

```
pi, p = ford(graph3, point_depart='a')
```

Le résultat donné par le programme est alors le suivant :

```
{'a': 0, 'b': 2, 'c': 3, 'd': 1, 'e': 6, 'f': 3, 'g': 4, 'h': 6}
{'a': None, 'b': 'a', 'c': 'd', 'd': 'b', 'e': 'b', 'f': 'h', 'g': 'c', '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.