TP 2 Recherche Opérationnelle
Licence III Informatique
Etude et implémentation d'algorithmes de plus courts chemins

Serigne A. Gueye

4 mars 2018

Soit un graphe $G = (V, E)$, connexe, où $V = \{1, 2, ..., n\}$ et dont chaque arc (arête) est valué par un coût (ou longueur) $c(e) \in \mathbb{R}$. L'objectif du TP est le développement en C++, et le test, des algorithmes de plus courts chemins vus en cours. Pour représenter, le graphe vous adapterez les structures de données utilisées dans le TP sur les flots.

1 Méthode Moore-Dijkstra

On considère dans cette section l'algorithme de Moore-Dijsktra vu en cours.

1.1 Implémentation

1- Ecrire une méthode :

MooreDijkstra(s)

calculant les plus courts chemins d'un sommet donné s à tous les autres sommets avec cette méthode.

• La méhode renverra le tableau “pere”. Pour tout sommet i, pere[i] contient l’indice du prédécesseur de i dans le plus court chemin de s à i.

• Le vecteur d donnant pour chaque indice i la valeur du plus court chemin de s à i sera dans un premier temps implémenté avec un tableau.

2- Les plus courts chemins devront être affichés par une méthode

display_shortest_paths(pere)

affichant à l’écran les plus courts chemins calculés.

3- Montrer que la complexité de l’algorithme de Dijkstra en terme de nombre de comparaisons est en $O(m) + O(n^2)$ où m est le nombre d’arcs du graphe, et n le nombre de sommets.

1.2 Amélioration de l’algorithme

Pour représenter d, on utilise maintenant une structure de tas (heap) [1] (au lieu d’un tableau).

4- Montrer que la complexité de l’algorithme de Dijkstra en terme de nombre de comparaisons devient $O(m\log(n))$.

5- Changer la structure de données utilisée pour d par un tas. Vous implémenterez au choix votre propre structure de tas, ou utiliserez la fonction template “make_heap” [2] de C++.

2 Méthode de Moore, 1957

On considère l’algorithme suivant.

1. voir notes de cours sur le site
2. voir http ://www.cplusplus.com/reference/algorithm/make_heap/

```
1  Algorithm: Moore(s)
2  C = {s} ;
3  C̄ = {1, 2, ..., n}\{s} ;
4  pour i ∈ C̄ faire
5      si i ∈ Γs alors
6          d(i) = c_si
7      fin
8      sinon
9          d(i) = +∞
10     fin
11 fin
12 tant que C̄ ≠ ∅ faire
13     d(j) = min_{i∈C̄} d(i) ;
14     C̄ = C̄\{j};
15     pour i ∈ Γ(j) faire
16         si d(j) + c_ji < d(i) alors
17             d(i) = d(j) + c_ji;
18             C̄ = C̄ ∪ {i};
19         fin
20     fin
21 fin
```

4 - Cet algorithme converge-t-il quand le graphe est sans circuit de longueur négative ? Si oui pourquoi ? [3].

5- Que se passe-t-il quand il est appliqué sur un graphe contenant un circuit de longueur négative ?

6- Si l'on remplace le sommet j choisi à la ligne 13 par un sommet quelconque de $\overline{C}$ cela change-t-il quelque chose à la convergence ?

7- Ajouter, comme pour l'algorithme de Moore-Dijkstra, une structure "pere" et des instructions permettant de stocker les plus courts chemins.

3. Montrer qu'il est équivalent à l'algorithme de Berge vu en cours

8- Si toutes les longueurs c_{ij} des arcs sont positives à quel algorithme est-il équivalent ?

3 Méthode de Bellman, 1958

9- Ecrire une méthode :

Bellman(s)

calculant les plus courts chemins d'un sommet donné s à tous les autres sommets avec cette méthode [4].

4. La méthode a implémenter est celle vue en cours