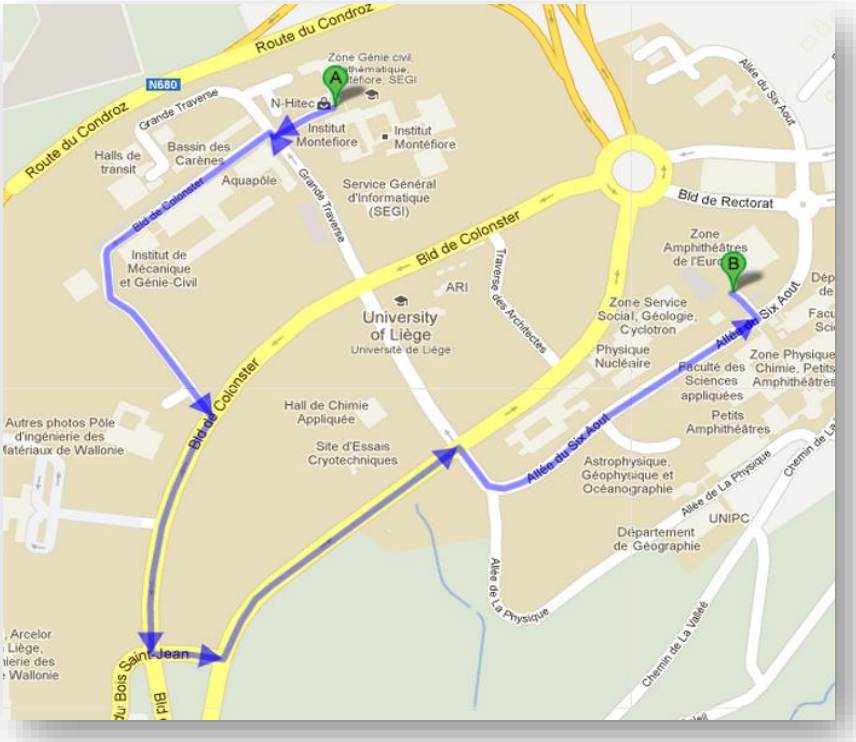
Projet de programation numérique

PLUS COURT CHEMIN



ARMAND AGBOGBA- AICHA LAMMAMRA- ALAIN KONAN SERGE ATTIEN- VANESSA TCHABOU

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

université PARIS-SACLAY

ENCADRANTE: *Laurence Pilard*

Année universitaire 2018/2019

TABLE DES MATIERES

LISTE DES FIGURES

I.INTRODUCTION

Le problème de plus court chemin en théorie des graphes est un problème algorithmique, qui consiste à trouver dans un graphe G la distance minimale d'un nœud source à tous les autres nœuds du Graphe, un graphe étant la donnée d'un ensemble V constitué des nœuds ou sommets du graphe et d'un ensemble A tel que pour deux nœuds u et v du graphe, (u,v) ∈ A signifie qu'il existe une arête entre les nœuds u et v. Dans le cadre de notre projet, nous utiliserons deux algorithmes de recherche de plus court chemin que sont les algorithmes de Dijkstra et de Bellman Ford. Notre travail ici consiste à implémenter une version séquentielle de chacun de ces deux algorithmes appliqués sur les graphes non orientés, tout en analysant les performances et en comparant les caractéristiques telles que la complexité et le temps d'exécution de ces algorithmes. Ici, le langage choisi est le C++, langage essentiellement utilisé en programmation orienté objet et pour le calcul scientifique.

II. ETUDE THEORIQUE DE L'ALGORITHME DE DIJKSTRA ET DE BELLMAN-FORD

1. Algorithme de Dijkstra

Description de l'algorithme

Etant donné un graphe G et un nœud s_0 (source), on associe à chaque sommet s_i du graphe une valeur d[si] qui représente la valeur maximale de la distance entre s_0 et s_i. Le principe de l'algorithme de Dijkstra est de minimiser les valeurs d[s_i]. Pour cela, on se donne deux ensembles E et F tel que EUF=V. On part de s_0 et pour chacun des nœuds s_i tel que s_0- s_i Є A, on améliore la distance minimale d[si], s_i sa valeur dépasse la somme de d[s_0] et du poids de l'arête s_0- s_i, d[S_0] étant nulle par définition. Ensuite on cherche le nœud si ayant la plus petite valeur de d[s_i] et on répète ainsi de suite de façon progressive le même calcul que précédemment à partir de s_i en considérant tous les nœuds s_j qui sont adjacents à s_i.

Algorithme de Dijkstra

// Déclaration de l'algorithme

G= Graphe (S, A) // S ensemble des sommets, A ensembles des arêtes
$\upsilon(s_i, s_j)$ // υ est la valeur du coût de l'arête entre le sommet s_i et s_j
s_0 // Sommet de départ

// Initialisation de l'algorithme

d $[s_0] \leftarrow 0$
Pour chaque sommet $s_i \in$ S, **faire**
  $d[s_i] \leftarrow \infty$
  II $[s_i] \leftarrow nil$

// Relaxation de l'algorithme

E $\leftarrow \emptyset$
F $\leftarrow S$
Tant que F $\neq 0$ **Faire**
  $s_i \leftarrow$ distance_mini (F, dis)
  E $\leftarrow$E $\cup \{s_i\}$
  Pour tout sommet $s_i \in$ succ $(s_i) \cap F$ **Faire** **// Relâchement successifs d'arête**
    Si $d[s_j] > d[s_i] + \upsilon(s_i, s_j)$ **Alors**
      $d[s_j] \leftarrow d[s_i] + \upsilon(s_i, s_j)$
      II $[s_j] \leftarrow d[s_i]$

1.2. Complexité de l'algorithme

Si on suppose que le graphe possède n sommets, en fonction du choix de notre implémentation qu'est la matrice, la complexité de l'algorithme de Dijkstra est de $O(n^2)$ étant donné que chaque arête est relâchée une fois et que le graphe comporte n sommets.

2. ALGORITHME DE BELLMAN FORD

2.1. Description de l'algorithme

L'algorithme de Bellman-Ford est basé sur le même principe que celui de Dijkstra: le relâchement. Contrairement à Dijkstra, l'algorithme de Bellman-Ford s'effectue en une série de n-1 relâchements successifs : on part de la source s_0, on relâche une première fois toutes les arêtes, après quoi tous les plus courts chemins de longueur 1 auront été trouvés. Après la k-ième série de relâchement des arêtes, tous les plus courts chemins de longueur k, partant de s_0, auront été trouvés. Si le graphe ne contient pas un cycle absorbant, l'algorithme donne les plus courts chemins ainsi que leurs poids. Au bout de n-1 relâchements, s'il existe une arête pour lequel un nouveau relâchement permettrait de diminuer la longueur minimale, alors le graphe contient nécessairement un cycle absorbant. L'algorithme utilise cette propriété pour détecter la présence de cycles absorbants.

Algorithme de Bellman Ford

// Déclaration des variables

G= Graphe (S, A) // S ensemble des sommets, A ensemble des arêtes

υ (s_i, s_j) =∞ // υ est la valeur des arêtes, s'il n'y a pas d'arête entre le sommet s_i et s_j

s_0 // Sommet de départ

// Initialisation de l'algorithme

$d\ [s_0] \leftarrow\ 0$

Pour chaque sommet s_i∈ S Faire

- $d[s_i] \leftarrow \infty$
- II $[s_i\ ] \leftarrow nil$

// Relaxation de l'algorithme

pour i ← 1 à S − 1 faire

Pour chaque arête $(s_i\ , s_j)$ ∈ A Faire

Si $d[s_j] >$ d$[s_i]$ + υ (s_i, s_j) Alors

- $d[s_j\ ] \leftarrow d[s_i\ ] + \upsilon\ (s_i, s_j\ )$
- II $[s_j\ ] \leftarrow d[s_i\ ]$

// Contrôle de la présence d'un cycle absorbant

Pour chaque arête$(s_i\ , s_j)$ ∈ A, faire

Si $d[s_j] >$ d$[s_i]$ + υ (s_i, s_j) Alors **// Relâchement successifs d'arête**

- Afficher ("Cycle absorbant")

Sinon

- Retour $d[s_j]$

2.2. Complexité de l'algorithme

Si le graphe comporte **n** sommets et **m** arêtes, chaque arête sera relâché **n-1** fois, et on effectuera donc au total **(n-1)m** relâchements successifs. En fonction du choix de notre implémentation qu'est la liste d'adjacence, on aura une complexité en **O(nm).**

3. Comparaison théorique entre les deux algorithmes: Dijkstra et Bellman-Ford

L'algorithme de Dijkstra et l'algorithme de Bellman-Ford ont pour but commun de déterminer le plus court chemin d'une source aux autres nœuds du graphe et procèdent par relâchement, Cependant, l'algorithme de Dijkstra est avidement relâché chaque arête une ou une seule fois.la tandis que dans L'algorithme de Bellman-Ford chaque arête est relâchée |S|-1 le nombre de sommets avec des distances correctement calculées augmente, d'où il résulte que tous les sommets auront finalement leurs distances correctes. Cette méthode permet d'appliquer l'algorithme Bellman – Ford à une classe d'entrées plus large que Dijkstra. L'autre différence est que l'algorithme de Dijkstra ne peut pas gérer les poids négatifs que Bellman-Ford gère. Bellman-Ford nous indique également si le graphique contient un cycle négatif. Si le graphique ne contient pas d'arêtes négatives, alors Dijkstra est toujours meilleur

En matière de vitesse d'exécution, nous voyons que l'algorithme de Dijkstra est toujours plus rapide que l'algorithme de Bellman-Ford.

III.IMPLEMENTATION

1. Langage

Nous avons eu besoin d'un langage polyvalent, performant, libre et suffisamment de haut niveau pour que l'on puisse faire toutes nos tâches librement, nous avons aussi utilisé des morceaux de codes provenant d'autres auteurs. Nous nous sommes donc tournés vers le C++, qui regroupe tous les avantages cités.

Pour l'algorithme de Djikstra, nous avons choisi la matrice d'adjacence comme structure de données car elle nous permet plus facilement de récupérer les points (s_i,s_j) de chaque arête. La matrice d'adjacence consiste un tableau à deux dimensions dont les éléments ne sont autres que les coefficients de la matrice d'adjacence du graphe considéré. Ensuite nous utilisons deux autres tableaux : le premier est initialisé avec des valeurs correspondant à une estimation de la distance minimale du nœud source à chacun des autres nœuds dans lequel nous calculons la distance minimale de chaque nœud depuis la source ; le deuxième nous permet de récupérer tous les prédécesseurs d'un nœud dont la distance minimale a été trouvée à partir du nœud source.

Par contre, pour l'implémentation de l'algorithme de Bellman Ford, nous avons choisi la liste d'adjacence pour simplifier notre code. Une liste d'adjacence est un tableau de taille |S| dont chaque élément T[si] est une liste (chaînée) de tous les sommets s_j tels qu'il existe une arête $\{s_i, s_j\} \in A$.[1]

3. Analyse des résultats

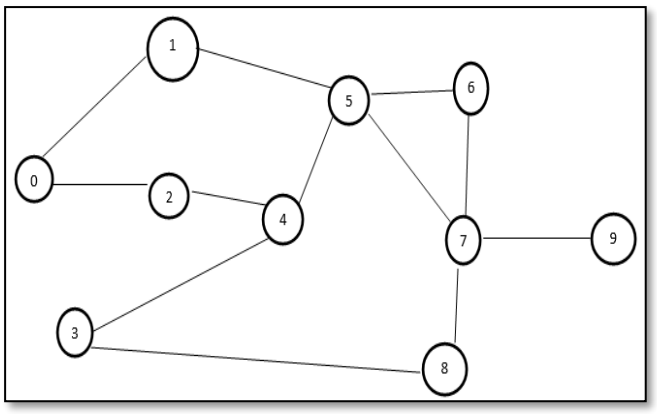


Figure II.1. Graphe de 10 nœuds

Dans un premier temps, pour les phases de test de l'implémentation des algorithmes, nous avons construit un graphe non orienté de 10 (**Figure 1**). Le programme prend alors en entrée ce graphe, un nœud source et affiche le plus court chemin et sa longueur de la source vers les autres sommets.

```
*****Resultats******
Les distances les plus courtes par rapport à la source sont les suivantes
Chemin          Distance
0 0                             Distance minimale:0
0 1                             Distance minimale:1
0 2                             Distance minimale:1
0 2 4 3                                 Distance minimale:3
0 2 4                           Distance minimale:2
0 1 5                           Distance minimale:2
0 1 5 6                                 Distance minimale:3
0 1 5 7                                 Distance minimale:3
0 2 4 3 8                               Distance minimale:4
0 1 5 7 9                               Distance minimale:4
```

Figure II.2. Résultats des tests avec l'algorithme de Dijkstra

```
Entrer la source :
0
Les distances les plus courtes par rapport à la source sont les suivantes 0
******Resultats******
Sommet          Distance minimale               Plus court chemin
0                0                              0
1               1                               0 1
2               1                               0 2
3               3                               0 2 4 3
4               2                               0 2 4
5               2                               0 1 5
6               3                               0 1 5 6
7               3                               0 1 5 7
8               4                               0 1 5 7 8
9               4                               0 1 5 7 9
```

Figure II.3. Résultats des tests avec l'algorithme de Bellman-Ford

Après une étude comparative des résultats d'exécution obtenue des deux algorithmes, nous observons que les deux algorithmes donnent les mêmes valeurs des longueurs des plus courts chemins à partir de la source 0.

Cependant, en regardant de plus près les résultats, nous voyons que le plus court chemin de la source 0 vers le sommet 8 n'est pas le même au niveau des deux algorithmes, ce qui relève une différence entre les deux algorithmes.

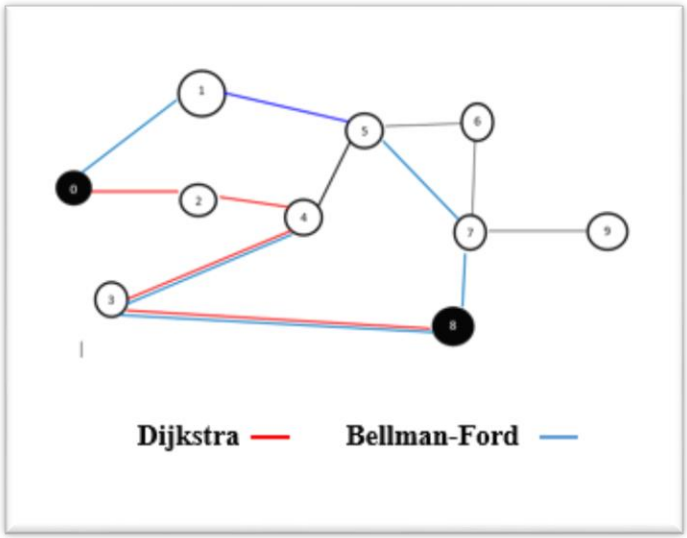


Figure II.4. Plus court chemin de 0 à 8 pour chaque algorithme

Nous avons jusqu'ici considéré les algorithmes de Dijkstra et de Bellman-Ford comme équivalents en termes de résultats. Nous allons maintenant comparer les performances de ces algorithmes en termes de temps d'exécution.

Pour cela, nous étudierons le temps d'exécution de chacun de ces deux algorithmes pour des tailles de graphes différents. Nous quantifierons cette taille par le nombre de nœuds qu'ils contiennent.

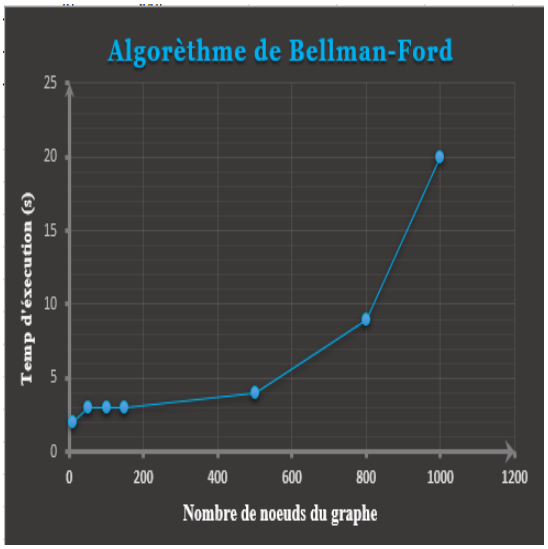


Figure.II.5. Temps d'exécution de l'algorithme de Dijkstra en fonction du nombre de nœuds des graphes testés

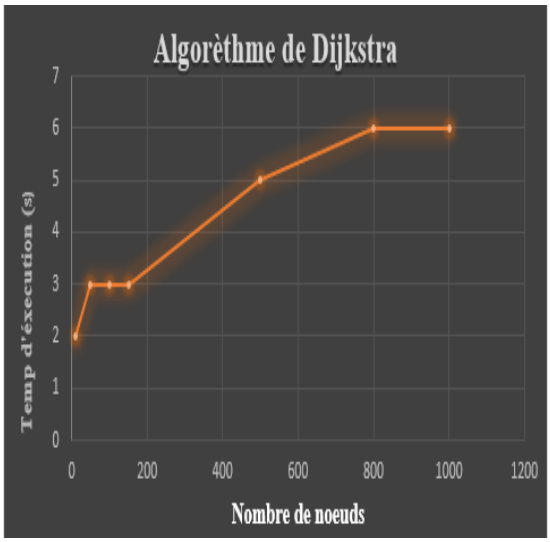


Figure.II.6. Temps d'exécution de l'algorithme de Bellman-Ford en fonction du nombre de nœuds des graphes testés

Dans le courbe temps d'exécution de l'algorithme de Bellman-Ford (FigureII.3), montre que le temps d'exécution augmente en fonction du nombre de nœuds. La théorie montre que l'algorithme de Bellman-Ford est en O(nm), or la courbe obtenue n'est pas linéaire.

La théorie montre que l'algorithme de Dijkstra est en O(n^2) Or, dans le **Figure II.4,** il apparait clairement que le temps d'exécution de l'algorithme dépend linéairement du nombre de nœuds traités.

- **Comparaison du temps d'exécution des deux algorithm**es

	Algorithme de Dijkstra		Algorithme de Bellman-Ford	
	Théorie	Application	Théorie	Application
La Complexité	O(n^2)	O(n)	O(nm)	O(n^2)

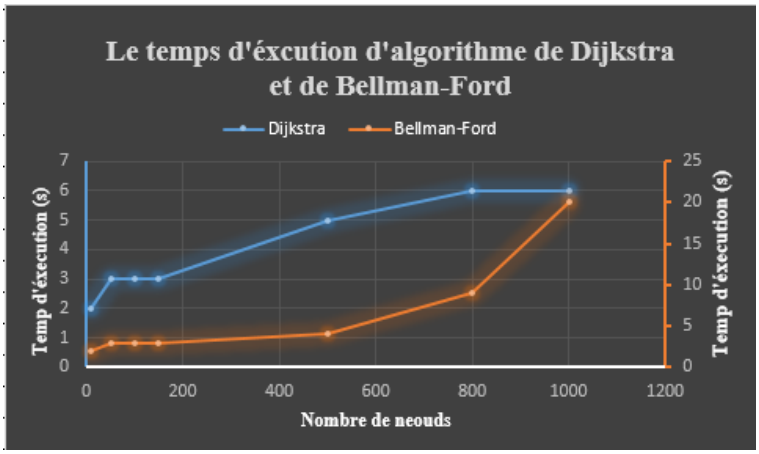


Figure.II.7. Temps d'exécution de l'algorithme de Bellman-Ford et Dijkstra en fonction du nombre de nœuds des graphes testés

L'avantage ici est à l'algorithme de Dijkstra, nettement plus rapide que celui de Bellman-Ford.

IV.CONCLUSION

La démarche de notre travail s'est portée sur trois grands points :

- Présentation du problème à résoudre.
- Implémentation du problème.
- Analyse des résultats d'implémentation

L'avancement de notre travail nous a mené à découvrir de plus en plus de problématiques (notamment pratiques) telle que la gestion de la mémoire due à la taille des matrices d'adjacences.

Finalement, notre étude nous a permis de conclure que les résultats théoriques ne sont pas toujours vrais dans la pratique.

V.BIBLIOGRAPHIQUE

https://www.normalesup.org/~dconduche/informatique/PT/Cours/Dijkstra.pdf

https://perso.crans.org/brameret/TER2009/GM06_HauteResolution.pdf

www.numdam.org/article/RO_1996__30_4_333_0.pdf

http://webdevtipstricksandinfo.blogspot.com/2013/06/simple-bellman-ford-implementation-in-c.html

http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/

https://perso.liris.cnrs.fr/samba-ndojh.ndiaye/fichiers/App_Graphes.pdf