école dE TECHNOLOGIE SUPÉRIEURE

UNIVERSITÉ DU QUÉBEC

Rapport de laboratoire 3

PRÉSENTÉ À

SAbeur Lafi

Dans le cadre du cours :

ALGORITHMES

ELE440-01

par

Julien lemay (LEMJ16059303)

Alexandre Lessard (LESA30099400)

montréal, le 20 novembre 2015

cid:image002.jpg@01CCE021.010CA410Julien Lemay et Alexandre Lessard, 2015

Sommaire

[1 Introduction 3](#_Toc435773714)

[2 L’algorithme de recherche du plus court chemin 4](#_Toc435773715)

[2.1 Profondeur 4](#_Toc435773716)

[2.2 Largeur 5](#_Toc435773717)

[2.3 Glouton 5](#_Toc435773718)

[2.4 Dijkstra 6](#_Toc435773719)

[2.5 Floyd-Warshall 6](#_Toc435773720)

[3 L’analyse Théorique 7](#_Toc435773721)

[3.1 Profondeur 7](#_Toc435773722)

[3.2 Largeur 7](#_Toc435773723)

[3.3 Glouton 7](#_Toc435773724)

[3.4 Dijkstra 7](#_Toc435773725)

[3.5 Floyd-Warshall 7](#_Toc435773726)

[4 Conclusion 7](#_Toc435773727)

# Introduction

Ce laboratoire porte sur l’exploration des graphes. Le but de celui-ci est de se familiariser avec les graphes pondérés, non-pondérés ainsi que les graphes orientés et non-orientés et d’implémenter des algorithmes du plus cours chemin.

Un algorithme dynamique et un glouton auront à être programmés ainsi que un algorithme de parcours d’arbre en profondeur et un en largeur. Ce laboratoire testera aussi l’algorithme de Dijkstra et Floyd-Warshall. Le tout sera utilisé pour trouver le plus court chemin entre deux points d’un graphe. Finalement, les statistiques des tests permettront de qualifier les algorithmes en trois catégories : optimal, sous-optimal et non-optimal.

# L’algorithme de recherche du plus court chemin

Dans cette section, nous décrirons les différents algorithmes utilisés soit la recherche en profondeur, en largeur, en profondeur de façon glouton, Dijkstra et Floyd-Warshall. Les pseudocode suivant sont ou se sont inspirés de ceux fourni dans l’énoncé du laboratoire.

## Profondeur

Parcours\_Profondeur(A[0..N-1],Source,Destination,Chemin[0..N-1],p):Booléen

Si A[Source].visité

Retourner Faux

A[Source].visité = Vrai

Trouvé = Faux

k = 0

Tant que (k ≤ A[Source].NVoisins) et (Pas Trouvé)

v = A[Source].lien[k]

Si (A[v].nom == Destination)

Trouvé == Vrai

Chemin[p+1] = A[v].nom

Sinon

Trouvé = Parcours\_Profondeur(A, v, Destination, Chemin, p+1)

Si Pas Trouvé

k = k + 1

Si Trouvé

Chemin[p] =A[Source].nom

A[Source].visité = Trouvé

Cet algorithme parcours l’arbre de manière a visiter tous les nœuds à gauche de l’arbre en premier toujours allant de plus en plus bas. Lorsqu’il rencontre le bas de l’arbre, il remonte jusqu’à ce qu’il trouve un branche non exploré, puis l’explore. Ce procédé se répète jusqu’à ce qu’il trouve la destination.

## Largeur

ExplorerLargeur(G,départ) :

Initialiser Prédécesseur à NUL; Visité à FAUX et Distance à ∞;

Distance[départ] = 0; Prédécesseur[départ] = NUL;

ÀExplorer = nouvelle file vide

Visité[départ] = VRAI

Enfiler(ÀExplorer, départ)

tant que File-Vide(ÀExplorer) == FAUX

u = Défiler(ÀExplorer)

pour chaque v voisin de u dans G

si Visité[v] == FAUX

Visité[v] = VRAI

Distance[v] = Distance[u] + 1

Prédécesseur[v] = u

Enfiler(ÀExplorer,v)

L’algorithme utilisé n’a pas été codé à l’aide de ce pseudo-code. On s’est plutôt inspiré de celui de la recherche en profondeur qui avait été donné dans l’énoncé de laboratoire.

L’algorithme de parcours en largeur permet de visiter tous les nœuds de la même profondeur à chaque itération jusqu’à ce qu’il trouve la destination puis reviens sur ses pas en marquant le chemin à emprunter.

## Glouton

Parcours\_Profondeur\_Glouton(A[0..N-1],Source,Destination,  
Chemin[0..N-1],p):Booléen

Si A[Source].visité

Retourner Faux

A[Source].visité = Vrai

Trouvé = Faux

k = 0

Trié en ordre de poids les voisins de A[source]

Tant que (k ≤ A[Source].NVoisins) et (Pas Trouvé)

v = A[Source].lien[k]

Si (A[v].nom == Destination)

Trouvé == Vrai

Chemin[p+1] = A[v].nom

Sinon

Trouvé = Parcours\_Profondeur(A, v, Destination, Chemin, p+1)

Si Pas Trouvé

k = k + 1

Si Trouvé

Chemin[p] =A[Source].nom

A[Source].visité = Trouvé

Cet algorithme est en fait exactement le même que l’algorithme en profondeur normal sauf que l’on trie par la valeur des poids les voisins de chaque nœuds.

## Dijkstra

Dijkstra(L[0..N-1,0..N-1],Source,Destination):Chemin[0..N-1],D[0..N-1],

Visité[0..N-1], Chemin[0..N-1]

Pour n = 0 à N-1

D[n] = L[Source,n]

Chemin[n] = Source

Visité[n] = FAUX

Visité[Source] = VRAI

nombreVisité = 1

Tant que nombreVisité < N

plusProche = sommet non visité avec le plus petit D

Visité[plusProche] = VRAI

nombreVisité = nombreVisité + 1

Pour tous les prochainSommet dans la liste des voisins de plusProche

Si D[prochainSommet]>D[plusProche]+L[plusProche,prochainSommet]

D[prochainSommet]=D[plusProche]+L[plusProche,prochainSommet]

Chemin[prochainSommet] = plusProche

L’algorithme de Dijkstra permet de trouver le chemin le plus court de tous les nœuds d’un graphe à partir d’un point du tableau. Le fonctionnement de cet algorithme consiste à constamment se déplacer vers la branche qui coute la moins cher jusqu’à ce que la destination ai été atteinte.

## Floyd-Warshall

Floyd-Warshall(L[0..N-1,0..N-1],Source,Destination):Chemin[0..N-1,0..N-1] D[0..N-1,0..N-1], Chemin[0..N-1,0..N-1]

Pour i = 0 à N-1

Pour j = 0 à N-1

D[i,j] = L[i,j]

Chemin[i,j] = -1

Pour k = 0 à N-1

Pour i = 0 à N-1

Pour j = 0 à N-1

Si (D[i,j] > D[i,k]+D[k,j])

D[i,j] = D[i,k]+D[k,j]

Chemin[i,j] = k

Cet algorithme permet de calculer le poid minimal entre chaque pair de sommets puis lorsque le la destination est atteinte, Il est facile d’évaluer le plus court chemin pour y arriver.

# L’analyse Théorique

## Profondeur

L’algorithme de parcours en profondeur peut ressembler à la recherche séquentielle c’est-à-dire, passer à travers l’entité de l’arbre jusqu’à ce qu’il trouve la destination. Aucune autre logique pour permettre de gagner du temps ou de l’efficacité est utilisé.

## Largeur

L’algorithme de parcours en largeur peut être rapide si nous savons que la source et la destination sont relativement proches puisque l’algorithme n’aura pas à traverser tout l’arbre avant de la trouver.

## Glouton

Comme dans la majorité des cas, l’algorithme glouton est un des moins performants mais un des plus rapide.

## Dijkstra

L’algorithme de chemin le plus court de Dijkstra est un des plus efficace pour les graphes pondérés. L’algorithme aussi étant du temps d’exécution de O(V lg V + E) V = nœud et E = lien

## Floyd-Warshall

L’algorithme de plus court chemin de Floyd-Warshall est très performante mais plus lente que les autres algorithmes puisque, pour chaque pair de nœuds, il calcule la meilleure solution or, même les nœuds qui sont pas nécessaire pour atteindre la destination.

# Conclusion

Ce laboratoire à traité d’algorithmes de plus court chemin en utilisant des graphes pondérés, non-pondérés autant qu’orienté ou non-orienté. Cinq algorithmes ont été implémentés, testés et analyser : Parcours en profondeur, Parcours en largeur, Algorithme glouton, Dijkstra ainsi que Floyd-Warshall.