école dE TECHNOLOGIE SUPÉRIEURE

UNIVERSITÉ DU QUÉBEC

Rapport de laboratoire 2

PRÉSENTÉ À

SAbeur Lafi

Dans le cadre du cours :

ALGORITHMES

ELE440-01

par

Julien lemay (LEMJ16059303)

Alexandre Lessard (LESA30099400)

montréal, le 2 octobre 2015

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

# Introduction

Cette première section contient une description des buts du laboratoire et des objectifs visés

ainsi que sert à introduire les sections qui suivent.

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. Un algorithme dynamique et un glouton auront à être programmé 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

(environ 1 à 2 pages/algorithme)

La deuxième section explique le principe de fonctionnement de chaque algorithme et donne

son pseudocode tel qu'il a été implémenté, sans oublier de mentionner sa provenance et

en mettant l'accent sur les ajustements qui ont dû être apportés à la source originale pour

les besoins particuliers de l'équipe. Elle donne aussi un court rapport sur les difficultés et

autres curiosités rencontrées lors de l’implémentation.

## 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 plustôt 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

PSEUDO CODE ET EXPLICATION

## 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

Cette section fournit le détail de l’analyse théorique de chaque algorithme et justifie les

conclusions de l’analyse.

## 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 proche 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 performant

## Dijkstra

## Floyd-Warshall

# Conclusion

Cette section établit des conclusions quant à l’atteinte des objectifs de départ. Il faut se

focaliser particulièrement sur les avantages et inconvénients de chaque algorithme et leur

utilité dans différents cas de figure.

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.

# Références

Cette dernière section doit énumérer toutes les références de vos sources. Dans le corps du

rapport, vous devez également mettre un renvoi après chaque élément emprunté, ex. [1],

[2], etc.