École Polytechnique de Montréal
Département de Génie Informatique et Génie Logiciel

LOG2810

STRUCTURES DISCRETES

Hiver 2017

TP1 : Graphes

Remis par :

Matricule	Prénom & Nom
1827808	Audrey Labrie
1804702	Sébastien Chagnon (groupe 2)
1818055	Sida Eric Li

À :

David Johannès

Le 1er mars 2017

Introduction

L'objectif de ce travail pratique est d'appliquer nos connaissances de la théorie des graphes. Pokémon Go est un jeu dont le but est d'amasser le plus de points en attrapant des Pokémons et en affrontant des adversaires. Le but de ce TP est donc de faciliter la vie de l'utilisateur en lui proposant des parcours qui optimiserait son gain de points. Pour ce faire, nous avons implémenté deux méthodes, une qui propose à l'utilisateur le plus court chemin pour atteindre un gain minimal donnée et le plus grand gain que l'utilisateur peut obtenir en parcourant au maximum une distance donnée.

Pour ce faire, nous avons créé une interface graphique permettant de visualiser les chemins proposés à l'utilisateur.

Présentation de la solution

Le problème est divisé en 2 parties. Tout d'abord, il faut générer le Graph à partir des informations qui sont stockées dans le fichier texte. Par la suite, à partir de se Graph, il faut trouver le chemin le plus optimale pour faire le meilleur ratio de points par distance.

Afin de générer le Graph, les arcs lus dans le fichier sont d'abord triés en ordre de distance. On prend ensuite l'arc le plus court et on sait qu'il y a un arc entre ses deux extrémités (Nodes). Par la suite, on prend le second plus petit d'arc. On considère la première extrémité comme le point A et la seconde comme le point B. Le programme essaie de trouver, a partir des arcs déjà présents sur la graphique, un chemin du point A au point B avec comme distance même que l'arc présentement traité. Cette recherche de distance est faite méthodiquement entre tous les arcs afin d'être certain de ne pas en laisser de côté.
Cet algorithme a une complexité exponentielle mais on gère avec un petit nombre de Nodes donc l'algorithme est raisonnable pour nos besoins. Il s'agit d'un algorithme récursif.
Cet algorithme est répété pour chacun des arcs lus dans le fichier en ordre de distance.

Une fois le graph crée, il faut trouver le chemin optimal selon les requis de l'utilisateur. Que ce dernier veule un maximum de points pour une certaine distance ou la plus petite distance pour un certain nombre de points, l'algorithme est le même. Un chemin (Path) est créé à partir du départ. Le Path a comme attribut une distance totale parcourue et un nombre de point accumulé, ce qui permet de calculer le ratio distance/points. Le Path a aussi une Map des Nodes désactivés. Il a finalement une list des Nodes précédemment visités
A partir de ce Path, *n* nouveaux Path sont créés où *n* est le nombre de Nodes reliés au Node actuel du Path traité. Chacun de ses Path ont un ratio distance/points différent au premier. L'algorithme retire le Path traité, ajoute les nouveaux Path a sa liste de Paths potentiels, puis les trie en ordre de ratio.
Le Path avec le meilleur ratio est ensuite traité comme l'a été le chemin de l'itération précédente. Une fois qu'un Path atteint l'objectif désiré, il est mis en mémoire par l'algorithme et l'algorithme passe à la prochaine itération. Une fois que l'algorithme a trouvé 500 Path complets, il les trie en fonction de ce que l'utilisateur a demandé et renvoie le meilleur.
Le nombre de Path potentiels gérés par l'algorithme grandit très rapidement. Afin d'éviter les

erreurs OutOfMemory et accélérer le processus, seulement 500 Paths potentiels sont gardés. Cela n'a pas vraiment d'incidences sur la qualité du chemin trouvé car les Path qui sont continués sont ceux avec les meilleurs ratios et ceux qui sont enlevés sont ceux avec les pires.

Voir la dernière page pour le diagramme de classe

Difficultés et solutions

Nous avons rencontré trois difficultés lors de ce TP.

Premièrement, nous avons eu de la difficulté à modéliser le graphique. Il y avait trop d'arcs pour qu'il soit visuellement organisé. Nous avons donc décidé de seulement connecter les arcs des chemins les plus courts entre les différents sommets. Pour faire cela, nous avons classé les arcs en ordre croissant de leurs distances entres les sommets. Par après, nous itérons à travers cette liste d'arcs ordonnée, puis nous connectons les sommets seulement si la distance de l'arc est plus petit que la distance minimale entre les deux sommets déjà connectés par d'autres arcs passant par d'autres chemins. Par exemple, si sommet A et sommet B sont connectés par un arc de distance 10 et sommet B et sommet C sont connectés par un arc de distance 10 et qu'on rencontre un arc qui connecte sommet A et sommet C avec une distance de 25, on sait que le chemin le plus court va de sommet A à sommet B à sommet C qui est de 20, donc on n'ajoute pas un arc connectant sommet A à sommet C directement (distance 25 plus grand que distance 20).

Deuxièmement, il était difficile de trouver une librairie qui pouvais nous aider à représenter graphiquement notre solution. Donc, nous avons décidé de représenter le graphique en cercle à l'aide librairies Java. En fait, tous les sommets forment un cercle et au centre de celui-ci, on peut y observer des arcs qui sont des chemins entre les différents sommets. Le chemin choisit par la méthode plusCourtChemin() ou plusGrandChemin() devient coloré, ce qui permet à l'utilisateur de bien visualiser la solution.

Finalement, au tout début, nous passions des pairs et des tuples en paramètre de certaines méthodes. On a réalisé que c'était inutile de faire ceci, car on pouvait juste passer les paramètres simples.

Conclusion

Ce laboratoire a été utile car nous avons appris à adapter un algorithme de graphe à une application concrète. Nous avons étudié l'algorithme Dijkstra afin de créer le nôtre.

Pour le prochain laboratoire, nous aimerions devoir passer moins de temps sur l'interface graphique et plus d'accent sur les algorithmes.