LOG 2810

STRUCTURES DISCRETES

TP1 : GRAPHES

Remis le 29 octobre 2018

Par
William Valiquette 1847812
Nicole Joyal 1794431
Jérôme Désilets 1648912

Introduction

Ce travail a essentiellement pour but d'appliquer la théorie sur les graphes vue en cours en nous présentant un contexte de la vie réelle dans lequel un tel savoir serait indispensable. Dans notre cas, le contexte est l'optimisation du parcours de véhicules dans la ville de Montréal.

Dans ce contexte, nous venons en aide à un étudiant en génie logiciel faisant un stage pour une compagnie start-up qui fait des véhicules médicalisés autonomes se déplaçant d'un CLSC à un autre. Le but de la compagnie est de réduire le coût et augmenter l'efficacité des services médicales de Montréal en rendant les véhicules sans chauffeurs, donc moins coûteux, et plus intelligents pour que leurs trajets soient plus courts. En effet, connaissant la destination d'avance et étant intelligents, ils seraient en mesure de déterminer le meilleur chemin à emprunter. Également, tous les aspects légaux du projet ont été approuvés par les autorités de Montréal.

Passons maintenant au mandat exact de l'élève en question. Il est chargé de construire et d'implémenter l'algorithme dont se servira les véhicules intelligents pour déterminer le chemin le plus efficace à emprunter. Malheureusement, l'étudiant en question ne maîtrise pas bien la matière du cours de structure discrète, malgré le fait qu'il l'a déjà passé. Cependant, étudiants modèles que nous sommes, les graphes n'ont pas de secrets pour nous. Dans le but d'aider une âme en détresse et de propager la connaissance incroyable que nous possédons, nous allons venir en aide à cet étudiant.

L'étudiant a déjà établit quelques points de départs pour la solution à ce problème que nous utiliserons comme base pour continuer à développer l'algorithme. Tout d'abord, il a fait correspondre à chaque chemin possible entre chaque CLSC un coût en temps, proportionnel à la distance. L'étudiant n'a pas lésiner sur le travail et s'est renseigné pour bien prendre que les chemins pouvant être empruntés par des véhicules médicalisés autonome. L'étudiant désire aussi que l'algorithme soit capable de tenir en compte du coût d'exploitation des trajets. Il a donc décidé d'utiliser un graphe des distances pour représenter les chemins entre les CLSC.

Présentation de nos travaux

C1 - Fonction creerGraphes

Pour cette fonction, nous avons créé 3 classes qui nous permettent de constituer un graphe.

La première classe, Sommets, contient l'identifiant du sommet ainsi qu'une variable borne qui indique s'il y a une borne ou non. La deuxième classe, Arete, contient la longueur de l'arête. La troisième classe, Graphe, est composée d'une liste de sommets et d'arêtes.

Pour créer le graphe, nous itérons au travers du fichier texte fourni. Jusqu'à ce qu'il y ait une ligne constitué uniquement d'une fin de ligne, nous créons un sommet constitué des caractères avant la virgule pour l'identifiant et du 0 ou du 1 suivant la virgule pour la signaler la présence de borne ou non. Chaque sommet est ajouté à la liste listeSommets pour un accès consécutif et facile.

Quand la ligne constituée uniquement d'un caractère de fin de ligne est atteinte, on commence à remplir la liste de listes matriceArete. Pour chaque ligne, on sépare celle-ci en 3 variables. La première et deuxième variable indiquent à quel endroit placer le arête dans la matrice et la 3e indique la valeur de la longueur avec laquelle l'objet arête à insérer doit être initialisé.

Lorsque listeSommets et matriceArete sont remplis, la fonction creerGraphe est complète.

C2 - Fonction lireGraphe

Pour cette fonction, il ne suffit que d'itérer au travers de la variable listeSommets et pour chaque itération, on imprime ledit sommet, puis chaque sommet qui y est relié. Ceci est fait en itérant dans la matriceArete à la ligne associée au sommet (la ligne correspond à l'identifiant du sommet - 1), et pour chaque fois où la case n'est pas vide, on imprime l'identifiant du sommet (qui correspond à la position de l'itération) et la longueur de l'arête (qui correspond à l'objet à cette position).
Quand il y a eu une itération complète au travers de listeSommets, l'exécution de lireGraphe est terminée.

C3 - Fonction trouverPlusCourtChemin

Afin de résoudre le problème, il a d'abord fallu implémenter Dijkstra en fonction pour pouvoir obtenir le plus court chemin entre deux points. Ensuite, classe véhicule a été créée avec les attributs pertinents pour déterminer, en exécutant parcourirChemin, si le chemin était suffisamment court pour que le type de voiture, avec le type de patient, puisse exécuter le trajet, et dans quel temps.

Ensuite, il a fallu inclure la possibilité que le chemin direct ne soit pas suffisant pour atteindre la destination, et il a donc fallu implémenter la fonction obtenirCheminsComplets, qui entrepose, dans la variable etiquettesSommetsPossibles, la liste de tous les chemins possibles. Cette liste a été assemblée en réunissant tous les parcours (les plus courts grâce à la fonction Dijkstra) entre chaque bornes et jusqu'à la destination.

Il a donc fallu fusionner la liste du chemin vers les bornes au prochain chemin qui mènera à une nouvelle borne ou bien à la destination. En créant les nouveaux chemins, si ceux-ci contenaient des bornes déjà visitées, ils ont simplement été abandonnés dans la récursion puisqu'ils sont automatiquement plus long que le plus court trajet possible et n'aident en rien à trouver le chemin le plus court.

Quand la liste etiquettesSommetsPossibles est complétée par la fonction obtenirCheminsComplets, on parcourt tous les chemins possibles et on entrepose le plus court pour le réexécuter et afficher ses informations à la fin. Si aucun chemin est trouvé avec la première itération, on essaie dans une seconde itération et tout est recommencé avec un véhicule LI-ion. Si aucun chemin n'est enfin trouvé, on affiche un message d'erreur.

Difficultés rencontrées

La première difficulté que nous avons rencontrée était l'apprentissage de Python. Ce n'était pas très demandant mais ça nous a pris un moment pour s'habituer et être fluide dans le langage. De plus, il a fallu installer un nouvel environnement de développement et apprendre son usage pour mieux le naviguer et l'utiliser à bon escient. Par la suite, nous avons eu la lourde tâche d'implémenter en code l'algorithme de Dijkstra, qui est

un concept complexe. Ce que nous avons réussi à faire en suivant chaque instruction donné dans l'énoncé et en réfléchissant à sa traduction dans un langage de programmation.

Le principal problème que nous avons eu a été la trop grande complexité de nos algorithmes de code. En effet, dans les cas extrêmes d'extraireSousGraphe, le code prend plusieurs dizaines de minutes à exécuter, car le temps d'exécution devient immense quand la consommation d'électricité de la voiture diminue, puisque. Nous avons réussi à améliorer quelques points mais la lenteur d'exécution demeure toujours dans ce cas.

Le temps d'exécution trop grand dans trouverPlusCourtChemin a également été problématique, mais celui-ci a été réduit en ajoutant dans la fonction une liste des bornes visitées et en n'explorant pas les chemins contenant ces bornes déjà visitées, ce qui était un cas extrêmement récurrent et a donc pu diminuer l'exécution, qui pouvait prendre des minutes à la fois, à moins d'une seconde.

Conclusion

Maintenant est venu le temps de conclure sur ce beau projet. Quelle sacrée aventure que ce fût !

Par le biais de nos travaux, nous sommes parvenus à aider un étudiant en peine en implémentant une façon de créer un graphe à partir d'un fichier texte, d'afficher ce graphe, de trouver le plus court chemin entre deux points dans ce graphe, ainsi que trouver le plus long chemin possible à partir d'un point prédéterminé. Le tout peut également être utilisé facilement par l'entremise d'une interface que nous avons créée.

En cours de route, nous avons appris plusieurs principes sur les graphes et gagné beaucoup d'expérience concrète dans ce même sujet. En effet, nous avons appris à manipuler et implémenter l'algorithme de Dijkstra dans un contexte concret. Également, nous avons appris comment implémenter un graphe et le représenter dans un contexte de programmation et d'utiliser la récursion afin de mieux parcourir ses diverses arêtes.