LOG2810

STRUCTURES DISCRÈTES

TP1 : Graphes

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

JEAN-PAUL KHOUEIRY - 2011397

ROY BOUTROS - 2012196

REMIS LE MARDI 3 NOVEMBRE 2020

1. INTRODUCTION

Avec la popularisation des voitures autonomes et électriques, les problématiques concernant les trajets à longue distance se sont multipliées. Ces véhicules nécessitent une nouvelle application qui pourrait mieux planifier et proposer leurs trajets. Ces véhicules ont des limites d'autonomie, ce qui rend l'optimisation du trajet très important. Nous avons une carte, qu'on peut voir comme un graphe orienté et pondéré où les arcs sont des chemins possibles à emprunter par un véhicule. Notre programme permettra de créer un graphe de la sorte, de le lire, d'en faire une extraction et de proposer le plus court chemin possible à emprunter pour un utilisateur. Le programme aura aussi un menu qui va être affiché à l'utilisateur pour qu'il puissent entrer les paramètres de son véhicule et faire des choix en lien avec le trajet à emprunter.

2. PRÉSENTATION DU PROJET

Nous avons développé le projet en orienté objet, ce qui nous a permis de diviser le code en quatres classes principales:

- Vehicule:

  Cette classe sert à représenter le véhicule qui sera utilisé lors du trajet. Il y a un constructeur de véhicules, ainsi que tous les attributs nécessaires pour le construire et déterminer son autonomie. Cette classe contient surtout des méthodes d'accès. Il existe aussi une méthode pour trouver l'autonomie restante en pourcentage.

- Sommet:

  Cette classe représente les sommets du graphe proposé. Autre que les constructeurs, les méthodes d'accès et les modificateurs. La méthode *getArcDeDeuxSommets()* permet d'identifier l'arc qui rejoint deux sommets et la méthode *getSommetsVoisinsSortant()* permet de trouver les sommets des voisins sortants.

- Arc:

  Cette classe contient évidemment les constructeurs, les méthodes de base et tous les attributs nécessaires. La méthode *getSommets()*, elle, sert à donner les deux sommets à chaque bord de l'arc.

- Graphe:

  Cette classe représente la plus grande partie du travail. Elle contrôle la majorité des opérations sur les arcs, les sommets et le véhicule. Elle contient les fonctions

principales, soit *creerGraphe()*, *lireGraphe()*, *extractionGraphe()* et *plusCourtChemin()*:

i. creerGraphe( ):

Cette fonction lit un fichier texte du même format que celui donné, et sauvegarde l'information lue concernant les sommets et les arcs. Elle retient ces informations dans un vecteur pour les sommets et un autre pour les arcs. Elle trie aussi les sommets voisins par sommets entrants et sortants.

ii. lireGraphe( ):

Cette fonction interprète les données qui sont contenues dans les vecteurs de sommets et d'arc, comme le type de sommets et la lettre qui représente chaque sommet, pour les représenter comme couple à l'utilisateur.

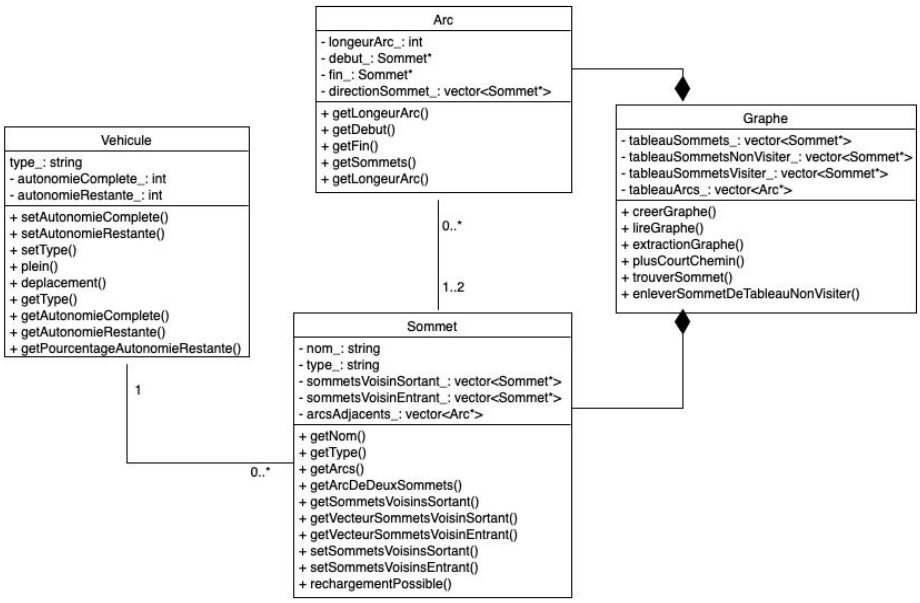
iii. extractionGraphe( )

Cette fonction retourne plutôt le plus long chemin qu'un véhicule pourrait parcourir à partir d'un sommet donné. Elle utilise un algorithme similaire à celui de Dijkstra, mais en mettant la distance des arcs en négatif. Ensuite, on trouve le plus petit chemin avec ces valeurs puis quand on remet ces valeurs positives, on obtient le plus long chemin.

iv. plusCourtChemin( )

Cette fonction est la plus importante du programme. Elle utilise l'algorithme de Dijkstra afin de trouver le chemin le plus court d'un sommet de départ quelconque à un sommet final. L'information nécessaire pour exécuter l'algorithme est contenue dans une unordered_map formé du sommet puis d'une paire de la longueur de l'arc du sommet débutant et la racine de cet arc (sommet précédent).

Cette figure représente le diagramme de classes du programme et des relations entre les solutions:



- Main:

  La fonction *main()* est le menu interactif qui se présente à l'usager. Cette fonction s'assure que les options choisies par l'usager sont valides à travers tout le programme et s'assure de redemander les questions auxquels l'usager aurait pu insérer une donnée non-valide (mettre une série de lettres au lieu d'un numéro lorsqu'on demande l'autonomie de son véhicule, par exemple.).

3. DIFFICULTÉES RENCONTRÉE ET LEURS SOLUTION

Il y a eu divers bogues en général dont on a dû passer beaucoup de temps à résoudre. Les fonctions pour trouver le plus long chemin et le plus court chemin, entre-autres, ont été plutôt difficiles à implémenter. Même si la théorie n'était pas mal comprise, c'était quand même un challenge de les mettre en code.

4. CONCLUSION

Ce labo nous a permis de renforcer nos connaissances sur les graphes et l'optimisation en général. À l'aide de l'algorithme de Dijkstra, nous avons pu proposer le chemin le plus court qu'un véhicule pourrait emprunter en s'assurant que l'autonomie de son véhicule soit respectée.

En général, la programmation du TP n'a pas été la plus simple et nous a quand même poussé à passer beaucoup de temps dessus. Il serait bien d'avoir une tâche un peu plus courte la prochaine fois, tout en maintenant l'apprentissage des notions.