Travail Pratique #1 : Graphes
LOG2810

École Polytechnique de Montréal

Trimestre : Automne 2020

Maximiliano Falicoff 2013658
Corentin Glaus 2021624
Gabriel Rouleau 2014536

Présenté à : Aurel Josias Randolph

Remis le vendredi 30 octobre

Introduction :

Ce laboratoire a pour objectif d'appliquer les notions théoriques vues en classe des graphes sur des cas concrets. Effectivement, c'est avec l'arrivée des voitures autonomes que l'intérêt de l'optimisation des déplacements vient prendre plus d'importance puisque l'on doit maintenant prendre en compte l'autonomie des voitures afin d'en prévoir les ravitaillements en énergie. L'objectif de créer une application qui calcule le meilleur chemin à emprunter pour un certain type de voiture. L'application devra considérer les différents types de véhicule possible soit hybride, électrique et essence. Elle devra aussi prendre en compte l'autonomie des différentes voitures et prévoir des arrêts afin de faire le ravitaillement d'énergie afin d'arriver à destination puisqu'il faut bien noter que ce ne sont pas toutes les villes qui ont des rechargements pour x voiture. Le système utilisera une carte de la région sous forme de graphe afin de faire les calculs. Les sommets seront les villes et les arcs les routes.

Présentation de la solution :

Notre solution est composée de 5 classes. Il s'agit des classes Interface, Graphe, Véhicule, Sommet ainsi que Arc.

La classe Interface est responsable de l'affichage au terminal des différentes options qui s'offrent à l'utilisateur. L'interface est donc en charge des interactions avec l'utilisateur, il lui faut donc demander les caractéristiques du véhicule. L'utilisateur pourra sélectionner une option dans le menu afin que l'interface ordonne certaines actions. L'interface vient donc agir comme contrôleur en ordonnant certaines actions comme la mise à jour de la carte de la région, l'extraction d'un sous-graphe, le calcul du chemin le plus court et la fermeture de celle-ci. L'interface est en relation de composition avec la classe Véhicule ainsi qu'avec la classe Graphe.

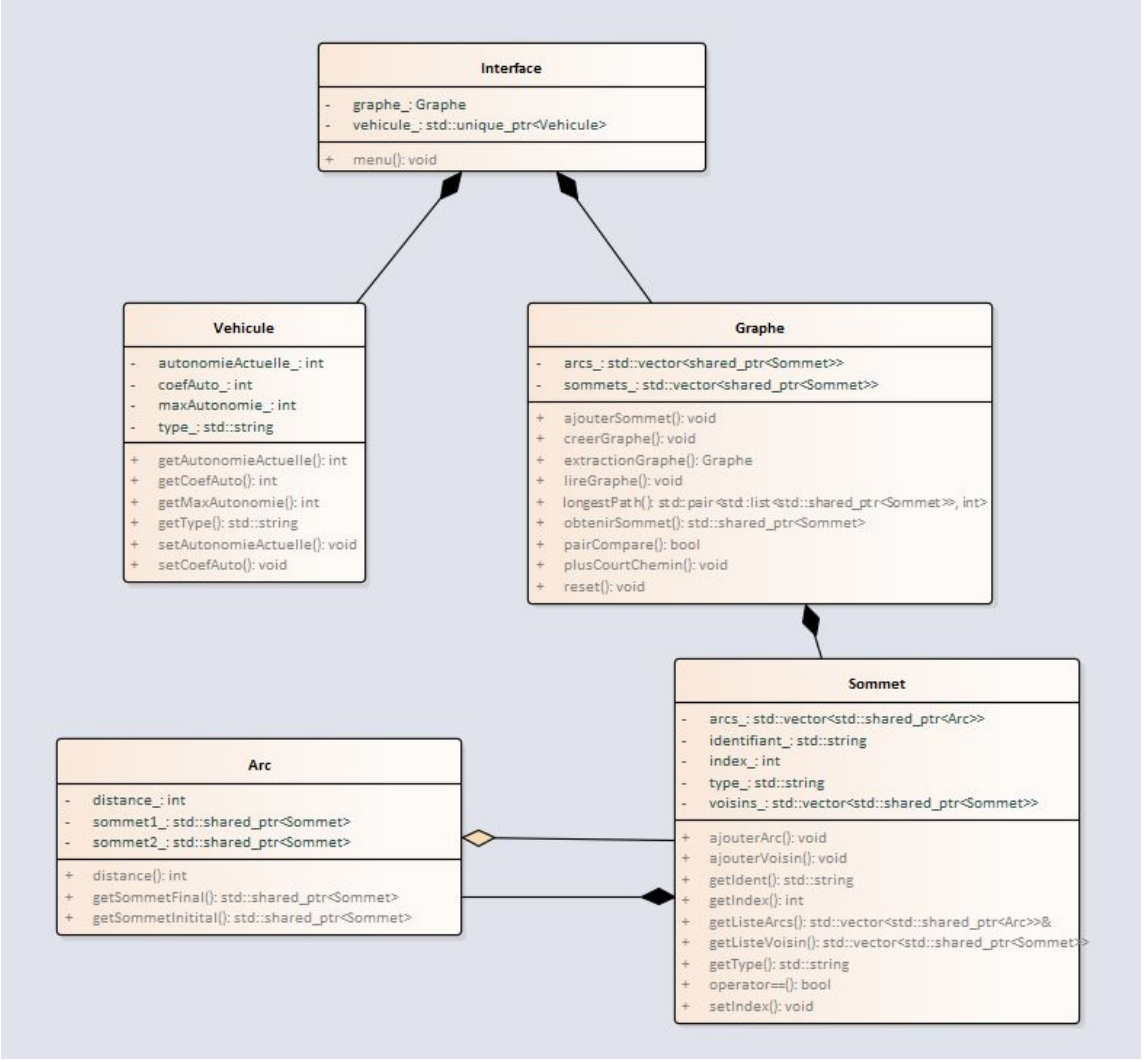
La classe véhicule est en charge de contenir les informations propre au véhicule de l'individu comme l'autonomie ou bien le type de la voiture utilisé.

La classe Graphe représente directement la carte de la région. Les distances optimales seront calculés directement à partir de cette classe. C'est aussi cette classe qui sera responsable d'afficher toutes les informations attachés au graphe. La classe Graphe est en relation de composition avec la classe Sommet.

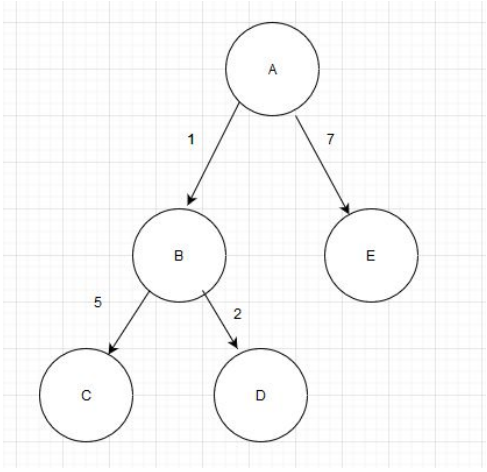
La classe sommet est en charge de contenir les informations sur la ville à laquelle il correspond incluant le nom, les villes voisines, le type de borne de rechargement qu'elle contient ainsi que les routes (arcs) allant et partant de la ville. La classe Sommet est donc en relation d'agrégation avec la classe Arc puisque les sommets (villes) peuvent continuer à être là sans la présence des arcs (routes).

La classe Arc est elle-même composé de la classe Sommet puisque avoir aucune destination défit l'objectif même de celle-ci. La classe Arc contient les informations sur la route comme la longueur ainsi que la ville de départ et d'arrivée.

Diagramme de classe :



Au niveau de la méthode extraction Graphe qui s'en charge de trouver la plus longues distance dont le véhicule peut traverser à partir d'un certain sommet partant.



On peut alors modéliser notre graphe en arbre avec notre racine étant le sommet Initial. On vient de transformer notre problème a un problème de traversement d'arbre. Dans notre cas on a choisi d'utiliser un depth first search pour notre parcours. On peut prendre un exemple simple.

Ici notre root est A et on cherche le chemin le plus long. On peut alors parcourir tous les voisins de notre root et puis appeler récursivement longest Path en passant une copie de la distance actuelle et du path. Cela va faire en sorte de parcourir A->C->C en premier puis vu que c'est le premier chemin, il sera évidemment le plus long, ensuite on revient à un Sommet dont il a encore des voisins à parcourir, donc B pour avoir le deuxième chemin A->B->D mais la distance totale est plus petite, il sera donc ignoré. On doit toujours tenir en compte les sommets visite pour ne pas revenir sur nos pas dans un cas ou cela peut arriver. Quand on a fini notre parcours et que l'on possède le plus long chemin, il ne reste plus qu'à construire le graphe que retourne la fonction, on fait alors un deep copy puis on reconstruit le graphe.

Au niveau de la méthode la plus court chemin, il s'agit d'une implémentation de l'algorithme de Dijkstra. L'algorithme de Dijkstra est utilisé pour trouver le chemin le plus court entre deux sommets du même graphe. Dans cette situation, il devait aussi prendre en compte la distance entre deux villes voisines ainsi que les points de rechargement pour les différents types de voitures. Ainsi, une voiture n'ayant pas assez d'autonomie se verrait introduite un détour afin de faire le plein d'énergie. L'algorithme de Dijkstra débute en marquant tous les noeuds comme étant non visité. Il assigne ensuite la valeur infini à tous les noeuds sauf de celui de départ, qui sera assigné la valeur 0. Il compare ensuite tous les coûts des différents noeuds voisins non visités et se dirige vers celui ayant le plus petit et le marque comme étant visité. Le noeud ayant le plus petit coût devient ensuite le noeud courant et on reprend jusqu'à ce qu'on ai trouvé le noeud recherché. Dès que le noeud est trouvé,

on peut arrêter de chercher puisqu'on sait que tous les chemins qui suivront seront plus coûteux.

Difficultés rencontrés :

Une des complexités survenues lors du développement de notre solution est le debuggage de notre méthode Extraction Graphe simplement dû à la nature récursive de l'algorithme. Un des moyens utilisés pour éviter de faire des erreurs au niveau de son implémentation est de bien modéliser le problème en faisant plusieurs example a la main en dessinant l'arbre et ensuite de faire l'algorithme a la main que l'on puisse assez facilement l'implémenter en terme de code. Mais même avec cette préparation, l'utilisation d'un debuggers et de regarder les calls stacs fut nécessaire à certains points pour bien voir l'état actuel de notre chemin.

Il fut aussi notablement difficile de déterminer par où commencer le projet. Nous étions incertain sur la façon de faire pour bien séparer le travail en classes distinctes ayant chacune des responsabilités dans le système. Nous avons donc procédé par modélisation UML en faisant ressortir les concepts et les objectifs du système. Ce n'est qu'après avoir mis de l'avant les requis que nous avons pu faire un diagramme de classe représentant la situation.

La problématique du changement des requis du laboratoire a aussi été un défi puisque le travail était pratiquement complété lorsqu'ils sont survenus. Nous avons donc dû redoubler d'effort afin de remettre le laboratoire à la date prévu.

Conclusion :

En conclusion, ce laboratoire a été utile puisqu'il nous a aidé à mettre au clair la matière vue en classe sur les graphes. Il a été aussi très utile selon nous, de mettre la théorie à la pratique dans un laboratoire comme celui-ci puisque les cours magistraux peuvent parfois être assez éloignés d'une expérience en entreprise. Il est donc certain qu'un travail comme celui-ci met de l'avant les requis du travail professionnel comme par exemple la communication. Nous nous attendons donc à un travail similaire pour le prochain laboratoire.