POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

LOG 2810
STRUCTURES DISCRETES

TP1 : GRAPHES

Session	Automne 2020
Pondération	10 % de la note finale
Taille des équipes	3 étudiants
Date de remise du projet	30 octobre 2020 (23h55 au plus tard)
Directives particulières	Soumission du livrable par moodle uniquement (https ://moodle.polymtl.ca).
	Toute soumission du livrable en retard est pénalisée à raison de 10% par jour de retard.
	Programmation C++
Les questions sont les bienvenues et peuvent être envoyées à : Aurel Josias Randolph (aurel.randolph@polymtl.ca), Saif-eddine Sajid (saif-eddine.sajid@polymtl.ca), Mohameth-Alassane Ndiaye (mohameth-alassane.ndiaye@polymtl.ca).	

1 Connaissances requises

- Notions d'algorithmique et de programmation C++.
- Notions de théorie des graphes.

2 Objectif

L'objectif de ce travail pratique est de vous permettre d'appliquer les notions théoriques sur les graphes que nous avons vues en cours, sur des cas concrets mais hypothétiques, tirés de votre quotidien. Dans cette optique, il est question dans ce travail de développer un outil permettant d'assister une voiture autonome dans ses trajets quotidiens.

3 Mise en situation

Avec l'arrivée des véhicules autonomes et la popularisation des véhicules électriques, les ingénieurs font face à des défis majeurs, comme la navigation et le ravitaillement en énergie. Les voitures autonomes futures devront être capable de déterminer des trajets à longue distance tout en tenant compte de l'autonomie du véhicule. Cependant, parce que toutes les voitures ne consomment pas le même carburant, les trajets devront être établis en fonction de la présence des points de charge et des stations

d'essence. Par exemple, une voiture autonome électrique devra assurer la présence de points de charges en créant le trajet.

4 Description

Un étudiant du département de génie informatique et génie logiciel émet l'idée de mettre en oeuvre une nouvelle application de résolution de chemin pour ce type de véhicules. Le système se veut modulaire, de sorte qu'il ne doit pas être limité à une seule carte routière : il faut donc fournir au système une carte de la région avant la recherche d'itinéraire. L'autonomie d'un véhicule étant limitée, cette carte doit fournir au système l'emplacement de divers stations de rechargement. Toujours dans un but d'utilisation modulaire, on veut pouvoir utiliser le système aussi bien pour des véhicules 100% électriques, hybrides ou essence. Ainsi, le système doit pouvoir reconnaître à la fois l'autonomie du véhicule et sa source d'énergie (essence ou électricité ou hybride), afin de pouvoir évaluer les stations utilisables par un véhicule donné. Afin de faciliter les notations, on indique l'autonomie restante du véhicule en kilomètres, de sorte que les distances sont directement reliées à l'autonomie. En recoupant toutes ses informations, tout utilisateur demandant une destination peut alors être transporté par le véhicule assisté du système de résolution de chemin en un temps minimal.

L'étudiant se rappelle des notions de structures discrètes acquises et décide de représenter la carte routière par un graphe, où les sommets du graphe représentent des points de départ ou de destination éventuels, et peuvent contenir une station service quelconque, et les arcs, les routes. Toutefois, à court de temps, il sollicite votre aide pour l'aider avec l'implémentation de certaines fonctionnalités de son application. Lors de votre rencontre, il vous expose son projet et vous rappelle les concepts suivants :

- **Représentation de la carte**
  La carte routière est représentée par un graphe orienté : une route à sens unique reliant deux sommets est représentée par un arc orienté dans le sens correspondant, une route à double-sens est représentée par deux arcs orientés. Un sommet contient en paramètres l'ensemble des sources d'énergie qu'il peut distribuer (`null`, `essence`, `elec`). À noter qu'un sommet peut ne rien distribuer, ou bien distribuer plusieurs sources différentes. On suppose qu'une station qui ditribue la source d'énergie correspondante au véhicule considéré réalise toujours le plein.

- **Degré d'un sommet**
  Dans un graphe orienté, un sommet possède un degré entrant (qui est le nombre d'arcs qui se dirigent vers le sommet) et un degré sortant (qui est le nombre d'arcs sortant du sommet). Le degré du sommet est alors la somme des deux valeurs précédentes.

- **Sous-graphe**
  Un sous-graphe est un graphe contenu dans un autre. Plus formellement, H est un sous-graphe d'un graphe G, si c'est un graphe, si l'ensemble des sommets de H est un sous-ensemble de l'ensemble des sommets de G, et si l'ensemble des arêtes (arcs) de H est un sous-ensemble de l'ensemble des arêtes de G.

- **Chemin**
  Un chemin reliant un sommet a à un sommet b, est l'ensemble des arêtes consécutives, reliant a à b. La séquence des arêtes parcourus définit le chemin entre a et b.

- **Connexité**
  Un graphe est connexe s'il existe toujours un chemin entre chaque paire de sommets distincts du graphe. Dans le cas contraire, un graphe est constitué de l'union de plusieurs sous-graphes disjoints, lesquels représentent les composantes connexes du graphe.

- **Graphe valué ou pondéré**
  Un graphe valué est un graphe dans lequel chacun des arcs présente une valeur. Cette valeur peut symboliser une distance, un coût, une durée etc. Dans le contexte de ce travail, les poids sur les arcs représentent la distance de la route séparant deux stations de rechargement.

- **Longueur d'un chemin**
  La longueur d'un chemin, dans un graphe pondéré, est la somme des poids des arêtes parcourues.

- **Propriété d'un sommet**
  Dans notre étude, chaque sommet est une station de rechargement : elle peut distribuer au choix exclusivement de l'électricité, ou bien exclusivement de l'essence, ou bien les deux carburants. On parle de propriété du sommet.

5 Composants à implémenter

C1. Écrire une fonction "creerGraphe()" qui permet de créer le graphe représentant les routes et les stations de chargement à partir d'un fichier, dont le nom est passé en paramètre.

C2. Écrire une fonction "lireGraphe()" qui permet d'afficher le graphe (*cf.* annexe a. pour un exemple d'affichage de la carte sous forme de graphe).

C3. Écrire la fonction "extractionGraphe()" qui permet d'extraire le sous-graphe résultant d'un graphe contenant uniquement les routes empruntables suivant l'autonomie maximale du véhicule.

C4. Écrire la fonction "plusCourtChemin()" qui permet de déterminer, à l'aide de l'algorithme de Dijkstra, le plus court chemin entre deux points dans un graphe, avec l'origine et la destination passés en paramètres. La fonction affiche l'autonomie finale restante du véhicule, le plus court chemin utilisé et la longueur de ce dernier, s'il existe, ou affiche un message d'erreur, dans le cas contraire.

C5. Faire une interface qui affiche le menu suivant:

   (a) Demander les caractéristiques du véhicule.
   (b) Mettre à jour la carte.
   (c) Adapter la carte aux caractéristiques.
   (d) Déterminer le plus court chemin.
   (e) Quitter

Notes

- Le programme doit toujours réafficher le menu, tant que l'option 5 n'a pas été choisie.
- L'utilisateur doit entrer un index valide, sinon le programme le signale et affiche le menu de nouveau.
- L'option (a) demande à l'utilisateur de choisir les caractéristiques du véhicule. L'utilisateur doit rentrer successivement une chaîne de caractères désignant le type de carburant utilisé (précisez dans le menu les chaînes de caractères acceptées par votre solution), l'autonomie maximale sous forme d'entier naturel, puis l'autonomie actuelle sous forme d'entier naturel.
- L'option (b) permet permet de lire une nouvelle carte afin de créer le graphe correspondant. Pour ce faire, le nom du fichier contenant les informations de la carte doit être demandé. Il est demandé d'afficher le graphe obtenu à la lecture d'un fichier. Le format du fichier est décrit en annexe.
- L'option (c) permet de déterminer, si possible, le plus court chemin entre le point de départ et la destination, tout en évitant de tomber en panne de carburant. Pour ce faire, les informations sur l'origine et la destination doivent être demandées. L'autonomie restante doit être prise en compte dans le calcul : si ce n'est pas le premier trajet effectué avec ce véhicule, l'autonomie restante en début du nouveau trajet doit correspondre à l'autonomie obtenue à la fin du précédent.

- L'option (a) doit être choisie avant l'option (c). Dans le cas contraire, le programme signale l'erreur et affiche le menu de nouveau.
- Si une nouvelle carte est lue ou si les caractéristiques sont mises à jour, l'option (d) doit être réinitialisée.

Prenez note que selon votre convenance, le mot "fonction" peut être remplacé par "méthode" et vice-versa, et que plusieurs autres fonctions/méthodes peuvent être ajoutées pour vous faciliter la tâche. La figure 1 présente un exemple de diagramme de classes à partir duquel vous pouvez travailler.

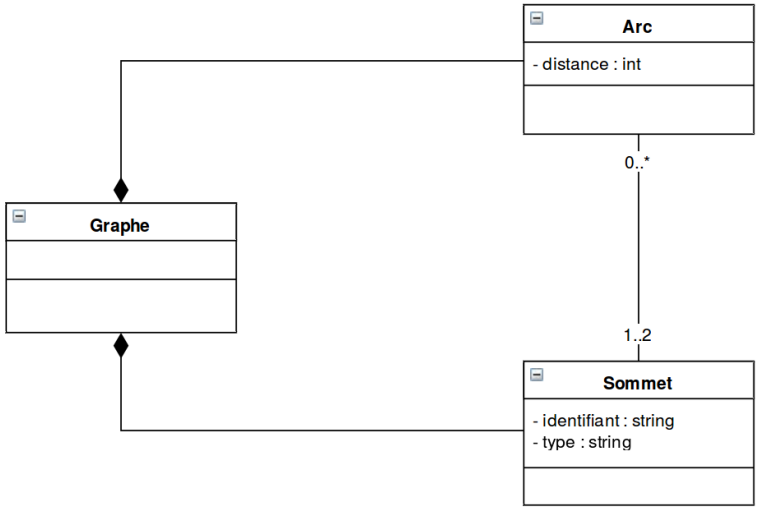

Figure 1: Exemple du diagramme de classes de la solution

6 Livrable

Le livrable attendu est constitué du code source et du rapport de laboratoire. Le livrable est une archive (ZIP ou RAR) dont le nom est formé des numéros de matricule des membres de l'équipe, séparés par un trait de soulignement (_). L'archive contiendra les fichiers suivants:

- les fichiers .cpp;
- les fichiers .h le cas échéant;
- le rapport au format PDF;
- les deux fichiers .txt formatés selon votre convenance et traduisant les graphes.

L'archive ne doit pas contenir de programme exécutable, de fichier de projet ou solution de Visual Studio, de répertoire Debug ou Release, etc. Les fichiers .cpp et .h suffiront pour l'évaluation du travail.

6.1 Rapport

Un rapport de laboratoire rédigé avec soin est requis à la soumission (format .pdf, maximum 8 pages). Sinon, votre travail ne sera pas corrigé (aussi bien le code source que l'exécutable). Le rapport doit obligatoirement inclure les éléments ou sections suivantes :

1. Page de présentation : elle doit contenir le libellé du cours, le numéro et l'identification du TP, la date de remise, les matricules et noms des membres de l'équipe. Vous pouvez compléter la page de présentation qui vous est fournie.
2. Introduction avec vos propres mots pour mettre en évidence le contexte et les objectifs du TP.
3. Présentation de vos travaux : une explication de votre solution. Ajoutez le diagramme de classes complet, contenant tous les attributs et toutes les méthodes ajoutés.
4. Difficultés rencontrées lors de l'élaboration du TP et les éventuelles solutions apportées.
5. Conclusion : expliquez en quoi ce laboratoire vous a été utile, ce que vous avez appris, vos attentes par rapport au prochain laboratoire, etc.

Notez que vous ne devez pas mettre le code source dans le rapport.

6.2 Soumission du livrable

La soumission doit se faire uniquement par moodle.

7 Évaluation

Éléments évalués	Points
Qualité du rapport : respect des exigences du rapport, qualité de la présentation des solutions, orthographe, grammaire	2
Qualité du programme: compilation, structures de données, gestion adéquate des variables et de toute ressource (création, utilisation, libération), passage d'arguments, gestion des erreurs, documentation du code, etc.	2
Composants implémentés : respect des requis, logique de développement, etc.	
C1	2
C2	2
C3	3
C4	2
C5	3
C6	4
Total de points	20

8 Documentation

- http://www.cplusplus.com/doc/tutorial/
- http://public.enst-bretagne.fr/∼brunet/tutcpp/Tutoriel%20de%20C++.pdf
- http://fr.openclassrooms.com/informatique/cours/programmez-avec-le-langage-c
- Algorithme de Dijkstra illustré : https://www.youtube.com/watch?v=0nVYi3o161A

Annexe

1 Plus court chemin

Soient un graphe valué G et deux sommets a et b de G. Pour calculer le plus court chemin entre a et b, utilisons l'algorithme de Dijkstra. Soit $d(x)$, la distance du sommet x par rapport au sommet de départ a, $w(u, v)$, la longueur d'une arête $\{u, v\}$ et $ch(a, x)$, le chemin (liste des arêtes traversés) du sommet a au sommet x.

- Au début de l'algorithme, les distances de chaque sommet x au sommet de départ a sont fixées à la valeur infinie ($d(x) = \infty$), à l'exception du sommet de départ, a, dont la distance (par rapport à lui-même) est 0. Pour chaque sommet, on associe également la liste des sommets traversés (le chemin emprunté) du sommet initial a jusqu'au sommet en question. À cette étape de l'algorithme, cette liste est vide pour tous les sommets, sauf pour le sommet a, dont la liste se résume à lui-même. En d'autres termes, pour tous les autres sommets x de G, on associe une étiquette "$\{x,\ \infty,\ ()\}$", et pour le sommet a, on a "$\{a,\ 0,\ (a)\}$". On considère également un sous-graphe vide S.
- À chaque itération, on sélectionne le sommet x de G, qui ne figure pas dans S, de distance minimale (par rapport au sommet a). Ce sommet x est ajouté au sous-graphe S. Par la suite, si nécessaire, l'algorithme met à jour les étiquettes des voisins x_v du sommet x ajouté. Cette mise à jour s'effectue si $d(x_v) > d(x) + w(x, x_v)$. Dans ce cas, l'étiquette du sommet x_v est modifiée comme suit:
  $\{x_v,\ d(x) + w(x, x_v),\ (ch(a, x), x_v)\}$.
- On répète l'opération précédente jusqu'à la sélection du sommet d'arrivée b, ou jusqu'à épuisement des sommets. L'étiquette associée à b donne alors le plus court chemin de a à b, de même que la longueur de ce dernier.

2 Informations utiles

a. **Affichage de la carte**
Pour afficher la carte, il faut indiquer, pour chaque sommet, si c'est une station, de quel type, et la liste des sommets voisins, comme illustré ci-dessous (par exemple) :

(station$_1$, type$_{station_1}$, (station_voisine$_{1_1}$, station_voisine$_{2_1}$, ..., station_voisine$_{n_1}$))
(station$_2$, type$_{station_2}$, (station_voisine$_{1_2}$, station_voisine$_{2_2}$, ..., station_voisine$_{m_2}$))
...

b. **Affichage du parcours**
Pour afficher le plus court chemin, la liste des stations définissant le trajet doit être présentée comme suit :

$\text{point}_{or} \rightarrow \text{point}_1 \rightarrow \text{point}_2 \rightarrow ... \rightarrow \text{point}_n \rightarrow \text{point}_{dest}$.

c. **Structure des fichiers**
Les cartes sont présentées sous forme de fichiers textes (*cf.* les fichiers textes fournis sur Moodle). Chaque fichier contient deux listes.

- La première ligne contient la liste des sommets. Chaque sommet est constitué d'un couple $(identifiant, type)$. Si le sommet n'est pas une station, le *type* est noté `null`. Sinon, le *type* désigne `essence`, `elec` ou `multi` – dans ce dernier cas, cela signifie que la station propose à la fois de l'essence et de l'électricité.
- La seconde ligne contient la liste des routes, séparées par des points-virgules ; chaque route est constituée de son sommet d'origine, son sommet d'arrivée, et la distance séparant les deux sommets, en kilomètres.