POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

LOG 2810
STRUCTURES DISCRETES

TP1 : GRAPHES

Session	Hiver 2020
Pondération	20 % de la note finale
Taille des équipes	3 étudiants
Date de remise du projet	22 mars (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++, Python ou Java
Les questions sont les bienvenues et peuvent être envoyées à : Saif-eddine Sajid (saif-eddine.sajid@polymtl.ca).	

1 Connaissances requises

- Notions d'algorithmique et de programmation C++, Python ou Java.
- Notions de théorie des graphes.
- Notions sur les structures de données.

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. Il s'agira ici d'optimiser le parcours d'un service du type Uber avec des voitures électriques dans Montréal. Cette tâche vous sera précisée dans la partie 5.

3 Mise en situation

Un étudiant du département de génie informatique et génie logiciel vient de trouver un stage auprès d'une compagnie qui développe une application afin d'aider les conducteurs de voitures électriques qui offrent un service tel qu'Uber à maximiser le nombre de clients qu'ils peuvent servir (et donc leur revenu !), tout en prenant en compte la satisfaction des clients. On le met en charge de l'algorithme qui déterminera les chemins qu'emprunteront les conducteurs et de la gestion des requêtes et leur acheminement puisqu'il a indiqué dans son C.V. avoir fait le cours de structures discrètes. Malheureusement, il ne maîtrise pas très bien cette matière. Il a donc désespérément besoin de votre aide.

Il décide d'associer à chaque chemin possible entre deux points de la ville un coût en temps, qui dépend bien évidemment de la distance entre les deux points (le coût est proportionnel à celle-ci). L'étudiant a fait beaucoup de recherches sur Internet pour déterminer les voies empruntables par les conducteurs, et voudrait que son algorithme soit capable de proposer un chemin optimal à ceux-ci pour minimiser les coûts d'exploitation. Pour cela, il décide de représenter les chemins empruntables par un graphe des durées de déplacement entre les quartiers de Montréal. Ainsi, Montréal est représentée dans le petit logiciel console qu'il élabore par un graphe dont les sommets correspondent aux points centraux des différents quartiers de Montréal, et les arêtes aux temps entre deux points centraux (sommets). L'étudiant vous fournira bien évidemment ce graphe.

4 Description

Lors de la séance de présentation du projet, il vous explique les concepts suivants, qui sont des concepts simplifiés de l'application qu'aimerait créer la compagnie, mais suffisants pour faire une première ébauche de son travail de stage. Il vous recopie à partir de son manuel un rappel sur certaines notions de la théorie des graphes et vous donne les informations sur certaines particularités des taxis qui seront utilisés pour les transports.

- Comme dit précédemment, chaque déplacement vers un autre quartier de Montréal possède un coût en temps proportionnel à la distance parcourue.
- La carte du lieu dans lequel progressera un conducteur repose sur une matrice de coût (en temps) entre les différents points centraux des quartiers et est donc représentable par un graphe non orienté, puisque les conducteurs savent respecter leur couloir de circulation à droite de la rue. Les sommets sont les points centraux des quartiers et les arêtes sont valuées par les coûts. Un tel graphe sera connexe, à savoir que l'on connaîtra le temps entre un quartier et les autres de Montréal.
- Les conducteurs peuvent faire le transport de plusieurs clients en même temps dans la journée, mais ont **une capacité maximale de 4 clients à la fois**.
- Les voitures ont un besoin de rechargement à prendre en compte. En effet, les voitures ont en général une certaine autonomie et on doit les recharger afin d'éviter de tomber en panne en plein transport. Certains points centraux des quartiers sont pourvus de stations de recharge où les voitures pourront faire le plein d'énergie dans leurs batteries.
- Les clients transportés ont un temps auquel ils s'attendent à arriver à leur destination.
- La compagnie estime que les clients n'acceptent pas les retards pour le temps d'arrivée et qu'ils sont toujours prêts à partir dès que le conducteur arrive.
- Les requêtes des clients doivent aussi être traitées selon leur ordre d'arrivée (premier arrivé, premier servi).
- **Dans les cas où le conducteur ne peut pas respecter les contraintes temporelles de certains clients, un autre conducteur sera envoyé à ceux-ci (on ne se préoccupe plus de ceux-ci). L'algorithme devra également vérifier que la voiture peut faire le chemin en respectant les contraintes d'autonomie, on doit se diriger vers une borne de recharge avant de passer sous la barre critique de 15% pour éviter une panne**
- Les sommets du graphe représentant la carte devront contenir un paramètre de recharge si ce point central de quartier est pourvu d'une station de recharge. Lorsqu'un conducteur s'arrête à une station, il attend que ses batteries récupèrent leur plein potentiel. Le potentiel d'une batterie est exprimé sur 100. À 100% d'énergie, les batteries de la voiture sont pleines. **À 0% d'énergie, la voiture tombe sur la voie publique, ce qu'il faut de toute évidence éviter à tout prix.**

- À cet effet, **l'algorithme doit éviter de laisser le niveau de la batterie baisser en bas de 15%**. En revanche, il est possible de recharger avec des clients à l'intérieur de la voiture. Une recharge de la batterie prend 10 minutes en tout, peu importe le niveau original de celle-ci.
- Au départ de chaque trajet, on considère que le niveau des batteries est à 100% d'énergie.
- La perte d'autonomie d'une voiture est de 1% par minute.
- 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 de H est un sous-ensemble de l'ensemble des arêtes de G.
- Un chemin reliant un sommet a à un sommet b est l'ensemble des arêtes consécutives reliant a à b. La séquence des arêtes parcourues définit le chemin entre a et b.
- 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.
- Un graphe valué (ou pondéré) est un graphe dans lequel chacunes des arêtes présente une valeur. Cette valeur peut symboliser une distance, un coût, une durée, etc.
- La longueur d'un chemin, dans un graphe pondéré, est la somme des poids des arêtes parcourues.

Le fichier `arrondissements.txt` contient les informations nécessaires pour l'algorithme.

- Les premières lignes contiennent la liste des sommets. Chaque ligne contient le numéro d'un arrondissement et si cet arrondissement contient une borne de recharge (1) ou non (0). Ces deux nombres sont séparés par une virgule.
- Il y a une ligne vide entre les lignes sur les sommets et celles sur les arêtes.
- Les lignes suivantes contiennent la liste des arêtes. Chaque ligne contient le numéro du premier sommet, le numéro du deuxième sommet, et le temps de parcours entre les deux (en minutes). Les arêtes ne sont pas orientées.

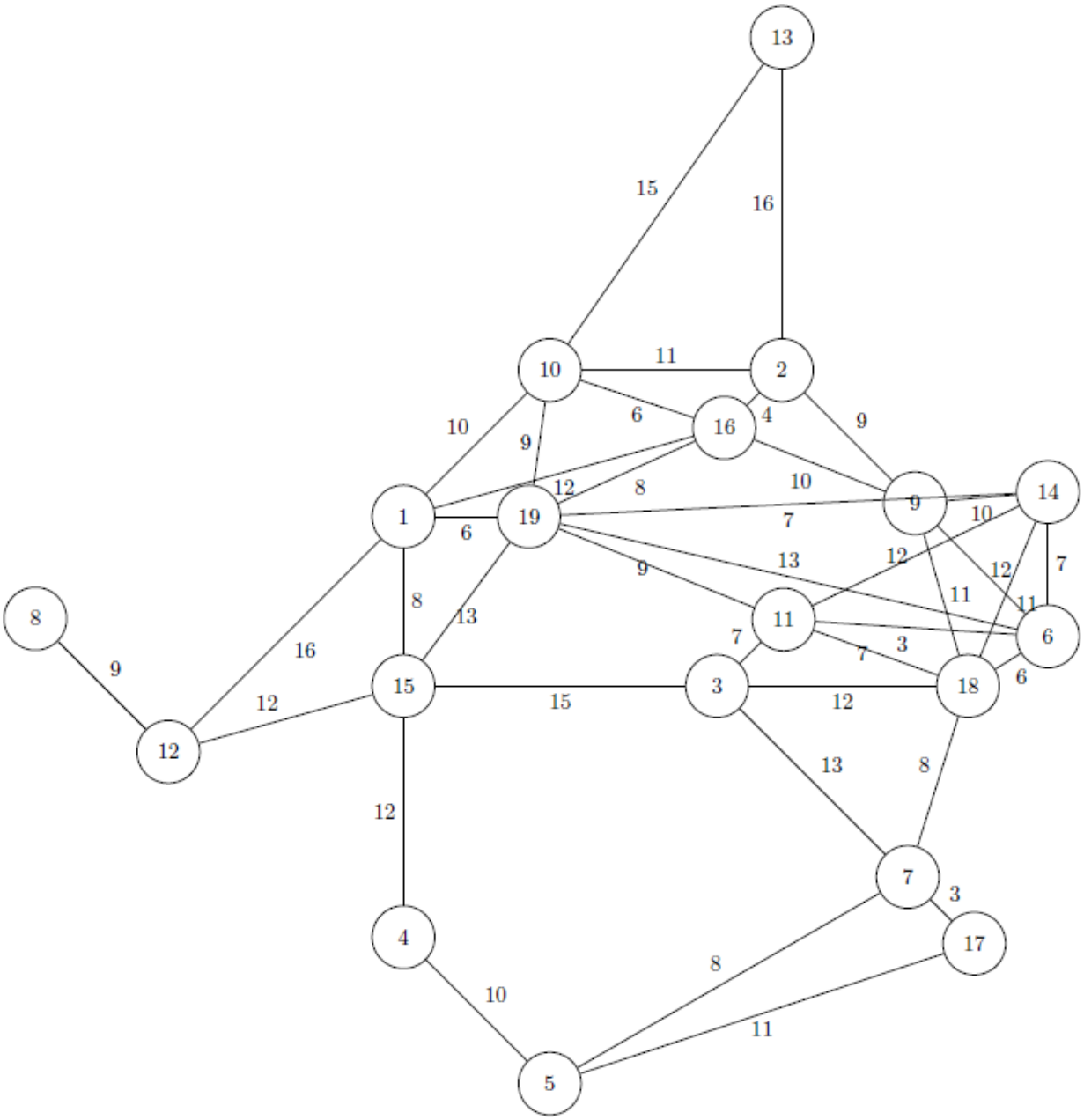

FIG. 1 : Ébauche du graphe dans le fichier arrondissements.txt

Le fichier `requetes.txt` contient les informations nécessaires pour les requêtes des clients.

- La première ligne contient le noeud de départ du conducteur.
- Par la suite, chaque ligne contient l'identifiant d'un client, son point de départ, son point d'arrivée et dans combien de temps en minutes il aimerait arriver à destination (depuis la réception de la série de requêtes qui est le moment 0).

5 Composants à implémenter

C1. Écrire une fonction récursive “creerGraphe()” qui permet de créer le graphe représentant les routes et les points centraux des arrondissements (sommets) à partir d'un fichier dont le nom est passé en paramètre.

C2. Écrire une fonction “afficherGraphe()” 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 ”plusCourtChemin()” qui permet de déterminer, en vous inspirant de l'algorithme de Dijkstra, le plus court chemin entre un sommet et un autre. L'origine (point de départ) et la destination (sommet d'arrivée) doivent être passées en paramètres. La fonction affiche le pourcentage final d'énergie dans les batteries de la voiture, le plus court chemin utilisé (d'après la liste de ses sommets, selon le format de l'annexe) et la longueur de ce dernier en minutes.

C4. Écrire la fonction ”traiterRequetes()” qui permet de déterminer, en vous inspirant de l'algorithme de Dijkstra, la faisabilité de requêtes de clients (nombre maximum de quatre clients à la fois à l'intérieur de la voiture, charge qui ne baisse pas en dessous de 15%, contraintes temporelles des clients) et de déterminer le chemin qu'il faut prendre pour desservir ceux-ci. Les requêtes proviennent d'un fichier qui s'intitule requetes.txt et elles devront être traitées selon leur ordre d'arrivée (leur ordre dans le fichier). La fonction affiche le pourcentage final d'énergie dans les batteries de la voiture, le plus court chemin utilisé (les sommets où les clients ont été ramassés, ceux où la voiture a été rechargée et les identifiants des clients doivent être clairment identifiés) et la longueur de ce dernier en minutes.

C5. Faire une interface qui affiche le menu suivant :

   (a) Mettre à jour la carte.

   (b) Déterminer le plus court chemin sécuritaire.

   (c) Traiter les requêtes.

   (d) Quitter.

   Notes

- Le programme doit toujours réafficher le menu, tant que l'option (d), ou « Quitter », 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) permet de lire une nouvelle carte afin de créer le graphe correspondant. Pour lire une nouvelle carte, on doit lire le fichier se nommant ”arrondissements.txt” qui se trouve dans le dossier courant. Il est demandé d'afficher le graphe obtenu à la lecture d'un fichier. Le format du fichier est décrit plus haut.
- L'option (b) permet de déterminer le plus court chemin sécuritaire d'après les points de départ et d'arrivée. Pour ce faire, les paramètres doivent être demandés par la console.
- L'option (c) permet de traiter les requêtes reçues. Pour ce faire, on doit lire le fichier se nommant ”requetes.txt” qui se trouve dans le dossier courant. Le format du fichier est décrit plus haut.
- Si une nouvelle carte est lue, les options (b) et (c) doivent être réinitialisées.

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.

6 Livrable

Le livrable attendu est constitué du code source et du rapport de laboratoire. Le livrable est une archive (ZIP ou RAR ou tar.gz) 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 ou .py ou .java ;
- les fichiers .h le cas échéant ;
- le rapport au format PDF ;
- les fichiers .txt passés en argument.

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, ou .py, ou. java suffiront pour l'évaluation du travail.
Les fichiers .txt qui seront utilisés pour les tests ne seront pas nécessairement ceux qui sont fournis, mais leur format sera pareil.

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. 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 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 ajoutées.
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. Vous pouvez également indiquer le temps passé sur ce TP à des fins d'ajustement pour le prochain qui aura lieu vers la fin de la session.

Notez que vous ne devez pas mettre de 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	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.	3
Composants implémentés : respect des requis, logique de développement, etc.	
C1	3
C2	2
C3	3
C4	5
C5	2
Total de points	20

8 Documentation.txt

- `http://www.cplusplus.com/doc/tutorial/`
- `http://public.enst-bretagne.fr/$\sim$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`
- `learnxinyminutes.com`

Annexe

1 Plus court chemin

Soient un graphe valué (ou pondéré) 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 traversées) 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, ∞, ()", 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 n'apparaît 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 du graphe**
Pour afficher le graphe, il faut indiquer, pour chaque sommet, quel est son numéro, et la liste des sommets voisins avec les temps associés, comme illustré ci-dessous :
(noeud1, borne ou pas, ((noeud_voisin$_1$, durée$_1$), (noeud_voisin$_2$, durée$_2$), ..., (noeud_voisin$_n$, durée$_n$)))
(noeud2, borne ou pas, ((noeud_voisin$_1$, durée$_1$), (noeud_voisin$_2$, durée$_2$), ..., (noeud_voisin$_n$, durée$_n$)))
...

(b) **Affichage du parcours**
Pour afficher le plus court chemin, la liste des sommets (ou objets) définissant le trajet doit être présentée comme suit :
$point_{départ}$ → $point_1$ → $point_2$ → *recharge* → ... → *débarquement client # 7* → $point_n$ → $point_{arrivée}$