POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

LOG 2810
STRUCTURES DISCRETES

TP1 : GRAPHES

Session	Hiver 2018
Pondération	10 % de la note finale
Taille des équipes	3 étudiants
Date de remise du projet	4 Mars 2018 (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 : Loïc Lerat (loic.lerat@polymtl.ca), Juliette Tibayrenc (juliette.tibayrenc@polymtl.ca).	

1 Connaissances requises

- Notions d'algorithmique et de programmation C++, Python ou Java.
- 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. À cet effet, il est question dans ce travail de permettre d'optimiser le parcours d'un braqueur de banque à travers le Canada.

3 Mise en situation

Joe, un étudiant du département de génie informatique et génie logiciel, se prépare à réaliser une série de braquages de banques à travers le Canada pendant la semaine de relâche, afin de payer ses études. Avant de se lancer, il cherche à déterminer le chemin qu'il empruntera entre les différentes villes du Canada. Malheureusement, il n'a pas été très attentif au cours de structures discrètes et 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 villes 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 routes empruntables avec son fourgon, et

voudrait que son algorithme soit capable de lui proposer un chemin optimal pour minimiser le temps passé sur la route et échapper plus facilement à la police. Pour cela, il décide de représenter les routes par un graphe des temps de parcours entre les villes du Canada. Ainsi, le Canada est représenté dans le petit logiciel console qu'il élabore par un graphe dont les sommets correspondent aux différentes villes, et les arêtes aux temps de parcours (en heures) entre deux villes (sommets). L'étudiant vous fournira bien évidemment ce graphe des distances, et il a besoin de vous pour l'aider à implémenter certains composants de son algorithme.

4 Description

Lors de votre rencontre, Joe vous explique les concepts suivants à prendre en compte pour la réalisation d'un braquage. 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 véhicules qui seront utilisés pour transporter l'argent et prendre la fuite.

- Comme dit précédemment, chaque déplacement vers une autre ville par la route possède un temps proportionnel à la distance parcourue.
- La carte routière du Canada qu'il utilisera pour s'orienter repose sur une matrice de temps de parcours entre les différentes villes, et est donc représentable par un graphe non orienté car une route peut être empruntée dans un sens ou dans l'autre. Les sommets sont les villes et les arêtes les temps de parcours entre ces villes. Un tel graphe sera connexe, à savoir que l'on connaîtra le temps de parcours entre une ville et toutes les autres au Canada.
- Les véhicules utilisés par le braqueur consomment de l'essence, leur **autonomie** est donc limitée et il est nécessaire de s'arrêter régulièrement pour faire le plein. Certaines villes sont pourvues de **stations service** où il pourra s'arrêter et faire le plein (on suppose qu'il n'existe pas d'autres stations services que celles indiquées).
- Pour transporter l'argent et fuir après un braquage, Joe doit louer un véhicule. Pour cela il a le choix entre **2 compagnies** de location : "Cheap Car" qui ne possède que de vieux modèles ayant une consommation d'essence élevée, et "Super Car" qui possède des véhicules haut-de-gamme qui ont une consommation plus faible.
- En fonction du système de sécurité de la banque, Joe a plus ou moins de temps pour charger son véhicule avec le contenu du coffre-fort. Selon la quantité d'argent à récupérer, il choisira donc différents types de véhicules. Les **3 types de véhicules** pouvant être utilisés sont : la **voiture** classique qui ne permet de transporter qu'une somme limitée d'argent, le **pick-up** qui permet de transporter une bonne quantité d'argent, et le **fourgon** qui permet d'emporter la totalité du contenu du coffre-fort de la banque.
- Ainsi, la voiture consomme moins d'essence que le pick-up qui lui-même consomme moins que le fourgon. L'autonomie dépendra donc du véhicule choisi.
- La location chez "Super Car" est à éviter car les prix sont beaucoup plus chers et Joe voudrait économiser le plus possible. L'algorithme doit conséquemment être capable de détecter les situations où un véhicule de chez "Cheap Car" ne disposerait pas d'assez d'autonomie et ne pourrait pas passer par suffisamment de stations service pour arriver à destination.
- Dans les cas où le véhicule loué chez "Cheap Car" ne suffirait pas, l'algorithme devra également vérifier que le même véhicule loué chez "Super Car" pourrait quant à lui achever le trajet. Si ce n'est pas le cas, alors le braquage ne pourra pas être effectué.
- Les sommets du graphe représentant la carte devront contenir un paramètre de recharge si cette ville est pourvue d'une station de recharge. Lorsque Joe s'arrête à une station, il remplit complètement le réservoir. Le niveau d'essence est exprimé sur 100. À 100% d'essence, le réservoir du

véhicule est plein. À 0%, c'est la panne sèche, Joe se retrouve bloqué au milieu de la route et se fera rattraper par la police, ce qu'il faut de toute évidence éviter à tout prix.

- À cet effet, l'algorithme doit éviter de donner à Joe un chemin qui fait passer le niveau d'essence de son véhicule en dessous de **12%**. Si c'est le cas, on préférera une autre chemin plus coûteux en termes de temps mais plus sécuritaire pour la réussite de fuite. Si une telle trajectoire n'existe pas, c'est seulement à ce moment-là qu'on utilisera un véhicule de chez "Super Car". Si les mêmes conditions ne sont pas respectées non plus pour le véhicule de chez "Super Car", on ne devra pas effectuer le braquage.
- Chaque arrêt dans une station service représente un coût en temps dont on doit tenir compte pour calculer le plus court chemin. Le temps nécessaire pour s'arrêter et faire le plein est de **15 minutes**, quel que soit le type de véhicule et son niveau d'essence. Joe s'arrête pour faire le plein dès qu'il passe par une station service, peu importe le niveau d'essence. En quittant la station, l'essence est à 100%.
- Au départ de chaque trajet, on considère que le véhicule est à 100% d'essence.
- La consommation d'essence d'un véhicule de chez "Cheap Car" va comme suit : une voiture perd **5%** d'essence par heure de route, un pick-up perd **7%** d'essence par heure de route, un fourgon perd **8%** d'essence par heure de route,
- La consommation d'essence d'un véhicule de chez "Super Car" va comme suit : une voiture perd **3%** d'essence par heure de route, un pick-up perd **4%** d'essence par heure de route, un fourgon perd **6%** d'essence par heure de route.
- 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 chacune 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.

Voici la carte des villes du Canada entre lesquelles Joe doit se déplacer :

FIG. 1 : Villes du Canada[1]

1	Montréal	11	Saskatoon
2	Québec	12	Calgary
3	Ottawa	13	Vancouver
4	Toronto	14	Edmonton
5	Halifax	15	Fort McMurray
6	Sept-Îles	16	Churchill
7	Thunder Bay	17	Prince George
8	Sandy Lake	18	Fort Nelson
9	Winnipeg	19	Whitehorse
10	Regina	20	Yellowknife

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

- Les premières lignes contiennent la liste des sommets. Chaque ligne contient le numéro d'une ville et si cette ville contient une station service (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 heures (ces temps ne sont pas nécessairement fidèles à la réalité). Les arêtes ne sont pas orientées.

1. Source : `http://www.carte-du-monde.net/pays-153-carte-grande-villes-canada.html`

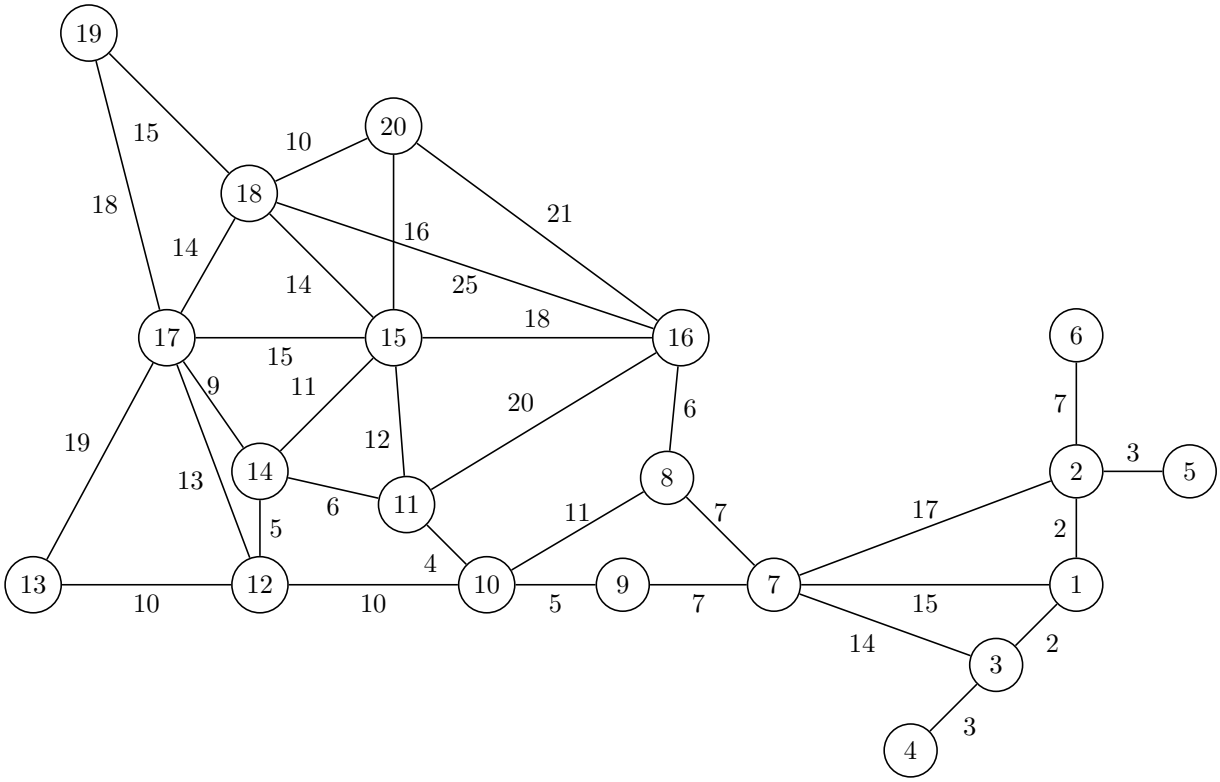

Fig. 2 : Graphe des villes du Canada et temps de parcours entre les villes

5 Composants à implémenter

C1. Écrire une fonction “creerGraphe()” qui permet de créer le graphe représentant les routes et les villes (sommets) 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 ”plusCourtChemin()” qui permet de déterminer, en vous inspirant de l'algorithme de Dijkstra, le plus court chemin sécuritaire pour fuir avec un véhicule dont le type est passé en paramètre. L'origine (point de départ) et la destination (sommet d'arrivée) doivent aussi être passés en paramètres. La fonction affiche la compagnie de location choisie (”Cheap Car” ou ”Super Car”), le pourcentage final d'essence dans le réservoir du véhicule, 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 temps (minutes). Si le braquage est annulé, rien de tout cela ne doit être affiché ; on doit plutôt recevoir un message d'avertissement justifiant l'impossibilité d'effectuer le braquage.

C4. Faire une interface qui affiche le menu suivant :

(a) Mettre à jour la carte.

(b) Déterminer le plus court chemin sécuritaire.

(c) Quitter.

Notes

- Le programme doit toujours réafficher le menu, tant que l'option (c), 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. La génération d'une nouvelle carte pourrait être importante dans les cas où des travaux importants seraient en cours par exemple. Pour lire une nouvelle carte, 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 (b) permet de déterminer le plus court chemin sécuritaire d'après les points de départ (la ville où est commis le braquage) et d'arrivée ainsi que le type de véhicule utilisé. Pour ce faire, les paramètres doivent être demandés par la console.
- 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. La figure 3 présente un exemple de diagramme de classes à partir duquel vous pouvez travailler.

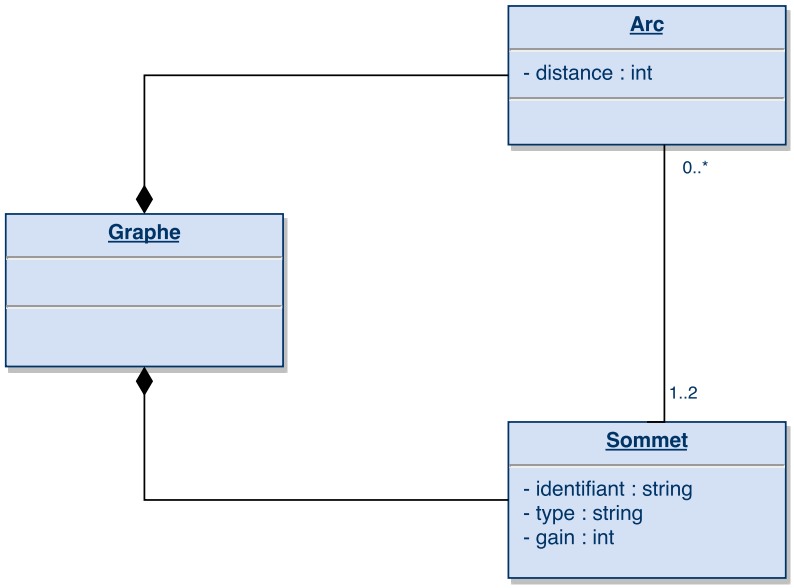

FIG. 3 : Exemple du diagramme de classes de la solution

6 Livrable

Le livrable attendu est constitué du code source et du rapport de laboratoire. Le livrable est une archive (ZIP ou RAR 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 ;

- le fichier .txt passé en argument (option (a)), s'il est différent de celui fourni par l'instructeur du laboratoire (vous pouvez en effet modifier le format des informations contenues dans le fichier fourni sur Moodle, mais vous devez à ce moment-là le remettre avec vos travaux).

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.

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	4
C2	3
C3 Divisé comme suit	5
C3.1 Implémentation correcte de Dijkstra	2
C3.2 Choix de chemin selon le véhicule/refus	2
C3.3 Choix de l'entreprise	1
C4	3
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 :
(objet1, numéro1, ((objet_voisin$_{1_1}$, durée$_{1_1}$), (objet_voisin$_{2_1}$, durée$_{2_1}$), ..., (objet_voisin$_{n_1}$, durée$_{n_1}$)))
(objet2, numéro2, ((objet_voisin$_{1_2}$, durée$_{1_2}$), (objet_voisin$_{2_2}$, durée$_{2_2}$), ..., (objet_voisin$_{n_2}$, durée$_{n_2}$)))
...

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

(c) **Structure des fichiers**
Les cartes sont présentées sous forme de fichier textes (cf. le fichier texte fourni sur Moodle). Un tel fichier contient deux listes.