Automne 2016

Algorithmes et structures de données pour ingénieur

GLO-2100

Travail à faire en équipes de 1 ou 2
À rendre avant la date indiquée
sur le site ENA du cours
(Voir modalités de remise à la fin de l'énoncé)

Travail Pratique #2

Mise en œuvres de classes pour le travail de session

UNIVERSITÉ LAVAL
Faculté des sciences et de génie
Département d'informatique
et de génie logiciel

1 - Objectif général

Ce TP constitue la deuxième partie de votre travail de session. On vous rappelle que le travail de session consiste à produire un logiciel qui permettra d'effectuer **efficacement** des opérations choisies par les utilisateurs du réseau de transport en commun de la ville de Québec. Ce travail mets l'emphase sur l'opération la plus effectuée par les utilisateurs du réseau: la recherche d'un trajet entre deux points A et B. Le défi de ce travail réside donc dans la construction d'une structure de données "Graphe" et l'implémentation des algorithmes de plus court chemin.

2 - Utilisation de votre TP1

Ce TP2 est une continuité de votre TP1, vous devez donc réutiliser toutes les classes que vous aviez préalablement développées. Même le code dans votre programme principal pourra être partiellement réutilisé. Nous vous prions donc de porter attention aux remarques des correcteurs afin d'améliorer votre solution pour le TP1 et obtenir les résultats désirés pour le présent TP.

3 - Ajouts au TP2

Vous devez principalement ajouter à votre travail, deux nouvelles classes (Reseau et Gestionnaire) et réécrire un nouveau programme principal.

3.1. Classe Réseau

Conformément à l'interface qui vous est fournie avec cet énoncé, cette classe est une implémentation d'un graphe qui représente le réseau de transport de la ville. Ce graphe doit présenter les spécifications suivantes:

- être orienté, pondéré et étiqueté (voir ci-dessous);
- les sommets sont identifiés de manière unique par un entier positif;
- l'accès à un sommet par l'intermédiaire de son identifiant doit se faire en O(1);
- les arcs sont étiquetés par une paire d'entiers positifs. Le premier élément de la paire représente le coût de l'arc, et le second représente le type de l'arc. Le type de l'arc est 0 si l'arc est traversé par un bus, 1 sinon. Un arc de type 1 représente un parcours à pieds entre deux stations.
- soit Nadj le nombre de sommets adjacents à un noeud, tous les arcs de ce sommet doivent être parcourues en O(Nadj).
- les algorithmes de plus court chemin Disjktra et Bellman Ford doivent être respectivement en $O(n^2)$ et O(nm) pour le graphe de n sommets et m arcs. Pour plus d'efficacité, votre algorithme de Bellman-Ford doit s'arrêter dès qu'il n'y a plus de relâchements possibles et celui de Disjktra doit s'arrêter dès qu'on atteint la destination.
- l'algorithme de Kosaraju doit être en O(n+m). Il en est de même pour l'opération de trouver le graphe inverse.

Il est de votre responsabilité de trouver les structures de données à utiliser dans la partie privée de votre implémentation pour respecter les spécifications ci-dessus. Vos choix pourraient entraîner une pénalité si votre implémentation ne respecte pas à la fois l'interface fournie et ces spécifications demandées.

3.2. Classe Gestionnaire

Cette classe représente un gestionnaire du RTC qui devrait satisfaire les requêtes des utilisateurs de votre application. Dans sa partie privée, nous vous imposons les attributs suivants:

- m_lignes: conteneur de toutes les lignes du RTC. Bien que le fichier routes.txt puisse contenir plusieurs entrées pour une même ligne, on vous demande d'avoir un seul objet Ligne pour un certain numéro de ligne dans votre conteneur. Cette contrainte devra être pris en compte lorsque vous enregistrez les voyages de cet objet Ligne. Exemple: la ligne "800" a deux entrées avec les identifiants 111 et 145. On veut que le conteneur ait un seul objet Ligne "800", peut importe l'id que vous conservé parmi les deux. Mais sachant que des voyages de la "800" refère à l'id 111 alors que d'autres réfère à l'id 145, votre objet Ligne "800" doit enregistrer tous ces voyages indépendamment de l'id conservé.
- m_stations: conteneur de toutes les stations du RTC
- m_voyages: conteneur de tous les voyages du RTC
- m_voyages_date: conteneur permettant de retrouver très rapidement les voyages à une date donnée (contrairement à l'utilisation du conteneur précédent).
- m_reseau: une instance de la classe Reseau décrivant les liens entre les stations du RTC

Votre tâche consiste à trouver le type approprié pour les conteneurs ci-dessus afin de respecter les spécifications de complexité des méthodes dans l'interface.
L'interface de votre classe devra contenir les méthodes suivantes:

```
/*!
* \brief Constructeur de la classe Gestionnaire. C'est ici que vous devez lire et charger les fichiers gtfs utiles au tp.
* \param chemin_dossier: chemin d'accès vers le dossier gtfs. Dans le cadre du tp, vous devez vous intéresser aux fichiers
* routes.txt, trips.txt, stop_times.txt, stops.txt, calendar_dates.txt
*
*/
Gestionnaire(std::string chemin_dossier)
```

```
/*!
* \brief Permet de vérifier si une date existe dans l'ensemble des dates de "calendar.txt" lors de la construction du gestionnaire.
* \param date: la date d'intérêt
* \return True ssi la date est prise en charge dans l'application
*/
bool date_est_prise_en_charge(const Date& date)
```

```
/*!
* \brief Permet de vérifier si une ligne avec un certain  numéro existe dans l'ensemble des lignes chargées dans le constructeur.
* \param num_ligne: le numéro de la ligne d'intérêt
* \return True ssi le numéro est pris en charge
*/
```

```
bool Gestionnaire::bus_existe(std::string num_ligne)
```

```
/*!
* \brief Permet de vérifier si un numéro de station existe dans l'ensemble des stations chargées dans le constructeur.
* \param station_id: l'identifiant de la station d'intérêt
* \return True ssi la station est prise en charge
*/
bool Gestionnaire::station_existe(int station_id)
```

```
/*!
* \brief Accès à une ligne à partir de son numéro
* \param num_ligne: le numéro de la ligne d'intérêt
* \return objet Ligne correspondant au numéro d'intérêt
* \exception Il n'existe pas de ligne ayant ce numéro
*/
Ligne Gestionnaire::getLigne(std::string num_ligne)
```

```
/*!
* \brief Accès à une station à partir de son id
* \param station_id: l'identifiant de la station d'intéret
* \return objet Station correspondant à l'id d'intérêt
* \exception Il n'existe pas de station ayant cet identifiant
*/
Station Gestionnaire::getStation(int station_id)
```

```
/*!
* \brief Permet de trouver les voyages d'une ligne qui passe par une station
* \param station_id: l'identifiant de la station d'intérêt
* \param num_ligne: le numéro de la ligne d'intérêt
* \return un vecteur contenant des pointeurs vers les voyages trouvés
* \exception logic_error si la station est inexistante
* \exception logic_error si la ligne est inexistante
*/
std::vector<Voyage*> Gestionnaire::trouver_voyages(int station_id, std::string num_ligne)
```

```
/*!
* \brief Permet d'obtenir les destinations des voyages d'une ligne s'arrêtant à une station.
* Notez qu'une ligne de bus ne peut pas avoir plus de deux destinations possibles: une pour l'aller et l'autre pour le retour.
* Dans le cas de certains bus (ex: couche-tard), une seule destination est possible.
* \param num_ligne: numéro de la ligne d'intérêt
* \param station_id: numéro de la station d'intérêt
* \return Une paire de chaîne de caractères.
* Si le bus ne passe pas par la station, alors une paire de chaîne vide est retournée. \n
* Sinon Si le bus a deux destinations possibles les deux éléments de la paire doivent être différents de la chaîne vide.
* Si le bus a une seule destination possible, le dernier élément de la paire seulement est
```

```
égale à une chaîne vide.
*
*/
std::pair<std::string, std::string> Gestionnaire::get_bus_destinations(int station_id,
std::string num_ligne)
```

```
/*
* \brief Trouver des stations environnantes étant donnée une coordonnée gps et un rayon
* \param coord: Coordonnée gps d'intérêt
* \param rayon: cette distance défini la circonférence a l'intérieure de laquelle on se trouve
les stations que l'on cherche
* \return un vecteur de paires (distance, pointeur vers une station) trié par distance
*/
std::vector<std::pair<double, Station*>>
Gestionnaire::trouver_stations_environnantes(Coordonnees coord, double rayon)
```

```
/*!
* \brief trouver l'horaire d'un bus à une station
* \param date: la date d'intérêt
* \param heure: l'heure à partir de laquelle on veut l'horaire
* \param numero_ligne: numéro de la ligne dont on cherche l'horaire
* \param station_id: l'identifiant de la station où on veut connaitre l'horaire de passage du
bus
* \param destination: permet de spécifier dans quelle direction on veut l'horaire.
* Ceci est pertinent car à certaines stations, la même ligne de bus peut passer dans les deux
sens.
* \return un vecteur contenant les heures d'arrivée (en ordre croissant) du bus à la station
d'intérêt
* \exception logic_error si la station est inexistante
* \exception logic_error si la ligne est inexistante
*/
std::vector<Heure> Gestionnaire::trouver_horaire(Date date, Heure heure, std::string
numero_ligne, int station_id, std::string destination)
```

```
/*!
* \brief permet de déterminer si le réseau de transport est fortement connexe.
* \param date: la date d'intérêt
* \param heure: l'heure à partir de laquelle on veut faire le calcul
* \param[in] considerer_transfert: indique si vous devez considérer ou pas les arcs de
transfert dans votre calcul
* \return true ssi à partir de n'importe quelle station du réseau on peut atteindre n'importe
quelle autre.
*/
bool Gestionnaire::reseau_est_fortement_connexe(Date date, Heure heure_deb, bool
considerer_transfert)
```

```
/*!
```

```
* \brief permet de déterminer les composantes fortement connexes du réseau de transport.
* \param date: la date d'intérêt
* \param heure: l'heure à partir de laquelle on veut faire le calcul
* \param[in] considerer_transfert: indique si vous devez considérer ou pas les arcs de transfert dans votre calcul
* \param[out] composantes: vecteur pour stocker les différentes composantes fortement connexes.
*/
void Gestionnaire::composantes_fortement_connexes(Date date, Heure heure_deb, std::vector< std::vector<unsigned int> >& composantes, bool considerer_transfert)
```

```
/*!
* \brief Trouver le plus court chemin en autobus pour aller d'un point A vers un point B dans l'interval de temps défini par  interval_planification_en_secondes
* à partir d'une heure de départ et pour une date donnée
* Pour ce faire, il faut initialiser le réseau, puis faire appel à ses routines de plus courts chemin
* \param date: la date de planification
* \param heure_depart: l'heure de début de planification
* \param depart: coordonnées gps du point de départ de votre déplacement
* \param destination: coordonnées gps du point de d'arrivée de votre déplacement
* \return Un vecteur contenant les stations du chemin trouvé, le vecteur est vide si aucun chemin n'est trouvé
*/
std::vector<unsigned int> Gestionnaire::plus_court_chemin(Date date, Heure heure_depart, Coordonnees depart, Coordonnees destination)
```

```
*** Cette méthode est privée ***
/*!
* \brief initialiser ou réinitialiser l'attribut m_reseau en fonction des paramètres.
* La date et l'interval de temps spécifié par l'utilisateur servent à trouver les arcs empruntés par les bus dans le réseau.
* Les coordonnées de départ et de destination servent à ajouter des stations fictives ayant respectivement les numéros 0 et 1.
La dist_de_marche sert à ajouter les arcs entre ces stations fictives et toutes les autres stations dans un rayon de dist_de_marche.
* La dist_transfert permet d'ajouter des arcs de transfert entre les stations qui sont à une distance l'une de l'autre inférieure à dist_transfert
* \param date: la date d'intérêt
* \param heure_depart: l'heure de début.
* \param heure_fin: l'heure de fin.
* \param depart: coordonnées gps du point de départ du déplacement pour lequel on initialise le réseau
* \param dest: coordonnées gps du point de d'arrivée du déplacement pour lequel on initialise le réseau
* \param dist_de_marche: permet de spécifier qu'on ne veut pas marcher plus de cette distance avant de prendre le bus à partir du point de départ, ou pour se rendre au point de destination à partir de la sortie du bus.
```

```
* \param dist_transfert: distance maximale de marche pour un transfert de bus
*/
void Gestionnaire::initialiser_reseau(Date date, Heure heure_depart, Heure heure_fin,
Coordonnees depart, Coordonnees dest, double dist_de_marche, double dist_transfert)
```

Aussi, vous devez respecter les spécifications suivantes:

- Le constructeur de la classe doit initialiser tous les conteneurs à partir des fichiers gtfs fournis. Veillez à lire et à traiter un seul fichier gtfs à fois afin d'éviter que votre application utilise beaucoup de mémoire. m_reseau pourrait être un réseau vide à la construction du gestionnaire. Bien qu'une partie du travail aie déjà été effectuée lors du TP1, vous devez être astucieux afin d'améliorer votre temps de chargement des données.
- les méthodes bus_existe, station_existe, date_est_prise_en_charge, getLigne, getStation doivent s'exécuter en temps constant: O(1).

4 - Livrable et critères de correction

Dans ce livrable, vous aurez principalement à faire des choix de structures de données qui auront un grand impact sur la performance de votre application. Aussi, en raison de la quantité de données, vous devez faire attention à ne pas trop dupliquer les données en mémoire.
Les fichiers .h des classes que vous devez implémenter sont fournies en archive avec cet énoncé. Vous devez implémenter toutes les méthodes publiques dont le prototype se trouve dans un fichier .h. Il est strictement interdit de modifier ces prototypes. Afin de supporter les méthodes publiques demandées, vous pouvez ajouter des méthodes privées et d'autres attributs privés au besoin, mais l'interface des classes, telle que spécifiée par la partie publique du fichier .h doit être préservée.
Pour utiliser les nouvelles fonctionnalités que vous allez ajouter, nous vous demandons aussi d'écrire un programme principal qui construit un objet Gestionnaire, et affiche un menu interactif permettant de :

- trouver les stations environnantes à une position GPS fournie par l'utilisateur;
- consulter l'horaire d'un bus à une station donnée et à partir d'une heure d'une journée choisie;
- trouver le plus court chemin pour aller d'un point GPS à un autre;
- afficher si le réseau est connexe ou pas, avec et sans les arcs de transfert.

Un exemple d'utilisation de ce programme est disponible à la fin de ce document. Notez qu'il est de votre responsabilité de valider les entrées de l'utilisateur afin d'éviter que votre programme plante en pleine exécution.

Vous serez évalué sur:
- Efficacité du code : choix judicieux des structures de données, respect des spécifications sur de complexité des opérations demandées;

- Clarté du code : il est important que votre code soit clair, lisible et documenté (sans excès) afin de le corriger au mieux;

- Respect des prototypes fournis et de l'énoncé: il est capital de respecter les prototype afin d'aider la correction.

- Critère de complétion: facteur multiplicatif sur la note globale, par exemple: « tout ce qui est demandé est implémenté » équivaut à 1.0, « la moitié » à 0.5, « rien » à 0.0) (le facteur multiplicatif n'est pas limité à ces trois exemples, il sera calculé.

- Critère de compilation: vous risquez de perdre entre 20% et 100% de la note si votre code ne compile pas (affecte donc le facteur multiplicatif).

- Critère d'exécution: vous risquez de perdre un pourcentage de la note si votre code s'interrompt pendant l'exécution (affecte le facteur multiplicatif). La gravité de l'erreur détermine le pourcentage perdu.

5 - Fichiers à remettre

Dans la boîte de dépôt du portail du cours, vous devez remettre les fichiers des classes du TP1 en plus des classes de ce TP2 et le programme principal. Il s'agit donc des fichiers suivants :

- arret(.cpp et .h) contenant la classe Arret;
- auxiliaires (.cpp et.h) contenant les classe Date et Heure et la fonction lireFichier;
- coordonnees (.cpp et.h) contenant la classe Coordonnees;
- ligne (.cpp et .h) contenant la classe Ligne;
- station (.cpp et .h) contenant la classe Station;
- voyage (.cpp et .h) contenant la classe Voyage;
- reseau (.cpp et .h) contenant la classe Reseau
- gestionnaire (.cpp et .h) contenant la classe Gestionnaire
- main.cpp qui contient le programme principal.

Vous devez insérer ces fichiers dans une archive ayant pour nom NomDeFamille_Prénom.zip. SVP, ne remettre aucun autre fichier. Bien que nous encourageons l'utilisation de Google Test pour vos tests unitaires, on vous demande de ne pas remettre vos testeurs (même si nous vous recommandons de bien tester votre code). Vous devez remettre votre travail dans la boîte de dépôt du portail du cours. Aucune remise par courriel ne sera acceptée. Tout travail remis en retard perdra 2 points par heure de retard. Donc, au bout de 10 heures de retard, la note maximale du TP sera de 80, et après 50 heures de retard, la note maximale sera de 0. Vous pouvez déposer autant de versions que vous le désirez. Seul le dernier dépôt sera corrigé.

Tout comme pour le TP1, ce travail est fait en équipe de 1 ou 2. Cependant, les équipes formées sont celles déjà formées pour le TP1. Si votre équipier a abandonné le cours et que vous désirez former une nouvelle équipe (de 2) pour ce TP2, SVP en informer le professeur par courriel. Ce dernier fera le nécessaire pour vous regrouper en équipe de 2 (seulement si il y a eu abandon d'un coéquipier).

ATTENTION : vous avez la responsabilité de vérifier l'intégrité des fichiers que vous avez remis dans la boîte de dépôt. Donc, nous vous recommandons fortement que vous examiniez ce que vous avez remis afin d'en vérifier l'intégrité. Le mécanisme de téléversement du portail ne corrompt pas les fichiers. Cependant, il peut arriver que votre archive ait été corrompue si, lorsque vous l'avez créée, certains des fichiers étaient ouverts par d'autres applications (Eclipse par exemple...). Vérifiez donc l'intégrité de votre archive après l'avoir construite. SVP : ne pas déposer tout votre «workspace» !!

6 - Plagiat

Tel que décrit dans le plan de cours, le plagiat est interdit. Une politique stricte de tolérance zéro est appliquée en tout temps et sous toutes circonstances. Tous les cas seront référés à la direction de la Faculté.

7 - Exemple d’exécution du programme principal

```
Bienvenue dans l'application RTC
Chargement des données terminé en 11.5603 secondes
Menu
1 - Stations à proximité
2 - Consulter horaire du bus
3 - Itinéraire
4 - Stats de connectivité
Sélectionner une option en indiquant un chiffre ou autre chose pour quitter:1
Entrez vos coordonnées GPS
Latitude: 46.778808
Longitude: -71.270014
Entrez un rayon (en Km) pour la recherche:0.5
À une distance de 0.0669453km:
1561 - de l'Université / des S.-Humaines
À une distance de 0.0837592km:
1515 - de l'Université / des S.-Humaines
À une distance de 0.200121km:
1783 - Laurier / du Parc-De Lotbinière
À une distance de 0.209594km:
1559 - de l'Université
À une distance de 0.236281km:
1831 - Laurier / du Parc-De Lotbinière
À une distance de 0.246752km:
1999 - Laurier - Université-Laval
À une distance de 0.263352km:
1830 - Laurier / J.-Vézina
À une distance de 0.338869km:
1501 - du Séminaire / des Bibliothèques
À une distance de 0.377467km:
7010 - de la Médecine / de l'Université
À une distance de 0.386508km:
1542 - du Séminaire / des Bibliothèques
À une distance de 0.388108km:
1558 - de la Médecine
À une distance de 0.403464km:
1832 - Laurier / R.-Lévesque Ouest
À une distance de 0.426433km:
1782 - Laurier / R.-Lévesque Ouest
À une distance de 0.45718km:
7011 - de la Médecine / de l'Agriculture
À une distance de 0.47401km:
1557 - de la Médecine
Menu
1 - Stations à proximité
2 - Consulter horaire du bus
```

```
3 - Itinéraire
4 - Stats de connectivité
Sélectionner une option en indiquant un chiffre ou autre chose pour quitter:2
Entrez la date qui vous intéresse!
annee [défaut=2016] :
mois [défaut=10] :
jour [défaut=5] :
Entrez l'heure de début de l'horaire!
heure [défaut=20] :20
minutes [défaut=20] :0
secondes [défaut=44] :0
Entrez le numéro du bus: 800
Entrez le numéro de la station: 1515
METRO_BUS 800 : Beauport - Pointe-de-Sainte-Foy
1515 - de l'Université / des S.-Humaines

20:03:00
20:19:00
20:33:00
20:48:00
21:03:00
21:17:00
21:33:00
21:48:00
22:02:00
22:16:00
22:31:00
22:46:00
23:01:00
23:16:00
23:31:00
23:46:00
24:01:00
24:16:00
24:31:00
24:46:00
25:01:00
25:16:00
Menu
1 - Stations à proximité
2 - Consulter horaire du bus
3 - Itinéraire
4 - Stats de connectivité
Sélectionner une option en indiquant un chiffre ou autre chose pour quitter:3
Choisir  votre point de départ
Carnets d'adresse
1 - 3475 avenue maricourt, Québec: (46.760074, -71.319867)
2 - 2325 vie étudiante, Québec: (46.778398, -71.26853)
```

```
3 - Cineplex odéon sainte-foy: (46.785923, -71.354046)
4 - Pavillon pouliot: (46.776635, -71.270671)
5 - 2476, avenue de lisieux, québec: (46.857245, -71.206804)
6 - Pavillon desjardin: (46.778808, -71.270014)
Sélectionner une adresse en indiquant un chiffre:6
Choisir  votre point de d'arrivé
Carnets d'adresse
1 - 3475 avenue maricourt, Québec: (46.760074, -71.319867)
2 - 2325 vie étudiante, Québec: (46.778398, -71.26853)
3 - Cineplex odéon sainte-foy: (46.785923, -71.354046)
4 - Pavillon pouliot: (46.776635, -71.270671)
5 - 2476, avenue de lisieux, québec: (46.857245, -71.206804)
6 - Pavillon desjardin: (46.778808, -71.270014)
Sélectionner une adresse en indiquant un chiffre:1
Entrez la date qui vous intéresse!
annee [défaut=2016] :
mois [défaut=10] :
jour [défaut=5] :
Entrez l'heure de départ!
heure [défaut=20] :20
minutes [défaut=22] :0
secondes [défaut=29] :0
Initialisation du réseau ....
Recherche du plus court chemin
0 -
1999 - Laurier - Université-Laval
1787 - Laurier / R.-Bourassa
1776 - Laurier / B.-Morin
1790 - Laurier - Laurier Québec
1791 - Laurier / de G.-des-Prés
1793 - Laurier / de l'Église
1794 - Laurier / Lavigerie
1795 - Lavigerie / Hochelaga
1796 - Hochelaga / Lavigerie
1797 - Hochelaga / G.-Dumont
1798 - Hochelaga / Duchesneau
1799 - Neilson / de Namur
1800 - Neilson
1801 - Neilson / de Liège
1802 - Neilson / Pie-XII
1804 - Pie-XII /  Neilson
1803 - Pie-XII / au 991
1 -
Menu
1 - Stations à proximité
2 - Consulter horaire du bus
3 - Itinéraire
```

```
4 - Stats de connectivité
Sélectionner une option en indiquant un chiffre ou autre chose pour quitter:4
Entrez la date qui vous intéresse!
annee [défaut=2016] :
mois [défaut=10] :
jour [défaut=5] :
Entrez l'heure de début qui vous intéresse!
heure [défaut=20] :21
minutes [défaut=29] :0
secondes [défaut=57] :0
Avec les arêtes de transfert, le réseau n'est pas fortement connexe.
Sans les arêtes de transfert, le réseau n'est pas fortement connexe.
```

Bon travail