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 #1

Mise en œuvres des classes de base 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 première partie d'un travail de session. Le travail de session consiste à produire un logiciel utilisateur pour le réseau de transport en commun dans la ville de Québec. Le but du logiciel est de permettre à un utilisateur quelconque de se renseigner facilement sur le réseau à des fins de déplacements. Vous devez utiliser les notions que nous avons vues en classe pour implémenter les structures de données et les algorithmes appropriés qui permettront d'effectuer **efficacement** les opérations choisies par les utilisateurs. Le défi réside donc principalement dans les choix que vous ferez pour les structures de données et les algorithmes. Pour compléter ce travail de session, vous aurez à faire trois remises dont la dernière remise inclura un rapport final.

2 Données ouvertes de la RTC

Nous proposons d'utiliser les données ouvertes du Réseau de Transport de la Capitale (RTC), qui répertorient, entre autres, l'emplacement des stations de bus dans la ville et les informations sur les lignes de bus (comme les horaires et les arrêts). Ces données sont disponibles dans un format compressé (zip) à l'adresse suivante: http://www.rtcquebec.ca/Portals/0/Admin/DonneesOuvertes/googletransit.zip Une fois le dossier décompressé, vous verrez que celui-ci contient plusieurs fichiers textes CSV (Comma Separated Values) au format CSV ayant les caractéristiques suivantes:

- le séparateur de colonnes est la virgule (,);
- les données de certaines colonnes sont entre guillemets;
- la première ligne précise les titres des différentes colonnes.

La liste des fichiers dans l'archive est la suivante: agency.txt, calendar_dates.txt, feed_info.txt, Horaire_Boucle_Partage.txt, __licence.txt, __lisez-moi.txt, routes.txt, shapes.txt, stops.txt, stop_times.txt, transfers.txt, trips.txt.

Notez que tous les fichiers adèrent au format GTFS (https://fr.wikipedia.org/wiki/General_Transit_Feed_Specification), un format standardisé pour communiquer des horaires de transports en commun et les informations géographiques associées. Ainsi, outre le RTC, la Société de Transport de Montréal (STM) utilise ce même format pour rendre publique ses horaires d'autobus et de métro. Ainsi, une application basée sur ce format pourra être utilisée pour traiter différents réseaux de transport en commun.

Dans le but d'uniformiser la correction, nous vous fournissons avec cet énoncé la version la plus récente des horaires publiés du réseau de transport. Il va falloir donc que vous utilisez exclusivement le dossier RTC fourni avec cet énoncé afin d'éviter de perdre des points.

3 Le livrable

Dans ce livrable, vous aurez principalement à vous mettre dans le contexte du TP en implémentant des classes en rapport avec les données à traiter. Vous n'aurez pas vraiment besoin de notions particulières sur les structures de données pour ce premier livrable.

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 aux besoin, mais l'interface des classes, telle que spépécifiée par la partie publique du fichier .h doit être préservée.

Vous devrez donc implémenter les classes suivantes:

- **Coordonnees**: utilisée pour représenter les coordonnées GPS d'un endroit. Principlament, cette classe est munie d'une méthode statique is_valid_coord permettant de valider si une paire potentielle (lat, long) forme une coordonnée GPS valide (voir https://fr.wikipedia.org/wiki/Coordonn%C3%A9es_g%C3%A9ographiques). Aussi, elle est munie d'un opérateur (-) de calcul de la distance entre deux coordonnées GPS. Les détails du calcul de la distance entre deux GPS peuvent être trouvés à l'adresse https://fr.wikipedia.org/wiki/Distance_du_grand_cercle. Notez que le constructeur ne construit un objet de la classe que si les données en paramètre sont valides.

- **Station**: Une station est un emplacement physique où un bus effectue des arrêts. Un objet station est construit à partir d'un vecteur de strings représentant une ligne du fichier stops.txt, qui est composé des colonnes suivantes :
    - stop_id : identifiant unique d'une station ;
    - stop_name : contient le nom de la station ;
    - stop_desc : contient la description de la station ;
    - stop_lat : contient la latitude de la station au format wgs84 (voir https://fr.wikipedia.org/wiki/WGS_84.);
    - stop_lon : contient la longitude de la station au format wgs84 ;
    - stop_url : contient l'URL d'une page décrivant une station en particulier ;
    - location_type : différencie les stations entre-elles ; si ce champ est vide, ce qui est le cas dans les données publiées par le RTC, c'est une station par défaut ;
    - wheelchair_boarding : identifie si l'embarquement de fauteuil roulant est possible à cette station ; une valeur
        - 0 (ou vide) indique que cette information n'est pas disponible pour cette station ;
        - 1 indique qu'un véhicule passant par cette station peut accueillir au moins un fauteuil roulant ;
        - 2 indique qu'aucun fauteuil roulant ne peut être pris en charge à cette station.

  Dans le cadre de ce travail, nous n'utilisons qu'une partie de ces données ; plus précisément stop_id (m_id), stop_name(m_nom), stop_desc(m_description), stop_lat et stop_long(m_coords). Notez que l'attribut m_voyages_passants est initialisé à un vecteur vide dans le constructeur et pourrait être rempli grâce à la méthode addVoyage. Aussi, cette classe doit permettre de calculer la distance entre deux stations, puis de trouver les lignes qui passent par cette station lorsque les voyages passants sont enregistrés.

- **Ligne**: Cette classe représente une ligne d'autobus du réseau de transport contenue dans le fichier routes.txt. Un objet Ligne est construit à partir d'un vecteur de strings représentant une ligne du fichier routes.txt. Pour votre information, le fichier routes.txt est composé des champs suivants :
    - route_id : identifiant unique de la ligne d'autobus ;
    - agency_id : identifiant unique de l'agence (RTC) ;

- route_short_name : texte permettant d'identifier un parcours sur les horaires et panneaux de signalisation ; typiquement, il s'agit du numéro de la ligne ("7", "800", "801", "13A", "13B", etc.) ;
- route_long_name : texte permettant d'identifier, avec plus de précisions, une ligne sur les horaires et panneaux de signalisation (ce champ est vide dans le fichier fourni par le RTC) ;
- route_desc : texte décrivant la ligne ; le RTC a choisi de préciser les destinations desservies ;
- route_type : identifie le type de véhicule de transport 4 ; la valeur est fixée à 3 pour indiquer qu'il s'agit d'autobus ;
- route_url : comprend une URL qui mène vers une page décrivant la ligne en question ;
- route_color : couleur permettant d'identifier visuellement la ligne ;
- route_text_color : couleur permettant d'identifier visuellement un texte décrivant la ligne.

Nous n'allons qu'utiliser: route_id (m_id), route_short_name (m_numero), route_desc(m_description), route_color(m_categorie de type CategorieBus: voir ci-dessous). Notez que, l'attribut m_voyages est initialisé à un conteneur vide dans le constructeur et pourra être rempli grâce à la méthode addVoyage qui ajoute un voyage à ce conteneur. Aussi dans le fichier routes.txt, il peut y avoir plus d'une entrée pour chaque ligne d'autobus. Cela est dû au fait qu'un dossier GTFS puisse chevaucher plusieurs saisons d'une année (des saisons différentes donnent des ids différents à une même ligne). Cette remarque ne présente aucune implication pour le livrable 1, donc ne vous préoccupez pas de cela, nous y reviendrons pour les autres livrables.

Pour finir, notez que lorsque l'attribut m_voyage est initialisé, la méthode getDestinations permettra de trouver les destinations des voyages de la ligne de bus. Toutefois, une ligne ne peut avoir plus de deux destinations possibles: une pour l'aller et l'autre pour le retour. Dans le cas de certains bus comme les couches tard, une seule destination est possible auquel cas le dernier élément de la paire retournée par la fonction est une chaîne de caractères vide.

- **CategorieBus**: il s'agit d'une énumération permettant de différencier les différents type de bus. Vous devez gérer la correspondance entre cette énumération et le champ route_color dans le fichier routes.txt:
  - - "97BF0D" (Verte)--> Métro bus
  - - "013888" (Bleue) --> Le bus
  - - "E04503" (Orange) --> Express
  - - "1A171B" (Noir) ou "003888" (Bleue) --> Couche tard

- **Voyage**: permet de décrire un voyage d'une ligne, i.e un déplacement entre deux stations terminales. Pour construire un objet voyage, vous aurez besoin de d'un vecteur de strings représentant une ligne du fichier trips.txt et d'un pointeur vers la ligne qui effectue ce voyage. Pour votre information, le fichier trips.txt est composé des champs :
  - route_id : identifiant unique d'une ligne ;
  - service_id : identifiant unique d'un ensemble de dates durant laquelle la RTC a un horaire de service défini ;
  - trip_id : identifiant unique d'un voyage ;
  - trip_headsign : indique aux passagers la destination du voyage ;
  - trip_short_name : texte permettant d'identifier un trajet sur les horaires et panneaux de signalisation ;
  - direction_id : valeur binaire indiquant le sens du voyage ;

- block_id : champ identifiant le bloc auquel appartient le voyage ; un bloc est constitué de deux ou plusieurs voyages successifs effectués avec le même véhicule ;
- shape_id : contient l'identifiant de la forme de l'itinéraire ;
- wheelchair_accessible : identifie si l'embarquement de fauteuil roulant est possible à une certaine station du voyage.

Dans le cadre de ce travail, nous n'utilisons qu'une partie de ces données ; plus précisément trip_id(m_id), trip_headsign(m_destination), service_id(m_service_id). Notez que l'attribut m_arrets n'est pas initialisé dans le constructeur, il pourra l'être par le biais de la méthode setArrets. Celle-ci doit stocker dans l'attribut m_arrets, les arrêts en ordre de numéro de séquence. En raison, de la structure des données de la rtc, si deux arrêts successifs se font dans la même minute, cette méthode doit ajouter 30 secondes à l'heure de départ et à l'heure d'arrivée du dernier arrêt parmi les deux. Cette classe est aussi composée des méthodes getHeureDepart et getHeureFin qui donnent respectivement l'heure de départ et de fin du voyage: il est évident que cela n'est possible que si les arrêts du voyage sont enregistrés. Aussi, la classe est munie des opérateurs de comparaison (< et >) servant à savoir si un voyage commence avant ou après un autre.

- **Arret**: Un arrêt est une composante d'un voyage, c'est une opération spatio-temporelle qui s'effectue lors d'un voyage d'une ligne donnée (ex: la ligne 800 fait, lors du voyage xxxxx, un arrêt à la station du desjardin à 11h32). Il est important de ne pas confondre la station et l'arrêt. Un objet arrêt est construit à partir d'un vecteur de strings représentant une ligne du fichier stop_times.txt. Pour votre information le fichier stop_times.txt comprend des données relatives aux arrêts effectués par les autobus ; il est composé des champs :
  - trip_id : identifiant du voyage ;
  - arrival_time : spécifie l'heure d'arrivée de l'autobus pour cet arrêt (d'un voyage en particulier) ;
  - departure_time : spécifie l'heure de départ de l'autobus pour cet arrêt (d'un voyage en particulier) ;
  - stop_id : identifiant unique d'une station ;
  - stop_sequence : identifie le numéro de séquence de cet arrêt dans l'ensemble des arrêts effectués pour un voyage en particulier (il permet ultimement de reconstituer l'ensemble des arrêts effectués pour un voyage dans le bon ordre) ;
  - pickup_type : indique si les passagers peuvent embarquer à l'arrêt en question ou si seules les descentes sont permises ;
  - drop_off_type : indique si les passagers sont déposés à l'arrêt selon l'horaire prévu ou que le débarquement n'est pas disponible.

  Nous n'aurons besoin que de arrival_time (m_heure_arrivee), departure_time (m_heure_depart), stop_id (m_station_id), et stop_sequence (m_numero_sequence). Cette classe devra contenir les opérateurs (< et >) qui permettent de comparer deux arrêts du même voyage sur la base des numéros de séquences.

- **Heure**: représente l'heure d'une journée, mais pour les besoins du TP nous permettront qu'elle puisse encoder un nombre d'heures supérieurs à 24 mais inférieure à 30 (i.e 26:02:00 est une heure valide). Cela est dû au fait qu'un service du réseau de transport qui a commencé dans une journée peut se terminer plus tard que minuit de cette journée mais surement avant le prochain service de la journée suivante qui commence à 6h. Le constructeur par défaut de la classe doit instancier un objet avec l'heure actuelle alors que celui avec des paramètres doit dépendre

de ces paramètres. Cette classe doit se munir des opérations de comparaisons et aussi d'un opérateur (-) qui retourne la différence entre deux heures en nombre de secondes.

- **Date**: représente une date. On doit pouvoir créer la date courante avec le constructeur par défaut ou une date quelconque avec le constructeur avec paramètres. Vous devez vous assurer de la validité des attributs de la classe en tout temps. Aussi, la classe doit être munie des opérateurs de comparaison. Le fichier calendar_dates.txt contient des données relatives aux dates où les bus sont opérationnels. Pour votre information, il est composé des champs :
    - service_id : identifiant unique d'un ensemble de voyages effectuées par diverses lignes selon un horaire de service défini. En gros, vu que les voyages du dimanche ont un horaire différent de ceux en semaine, il peut y avoir un service_id pour les voyages en semaine et un autre pour les voyages du dimanche ;
    - date : spécifie une date durant laquelle un service est offert ;
    - exception_date : indique si le service est disponible à la date spécifiée.

  Notez qu'un service prédéfini (service_id) peut être donné à plusieurs date différentes et une date peut être associée à plusieurs services prédéfinis. Ceci étant dit, étant donnée une date, ce sont les service_ids qui lui sont associés qui permettent de trouver les voyages à cette date car chaque voyage est effectué dans le cadre d'un horaire de service prédéfini (service_id).

L'interface de ces deux dernières classes se trouve dans auxiliaires.h

En plus de coder les classes ci-dessus, vous devez également coder une fonction capable de lire n'importe quel des fichiers dans le dossier de données du réseau de transport. Nous vous proposons le prototype suivant pour cette fonction qui se trouve dans auxiliaires.h:

```
/*!
* \brief Permet de lire un fichier au format gtfs
* \param[in] nomFichier: chemin d'accès au fichier qui est supposé contenir plusieurs
lignes et plusieurs colonnes
* \param[out] résultats: vecteur 2D destiné à contenir le fichier, l'élément [i][j] représente la
ième ligne et la jème colonne du fichier
* \param[in] délimiteur: le caractère délimiteur des colonnes dans ce fichier.
* \param[in] rm_entete:  un booléen qui spécifie s'il faut supprimer ou pas la première ligne
du fichier.
* \pre Le fichier existe
* \exception logic_error s'il y a un problème lors de l'ouverture du fichier.
*/
void lireFichier(std::string nomFichier, std::vector<std::vector<std::string>>& resultats,
char delimiteur, bool rm_entete);
```

4 Travail à faire

Vous devez implémenter les classes qui sont décrites dans cet énoncé ainsi que la fonction lireFichier, en tenant compte des remarques et explications de la section précédente. Les interfaces de classes

fournies ne doivent pas être modifiées (aucune exception ne sera tolérée). Toutefois, l'interface n'incluant que la partie publique d'une classe, vous pouvez ajouter selon vos besoins des éléments privés dans vos classes. Nous vous demandons aussi d'écrire un programme principal qui charge toutes les données des fichiers (du dossier RTC que nous vous fournissons) utilisées pour:

- construire des objets Lignes, Stations, Voyages, Arrets;
- afficher le temps requis pour charger les données et construire les objets
- écrire dans un fichier texte les lignes du réseau (en ordre de leur numéros), les stations (en ordre de leur numéros), et l'ensemble des voyages de la journée actuelle qui commenceront à partir de l'heure actuelle et qui se termineront dans au plus une heure(en ordre de l'heure de départ). Un exemple du résultat attendu est le suivant:

```
Chargement des données terminé en 65.5701 secondes
======================
 LIGNES DE LA RTC
 COMPTE = 295
======================
LEBUS 1 : Station Belvédère - Gare fluviale / Cap-Blanc
LEBUS 1 : Station Belvédère - Gare fluviale / Cap-Blanc
LEBUS 10 : Marie-de-l'Incarnation - Cégep Garneau
LEBUS 107 : Gare du Palais - Pointe-de-Sainte-Foy
LEBUS 107 : Gare du Palais - Pointe-de-Sainte-Foy
LEBUS 11 : Place D'Youville / Vieux-Québec - Pointe-de-Sainte-Foy
LEBUS 11 : Place D'Youville / Vieux-Québec - Pointe-de-Sainte-Foy
LEBUS 111 : Place D'Youville - Pointe-de-Sainte-Foy
LEBUS 111 : Place D'Youville - Pointe-de-Sainte-Foy
…
=======================
 STATIONS DE LA RTC
 COMPTE = 4604
=======================
1003 - T. D'Youville
1005 - Champlain/1005
1006 - Champlain/1006
1007 - Champlain/1007
1008 - Champlain/1008
1009 - Champlain/1009
1010 - Champlain/1010
1011 - T. D'Youville
1012 - La Croix-Rouge
1013 - Champlain/1013
1014 - Champlain/1014
1015 - Petit-Champlain
1016 - Gare fluviale
1017 - M. Civilisation
1019 - Dalhousie
1020 - des Navigateurs
```

1021 - St-Thomas
1022 - Gare-Palais/1022
1024 - J.-Lesage
1025 - St-Dominique
1026 - du Pont
1027 - R.-Lévesque
1028 - de la Couronne
1029 - Royale/1029
1032 - Dorchester
1034 - Caron
1035 - Langelier
1036 - Laviolette
1037 - Grande Allée
….

==========
VOYAGES DE LA JOURNÉE DU 2016-09-06
22:47:52 - 23:47:52
COMPTE = 57
==========
28: Vers Colline Parlementaire (Sud)
22:49:00 - 2157
22:49:30 - 2149
22:50:00 - 2150
22:50:30 - 3189
22:51:00 - 3232
22:51:30 - 1937
22:52:00 - 1938
22:52:30 - 1507
22:53:00 - 1508
22:53:30 - 1509
22:54:00 - 1510
22:54:30 - 1511
22:55:00 - 1512
22:55:30 - 1513
22:55:00 - 1514
22:56:00 - 4895
22:56:30 - 4896
22:56:00 - 4897
22:57:00 - 5176
22:58:00 - 1336
22:58:30 - 2955
22:59:00 - 1254
23:00:00 - 1257
23:01:00 - 1260
23:02:00 - 1262
23:03:00 - 1263

23:05:00 - 1265
23:07:00 - 1190
23:08:00 - 1517
23:11:00 - 2116
23:12:00 - 1582
87: Vers Loretteville (Nord)
22:49:00 - 1940
22:50:00 - 1941
22:51:00 - 1824
22:51:30 - 1826
22:52:00 - 1828
22:54:00 - 1561
22:55:00 - 1501
22:56:00 - 1502
22:56:30 - 1504
22:59:00 - 1550
23:00:00 - 7005
23:01:00 - 7007
23:03:00 - 7008
23:07:00 - 4350
23:08:00 - 4351
23:09:00 - 4352
23:09:30 - 4353
23:10:00 - 4354
23:11:00 - 4355
23:12:00 - 4356
23:13:00 - 4357
23:14:00 - 4358
23:14:30 - 1203
23:15:00 - 1208
23:17:00 - 1210
23:18:00 - 5727
23:23:00 - 5037
23:23:30 - 5063
23:23:00 - 5299
23:24:00 - 5300
23:25:00 - 5302
23:25:30 - 5072
23:25:00 - 5284
23:26:00 - 5073
23:26:30 - 5075
23:27:00 - 5076
23:27:30 - 3900
23:27:00 - 5077
23:28:00 - 5078
23:28:30 - 5079
23:29:00 - 5080

23:29:30 - 5081
23:30:00 - 4304
23:31:00 - 4295
23:32:00 - 5082
23:33:00 - 3652
23:33:30 - 3840
23:34:00 - 4275
23:35:00 - 4148
803: Vers Terminus Beauport (Est)
22:50:00 - 5726
22:52:00 - 4433
22:52:30 - 1225
22:52:00 - 1253
22:53:00 - 4630
22:54:00 - 1827
22:55:00 - 5264
22:55:30 - 5298
22:56:00 - 5044
22:58:00 - 5419
22:59:00 - 5555
23:02:00 - 2539
23:06:00 - 5554
23:06:30 - 2283
23:07:00 - 5744
23:08:00 - 4875
23:08:30 - 1247
23:09:00 - 1245
23:09:30 - 4901
23:10:00 - 4052
23:11:00 - 4053
23:11:30 - 2889
23:12:00 - 3197
23:12:30 - 3196
23:13:00 - 2976
23:15:00 - 2515
23:16:00 - 3174
23:17:00 - 3175
23:18:00 - 3178
23:19:00 - 3179
23:19:30 - 5033
23:20:00 - 5035
23:22:00 - 1306
23:23:00 - 1308
23:23:30 - 5036
23:25:00 - 3632
23:27:00 - 3328
...

5 Fichiers à remettre

Dans la boîte de dépôt du site Web/ENA du cours, vous devez remettre uniquement les 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;
- 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. Vous devez remettre votre travail dans la boîte de dépôt du site ENA 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 remettre autant de versions que vous le désirez. Seul le dernier dépôt sera corrigé.

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 de l'ENA 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 remettre 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.