IFT-2008 – Algorithmes et structures de données
Hiver 2020

Travail pratique 2

À remettre, par le portail du cours avant 23h50 le lundi 23 mars 2020

1 Mise en contexte

Nous vous demandons dans ce deuxième travail pratique d'implanter **un réseau d'une compagnie aérienne** en utilisant les notions que nous avons vues dans le cours concernant le type abstrait « Graphe ». En effet, le réseau aérien est un graphe (orienté, pondéré et étiqueté) constitué d'un ensemble de villes (sommets du graphe) et d'un ensemble de trajets d'avions reliant ces villes (arcs du graphe). Nous supposons que chacun des trajets du réseau aérien possède ces trois différents types de pondérations :

- Une estimation de la durée du vol en heures (exemple: 0.8 heure entre Québec et Montréal. C'est l'équivalent de 60*0.8 = 48 minutes).
- Une estimation du coût du trajet en dollars (exemple: 220.46 $ pour un vol entre Québec et Montréal).
- Un niveau de sécurité représentant un indice qui peut être négatif. L'augmentation de cet indice impliquerait des mesures supplémentaires pour assurer la sécurité du vol (exemple: les bagages des voyageurs pour un trajet qui a un niveau de sécurité égale à 8 doivent être tous fouillés. Par contre, seulement les bagages de quelques voyageurs pour un trajet qui a un niveau de sécurité égale à -2 sont fouillés). De ce fait et dans notre cas, les clients cherchent à minimiser le niveau de sécurité, car ils n'aimeraient pas passer beaucoup de temps à la douane à être fouillés. De plus, un niveau de sécurité élevé veut dire que le vol en question est potentiellement dangereux.

Nous voudrons donc modéliser un réseau afin qu'une compagnie aérienne puisse offrir un meilleur service à ses clients qui peuvent être soit des particuliers soit des agences de voyages. Ces clients aimeraient avoir des informations pertinentes sur le ou les trajets qu'ils pensent réserver. Nous vous demandons donc de leur offrir un système leur permettant de répondre à leurs questions sous forme de requêtes. Voici quelques exemples de requêtes que les clients peuvent demander à votre système :

Existe-t-il un trajet entre deux villes ?
Quel est le meilleur trajet entre deux villes qui nous donne la plus faible durée du vol ?
Quel est le meilleur trajet entre deux villes qui nous donne le plus faible coût ?
Quel est le meilleur trajet entre deux villes qui nous donne le meilleur niveau de sécurité ?

2 Modélisation du réseau aérien

La structure interne choisie pour implanter le type ReseauAerien (voir « ReseauAerien.h ») est principalement composée d'un type Graphe orienté dont la définition se trouve dans le fichier « Graphe.h ». La structure interne et le modèle d'implantation de ces 2 types ReseauAerien et Graphe, ne doivent être en aucun cas modifiés. Par contre, vous pouvez ajouter dans les classes définissant ces 2 types :

- D'autres méthodes publiques en plus de celles demandées.
- Des méthodes privées (on vous y encourage).
- Inclusion de nouvelles librairies de conteneurs de la STL.
- Des définitions de constantes utiles pour l'implémentation de ce TP.

Sommairement, le graphe est implémenté en utilisant un vecteur de listes d'adjacence. On trouve dans le nœud typique de chacune de ces listes une structure *Arc* contenant les trois pondérations mentionnées auparavant (durée, coût et niveau de sécurité) ainsi que le numéro du sommet destination. Nous utilisons également un vecteur de chaînes de caractères contenant les noms des sommets qui sont numérotés de 0 à n-1. Vous n'avez pas à appliquer la théorie du contrat pour ce TP.

3 Travail à faire

3.1 Interface du type Graphe

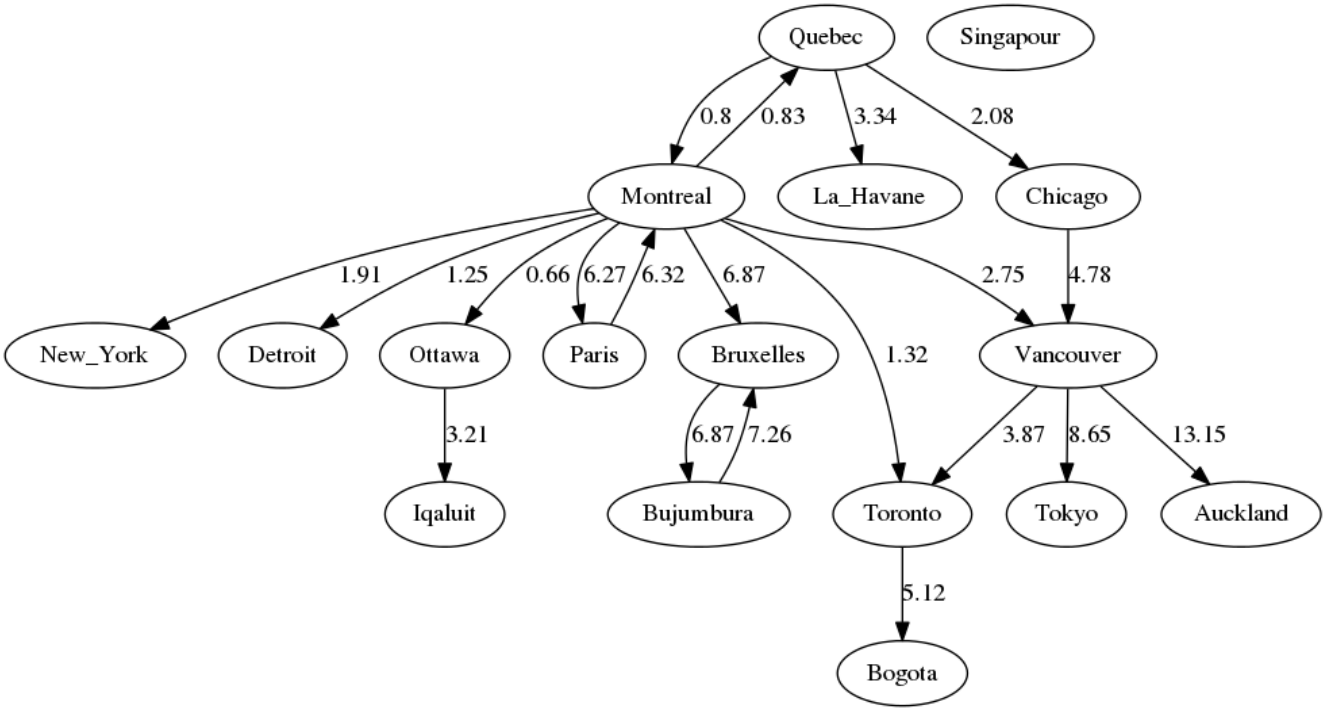
En tenant compte du modèle d'implantation proposé dans la définition de la classe Graphe, vous devez commencer le travail par écrire l'ensemble de méthodes sur le type Graphe dont la description se trouve dans le fichier « Graphe.h **»**. Notez que l'implémentation de cette interface doit être mise dans le fichier « Graphe.cpp » déjà fourni. Vous devez tester intensivement les opérateurs sur le type Graphe. Nous vous fournissons d'ailleurs la surcharge de l'opérateur << sur le type Graphe (*inline*) afin de visualiser vos graphes une fois mis en mémoire.

3.2 Interface du type ReseauAerien

Il s'agit maintenant d'implémenter dans le fichier « ReseauAerien.cpp » l'interface du type ReseauAerien dont les méthodes sont décrites dans le fichier « ReseauAerien.h ». Vous devez absolument coder les méthodes demandées de haut niveau par rapport au type Graphe, c'est-à-dire utiliser uniquement les opérations publiques décrites dans « Graphe.h ». En aucun cas, vous ne devez le faire en utilisant les détails internes du type Graphe. De plus et afin de vous aider, nous vous fournissons la méthode chargerReseau() dans le fichier « ReseauAerien.cpp ». Cette méthode charge le réseau en utilisant un fichier texte qui a le format suivant :

Reseau Aerien: Nom du reseau
Nombre de villes
Liste des villes:
Ville A
Ville B
.
.
.
Ville N
Liste des trajets:
Ville A
Ville B
Duree Cout Niveau
Ville B
Ville N
Duree Cout Niveau
.
.
.
Ville B
Ville A
Duree Cout Niveau

Nous vous fournissons d'ailleurs un exemple de réseau aérien se trouvant dans le fichier texte « ReseauAerien.txt ». Nous vous conseillons par contre de l'améliorer en ajoutant d'autres villes et d'autres trajets. Vous pouvez utiliser par exemple ce lien pour connaître les trajets pratiqués par Air Canada : https://ac.fltmaps.com/fr. Sinon et afin d'avoir une représentation graphique de votre réseau dans le but de faire une validation visuelle, nous vous conseillons d'utiliser l'outil Graphvis (http://www.graphviz.org/), mais ce n'est pas obligatoire. Voici un exemple de graphique que cet outil peut produire en se basant sur les données du fichier « ReseauAerien.txt »:



Il est à noter que le nœud indépendant « Singapour » veut tout simplement dire que la compagnie aérienne a ajouté cette ville dans son système, mais elle n'a pas encore planifié des vols pour cette ville.

3.3 Algorithmes de recherche de chemins

Il s'agit maintenant d'implémenter les algorithmes de recherche de chemins à origine unique.

3.3.1 Algorithme de Dijkstra

Nous vous demandons d'implémenter l'algorithme de Dijkstra afin de retrouver le plus court chemin entre deux villes en utilisant soit la durée du vol en heures soit le coût du vol en dollars comme pondération sur les trajets. Voici le prototype de la fonction :

```
Chemin rechercheCheminDijkstra(const std::string &origine,
                               const std::string &destination, bool dureeCout) const;
```

Le départ et l'arrivée doivent bien sûr faire partie du réseau aérien. Sinon, il faut lancer une exception *logic_error*. Cette fonction retourne le chemin trouvé dans une structure *Chemin* contenant entre autres la liste des villes représentant ce chemin. De plus, cette fonction a le booléen *dureeCout* comme paramètre d'entrée et qui vaut *true* s'il faut utiliser la durée du vol comme pondération sur les trajets. Dans ce cas, il faut mettre la durée totale du plus court chemin dans la variable *dureeTotale* et *true* dans le booléen *reussi* de la structure *Chemin*. D'ailleurs et afin de simplifier le travail demandé, vous pouvez supposer qu'un voyageur n'a pas besoin d'attendre (temps d'escale) dans chacune de ses correspondances. En outre, le booléen *dureeCout* vaut *false* s'il faut utiliser le coût du vol comme

pondération sur les trajets. Dans ce cas, il faut mettre le coût total du plus court chemin dans la variable *coutTotal* et *true* dans le booléen *reussi* de la structure *Chemin*. S'il n'existe pas de solution (ie. pas de chemin entre les deux villes), il faut mettre la valeur *false* dans le booléen *reussi* de la structure *Chemin.*

3.3.2 Algorithme de Bellman-Ford

Nous vous demandons d'implémenter l'algorithme de Bellman-Ford afin de retrouver le plus court chemin entre deux villes en utilisant les trois différents types de pondérations sur les trajets. Voici le prototype de la fonction :

```
Chemin rechercheCheminBellManFord(const std::string &origine,
                                  const std::string &destination, int dureeCoutNiveau) const;
```

Le départ et l'arrivée doivent bien sûr faire partie du réseau aérien. Sinon, il faut lancer une exception *logic_error*. Cette fonction retourne le chemin trouvé dans une structure *Chemin* contenant entre autres la liste des villes représentant ce chemin. De plus, cette fonction a l'entier *dureeCoutNiveau* comme paramètre d'entrée et qui vaut 1 s'il faut utiliser la durée du vol comme pondération, 2 s'il faut utiliser le coût du vol comme pondération et 3 s'il faut utiliser le niveau de sécurité comme pondération (il faut lancer une exception *logic_error* si *dureeCoutNiveau* est différent de ces valeurs).

Cet algorithme fait donc la même chose que Dijkstra pour le cas de la durée du vol et le coût du vol. Cependant, il est capable également de donner éventuellement un résultat pour le niveau de sécurité. Dans ce cas et s'il y a une solution, il faut mettre le niveau de sécurité totale du plus court chemin dans la variable *nsTotal* et *true* dans le booléen *reussi* de la structure *Chemin*. S'il n'existe pas de solution (ie. présence d'un circuit de poids négatif), il faut mettre *false* dans le booléen *reussi* de la structure *Chemin*.

3.4 Fichier Principal.cpp

Nous vous fournissons également un fichier Principal.cpp comprenant un main() qui vous permettra de vous aider à tester votre réseau. Voici un exemple de ce que le fichier principal.cpp est censé donner comme résultat dans la console de la VM :

```
Bienvenue dans votre reseau aerien !
------------------ Menu -------------------------------
1 - Charger le reseau à partir d'un fichier texte.
2 - Trouver le plus court chemin avec Dijkstra.
3 - Trouver le plus court chemin avec Bellman-Ford.
4 - Mesurer le temps d'exécution de Dijkstra.
5 - Mesurer le temps d'exécution de Bellman-Ford.
0 - Quitter.
-------------------------------------------------------
```

Entrer s.v.p. votre choix (0 a 5):? 1
Le dictionnaire a été chargé !
Affichage du dictionnaire:
Le graphe contient 17 sommet(s) et 20 arc(s)

Sommet: Quebec
Voisin(s): Montreal, La_Havane, Chicago
Sommet: Montreal
Voisin(s): Quebec, Paris, New_York, Detroit, Ottawa, Vancouver, Toronto, Bruxelles
Sommet: Paris
Voisin(s): Montreal
Sommet: La_Havane
Voisin(s): Rien
Sommet: Detroit
Voisin(s): Rien
Sommet: Ottawa
Voisin(s): Iqaluit
Sommet: Vancouver
Voisin(s): Toronto, Tokyo, Auckland
Sommet: Toronto
Voisin(s): Bogota
Sommet: Bogota
Voisin(s): Rien
Sommet: Bruxelles
Voisin(s): Bujumbura
Sommet: Bujumbura
Voisin(s): Bruxelles
Sommet: Iqaluit
Voisin(s): Rien
Sommet: Chicago
Voisin(s): Vancouver
Sommet: Tokyo
Voisin(s): Rien
Sommet: Auckland
Voisin(s): Rien
Sommet: New_York
Voisin(s): Rien
Sommet: Singapour
Voisin(s): Rien

Entrer s.v.p. votre choix (0 a 5):? 2
Recherche du plus court chemin avec Dijkstra.
Entrez la ville de depart:? Quebec
Entrez la ville de destination:? Bogota
Liste des villes du plus court chemin en utilisant la duree du vol: Quebec, Montreal, Toronto, Bogota
Duree totale du plus court chemin: 7.24
Liste des villes du plus court chemin en utilisant le cout du vol: Quebec, Montreal, Toronto, Bogota,
Cout total du plus court chemin: 1444.59

Entrer s.v.p. votre choix (0 a 5):? 3
Recherche du plus court chemin avec Bellman-Ford.
Entrez la ville de depart:? Quebec
Entrez la ville de destination:? Bogota
Liste des villes du plus court chemin en utilisant la duree du vol: Quebec, Montreal, Toronto, Bogota
Duree totale du plus court chemin: 7.24
Liste des villes du plus court chemin en utilisant le cout du vol: Quebec, Montreal, Toronto, Bogota
Cout total du plus court chemin: 1444.59
Liste des villes du plus court chemin en utilisant niveau de securite: Quebec, Chicago, Vancouver, Toronto, Bogota
Niveau de securite total du plus court chemin: 0

Entrer s.v.p. votre choix (0 a 5):?4
Mesurer le temps d'exécution de Dijkstra.
Temps d'execution: 19645 microsecondes

Entrer s.v.p. votre choix (0 a 7):? 5
Mesurer le temps d'exécution de Bellman-Ford.
Temps d'execution: 24102 microsecondes

Entrer s.v.p. votre choix (0 a 7):? 0
Merci et au revoir !

PS. Il est à noter que le temps d'exécution donné ici est juste à titre indicatif. Vous pouvez donc avoir un temps différent.

4 Ce que vous devez rendre

À l'aide de monportail, vous devez rendre un fichier .zip comportant **uniquement** les fichiers suivants :

- **Graphe.h** et **Graphe.cpp**
- **ReseauAerien.h** et **ReseauAerien.cpp**
- **Principal.cpp et ReseauAerien.txt**
- **NoteTp2-IFT2008.xls** (un fichier Excel contenant le barème de correction. Il faut juste ajouter les noms des membres de votre équipe sur la première ligne).

Vous ne devez en aucun cas ajouter un autre fichier ou répertoire. N'incluez aucun exécutable, ni aucun fichier généré par votre environnement de développement. De plus, vous n'avez pas à générer la documentation Doxygen en format html pour l'ensemble du projet (pour ne pas envoyer un gros fichier zip). Le correcteur pourra le faire lors de la correction.

Le nom du .zip doit respecter le format suivant : TP2-Matricule.zip. Nous vous rappelons qu'il est important de faire la remise par voie électronique uniquement en vous connectant à: http://monportail.ulaval.ca (aucun travail envoyé par courriel n'est accepté). Il est toujours possible de faire plusieurs remises pour le même travail. Vous pouvez donc remettre votre travail plus tôt afin d'être sûr qu'il a été remis en temps, et remettre par la suite une version corrigée avant l'heure limite de remise. De plus, il est de votre responsabilité de vérifier après la remise que vous nous avez envoyé les bons fichiers (**non vides et non corrompus**), sinon vous pouvez avoir un zéro.

Attention ! Tout travail remis en retard se verra pénalisé de -25% de la note. Le retard débute dès la limite de remise dépassée (dès la première minute). Un retard excédant une journée provoquera le rejet du travail pour la correction et la note de 0 pour ce travail.

PS. Vous devez considérer les mêmes remarques importantes du TP1 concernant les normes de programmation, la tolérance zéro vis-à-vis des erreurs de compilation et la portabilité des programmes.

Bon travail.