Travail pratique 3

Planification d’un réseau de panneaux publicitaires

1. Objectifs

- Utiliser des structures de données d'une bibliothèque normalisée (les conteneurs de la librairie standard de C++).
- Implémenter un type abstrait de données `CarteVisites` basé sur la représentation de graphe.
- Implémenter un algorithme de recherche d’un arbre de recouvrement.

2. Problématique

Vous devez écrire un programme C++ nommé `tp3`.

Le programme `tp3` doit chercher un arbre de recouvrement d’un graphe.

La mairie d’une ville a décidé d’installer des terminaux publicitaires dans un quartier. Elle veut relier tous les sites (places publiques) du quartier entre eux via un réseau filaire. Elle vous a fourni les données suivantes :

- La liste des rues les plus visitées du quartier; chaque rue est identifiée par un ensemble de sites.
- La position géographique des sites.
- Le flux d’achalandage associé à chaque rue, il est défini par la moyenne du nombre de personnes qui circulent quotidiennement sur cette rue.

Les sites du quartier sont reliés entre eux en formant un graphe connexe étiqueté où les nœuds représentent les sites du quartier, les arêtes sont les rues qui relient les sites entre eux et les poids sont le flux des personnes sur chaque rue. La ville veut relier, de façon optimale, tous les sites du cartier par un réseau publicitaire en favorisant les rues les plus achalandées. L’objectif du TP3 est de fournir un arbre de recouvrement à coût maximum (MST) du graphe des sites d’un quartier en se basant sur l’algorithme de Prim. Dans le processus du choix des arêtes dans l’algorithme Prim, si plusieurs arêtes ont le même poids vous devez choisir celle qui est la plus proche en distance.

3. Structure d’un programme

Pour bien amorcer ce travail, il est fortement recommandé de commencer avec le squelette de départ fourni dans `tp3.zip`. Vous pouvez modifier ces fichiers autant que vous le désirez. Toutefois, pour la correction automatique, vous devez préserver la syntaxe d'appel du programme et ses formats d'entrée et de sortie.

3.1 Syntaxe d'appel

Le programme `tp3` doit pouvoir être lancé en ligne de commande avec la syntaxe suivante.

```
./tp3 carteVisites.txt
```

où :

- le fichier `carteVisites.txt` spécifie une carte des sites d'un cartier et les données de flux de chaque rue.

Les résultats produits par votre programme doivent être écrits dans la sortie standard (*stdout*) à l'aide du flux de sortie C++ `std::cout`.

3.2 Fichier `carteVisites`

Un fichier **`carteVisite`**`.txt` est constitué de :

1. Une liste de sites (noeuds). Un site est spécifié par :
    - un nom (une chaîne de caractères);
    - une coordonnée de la forme (latitude, longitude).
2. Trois tirets (---) de séparation.
3. Une liste de rue. Une rue est spécifiée par :
    - un nom de rue (une chaîne de caractères);
    - un deux-points (:);
    - une liste de sites;
    - un nombre caractérisant le flux des personnes sur cette rue; dans le graphe ce chiffre représente le poids de toutes les arêtes de cette rue
    - un point-virgule (;).

À titre d'exemple, voici le fichier (**`uqam-carteVisites.txt`**) :

```
n1 (45.508377,-73.568755)
n2 (45.508662,-73.569259)
n3 (45.509128,-73.570289)
n4 (45.509331,-73.570793)
n5 (45.509944,-73.57056)
n6 (45.510452,-73.570348)
n7 (45.511113,-73.570010)
n8 (45.510579,-73.568894)
n9 (45.510106,-73.567939)
n10 (45.509910,-73.567435)
n11 (45.509346,-73.567929)
n12 (45.510004,-73.569399)
n13 (45.509561,-73.568428)
n14 (45.509497,-73.568256)
n15 (45.509474,-73.569881)
```

```
n16 (45.509523,-73.569694)
n17 (45.509636,-73.569715)
n18 (45.509741,-73.569656)
n19 (45.508940,-73.569045)
n20 (45.509170,-73.569173)
---
Jeanne-Mance : n1 n2 n3 n4 10 ;
Sherbrooke : n4 n5 n6 n7 7 ;
Saint-Urbain : n7 n8 n9 n10 9 ;
President-Kennedy : n10 n11 n1 4 ;
Grands-Batisseurs : n6 n12 n13 n14 n11 7 ;
UQAM1 : n3 n15 n16 n17 n18 n12 n8 10 ;
UQAM2 : n2 n19 n14 4 ;
UQAM3 : n9 n13 2 ;
UQAM4 : n19 n20 3 ;
```

Et voici la carte correspondante :

3.3 Format de sortie pour tp3

Le programme `tp3` doit afficher l'arbre MST de la façon suivante :

> Afficher d'abord les arêtes du MST une après l'autre dans l'ordre croissant des nœuds de départ. Si plusieurs arêtes ont le même nœud de départ, vous devez afficher ces arêtes selon l'ordre croissant des nœuds d'arrivées. Pour chaque arête vous devez afficher:
>
> a. à la première ligne, afficher les nœuds de l'arête dans l'ordre croissant

b. sur la deuxième ligne, le nom de la rue à laquelle appartient l'arête;
c. sur la troisième ligne, le poids associé à cette arrête;

Après avoir affiché, toutes les arrêtes :

1. Trois tirets (---) de séparation;
2. Le coût total de l'arbre sur la dernière ligne.

Exemple de sortie.

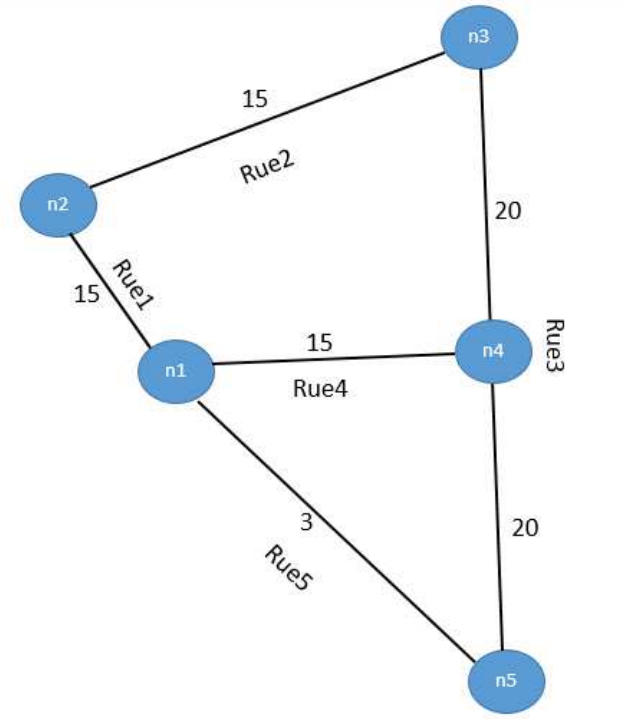
```
$ ./tp3a uqam-carteVisites.txt
n1 n2
Jeanne-Mance
10
n2 n3
Jeanne-Mance
10
n2 n19
UQAM2
4
n3 n4
Jeanne-Mance
10
n3 n15
UQAM1
10
//les autres arrêtes

---
159
```

Un deuxième exemple qui montre le cas où deux arêtes ont le même poids et le même nœud de départ :

```
n1 (45.509128,-73.570289)
n2 (45.509331,-73.570793)
n3 (45.511113,-73.570010)
n4 (45.510579,-73.568894)
n5 (45.509910,-73.567435)
---
Rue1 : n1 n2 15 ;
Rue2 : n2 n3 15;
Rue3 : n3 n4 n5 20 ;
Rue4 : n1 n4 15;
Rue5 : n1 n5 3;
```

Voici la carte correspondante :



La sortie :

```
n1 n2
Rue1
15
n1 n4
Rue4
15
n3 n4
Rue3
20
n4 n5
Rue3
20
---
70
```

Dans cet exemple votre programme ne doit pas choisir l'arbre (*n1*, *n4*) (*n2*, *n3*) (*n3*, *n4*) (*n4*, *n5*) comme MST car la distance entre *n2*, *n3* est la plus longue de toutes les arêtes qui ont le poids 15. Si votre programme prend comme nœud de départ le nœud *n1*, il aura le choix entre les deux arêtes (*n1*, *n4*) et (*n1*, *n2*) à cause que leurs poids sont les mêmes. Pour choisir entre elles, votre programme devra calculer la distance entre *n1* et *n4*, qui est de 0.195km, puis la comparer avec celle de *n1* et *n2*, qui est de 0.045km. Donc il choisira l'arête (*n1*, *n2*) vu qu'elle est la plus courte. En deuxième étape il devra

choisir entre les arêtes (*n2*, *n3*) et (*n1*, *n4*) car elles ont le même poids aussi. Il calculera la distance entre *n2* et *n3*, qui est de0.207km, vu que la distance entre *n1*et *n4* est la plus courte, il choisira l'arête (*n1*, *n4*). Pour la suite, votre programme n'aura qu'un seul choix à chaque fois, il choisira les arêtes (*n4*, *n3*) et (*n4*, *n5*).

4. Contraintes

4.1 Librairie standard C++ obligatoire

Contrairement aux TP1 et TP2, **vous devez maintenant utiliser, autant que possible, les conteneurs de la librairie standard de C++ (*Standard Template Library*)**. Cette contrainte vise à mettre en pratique l'utilisation d'une bibliothèque normalisée. Évidemment, vous pouvez créer vos propres structures lorsque justifié.

4.2 Environnement de développement

Vous avez la totale liberté d'utiliser l'environnement de développement de votre choix pour réaliser les travaux pratiques et les laboratoires.

4.3 Compilateur officiellement supporté :

Le compilateur officiellement supporté dans le cours est le compilateur GCC de GNU communément appelé «g++».

4.4Obligations pour la correction

Vos travaux pratiques seront corrigés de façon semi-automatisée à l'aide de scripts *bash* ou Oto sous Linux. Ainsi, afin d'être corrigés, vos travaux pratiques doivent pouvoir être compilés et exécutés sur les Linux (`{malt, oto}.labunix.uqam.ca`) de LabUnix. Si vos programmes ne fonctionnement pas sous ces environnements, alors les correcteurs n'auront pas l'obligation de corriger vos travaux. Un travail non corrigé peut obtenir une note de zéro, et ce, sans possibilité de reprise. Il est de votre responsabilité de vous vérifier que vos programmes fonctionnent correctement sous Linux.

4.5Taille des équipes

Vous pouvez faire ce travail en équipe de 1 ou 2. Toutefois, tous les membres de l'équipe doivent contribuer à l'ensemble du travail et non à seulement quelques parties. Le travail d'équipe vise à favoriser les discussions et l'entraide. Le travail d'équipe ne vise pas à réduire la tâche. Ainsi, se diviser la tâche en deux n'est pas une méthode de travail d'équipe appropriée dans ce cours. Tous les membres de l'équipe doivent être en mesure de comprendre et d'expliquer l'ensemble du travail. La participation inadéquate d'une étudiante ou d'un étudiant peut être considérée comme du plagiat. Le professeur et le correcteur pourront sélectionner quelques équipes au hasard afin de vérifier que tous les membres sont capables d'expliquer l'ensemble du travail.

6. Tests

Aucun test ne vous sera fourni pour ce travail. Vous devez créer vos propres tests et vérifier votre solution. La correction sera faite de manière automatique, donc le respect de la syntaxe d'appel et de la sortie est obligatoire. La note sera basée sur les résultats de passage de nos tests.

7. Remise

Vous devez respecter les règles de remise décrite sur le document Recommandations pour la remise de votre TP qui se trouve sur Moodle. Vous pouvez perdre jusqu'à 3points si vous ne respectez pas ces règles.

Vous devez remettre le TP3 via **moodle** au plus tard le **jeudi 26 avril à 23h55**. Une pénalité linéaire de 5% de la note maximale par heure de retard s'applique. Cela implique une note de zéro (0) après 20 heures de retard.

7.1 Remise papier

La partie papier est constituée de:

1. **Un formulaire à remplir, qui sera disponible sur *Moodle* bientôt.**
2. **Une analyse de la complexité algorithmique de votre solution.**
    - Identifiez les principaux facteurs qui influencent le temps d'exécution. Indice : la taille du problème.
    - Donnez la complexité temporelle en notation grand *O*. avec une justification.
3. **Code source.** Imprimez le code source complet de votre programme. Pour réduire la consommation de papier, vous pouvez imprimer dans un format recto-verso et 2 pages par côté de feuille (4 pages / feuille).

Remettre la partie papier au plus tard le **jeudi 26 avril à 23h55** dans la chute à travaux tout juste à côté du PK-4151. Les correcteurs vous feront des commentaires constructifs sur la partie papier. Celle-ci vous sera remise après la correction. Une pénalité linéaire de 5% de la note maximale par heure de retard s'applique. Cela implique une note de zéro (0) après 20 heures de retard.

7. Évaluation

Ce travail pratique vaut 15% de la note finale.

7.1 Grille de correction

Critère	Description	Pondération
A.	**Respect des directives pour la remise.** • Fichiers sources seulement (Makefile, .h, .cpp). Aucun fichier source manquant. Aucun fichier intermédiaire (.o, .obj, .gch, etc.) ou exécutable (tp3.exe). Aucun fichier test. • Remise par Oto. Pas de remise par courriel. • Compilable avec `make` sans modifications. • Exécutable sans modification.	/ 1

B.	**Appréciation générale.**
	
	• Structure du programme.
	  o Découpage du programme (tout n'est pas dans la fonction `main`).
	  o Justesse des types et des structures de données.
	  o Classes et fonctions.
	  o Usage du mot clé `const`.
	  o Usage des références et des pointeurs.
	• Qualité du code.
	  o Nomination des identificateurs (noms significatifs), lisibilité du code, etc.
	  o Présence et pertinence des commentaires; etc.
	• Encapsulation.
	  o Respect des principes de l'abstraction;
	  o Cachez le maximum de la représentation des objets en rendant un maximum d'attributs privés;
	  o Évitez autant que possible les bris d'abstraction, comme des *getters* et *setters* qui retournent ou affectent directement des attributs d'un type abstrait de donnée. Par exemple, les fonctions `getX()` et `getY()` ne devraient pas exister dans une classe `Point`. Mais, une fonction `getNom()`dans une classe `Station` peut être justifiée. La représentation d'une classe `Date` devrait être privée. Une structure ou classe `Intervalle` peut avoir deux attributs publics `debut` et `fin`.
	  o Utilisation appropriée des modificateurs d'accès `public`, `protected` et `private`, et du mot clé `friend`, etc.
	• Gestion de la mémoire.
	  o Toute la mémoire allouée dynamiquement doit être correctement libérée au moment approprié et avant la fin de la fonction `main`.
		/ 2

C.	**Fonctionnement correct.**	/ 7

E	**Analyse des algorithmes.** • Complexité temporelle en notation grand O. • Ordre de grandeur simplifié. Ex: O(2n) ==> O(n). • Justification claire et correcte.	/2
F	**Efficacité raisonnable.** Tous les points sont accordés dès que les temps d'exécution ne dépassent pas le double de ceux d'une solution raisonnable.	/ 3
	Total :	/ 15

Pour les cas problématiques, jusqu'à 2 points peuvent être retranchés pour la qualité de la langue et de la présentation.

8. Crédits et licences

- Les données des cartes fournies sont extraites d'OpenStreetMap et sont régies par la licence *Open Database License (ODbL) v1.0*.
- Les tuiles (images) des cartes proviennent d'OpenStreetMap et sont distribuées sous la licence *Creative Commons* paternité – partage à l'identique 2.0 (CC-BY-SA).