Travail pratique #1 : Positionnement de stations dans un réseau sans fil

Objectifs

- S'initier à la programmation en C++.
- Pratiquer l'encapsulation et concevoir des types abstraits de données.
- Gérer la mémoire.
- Implémenter et appliquer des structures de données simples en C++.
- Résoudre un problème simple.
- Analyser la complexité temporelle d'un algorithme.

Problématique

En télécommunications sans-fil, déterminer où installer des stations radio (antennes) est un problème d'optimisation complexe. Il faut entre autres maximiser la couverture du réseau sans-fil, minimiser les coûts, éviter les interférences radio, etc., et ce, tout en considérant le relief du terrain et la hauteur des bâtiments qui font obstacle aux signaux radio.

Le TP1 consiste à résoudre une version simplifiée de ce programme d'optimisation. Pour simplifier le problème, on assume que:

- les stations ont chacune un rayon de couverture fixe;
- les stations doivent être installées sur des immeubles ayant une hauteur minimale donnée;
- les stations de communication peuvent desservir un nombre illimité de clients dans son rayon de couverture;
- la région a un relief parfaitement plat (le rayon de couverture est un cercle parfait).

À noter qu'un client peut être desservi par plusieurs stations, mais il ne doit être compté qu'une seule fois.

Structure du programme

Vous devez écrire un programme C++ nommé tp1. Votre programme reçoit en entrée une liste d'immeubles à desservir et une liste de stations à installer. Le but du programme est de déterminer sur quels immeubles il faut installer les stations de communication afin de desservir un maximum de clients.

Un squelette de départ est disponible dans tp1.zip.

- Ce squelette est fourni pour vous aider à vous concentrer sur l'essentiel du TP1.
- Vous êtes libres de l'utiliser ou non.
- Vous pouvez y effectuer toutes les modifications que vous souhaitez ou jugez nécessaires.
- Vous ne pouvez pas modifier le format des entrées et sorties.
- Ce squelette vise d'abord la simplicité pour obtenir rapidement un programme fonctionnel.

Syntaxe d'appel du programme tp1

Votre programme doit pouvoir être lancé en ligne de commande avec la syntaxe suivante :

./tp1 [nomfichier]

où nomfichier est optionnel.

Si nomfichier est spécifié, alors votre programme doit lire depuis le fichier nomfichier au moyen d'un flux de lecture de type std::ifstream. Sinon, votre programme doit lire dans l'entrée standard (*stdin*) au moyen du flux d'entrée std::cin. À noter que le squelette implémente déjà cela.

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

Entrée

Un fichier d'entrée contient une liste* de stations de communication et une liste d'immeubles. Au tout début du fichier, on retrouve le nombre de stations. Ensuite, on retrouve chaque station dans le format suivant : son nom, son rayon de couverture, sa hauteur minimale d'installation, et sa fréquence d'opération en MHz. Ensuite, on retrouve la liste des immeubles. Attention, contrairement au nombre de stations, le nombre d'immeubles n'est pas explicitement écrit dans le fichier. Il faut lire le fichier au complet pour savoir combien d'immeubles il contient. Les immeubles sont formatés de la façon suivante : nom, position géographique notée (x,y), hauteur et le nombre de clients dans l'immeuble. Les coordonnées, les rayons de couverture et les hauteurs sont exprimés en mètres.

Ci-haut, il faut lire **liste au sens littéraire et non au sens technique, c'est-à-dire qu'il ne faut pas nécessairement utiliser une liste chainée. Ces informations peuvent être stockées dans une structure de données de votre choix (liste, tableau, etc.).*

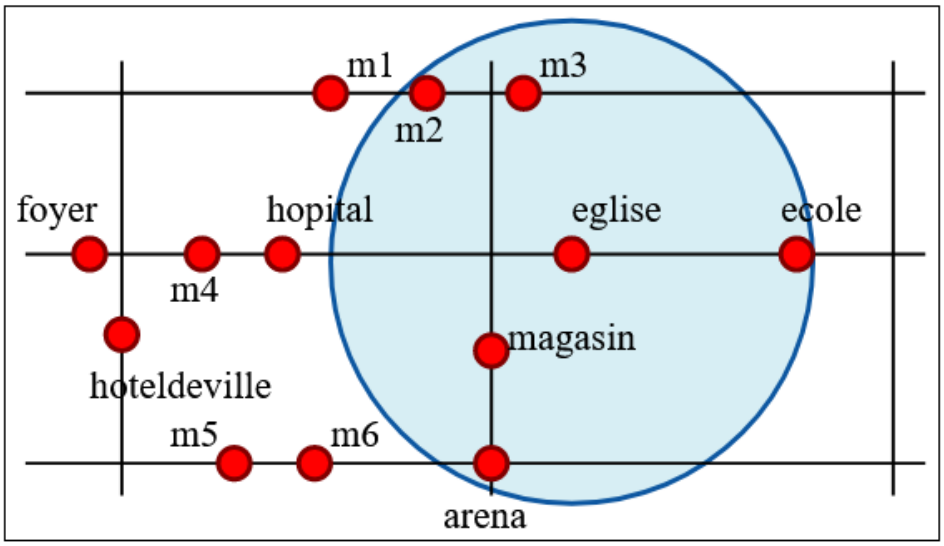
Pour faciliter le *parsing* au moyen de std::cin >>, il y a au moins un espace blanc (espace, tabulation ou retour de ligne) séparant chaque champ. De plus, il est garanti que le nom des stations et immeubles ne contiennent pas d'espaces blancs.

Voici un exemple d'entrée (testC00.txt).

```
1
station1 15.0 10.0 900.0

arena (29,2) 12.9 24
ecole (48,15) 10.1 32
eglise (34,15) 20.7 18
foyer (4,15) 9.1 42
hopital (16,15) 9.9 42
hoteldeville (6,10) 6.4 12
m1 (19,25) 7.9 2
m2 (25,25) 7.9 7
m3 (31,25) 7.9 4
m4 (11,15) 6.3 4
m5 (13,2) 8.3 5
m6 (18,2) 8.3 4
magasin (29,9) 4.1 6
```

Et voici la carte correspondante :



Le cercle bleu montre la couverture de la station station1 lorsqu'elle est installée sur l'église.

Sortie

Pour chaque station, le programme doit écrire, sur une ligne, son nom, un espace et le nom de l'immeuble où elle doit être installée. Ensuite, sur une seule nouvelle ligne le programme doit écrire le nombre total de clients desservis par l'ensemble des stations.

Voir un exemple de sortie dans le fichier testC00.txt.

```
station1 eglise
91
```

Et un exemple de sortie dans le fichier testD00.txt.

```
station1 eglise
station2 hopital
137
```

Pour certains problèmes, une ou plusieurs stations pourraient être inutiles ou ininstallables. Dans ces cas, certaines stations peuvent ne pas être assignées. Pour ce faire, il suffit d'écrire deux tirets (--) pour indiquer qu'une station n'est assignée à aucun immeuble. Le test testD05.txt est un exemple dont la solution est:

```
s1 --
s2 im1
12
```

Algorithme

S'il n'y a qu'une seule station à installer, ce problème se résout avec l'algorithme suivant:

```
lire stations
lire immeubles
immeuble_choisi = ?
nbclientsmax = -1
pour i = 0 à n-1 // n=nombre d'immeubles
  si immeubles[i].hauteur >= station.hauteurminiale
    nbclients = 0
    pour j = 0 à n-1
      d = distance entre immeubles i et j
      si d <= rayon de la station
        nbclients += nombre de clients dans l'immeuble j
    si nbclients > nbclientsmax
      nbclientsmax = nbclients
      immeuble_choisi = i
afficher nom de la station + " " + nom de l'immeuble i
```

Si le problème contient deux stations à installer, alors il faut ajouter une boucle **pour** supplémentaire. Toutefois, il faut faire attention pour ne pas compter deux fois un même

client. De plus, si les deux stations ont la même fréquence, elles doivent être suffisamment éloignées.

À noter que les solutions ne sont pas nécessairement uniques. Pour un problème, il peut y avoir plusieurs assignations possibles qui maximisent le nombre de clients desservis. Il suffit de retourner l'une de ces solutions. Ce qui prime, c'est le nombre total de clients desservis.

Interférences entre les stations

Les solutions retournées doivent ne contenir aucune interférence possible entre toute paire de stations. Deux stations interfèrent si et seulement si :

1. elles utilisent la même fréquence;
2. et la distance les séparant est inférieure ou égale à la somme de leur rayon de couverture.

Par exemple, dans le problème suivant :

```
2
s1 5.0 0.0 900.0
s2 5.0 0.0 900.0
im1 (0,0) 4.0 4
im2 (10,0) 4.0 4
im3 (10.1,0) 4.0 4
```

les stations `s1` et `s2` ne peuvent être simultanément installées sur les immeubles `im1` et `im2`. Par contre, leur installation sur `im1` et `im3` est correcte.

Contraintes

Librairies permises

Vous devez implémenter et utiliser vos propres structures de données. Pour l'instant, l'utilisation des conteneurs de la librairie standard de C++ (*Standard Template Library*) n'est pas permise. Ce sera permis plus tard dans le cours.

Environnement de développement

Relisez les Politiques et les directives sur les outils informatiques dans le cours INF3105.

Taille 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.

Tests

Des tests sont disponibles dans tp1-tests.zip.

- Les fichiers test[A-G][0-9][0-9].txt sont les fichiers test à être lu par le programme tp1.
- Les fichiers test[A-G][0-9][0-9]**+**.txt sont les résultats calculés par une solution du TP1.
- Les fichiers test[A-G][0-9][0-9]**=**.txt contiennent le décomte du nombre de stations, d'immeubles et de clients comptés par le programme de validation.
- Les lettres dans les noms de fichier test indiquent à quel critère ils font référence. Par exemple, le test testC00.txt est un test pour le critère C, c'est-à-dire avec une seule station.

Exécuter des tests sur votre propre machine

Pour exécuter les tests, vous devez décompresser le fichier tp1-tests.zip. Les fichiers tests et le script evaluer.sh ne doivent pas être dans le même répertoire que votre programme tp1.

./tests/evaluer.sh
Évaluation du TP1 d'INF3105...
Limite de temps par test : 120 secondes.
Limite de mémoire par test : 2097152 KiO.
Limite de taille de fichier : 1024 KiO.
...

Fichier_test	#Stat	#Imm	#Cli	Opt?	CPU	Mém.(k)
testC00.txt	1	13	91	1	0.00	1260k
testC01.txt	1	11	30	1	0.00	1272k
testC02.txt	1	1000	2322	1	0.00	1360k
testC03.txt	1	5000	4025	1	0.02	1884k

...

Temps d'exécution de référence

Voir fichier temps_exec.html pour le temps d'exécution de quelques solutions variées.

Remise

Vous devez remettre électroniquement le TP1 **au plus tard le dimanche 27 février 2022 à 23h59**. Votre projet, en format zip, doit être remis sur le site Moodle dans la section Travaux pratiques. Veuillez mettre vos codes permanents dans le nom de votre fichier.

Vous pouvez soumettre votre TP autant de fois que vous voulez. Seule la dernière soumission sera considérée.

Vous devez remettre tous vos fichiers sources, incluant un fichier Makefile.

Rapport

 Auto-évaluation indiquant si votre programme fonctionne correctement, partiellement ou aucunement.
 Un tableau montrant les temps d'exécution sur les fichiers tests fournis.

- Si un test prend plus de 60 secondes, vous pouvez l'arrêter ([Ctrl]+[C]) et simplement écrire **>60** dans votre rapport.
- Vous **pourrez** générer un rapport automatiquement en lançant la commande /home/inf3105/tp1/evaluer.sh sur malt.labunix.uqam.ca à partir du répertoire où vous avez compilé votre exécutable tp1.

 Analyse de la complexité temporelle (pire cas) en notation grand O

- La complexité temporelle doit être exprimée en fonction de la taille du problème :
  - *n* indique nombre de stations à installer;
  - *m* indique nombre de bâtiments.

Évaluation

Ce travail pratique vaut 15% de la note finale.

Grille de correction

Critère	Description	Pondération
A.	**Respect des directives pour la remise** • Fichiers sources (Makefile, .h, .cpp) seulement. Aucun fichier source manquant. Aucun fichier binaire (.o, exécutable). Aucun fichier test. • Remise par Moodle. Pas de remise par courriel. • Compilable avec make sans modifications. • Exécutable sans modification.	/ 2

B.	**Appréciation générale**
	
	• Structure du programme + Qualité du code :
	  ○ Découpage du programme (tout n'est pas dans la fonction main()).
	  ○ Choix des types de données; identificateurs (noms) significatifs, lisibilité du code, pertinence des commentaires; etc.
	  ○ Justesse de l'usage du mot-clé const, des références (&) et des pointeurs (*).
	• Encapsulation :
	  ○ Respect des principes de l'abstraction;
	  ○ Cachez le maximum de la représentation des objets en rendant un maximum d'attributs privés;
	  ○ 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.
	  ○ Utilisation appropriée des modificateurs d'accès public, protected et private, et du mot-clé friend, etc.
	• Gestion de la mémoire :
	  ○ Toute la mémoire allouée dynamiquement doit être correctement libérée au moment approprié et avant la fin du programme.
		/ 3

C.	**Fonctionnement correct avec 1 seule station**	/ 4
D.	**Fonctionnement correct avec 2 stations**	/ 3

<table>
<tr><td>E.</td><td>Exactitude de l'autoévaluation. Grille :<table>
<tr><td></td><td>Vous déclarez clairement que votre programme fonctionne avec 2 stations</td><td>Vous déclarez clairement que votre programme fonctionne avec seulement 1 station</td><td>Vous déclarez clairement que votre programme ne fonctionne pas</td></tr>
<tr><td>Nous constatons que votre programme fonctionne avec 2 stations</td><td>1</td><td>0.5</td><td>0</td></tr>
<tr><td>Nous constatons que votre programme fonctionne avec seulement 1 station</td><td>0.5</td><td>1</td><td>0.5</td></tr>
<tr><td>Nous constatons que votre programme ne fonctionne pas</td><td>0</td><td>0</td><td>1</td></tr>
</table></td><td>/ 1</td></tr>
<tr><td>F.</td><td>Analyse de l'algorithme</td><td>/ 2</td></tr>
<tr><td></td><td>Total :</td><td>/ 15</td></tr>
<tr><td>G.</td><td>Fonctionnement correct avec 3+ stations (boni)
Le nombre de stations peut être arbitrairement grand.</td><td>+1</td></tr>
<tr><td>H.</td><td>Efficacité (boni)
Algorithme significativement plus efficace qu'une solution naïve. Une solution naïve n'arrivera pas à résoudre tous les tests fournis en moins de 120 secondes chacun.</td><td>+1</td></tr>
<tr><td></td><td>Note maximale :</td><td>17 / 15</td></tr>
</table>

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