Automne 2024

Algorithmes et structures de données pour ingénieur

GLO-2100/ IFT-2008

Thierry EUDE, Ph.D
Kim Rioux-Paradis, M.Sc

Travail à faire INDIVIDUELLEMENT
À rendre avant la date indiquée
sur le portail du cours

(Voir modalités de remise à la fin de l'énoncé)

Ce travail doit être de votre propre création.
Tout étudiant qui commet une infraction au Règlement disciplinaire à l'intention des étudiants de l'Université Laval dans le cadre du présent cours, notamment en matière de plagiat, est passible des sanctions qui sont prévues dans ce règlement. Il est très important pour tout étudiant de prendre connaissance des articles 23 à 46 du Règlement disciplinaire. Celui-ci peut être consulté à l'adresse suivante:
http://ulaval.ca/reglement-disciplinaire
Tout étudiant est tenu de respecter les règles relatives au plagiat. De plus, les étudiants ou étudiantes ayant collaboré au plagiat seront soumis aux sanctions normalement prévues à cet effet par les règlements de l'Université.

Tous les travaux sont soumis à un logiciel de détection de plagiat.

Travail Pratique #2
Réseau IP

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

Objectifs

Ce travail pratique met l'emphase sur le type graphe et ses algorithmes. Par exemple, il sera question d'implanter un graphe ainsi que les principaux algorithmes rattachés à cette importante structure de données comme, par exemple, les algorithmes de **Dijkstra**. Il est question également d'explorer des propriétés dans un graphe telles que, entre autres, les composantes connexes

Étude du routage dans Internet

On veut observer le routage dans l'internet. Pour cela on dispose de données de mesure dans des fichiers. Celles-ci nous donnent le nom des routeurs, leur adresse dans l'internet (adresse IP) ainsi que le délai de transmission en ms entre différents routeurs. Il s'agit alors d'exploiter ces fichiers de mesures et de vérifier comment se comporte l'internet vis-à-vis du routage.
La partie de réseau observée peut être représentée par un graphe orienté pondéré dans lequel chaque sommet représente un routeur et chaque arc, une liaison. Le poids associé à chaque arc représente le délai de transmission en ms entre deux routeurs. Le graphe figure 1 pourrait en être un exemple de représentation.

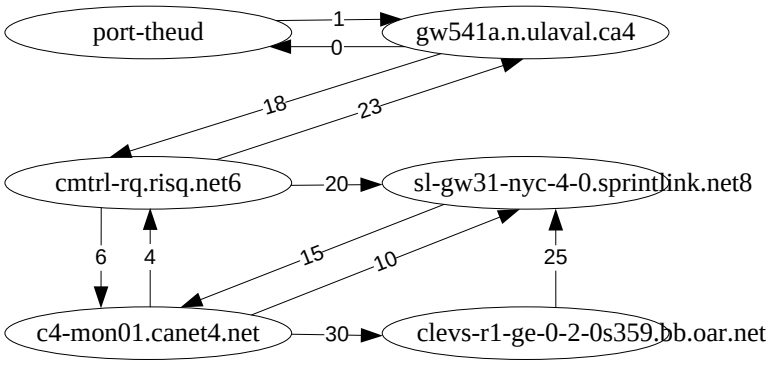

Figure 1

Travail à réaliser

Avertissements

Il est interdit de modifier le modèle d'implantation de la classe Reseau, Graphe ou Routeur, cependant, vous avez le droit d'ajouter :

1. des méthodes privées aux trois classes afin d'optimiser votre implantation, c'est même très encouragé ;
2. des définitions de constantes;
3. d'inclusion d'autres classes de la librairie STL que ceux déjà inclus dans Reseau.h si vous sentez la nécessité.

I. Format du fichier de sauvegarde

La sauvegarde des données représentant un réseau de routeurs se fera dans un fichier texte. On trouvera tout d'abord la liste de chaque routeur précisant respectivement, son nom, son adresse IP et sa localisation, puis la liste de toutes les liaisons (départ, arrivée, délai de transmission). L'exemple suivant représente un fichier de sauvegarde (il a deux arêtes de moins que l'exemple).

```
port-theud
132.203.120.179
Québec
gw541a.n.ulaval.ca4
132.203.244.145
Québec
cmtrl-rq.risq.net6
132.202.100.65
Montréal
sl-gw31-nyc-4-0.sprintlink.net8
160.81.43.177
New York
c4-mon01.canet4.net
205.189.32.226
OTTAWA
clevs-r1-ge-0-2-0s359.bb.oar.net
192.88.192.137
Cleveland
$
port-theud
gw541a.n.ulaval.ca4
1
gw541a.n.ulaval.ca4
port-theud
0
gw541a.n.ulaval.ca4
cmtrl-rq.risq.net6
18
cmtrl-rq.risq.net6
gw541a.n.ulaval.ca4
23
cmtrl-rq.risq.net6
c4-mon01.canet4.net
6
c4-mon01.canet4.net
sl-gw31-nyc-4-0.sprintlink.net8
10
cmtrl-rq.risq.net6
sl-gw31-nyc-4-0.sprintlink.net8
20
sl-gw31-nyc-4-0.sprintlink.net8
c4-mon01.canet4.net
15
clevs-r1-ge-0-2-0s359.bb.oar.net
sl-gw31-nyc-4-0.sprintlink.net8
25
```

On pourra remarquer que la liste des routeurs est séparée de la liste des liaisons par le caractère « $ »

Ce travail pratique comprend 3 parties.

II. Interface du patron de classe Graphe

Une interface de la classe de graphe vous est fournie. Vous devez implémentez des méthodes manquantes. Vous ne devez pas modifier l'interface sauf si vous voulez ajouter des méthodes privées. Vous devez également ajouter la théorie du contrat dans les méthodes manquantes.

```
bool Graphe::estFortementConnexe() const;
```
Renvoie:
Si le graphe est fortement connexe ou non

```
bool Graphe::dijkstra(const Objet& p_eOrigine,
const Objet& p_eDestination, std::vector<Objet>& p_chemin);
```
Précondition:
p_eOrgine et p_eDestination existent
Postcondition:
Le graphe original reste inchangé
Le chemin est retourné
Exceptions:
bad_alloc Il n'y a pas assez de mémoire pour retourner le chemin
Paramètres:

[in]	p_eOrigine	L'étiquette du sommet d'origine
[in]	p_eDestination	L'étiquette du sommet destination
[out]	p_chemin	le chemin de p_eOrgine à p_eDestination

Renvoie:
un vecteur contenant le chemin

```
bool Graphe::getPointsArticulation(std::vector<Objet>& p_sommets)
```
Exceptions:
bad_alloc Il n'y a pas assez de mémoire pour retourner le chemin
Paramètres:

[out]	p_sommets	Les sommets d'articulation

Renvoie:
un vecteur contenant les sommets d'articulations

Note : Les points d'articulations pour un graphe orienté sont définis comme suit:
Soit un graphe G fortement connexes. Les sommet s'articulations sont ceux pour lesquels le graphe G n'est plus fortement connexe si l'on enlève ces noeuds.

III. Interface du type Routeur

Cette classe est assez simple. Toutes les méthodes sont déjà implémentées. Il ne faut pas modifier cette classe.

IV. Interface du type Reseau

Il s'agit maintenant d'implémenter dans le fichier `Reseau.cpp` l'interface du type Reseau dont les méthodes sont décrites ci-après. Pour implanter ces méthodes, vous devez respecter la définition du modèle d'implantation du type Reseau. En d'autres mots, il ne faudrait en aucun cas modifier la partie privée de la classe Reseau à part d'ajouter éventuellement quelques définitions de constantes. Vous devez également ajouter la théorie du contrat dans les méthodes à développer.

 **Important.** Nous vous fournissons quelques méthodes du type Reseau, entre autres : `afficherRouteurs()`, pour vous en servir lors des tests, ainsi qu'un constructeur qui prend en argument un fichier en entrée et qui construit le réseau décrit dans ce fichier dans un graphe en mémoire. Leur déclaration ainsi que leur implémentation se trouve dans les fichiers `Reseau.h` et `Reseau.cpp` respectivement.

 **Tests unitaires**
Vous devez créer le fichier `ReseauTesteur.cpp`. Ce testeur contiendra vos google tests.

Plus court chemin

```
std::vector<Routeur> Reseau::determinerMinParcours(const Routeur& p_routeurOrigine,
            const Routeur& p_routeurDestination, int& p_nbSecondes)
```

Trouve le plus court chemin entre deux routeurs en utilisant l'algorithme de Dijsktra et le retourne.

Précondition:
p_orgine et p_destination existent

Postcondition:
Le temps de transmission total est retourné, -1 s'il n'y a pas de chemin
Le graphe original reste inchangé
La liste des routeurs à parcourir est retournée

Exceptions:

bad_alloc Il n'y a pas assez de mémoire pour retourner le chemin

Paramètres:

[in]	**p_routeurOrigine**	le routeur de l'origine
[in]	**p_routeurDestination**	le routeur de destination
[out]	**p_nbSecondes**	le temps de transmission total

Renvoie:
un vecteur contenant le parcours

Notez bien. Il s'agit bien entendu ici d'implanter l'algorithme de Dijkstra pour résoudre cette fonction. De plus, un réseau qui ne comporte pas de solution ne constitue pas un appel anormal de la fonction. Il s'agira d'une utilisation tout à fait normale de la méthode.

Tout routeur est-il accessible?

La première chose à vérifier dans le réseau est si tout routeur du réseau est accessible à partir de n'importe quel autre routeur. Vous devez donc écrire une méthode qui détermine si tous les routeurs sont accessibles dans le réseau. Les spécifications détaillées de la méthode sont les suivantes :

```
bool Reseau::routeursAccessibles()
```

Détermine si tous les routeurs du réseau sont accessibles à partir de n'importe où

Précondition:
Le réseau est non vide
Renvoie:
VRAI si tout routeur est accessible depuis un autre, sinon FAUX dans tous les autres cas.

Routeurs critiques

Dans un réseau, il est toujours possible qu'un routeur tombe en panne. Il s'agit alors d'identifier les routeurs critiques, qui, s'ils tombent en panne, remette en question l'intégrité du réseau. Connaissant ainsi les routeurs critiques on pourra décider d'investir pour sécuriser ces routeurs en prévoyant des redondances de secours par exemple.

Vous devez donc écrire une fonction qui retourne tous les routeurs qui, s'ils tombent en panne, rendent certains routeurs du réseau inaccessibles. Les spécifications détaillées de cette fonction sont les suivantes:.

```
std::vector<Routeur> Reseau::routeursCritiques()
```
Précondition:
Le réseau est non vide.
Postcondition:
Le réseau original reste inchangé
Exceptions:
bad_alloc Il n'y a pas assez de mémoire pour retourner la liste des routeurs
Renvoie:
Dans un vector, une liste de routeurs que l'on doit sécuriser, sinon une liste vide.

V. Investigation

Vous devez investiguer d'autres méthodes de plus court chemin qui s'applique à tous types de graphes. Vous devez présenter 3 autres algorithmes de plus court chemin dont au moins un qui n'a pas été vu en classe.
Votre démarche d'investigation, pour arriver à trouver l'algorithme de plus court chemin le plus efficace possible pour le réseau, devra être l'objet d'un **rapport écrit à rendre**. Nous vous conseillons de commencer votre démarche d'investigation en examinant, la structure du réseau et ses caractéristiques (densité, existence ou non de cycles, valeurs possibles des poids, nombre de nœuds et d'arcs, etc) avant de commencer votre recherche. Veuillez consigner vos observations dans votre rapport. Suite à vos observations, examinez les algorithmes et faites vos choix en fonction de leurs propriétés et de leur complexité algorithmique (coût d'exécution).
N'oubliez pas d'inclure vos références.

Votre rapport d'investigation doit être un document PDF d'au plus 3 pages qui contiendra les éléments suivants (reprenez les titres dans votre rapport).

1. **Définition de la question à étudier :**
   La formulation, en vos propres mots, de ce que vous devez faire pour cette investigation.
2. **Démarche d'investigation :**
   Un résumé sur les algorithmes de plus court chemin que vous avez envisagés et les raisons qui ont conduit au choix de votre algorithme dans le cadre du routage dans Internet.
3. **Application de la démarche d'investigation :**
   Vous devez décrire ces algorithmes (fonctionnement, analyse algorithmique). Vous devez également donner des avantages et inconvénients pour chacun d'eux ainsi que des cas d'utilisation.
4. **Analyse des données recueillies :**
   Une présentation des résultats obtenus suite à votre démarche d'investigation et votre analyse de ces résultats.
5. **Conclusion :**
   Vos conclusions et recommandations.

Récapitulatif des fichiers fournis

1. Le fichier Routeur.h contenant la description interne du type Routeur
2. Le fichier Routeur.cpp
3. Le fichier Reseau.txt contenant les données d'un réseau IP.
4. Reseau.h contenant la description interne du type Reseau.
5. Reseau.ccp à compléter.
6. Graphe.h contenant la description interne du type graphe.
7. Graphe.hpp à compléter
8. ContratException.h et ContratException.cpp pour l'implémentation du contrat.
9. Un fichier main.cpp pour exécuter votre code.

Bien livrable :

Le deuxième travail pour le cours GLO-2100/IFT-2008 est **à faire INDIVIDUELLEMENT.**
Tous vos développements doivent être faits dans le dépôt github privé qui vous est créé après avoir accepté « l'assignment »
https://classroom.github.com/a/tlLSDnr_
RAPPEL : Vous DEVEZ ABSOLUMENT CHOISIR VOTRE COURRIEL dans la liste avant d'accepter l'assignement sinon, on ne pourra pas vous identifier lors de la correction et vous donner votre résultat.
On devra y retrouver tous les fichiers sources nécessaires à la compilation de votre projet :

1. Le fichier éventuellement complété **Reseau.h** ainsi que le fichier **Reseau.cpp** contenant l'implémentation du type Reseau.
2. Les fichiers **Routeur.h** ainsi que le fichier **Routeur.cpp** contenant l'implémentation du type Routeur.
3. Les fichiers **Graphe.h** et **Graphe.hpp**
4. ContratException.h et ContratException.cpp
5. **TesteurReseau.cpp** contenant vos tests unitaires suivants :
   - **TestDijkstra()**, pour tester `determinerMinParcours()`.
   - **TestRouteursAccessibles ()** pour tester la méthode *routeursAccessibles ()*.
   - **TestRouteursCritiques ()** pour tester la méthode *routeursCritiques ()*.

6. Dans une archive zip au nom de Documentation, la documentation générée par Doxygen en format html pour l'ensemble du projet.
7. Vos fichiers CMakelist.txt
8. Tous autres fichiers nécessaires à la compilation et l'exécution de votre code.
9. Votre fichier pdf d'investigation des autres algorithmes de plus court chemin.

Spécifications sur l'ensemble du travail pratique

Les normes de programmation

Vous devez respecter les normes de programmation du cours. Une description de ces normes se trouve dans *contenu et activités/Notes de cours/Documents divers* du site Web du cours.

Tolérance zéro vis-à-vis des erreurs de compilation

Si un travail comporte ne serait-ce qu'une erreur de compilation, il sera fortement pénalisé, et peut même se voir attribuer la note zéro systématiquement.

Erreurs à l'exécution

Testez intensivement vos programmes. Rappelez-vous qu'un programme qui ne plante pas n'est pas nécessairement sans bogue, mais qu'un programme qui plante même une seule fois comporte nécessairement un bogue. D'ailleurs, si un programme ne plante nulle part sauf sur l'ordinateur de votre ami, c'est qu'**il comporte un bogue**. Il risque donc de planter un jour ou l'autre sur votre ordinateur et, encore pire, sur celui des correcteurs.
Si vous ne testez pas suffisamment votre travail, il risque de provoquer des erreurs à l'exécution lors de la correction. La moindre erreur d'exécution rend la correction extrêmement difficile et vous en serez donc fortement pénalisés.

Portabilité des programmes

Vos programmes doivent absolument pouvoir être compilés et exécutés sans erreurs sur n'importe quel système d'exploitation. Autrement, vous obtiendrez la note zéro pour votre travail. Notamment, nous exigeons que les programmes puissent être compilés sous g++ avec la machine virtuelle du cours. Rappel : Il est donc important de vérifier la portabilité de votre code avec l'environnement de correction ceci avant la remise.

Critères d'évaluation

1) Respect des biens livrables
2) Portabilité de votre code source
3) Structure, organisation du code, lisibilité
4) Exactitude du code (fonctionnalité)
5) La gestion dynamique de la mémoire.
6) La gestion des exceptions
7) La qualité de vos testeurs
8) Le respect des prototypes des méthodes quand ils sont imposés.
9) L'implémentation du contrat
10) Votre rapport d'investigation

 ***Particularités du barème***

- *Si des pénalités sont appliquées, elles le seront sur l'ensemble des points.*
- *Si un travail comporte ne serait-ce qu'une erreur de compilation, il sera fortement pénalisé, et peut même se voir attribuer la note zéro systématiquement.*
- *Il est très important que votre travail respecte strictement les consignes indiquées dans l'énoncé, en particulier les prototypes des fonctions, sous peine de fortes pénalités*

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.

Bon travail