POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

LOG 2810
STRUCTURES DISCRETES

TP2 : AUTOMATES – MACHINES À ÉTATS

Session	Automne 2016
Pondération	10 % de la note finale
Lieu de réalisation	L-3714
Taille des équipes	3 étudiants
Date de remise du projet	2 décembre 2016 (23h55 au plus tard)
Directives particulières	Soumission du livrable par moodle uniquement (https://moodle.polymtl.ca).
	Toute soumission du livrable en retard est pénalisée à raison de 10% par jour de retard.
	Programmation $C++$ avec Visual Studio 2013.
Les questions sont les bienvenues et peuvent être envoyées à: Alexandre Courouble (alexandre.courouble@polymtl.ca), Gaëtan Dupeuble (gaetan.dupeuble@polymtl.ca), Foutse Khomh (foutse.khomh@polymtl.ca), Aurel Josias Randolph (aurel.randolph@polymtl.ca),	

1 Connaissances requises

- Notions d'algorithmique et de programmation C++.
- Notions de théorie des langages.

2 Objectif

L'objectif de ce travail pratique est de vous permettre d'appliquer les notions théoriques du langage vues en cours, sur des cas concrets mais hypothétiques, tirés de votre quotidien. Dans cette optique, il est question dans ce travail de développer un système d'autopartage.

3 Mise en situation

Un service d'autopartage est un service qui met des véhicules à disposition des habitants d'une ville. Plutôt que d'acheter une voiture qui sera stationnée pendant la majorité du temps, les automobilistes

ont la possibilité de partager des voitures pour effectuer leurs déplacements. L'arrivée des véhicules autonomes s'apprête à changer complètement le fonctionnement de l'autopartage. Les voitures mises à disposition des utilisateurs seront maintenant autonomes et pourront venir chercher les clients à leur position actuelle.

4 Description

Vous répondez à un appel d'offres d'une entreprise désirant mettre en place un service d'autopartage de véhicules autonomes dans la ville de Montréal. Le conseil d'administration a découpé l'agglomération en différentes zones, chaque zone elle-même découpée en voisinages (*cf.* fichiers textes `zone1.txt` à `zone3.txt` : un voisinage est représenté par son zipcode).

Le but du service est le suivant. Un client peut demander à tout moment une voiture autonome livrée à sa position actuelle. L'application doit nécessairement choisir un véhicule libre, stationné dans la zone occupée par le client, préférentiellement dans le voisinage. Si aucun véhicule n'est stationné dans le voisinage, un autre véhicule libre de la zone est convoqué. Si aucun véhicule n'est disponible dans la zone, la demande du client est refusée.

Puisque le refus d'un client est un coût énorme à l'entreprise en terme de réputation, celle-ci a décidé de mettre en place un système d'équilibrage de la flotte, afin d'éviter tout scénario désastreux. Le but de ce système secondaire est d'assurer à tout instant que chaque zone possède un nombre équivalent de véhicules libres.

Avant d'implémenter les premiers véhicules, le conseil d'administration désire réaliser une étude de la viabilité du système. Pour cela, il vous est demandé de réaliser une application permettant la simulation de différentes stratégies, suivant le nombre d'utilisateurs potentiels et le nombre de véhicules de la flotte.

Le porteur du projet au sein de l'entreprise, diplômé de l'École Polytechnique, se rappelle des notions de structures discrètes et vous conseille de modéliser le problème à l'aide d'automates et de machines à états. Lors de votre rencontre, il vous expose le projet et vous rappelle les concepts liés.

- **Implémentation des zones**

  L'agglomération est découpée en plusieurs zones, chaque zone regroupant un nombre quelconque de quartiers (reconnus par leur zipcode). Les zones sont disjointes deux à deux (vous n'avez pas à le vérifier). Pour chaque zone, l'ensemble des quartiers représente un lexique qui peut être organisé sous la forme d'un automate à états finis, où chaque arc correspond à un caractère du zipcode, et tel que tout zipcode complet est un état final. Par exemple, le lexique {H2V, H3V} est représenté par l'automate de la figure 1.

  L'objectif est ici de construire un automate par zone, qui est parcouru par l'application lorsqu'un utilisateur requiert un véhicule à sa position actuelle. L'automate vérifie alors si la position actuelle appartient bien à sa zone : si c'est le cas, l'application envoie un véhicule de ladite zone à la position de l'utilisateur.

  Note : il est bel et bien demandé d'implémenter un automate, et pas une simple recherche de chaîne de caractères dans un tableau de données.

- **Choix du véhicule à envoyer**

  À cette étape de l'exécution, le programme sait dans quelle zone est l'utilisateur. Il vérifie alors si, parmi les véhicules libres de la zone considérée, il y en a un qui est dans le voisinage de l'utilisateur – c'est-à-dire, dont le zipcode est le même que celui de l'utilisateur. Si ce n'est pas le cas, un autre véhicule *libre* de la zone est appelé.

- **Équilibrage des zones**

  Un utilisateur n'est jamais forcé de se déplacer uniquement dans sa zone : sa destination peut tout à fait être à l'autre bout de la ville. Ceci peut amener un fort déséquilibre entre les zones en terme de flotte, jusqu'à mener à une désertion complète d'une zone au profit d'une seconde. Afin

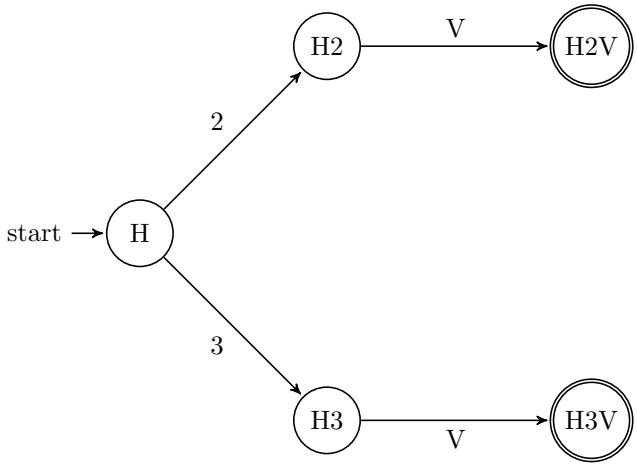

Figure 1: Automate représentant {H2V, H3V}

d'éviter ce problème et assurer un service relativement homogène, un système d'équilibrage est mis en place : si un déséquilibre est remarqué, un véhicule libre d'une zone en surplus doit réaliser le transfert vers une zone désertée. Sa position est alors prise au hasard parmi les voisinages de sa nouvelle zone.

- **Simulation**

  Aucune notion temporelle n'est à considérer pour la simulation. On considère que les utilisateurs réalisent leurs requêtes en même temps, rassemblés en plusieurs groupes. Les membres du groupe 1 réalisent leurs requêtes en même temps, ce qui provoque un changement global et immédiat de la flotte. L'équilibrage de la flotte est alors réalisé après chaque exécution de groupe. On peut passer au deuxième groupe et réaliser de la même manière toutes les requêtes. On compte qu'un véhicule donné a réalisé un trajet s'il a changé de voisinage durant l'exécution d'un groupe d'utilisateurs : on différencie alors les trajets occupés (avec un client) des trajets à vide (réalisé à cause d'un équilibrage).

  Il est possible de ne considérer aucun groupe : on réalise alors les requêtes une à une, en équilibrant la flotte après chaque requête, comme si chaque utilisateur était seul dans son groupe.

5 Composants à implémenter

C1. Écrire une fonction "creerLexiques()" qui permet de lire les fichiers textes correspondants aux zones des véhicules.

C2. Écrire une fonction "equilibrerFlotte()" qui permet d'équilibrer le nombre de véhicules pour chaque zone.

C3. Écrire une fonction "lancerSimulation()" qui démarre la simulation une fois que l'utilisateur a rentré les données nécessaires.

C4. Faire une interface qui affiche le menu suivant :

  (a) Créer les zones.

  (b) Entrer les clients et les véhicules.

  (c) Démarrer la simulation.

  (d) Quitter.

Notes

- La fonction "creerLexiques()" prend comme paramètre le chemin d'un répertoire, et extrait les automates des zones correspondantes aux différents fichiers textes du répertoire. Les éventuelles erreurs de l'utilisateur doivent être évidemment gérées.
- La fonction "equilibrerFlotte()" ne demande aucun paramètre *a priori*. Les véhicules qui changent de zone durant l'exécution de cette fonction doivent récupérer un nouveau voisinage au hasard : cela implique un parcours de l'automate correspondant réalisé par le programme.
- La fonction "lancerSimulation()" peut prendre comme paramètres les données de simulation entrées par l'utilisateur.
- Le programme doit toujours réafficher le menu, tant que l'option (d) n'a pas été choisie.
- L'utilisateur doit entrer un index valide (entre (a) et (d)), sinon le programme le signale et affiche le menu de nouveau.
- L'option (a) demande à l'utilisateur de rentrer les fichiers .txt contenant les lexiques des différentes zones. Le programme doit être capable de lire tous les fichiers .txt contenu dans un répertoire donné. Le chemin de ce répertoire sera donné par l'utilisateur. La manière d'entrer ce chemin est à votre discrétion mais doit être claire pour l'utilisateur.
- L'option (b) permet à l'utilisateur de rentrer les informations concernant les clients et les véhicules. Chaque client doit avoir un point de départ, une destination, ainsi qu'un numéro de groupe. Chaque véhicule doit avoir une zone de départ. Pour cette raison, l'option (b) ne peut être choisie avant l'option (a). L'utilisateur doit pouvoir rentrer autant de personnes et de véhicules qu'il le désire. Il est possible de rentrer les informations sous la forme d'un fichier texte : dans ce cas, le format et l'utilisation sont à votre discrétion, mais doivent être clairs pour l'utilisateur.
- L'option (c) permet de démarrer la simulation. Cette option ne peut être choisie avant l'option (b). Lorsque la simulation est terminée, le programme doit afficher les deux tableaux montrant les résultats de la simulation. Le premier tableau affichera pour chaque véhicule, le nombre de trajets avec un occupant et le nombre de trajets à vide. Le deuxième tableau montrera le nombre de voitures par zones au début et à la fin de la simulation.
- Un utilisateur a comme caractéristiques principales : son origine, sa destination, son numéro de groupe (éventuellement).
- Un véhicule autonome a comme caractéristiques principales : son occupation, sa zone actuelle, sa position actuelle.

Prenez note que selon votre convenance, le mot "fonction" peut être remplacé par "méthode" et vice-versa, et que plusieurs autres fonctions/méthodes peuvent être ajoutées pour vous faciliter la tâche.

6 Livrable

Le livrable attendu est constitué du code source et du rapport de laboratoire. Le livrable est une archive (ZIP ou RAR) dont le nom est formé des numéros de matricule des membres de l'équipe, séparés par un trait de soulignement (_). L'archive contiendra les fichiers suivants:

- les fichiers .cpp;
- les fichiers .h le cas échéant;
- le rapport au format PDF;

L'archive ne doit pas contenir de programme exécutable, de fichier de projet ou solution de Visual Studio, de répertoire Debug ou Release, etc. Les fichiers .cpp et .h suffiront pour l'évaluation du travail.

6.1 Rapport

Un rapport de laboratoire rédigé avec soin est requis à la soumission (format .pdf, maximum 8 pages). Sinon, votre travail ne sera pas corrigé (aussi bien le code source que l'exécutable). Le rapport doit obligatoirement inclure les éléments ou sections suivantes :

1. Page de présentation : elle doit contenir le libellé du cours, le numéro et l'identification du TP, la date de remise, les matricules et noms des membres de l'équipe. Vous pouvez compléter la page de présentation qui vous est fournie.
2. Introduction avec vos propres mots pour mettre en évidence le contexte et les objectifs du TP.
3. Présentation de vos travaux : une explication de votre solution. Ajoutez le diagramme de classes complet, contenant tous les attributs et toutes les méthodes ajoutés.
4. Difficultés rencontrées lors de l'élaboration du TP et les éventuelles solutions apportées.
5. Conclusion : expliquez en quoi ce laboratoire vous a été utile, ce que vous avez appris, vos attentes par rapport au prochain laboratoire, etc.

Notez que vous ne devez pas mettre le code source dans le rapport.

6.2 Soumission du livrable

La soumission doit se faire uniquement par moodle.

7 Évaluation

Éléments évalués	Points
Qualité du rapport : respect des exigences du rapport, qualité de la présentation des solutions, orthographe, grammaire	2
Qualité du programme: compilation, structures de données, gestion adéquate des variables et de toute ressource (création, utilisation, libération), passage d'arguments, gestion des erreurs, documentation du code, etc.	2
Composants implémentés : respect des requis, logique de développement, etc.	
C1	4
C2	4
C3	4
C4	4
Total des points	20

8 Documentation

- http://www.cplusplus.com/doc/tutorial/
- http://public.enst-bretagne.fr/∼brunet/tutcpp/Tutoriel%20de%20C++.pdf
- http://fr.openclassrooms.com/informatique/cours/programmez-avec-le-langage-c