École Polytechnique de Montréal
Département de génie informatique et génie logiciel

LOG2810
Structures Discrètes

TP2 : Automates – Machines à états

Soumis par :
Alexandre Hua (1795724)
Ayoub El Asry (1800595)
Gabriel Hélie (1791764)

Groupe 01

2 décembre 2016

Table de matière

Introduction

L'objectif de ce TP est d'appliquer les notions théoriques vues en cours dans un problème concret qui peut se produire dans la vie de tous les jours. Dans ce cas, il est question de se servir des notions de théorie des langues pour résoudre le problème, en particulier les notions de machines à états et d'automate.

La situation que nous sommes confrontés est la suivante :

Nous devons implémenter une simulation d'un système d'autopartage qui est un service qui permet aux clients de louer un véhicule stationné dans un endroit à proximité d'eux. Cela permet à ces gens d'éviter à se payer une voiture. Ce service permet au client de partir de la position de départ du client vers une destination désirée. Pour cela, il faut qu'une des véhicules du voisinage vienne chercher le client. Les quartiers sont identifiés par un zipcode.

La carte où ce service est disponible est répartie selon des zones et des quartiers. Les zones représentent les grands secteurs du lieu en service et les quartiers représentent des parties qui composent une zone. Il n'y a pas de limite du nombre de zone sur la carte et du nombre de quartier par zone. Le nombre de véhicule dans chaque zone doit équivalent en tout temps dans le but de maintenir un équilibre entre plusieurs zones.

Pour répondre aux exigences du contexte, notre application doit être capable de détecter la présence des véhicules stationnés dans les voisinages. Un client commence à partir d'un certain quartier de départ. Pour que le véhicule puisse récupérer le client, ce véhicule doit être dans le même quartier que celui-ci. S'il existe un véhicule stationné dans le même quartier que le client, le véhicule peut directement chercher le client, mais dans le cas contraire, un véhicule du voisinage est appelé pour chercher ce client. L'ensemble de tous les quartiers peut être représentés par un lexique. Ce dernier peut organiser le zipcode de tous les quartiers par un automate dont les arcs représentent un caractère du zipcode. En d'autres mots, il faut modéliser un automate par zone. Bien que dans le cas réel, le client peut décider d'utiliser ce service à n'importe quel moment, mais dans l'intérêt d'une simulation, on se contente d'exécuter des requêtes des clients en vague.

Notre solution

Pour créer notre solution, nous avons tenté de visualiser le problème dans un point de vue de la vie réelle, c'est-à-dire que nous avons tenté de considérer des différents facteurs qui pourraient se produire lors de la simulation. Par exemple, un des facteurs que nous avons considérés est le déplacement optimal des véhicules. Après qu'une vague de client se fait servie, il est temps d'équilibrer les nombres de voiture dans chaque zone. Cependant, on ne peut pas juste déplacer les voitures n'importe comment. Dans un point de vue virtuel, il possible d'équilibrer le nombre de véhicule dans chaque zone simplement en affectant à chaque zone un

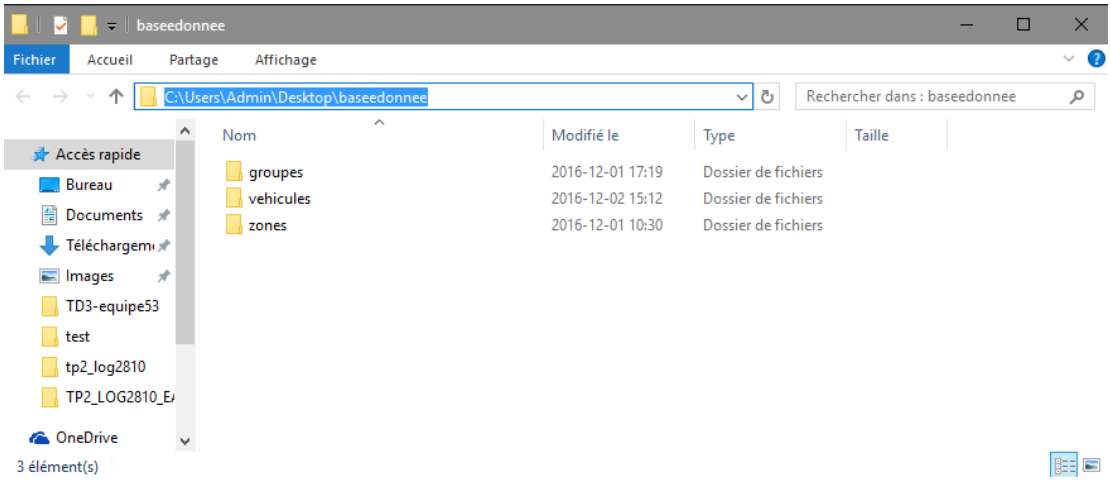
nombre idéal de véhicule un peu comme si on pouvait téléporter un véhicule à la zone appropriée, mais dans la réalité, ces véhicules ont besoin de faire des déplacements. Afin de minimiser les déplacements inutiles des véhicules, nous avons adopté une approche qui permet aux véhicules de rester dans la zone de destination de leur trajet et de ne les déplacer seulement s'il est nécessaire. Notre solution pour équilibrer le nombre de véhicule de chaque zone est la suivante. Tout d'abord, pour un nombre de zone existant et un nombre de véhicule créé, nous déterminons le nombre minimum et maximum de voiture par zone, ainsi que le nombre de zone qui peut accepter le minimum et le maximum de voiture. Pour l'expliquer plus concrètement, analysons l'exemple suivant. La carte possède 10 zones et 24 véhicules a été créés. Si nous répartissons correctement, il y aura 4 zones qui possèdent un nombre minimum de véhicule, soit 2, et 6 zones qui possèdent un nombre maximum, soit 4. Au moment de l'équilibrage, les zones dont le nombre de véhicule est supérieur au maximum envoient des véhicules aux zones dont le nombre de véhicule est inférieur au minimum. De plus, on respecte le nombre de de zone maximum et minimum. Nous commençons par combler les zones minimums, mais lorsque nous avons atteint la limite du nombre de zone minimum permis, nous commençons à combler les zones maximums.

Nous avons également considéré des facteurs importants lors du démarrage de la simulation. Notre but était de créer un algorithme très précis permettant d'afficher toutes les étapes importantes effectuées lorsque divers groupes de clients utilisent notre système. Premièrement, nous devions garder en mémoire toutes les opérations effectuées par chacune des voitures pour afficher le nombre de déplacements effectués par des clients et le nombre de déplacements vides effectués lors du transfert de voitures entre certains quartiers d'une même zone.

Ces informations sont très utiles puisqu'ils nous permettent de savoir si chacune des voitures ont déplacés des clients et ont été transférées à d'autres quartiers lorsque nécessaire. Deuxièmement, puisque notre automate ne contient que les informations finales à la fin (l'état de l'automate se modifie au cours du processus) de la simulation, nous avons dû penser à une alternative. Nous avons donc créé une structure de données de type map (indice = zone et valeur = nombre de véhicules initiaux). À la fin de la simulation, nous affichons deux tableaux différents : un contenant les informations liées aux déplacements des véhicules et le second contenant les informations liées au nombre de véhicule de chaque automate avant et après la simulation.

Concernant la logique, nous déplaçons chaque groupe un à la fois. De cette manière on traite chaque utilisateur d'un même groupe. En premier lieu, on essaye de trouver un véhicule qui se trouve dans le même quartier que le client. Si cette opération n'est pas possible nous vérifions s'il existe un véhicule présent dans la même zone que l'utilisateur. Si cela n'est pas possible alors nous ne pouvons pas traiter la demande de l'utilisateur.

Notes importante

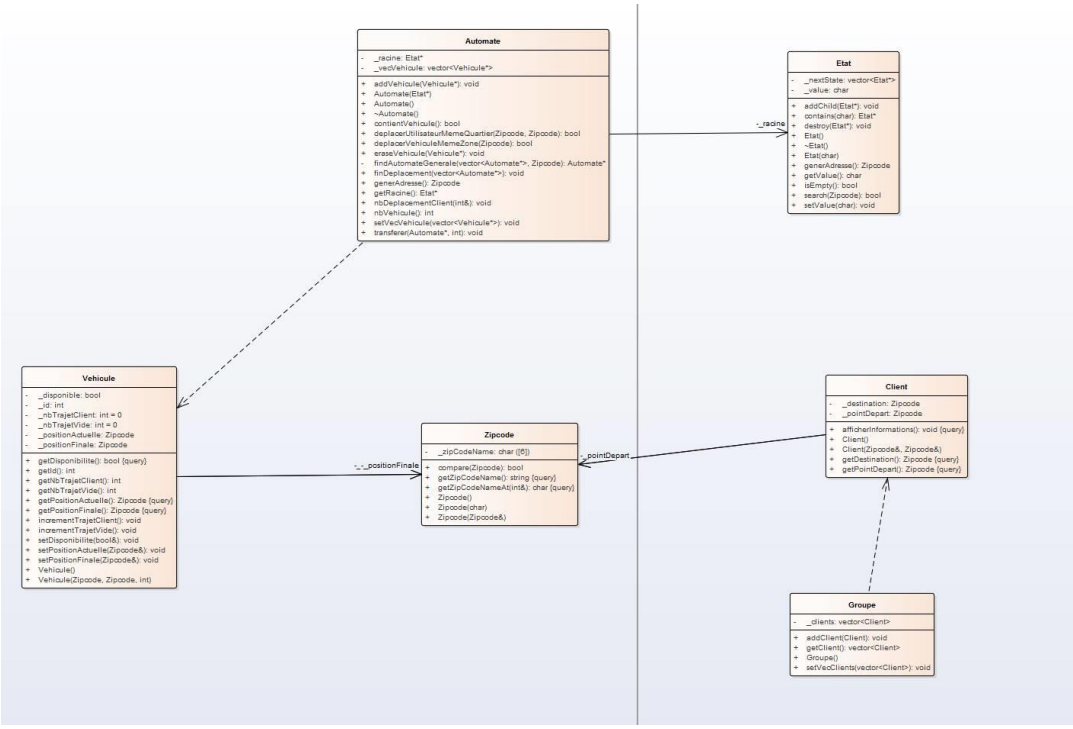


Comme il n'y a pas de format précisé dans l'énoncé, nous avons choisi comme solution d'obliger l'utilisateur de suivre une certaine façon pour placer ses fichiers textes. Tout d'abord, il faut créer 3 dossiers qui portent le nom de «zones», «groupes» et «vehicules» et pour chacun des dossiers, les fichiers doivent portés respectivement les noms «zone», «groupe» et «vehicules». Considérant il peut y avoir plusieurs fichiers dans le même dossier, il faut attacher à la fin du nom un numéro qui s'incrémente et qui commence par 1. Sauf pour le fichier texte «vehicules» qui est unique, donc il ne porte pas de numéro. Un exemple est fourni dans le fichier .zip. Ce dossier doit rester ensemble dans le même répertoire, mais il est possible de les déplacer dans n'importe quel répertoire. Pour effectuer les tests, il est nécessaire de fournir le chemin complet qui mène vers le répertoire où se trouvent les trois dossiers.

Dans le fichier de client, chaque ligne représente un client et il faut mettre le zipcode de départ et le zipcode de destination. Les 6 caractères du zipcode sont collés, mais il y a une espace entre les 2 zipcodes pour les séparer.
Dans le fichier de véhicule, chaque ligne représente un véhicule et il faut mettre le zipcode de sa position initiale.

Comme on peut le voir sur la figure, le lien qui est surligné bleu est le lien qu'il faut coller sur le console en tant qu'entrée.

Diagramme de classe



Difficultés rencontrées

Une des difficultés que nous avons rencontrées est lors de l'implémentation de la méthode pour créer les lexiques En effet, nous avons pensé tout d'abord de créer des états dont chacun représente un caractère unique. De cette façon, cela permet pour les zipcodes dont un même caractère revient souvent de se référencier au même état, ce qui évite de créer trop d'objet. Cependant, nous avons découvert que cette façon était encore plus difficile due au fait qu'il était nécessaire de faire une recherche dans l'arbre complet à chaque fois qu'on ajoute un nouveau zipcode pour vérifier si un état de même caractère avait déjà été créé. Finalement, notre solution à ce problème était d'utiliser une structure d'arbre avec des relations parentales. En d'autres mots, lors de la création de l'automate, pour chaque état où on est rendu lors du parcours, on regarde ses fils. Si l'état ne possède pas encore le caractère pour continuer la construction du zipcode, un nouvel état est créé pour ce caractère et sera ajouté dans les fils de l'état actuel. Cette structure peut créer plusieurs états qui représentent le même caractère, mais cela permet une meilleure performance pour la recherche et la création de l'automate.

Une autre difficulté que nous avons rencontrée est lors de l'implémentation de la méthode pour lancer la simulation. En effet, nous avons dû tenir compte de plusieurs facteurs. Par exemple, nous devons considérer les différents déplacements qu'un véhicule peut effectuer. Un véhicule peut déplacer un client dans un autre quartier ou dans une autre zone et il peut être importé d'un autre quartier, ce qui effectue des déplacements également. Cela permettait de savoir le nombre de déplacement par véhicules. Un autre facteur à considérer est le fait que la disponibilité des véhicules peut changer. Lorsque le véhicule serve le client, celui-ci ne devient plus disponible pour les autres clients avant la fin de la simulation. Donc, à chaque fois qu'un

client se fait servir, nous devons assurer de tenir compte seulement des véhicules qui sont encore disponibles.

Bien que l'interface du menu ne soit pas une chose difficile à implémenter ou à comprendre, mais il y a tout de même beaucoup de facteur d'erreur à considérer. L'utilisateur a la possibilité de créer 3 éléments différents, soit une zone, un client ou un véhicule. Pour chacun de ces éléments, il fallait gérer les erreurs possibles des entrées de l'utilisateur. Dans ce TP, il n'y avait pas de format officiel pour lire les fichiers textes, ce qui nous a fait réfléchir sur la meilleure façon pour non seulement lire les fichiers, mais aussi pour trouver le fichier dans l'ordinateur. Il y a également des gestions d'erreurs d'entrée pour les choix d'option dans les différents menus qui s'affichent. En effet, il fallait considérer le comportement de notre application lorsque l'utilisateur entre des données aberrantes tel qu'un mot ou un nombre au lieu d'un caractère comme il le faut.

Conclusion

En conclusion, nous avons établi un système d'autopartage pour la ville de Montréal. Pour ce faire, nous nous sommes assurés de créer un système qui allait respecter les demandes de plus de clients possibles.

Pour ce faire, nous avons représenté chaque zone de Montréal par un automate contenant plusieurs états. Chaque état contient une valeur, soit un caractère représentant une lettre d'un code postal ainsi qu'un vecteur contenant les prochains états. Nous avons également conçu un algorithme permettant de rééquilibrer les voitures après le déplacement d'une vague de clients afin de maximiser les chances de succès des demandes des clients des prochaines vagues.

Pour tester notre système, nous avons généré des simulations. Dans ces simulations, un utilisateur entre plusieurs données comme le nombre de voitures pour chaque zone ainsi que le nombre de clients pour se représenter une situation réelle.

Ce laboratoire nous a donc permis de nous familiariser avec l'utilisation d'algorithmes afin de maximiser les bénéfices tirés. Il nous a également appris à penser à des contraintes appartenant au monde réel. Si nous avions un autre laboratoire à faire, nous nous attendrions à ce qu'il soit aussi difficile et qu'il nous permette de faire un lien entre la matière vue en cours et la programmation afin de créer une petite application utile pour diverses situations pouvant s'appliquer au monde réel.