LOG2810

Structures discrètes

Polytechnique Montréal

TP2 : Automates et Machines à États

Gabriel Bourgault - 1794069

Frédéric Hallé – 1802024

6 décembre 2016

# Introduction :

Lors des dernières années, le problème du réchauffement climatique a pris beaucoup d’ampleur et de plus en plus de gens essaient de faire attention et porter des gestes plus écologiques. Ils cherchent à minimiser leur empreinte écologique. Le transport en commun fait partie de ces actions prises par les citoyens de la terre. Cependant, pour certains, parfois en régions éloignées, le transport en commun n’est pas la meilleure solution. Le transport collectif en région, n’est pas aussi développé, ni aussi flexible. La voiture reste donc bien souvent la meilleure solution. Une solution existe tout de même pour ceux qui se soucient de l’environnement et qui ont tout de même besoin de la voiture. Il s’agit de services d’autopartage. Ces services permettent de fournir un véhicule aux usagers selon leur demande. Ils peuvent donc ainsi minimiser les coûts liés à l’utilisation d’une voiture, puisqu’ils n’en sont pas propriétaires, et du même coup, porter des actions dans la bonne direction pour aider la planète.

Dans le cadre du cours de Structures discrètes, nous devons concevoir un système d’autopartage qui permettra aux citoyens d’une ville de pouvoir louer un véhicule pour effectuer leurs déplacements. Ce système, combiné aux voitures autonomes permettra aux usagers d’avoir la voiture à leur porte quand ils en auront besoin.

Nous verrons tout d’abord un aperçu de la tâche que nous avons à réaliser, en prenant soin de noter les particularités et éléments requis de notre système. Nous parcourrons certaines conditions de fonctionnement de ce système, qui sera suivi par l’explication du fonctionnement des automates et machines à états dans notre système et comment nous avons exploité certaines propriétés. Nous verrons aussi, par après les fonctions que nous devions obligatoirement implémenter dans notre système. Leur fonctionnement sera démontré avec un diagramme de classe, qui nous permettra aussi du même coup de comprendre la structure et le fonctionnement de tout ce système. Finalement, nous présenterons les principales difficultés que nous avons rencontré ainsi que les méthodes employées pour les surmonter.

# Présentation des travaux

## Énoncé de la tâche à réaliser :

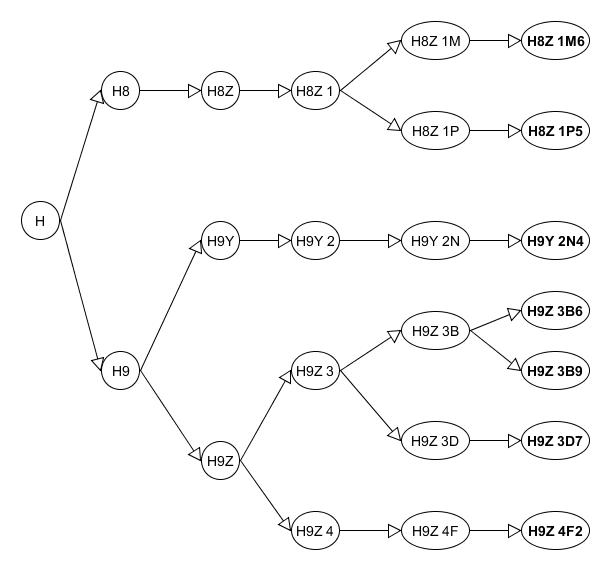
L’objectif de ce travail est de réaliser un test de viabilité dans le cadre de la mise en service d’un service d’autopartage pour la ville de Montréal. Tout d’abord, la ville est découpée en différentes zones. Chaque zone est ensuite sous-divisée en voisinages, dont chacun est défini par son code postal. Chacun de ces codes postaux sont présentés dans les fichiers zone1.txt à zone3.txt.

L’idée à la base de ce système est de toujours avoir une voiture disponible pour un usager. Bien entendu, pour minimiser le nombre de clients mécontents et maximiser le nombre de consommateurs satisfaits, il faut offrir un service de qualité. Ce service de qualité passe tout d’abord par l’accès aux véhicules. Il doit donc exister en permanence, un certain équilibre du nombre de véhicules disponibles, dans toutes les zones de la ville.

En effet, lorsqu’un utilisateur fait une demande pour un véhicule, un véhicule libre stationné dans son voisinage lui est acheminé. Advenant le cas où il n’y aurait pas de véhicule libre dans son voisinage actuel, ce sera donc un véhicule libre dans la même zone qui sera livré. Si par malheur, en grosse période d’achalandage, aucun véhicule n’est disponible dans la zone, alors la demande du client sera refusée. Le refus d’une demande peut avoir des impacts non seulement économiques, mais aussi sur la réputation de la compagnie du système. Afin d’assurer la meilleure expérience client possible, nous avons donc comme objectif d’équilibrer le plus possible le nombre de véhicules libres entre les zones.

Pour mettre en place un tel système, il faut tout d’abord s’assurer de compléter certaines tâches. Par exemple, la première consiste à implémenter les zones. Il faut s’assurer de créer un modèle qui soit facilement interprétable par l’humain, tout en étant facile à implémenter dans un contexte de programmation.

Comme mentionné plus tôt, la ville est séparée en zones. Chaque zone regroupe plusieurs quartiers, qui chacun, peuvent être associés à un code postal. C’est ce code postal qui nous permettra de déterminer dans quelle zone le client se trouve actuellement Nous fonctionnerons avec un système d’automates et de machines à états pour déterminer dans quelle zone le client se trouve actuellement. À l’aide de ce code postal, nous pouvons créer un automate, ou les états finis correspondent à un code postal complet. Les arcs correspondent aussi à chaque caractère du code postal. On pourrait donc représenter le lexique suivant : {H8Z 1M6, H8Z 1P5, H9Y 2N4, H9Z 3B6, H9Z 3B9, H9Z 3D7, H9Z 4F2} avec l’automate suivant :

Ainsi, nous générons un automate de ce genre par zone. Cependant, chaque zone contient bien plus que 7 codes postaux. On parle plus d’environ 25 par zone. On peut s’imaginer la complexité d’un automate pour 25 éléments, sachant que celui-ci n’en contient que 7.

Une fois l’automate généré avec le lexique qui nous a été fourni, et qu’un nouveau client fait une demande pour avoir un véhicule, il faut déterminer quel véhicule lui acheminer selon sa localisation qu’il connait maintenant. Si un véhicule est libre et dans le même voisinage que le client, cette voiture lui est directement acheminée. Si aucune voiture n’est disponible dans le voisinage actuel, l’usager recevra une voiture qui provient d’un autre voisinage, mais tout de même de la même zone. Dans cette situation, un cas peut sembler négligé. Le cas où il n’y aurait pas de voiture disponible dans la zone. Ce cas est couvert par l’algorithme d’équilibrage.

Figure 1 - Automate qui représente les codes postaux suivants : {H8Z 1M6, H8Z 1P5, H9Y 2N4, H9Z 3B6, H9Z 3B9, H9Z 3D7, H9Z 4F2}. Veuillez prendre note que les états en **gras** sont les états finaux

Comme mentionné, l’algorithme d’équilibrage permet d’éliminer le cas où il n’y aurait pas de voiture disponible dans la zone. Ce cas pourrait se produire, car les utilisateurs ne sont pas limités à leur zone une fois qu’ils sont au volant du véhicule. Ils peuvent sortir de la zone, ce qui pourrait débalancer les zones, laissant ainsi des zones avec aucun véhicule disponible. On peut comparer cet algorithme à celui des Bixi de Montréal. Le système doit s’assurer qu’il y ait toujours de vélos disponibles pour emprunter aux diverses stations, mais aussi qu’il y ait des places disponibles pour que les utilisateurs qui en ont terminé puissent le remettre. La façon de procéder est relativement simple. Dès qu’un déséquilibre est remarqué dans une zone, un véhicule provenant d’une zone où il y a plusieurs véhicules disponibles est acheminé vers cette zone en déficit.

Il est important de noter que nous ne fixons pas de restrictions par rapport au temps. En effet, lorsque nous exécutons la simulation, nous prenons pour acquis font tous leurs requêtes de véhicules en même temps. Pour faciliter la gestion de l’exécution de la simulation, nous séparons les clients en différents groupes. Chaque groupe fait donc ses requêtes en même temps, ce qui crée forcément un déséquilibre au niveau de la flotte de véhicules. Il faut donc équilibrer la flotte après que chaque groupe ait fait ses requêtes. Nous avons aussi la possibilité de considérer chaque utilisateur de façon indépendante en les mettant tous dans un groupe différent.

## Notre solution :

Tout d’abord, pour élaborer du mieux possible ce système, nous avons quelques conditions à respecter et quelques classes et fonctions que nous devons obligatoirement utiliser.

La première fonction que nous devons avoir est la fonction « creerLexiques() », qui permet de lire les fichiers qui nous sont fournis et qui contiennent les codes postaux de chaque zones. //TODO : Continuer d’ajouter dla sauce

Nous devons aussi avoir une fonction « equilibrerFlotte() ». C’est cette fonction qui sera exécutée chaque fois qu’un groupe de clients commandera un véhicule. Comme son nom l’indique, cette fonction fera équilibrer le nombre de véhicules dans chaque zone de la ville.//TODO : ajouter dautre sauce sur le fonctionnement de cette fonction

Il faudra aussi une fonction « lancerSimulation() » qui exécutera la simulation. L’usager doit entrer toutes les informations nécessaires au préalable. Pour ce faire, il devra passer par les différents menus. Ces menus permettront tout d’abord de créer les zones dans la ville. Cette option appellera donc la fonction qui fait la lecture des fichiers et qui lit les codes postaux. Il sera ensuite nécessaire d’entrer les clients, ainsi que les véhicules. Dans ce menu, nous demandons tout d’abord à l’utilisateur quel est son point de départ, sa destination, ainsi que son numéro de groupe. Les voitures ont ensuite aussi une zone de départ. Un détail important ici, est de s’assurer d’exécuter la première partie pour créer les zones avant la seconde, car ces zones sont nécessaires pour valider que l’utilisateur entre des données valides. Une fois toutes ces informations complétées, il est possible d’exécuter la simulation. Ces options seront donc représentées dans un menu semblable à celui ci-contre : //TODO : ajouter le menu

## Diagrammes de classes :

# Difficultés rencontrées et solutions apportées :

## 

# Conclusion :

Pour conclure, notre système est fonctionnel et permet de déterminer les différents parcours possibles, tout en s’assurant que la voiture ne tombe jamais en panne. Nous avons été en mesure d’implémenter les fonctions obligatoires, *creerGraphe()*, *lireGraphe()*, *extractionGraphe()* et *plusCourtChemin()*. Nous sommes aussi en mesure de modifier les paramètres de la voiture et d’utiliser cette dernière dans plusieurs régions en modifiant la carte de l’environnement. La structure du système a aussi été pensée et visualisée avec un diagramme de clases qui est aussi fourni en annexes. Nous avons discuté des principales difficultés que nous avons rencontré et trouvé une façon de régler le blocage pour arriver à une solution. Nous croyons rencontrer toutes les exigences de ce système. Une amélioration qui serait bien à apporter à ce projet pour le pousser davantage, serait de l’intégrer aux systèmes de covoiturage tels que Uber au moment d’assigner une course à un conducteur, pour s’assurer que le véhicule aura une assez grande autonomie pour porter le client à sa destination sans avoir à arrêter faire le plein.

# Annexe :