POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

TP1 : Graphes

Rapport de laboratoire

Présenté à
Loïc Lerat

Par
Samuel D'Amours-Fortier - 1810585
Justine Lambert - 1846574
Julie Rousseau - 1846102

Dans le cadre du cours
LOG2810 : Structures discrètes

Département de génie informatique et génie logiciel
Polytechnique Montréal

4 mars 2018

Introduction

Dans le cadre du cours, ce premier travail visait à faire le pont entre la théorie et la pratique en proposant une application informatique de la théorie des graphes ainsi qu'une implémentation de l'algorithme de Dijkstra. La mise en situation fournie présentait la problématique suivante.

Alors qu'un étudiant se prépare à réaliser une suite de vols de banque à travers le Canada, il cherche à optimiser son trajet de manière à minimiser son temps de déplacement et ainsi ses chances d'être arrêté par la police. Différents facteurs sont toutefois à prendre en considération, notamment la consommation d'essence en fonction du véhicule choisi et l'argent dépensé selon la compagnie de location sélectionnée (CheapCar, qui est moins dispendieuse mais dont les véhicules consomment plus d'essence, ou SuperCar, dont les véhicules disposent de plus d'autonomie, mais à moins bon prix).

Pour aider cet étudiant, nous disposons d'un fichier texte qui modélise un graphe représentant les différentes villes canadiennes (sommets) ainsi que le temps de parcours en heures entre chacune d'elles (arcs). Il nous était donc demandé de concevoir un programme permettant de générer un graphe à partir du contenu du fichier, de faire l'affichage de ce graphe en console et surtout, d'implémenter un algorithme permettant d'obtenir le plus court chemin à suivre par notre braqueur de banques. L'application permet à l'utilisateur de faire un choix à partir d'un menu en console.

Les sections suivantes présentent les détails de la conception et l'architecture logicielle de notre solution, puis exposent les difficultées rencontrées et leurs solutions.

Présentation des travaux

Avant de développer une application, il est primordial d'en faire la conception afin d'éviter les erreurs et d'oublier des aspects importants. C'est une étape d'organisation qui nécessite un certain temps afin de minimiser les problèmes de programmation et de logique. Au départ, nous avons défini les principaux domaines à modéliser. Ensuite, nous avons étudié chacun des cas pour en définir le comportement et les relations. Une description de chacun des domaines est abordée dans les prochaines lignes.

Véhicule

Pour faire un braquage, l'utilisateur a besoin d'un véhicule pour se déplacer entre les villes. Plusieurs types de véhicules sont disponibles: une automobile, un pick-up et un fourgon. Chaque véhicule est loué chez une compagnie donnée et possède une quantité d'essence restante. Ces informations sont importantes pour l'algorithme permettant de trouver le plus court chemin. Donc, une classe abstraite était nécessaire pour représenter les véhicules pour maximiser la réutilisation de code entre les classes liées au véhicule utilisé. Toutes les

méthodes dans les classes de véhicules servent à gérer les informations de base. La figure 1 démontre la structure utilisée pour représenter les véhicules disponibles.

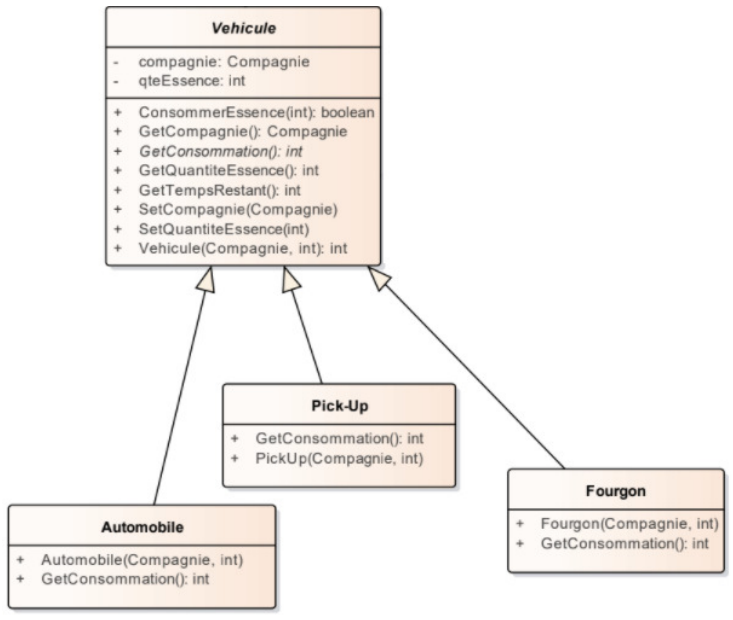

Figure 1. Diagramme de classes pour la section véhicule

Pour poursuivre, la classe Véhicule est la classe abstraite héritée par les classes Automobile, Pick-Up et Fourgon. Effectivement, une classe a été créée pour chaque type de véhicule.

Compagnie

Comme il a été mentionné ci-dessus, le véhicule est loué d'une compagnie. Deux choix sont utilisé pour ce faire: les entreprises SuperCar et CheapCar. Toutefois, les données de consommation pour un véhicule diffèrent entre les compagnies. Il était donc nécessaire de différencier les compagnies pour utiliser les bonnes données dans le système. La figure 2 représente l'architecture utilisée pour les compagnies.

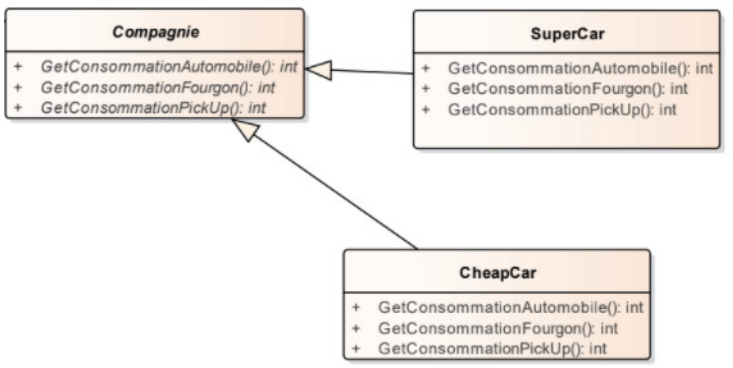

Figure 2. Diagramme de classes pour la section compagnie

L’abstraction a ici aussi été l’option choisie pour modéliser les compagnies. Les classes SuperCar et CheapCar contiennent les données pour les consommations d’essence respectives et héritent de la classe abstraite Compagnie. La classe Véhicule est composée d’une compagnie pour que le véhicule ait connaissance de sa consommation.

En ce qui concerne la consommation, nous avons décidé de la modéliser à l’aide d’un pourcentage. Par exemple, pour démontrer 7% de consommation automobile, la méthode GetConsommationAutomobile retourne la valeur 7 et ainsi de suite pour les autres méthodes de la classe Compagnie.

Villes et Routes

Dans un graphe, il y a des sommets et des arcs. Un sommet est lié à des arcs et un arc est lié à deux sommets, un de départ et un d'arrivée. Dans le cadre de ce système, les sommets sont les villes et les arcs sont les routes. Pour faciliter l’algorithme pour trouver le plus court chemin, les villes connaissent toutes leurs routes, dont elles sont un sommet, alors qu’une route connaît uniquement sa ville finale. De ce fait, les deux villes d’une route sont connues vu que la ville initiale est la ville qui contient la route et la ville finale est la ville dans la classe Route. La figure 3 exprime ce qui a été utilisé dans l’application.

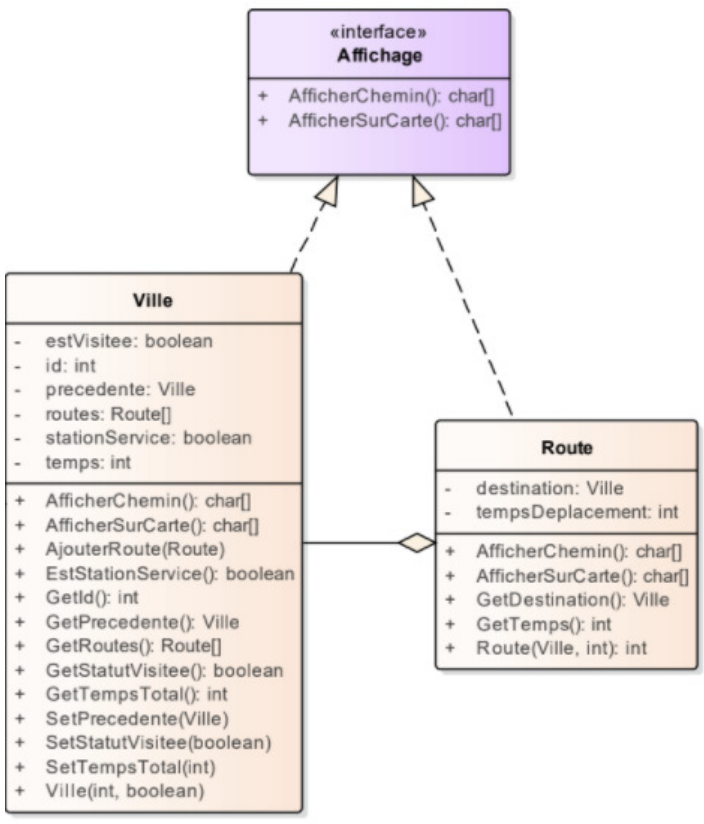

Figure 3. Diagramme de classes pour les villes et les routes

En ce qui concerne la ville, des informations de base doivent être enregistrées pour bien différencier les villes entre elles et pour pouvoir exécuter des algorithmes. Alors, chaque ville possède un identifiant et un drapeau indiquant si une station service s’y trouve. Pour faciliter

l’algorithme qui retourne le plus court chemin, il est nécessaire que la ville connaisse la ville qui a été précédemment visitée, si elle a été visitée ainsi que le temps total par rapport au point d’arrivée.

Pour ce qui est des routes, elles sont associées à une valeur en temps pour le déplacement de la ville initiale vers la ville finale. Comme il a été mentionné plus haut, la route connaît son point final. Des méthodes pour accéder à ses informations sont nécessaires dans le programme.

Dans l’application, deux types d’affichage sont utilisés. Le premier est l’affichage de la carte qui affiche les villes et les routes. Pour ce faire, une méthode AfficherSurCarte a été ajoutée dans les classes Ville et Route. Elles sont appelées en chaîne, c’est-à-dire que c’est la ville qui affiche ses routes et les routes affichent leur temps de déplacement et leur sommet final. Le deuxième affichage est celui du plus court chemin trouvé. Une méthode AfficherChemin est utilisée en chaîne également, mais cette fois-ci, la route affiche son point final. Pour simplifier les implémentations, une interface nommée Affichage a été ajoutée pour obliger la ville et la route à définir ces méthodes d’affichage.

Carte

La carte est le coeur de l’application. Elle fait le lien entre la majorité des domaines de l’application. Les responsabilités de la carte sont d’afficher les villes et les routes et de déterminer le chemin le plus court entre deux villes avec un certain véhicule. La figure 4 représente la modélisation de la carte et du menu de l’application. Le menu est inclus dans le domaine de la carte, car toutes les actions du menu sont redirigées vers la carte.

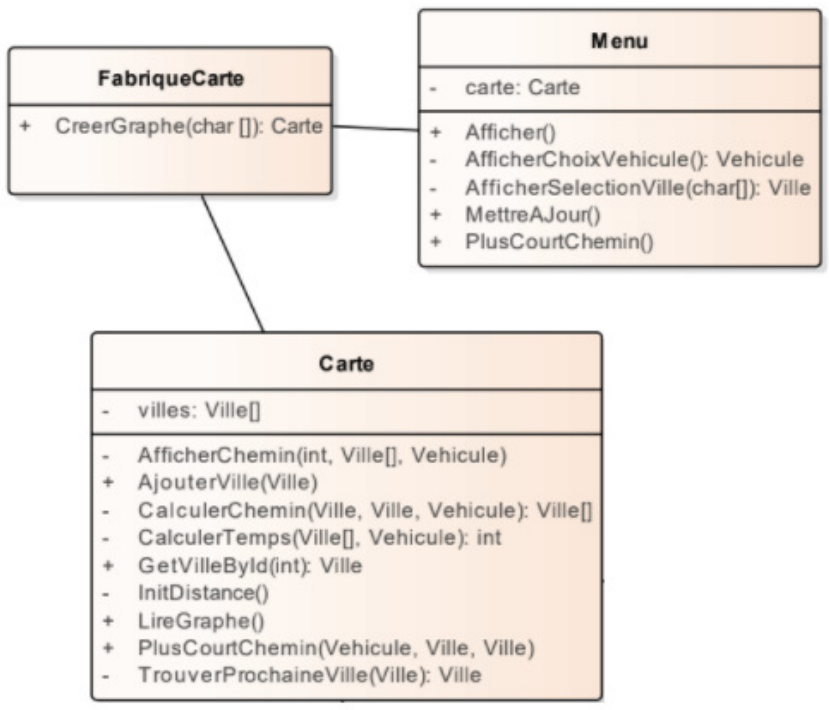

Figure 4. Diagramme de classes pour la section de la carte

La classe FabriqueCarte est inspirée du patron de conception de la fabrique. En effet, il génère une instance d'une carte en lisant un fichier de données pour assigner les informations nécessaires à la carte. Si les informations sont invalides, un message d'erreur est envoyée à l'utilisateur dans la console.

La classe Menu est l'interface utilisée par l'utilisateur. Lors du lancement de l'application, le menu est affiché et permet à l'utilisateur d'exécuter des actions précises: mettre à jour la carte, trouver le plus court chemin entre deux villes et quitter l'application. Pour le bien de l'application, il est nécessaire de mettre à jour la carte avant de trouver le plus court chemin entre deux villes, car la carte n'aura aucune donnée et l'algorithme sera dans l'impossibilité de s'exécuter adéquatement. L'accès à la carte est fait via la fabrique de carte et les actions possibles sont redirigées vers la carte générée.

La classe Carte contient l'affichage de la carte, c'est-à-dire de toutes les villes et les routes ainsi que l'algorithme pour trouver le plus court chemin entre deux villes avec un véhicule précis. L'affichage de la carte fonctionne en chaîne, comme les affichages des villes et des routes. La carte appelle l'affichage de la ville et celle-ci appelle l'affichage de ses routes.

Ensuite, la méthode pour trouver le plus court chemin est implémentée selon l'algorithme de Dijkstra. Afin de prendre en compte la restriction sur l'essence, l'algorithme parcourt non seulement de façon itérative les villes les plus proches du point d'origine pour trouver la ville suivante la plus proche, mais il vérifie à chaque étape si le véhicule a suffisamment d'essence pour se rendre à la ville suivante. Si l'algorithme échoue à trouver un chemin respectant la contrainte d'essence alors que le véhicule est de la compagnie CheapCar il effectue un nouveau calcul de chemin, cette fois en mettant la priorité sur la quantité d'essence restante selon ce parcours, une ville est donc considérée la suivante sur le chemin si la quantité d'essence est la plus grande qu'en passant par un autre chemin. Ce n'est que si cette nouvelle recherche de parcours échoue que la compagnie du véhicule est changée pour SuperCar et que la recherche du plus court chemin est relancée.

Architecture complète

Finalement, en reliant tous les domaines de l'application, on obtient un diagramme de classes complet qui parcourt tous les sujets nécessaires pour le bon déroulement du programme. La figure 5 de la page suivante représente les liens entre les différents domaines.

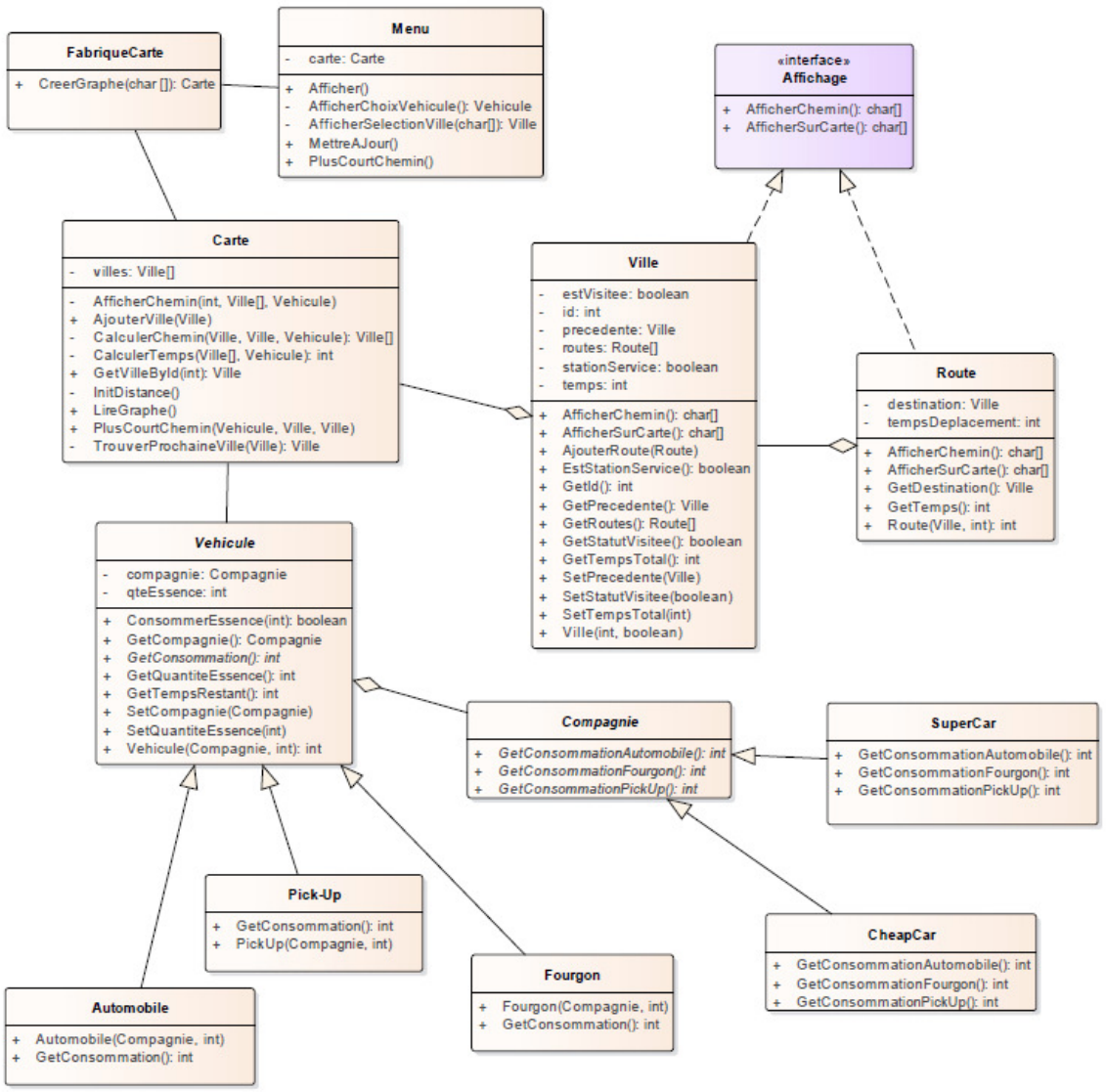

Figure 5. Diagramme de classes complet

Difficultés rencontrées

Lors du processus de conception et de développement, certaines problématiques ont fait surface quant à l'élaboration du programme. Tout d'abord, dès la conception, réaliser l'architecture du logiciel était une tâche qui pouvait paraître ardue. Nous avons donc décidé de diviser le problème en plusieurs blocs de concepts, par exemple une partie Véhicule, une partie Villes et Routes, une partie Carte, etc., de sorte que nous avons pu rapidement obtenir un diagramme de classes cohérent.

Ensuite, nous avons choisi de développer l'application avec le langage Java, que certains membres de l'équipe maîtrisaient moins bien que d'autres langages (par exemple le C++). Nous avons donc dû faire des recherches sur le Java, ses classes ou librairies et ses façons de faire. Par exemple, faire la lecture d'entrées en console et la lecture du graphe à partir du fichier *villes.txt* a posé certains défis. Nous avons donc utilisé les classes BufferedReader

pour obtenir les informations depuis le fichier texte et Scanner pour les entrées de la console. Or, l'utilisation du Scanner a nécessité une légère refactorisation du code du menu principal, puisque nous avions une fuite de mémoire à un certain point.

Puis, un autre défi a été de bien réussir la gestion des erreurs et des exceptions, puisque nous devions penser à tous les cas de figure où une erreur pourrait survenir: à l'ouverture du fichier, lors d'une entrée du mauvais type, lorsque les mauvais paramètres étaient passés aux méthodes, lorsque certains objets avaient une valeur de *null*, etc. Nous avons donc exécuté plusieurs tests afin de couvrir un maximum d'exceptions possibles.

Toutefois, le plus grand défi de ce travail pratique se situait sans nul doute autour de la méthode du plus court chemin. En effet, l'implémentation de l'algorithme a nécessité plusieurs tentatives de notre part et il nous semblait parfois que lorsqu'une difficulté était solutionnée, un autre problème apparaissait. La plus grosse difficulté a été la restriction au niveau de l'essence. Même en vérifiant à chaque route si le niveau d'essence était suffisant pour se rendre à la ville suivante, il arrivait que des trajets soient considérés impossibles avec un CheapCar alors qu'il y avait bien un trajet possible. Le problème venait du fait qu'il existait un chemin plus court mais ne passant pas par une station d'essence et n'ayant plus suffisamment d'essence pour la fin du trajet. Une ville était donc ignorée parce que son temps était plus long même si un trajet aurait été possible en passant par cette ville. C'est pour résoudre ce problème qu'un nouveau chemin est cherché en donnant la priorité à la quantité d'essence si aucun chemin n'est trouvé selon le temps seulement.

Nous sommes conscients qu'il y a encore place à l'optimisation de notre algorithme. Il y a entre autres beaucoup de calculs effectués pour connaître la quantité d'essence à chaque point du trajet, puisque nous avons décidé d'en laisser la responsabilité à la classe experte qu'est le véhicule. Il y a aussi la possibilité qu'un trajet plus court soit ignoré parce qu'une ville n'ait été considérée qu'avant le passage à une station service et non après. Une solution potentielle à cela aurait été d'implémenter un autre algorithme, celui de Yen par exemple, pour calculer plusieurs chemins potentiels, de vérifier la faisabilité selon l'essence et de prendre le plus court chemin parmi ceux-ci.

Conclusion

En bref, le développement de ce programme d'optimisation a fait en sorte de mettre en pratique les notions d'algorithmique et de théorie des graphes vus en cours, en appliquant un algorithme inspiré de Dijkstra à un problème d'optimisation basé sur le parcours d'un graphe. Fort utile, le travail pratique nous a donc permis de passer de la théorie à la pratique et d'apprendre à faire le traitement logiciel des graphes, en plus d'améliorer notre maîtrise du Java.

Nous avons trouvé la charge de travail raisonnable, malgré que l'algorithme du plus court chemin a tout de même exigé beaucoup de temps de développement et de tests à cause des nombreuses contraintes à considérer. Ainsi, nous nous attendons à un travail pratique similaire pour le second laboratoire, soit qui nous permet d'appliquer un algorithme à une situation de la vie réelle.