A black sign with white text

Description automatically generated

Paul Clas 1846912

Mazigh Ouanes 1721035

Simon Richtot 1637043

**Rapport du TP1 : GRAPHES**

LOG 2810 Structures Discrètes

**Présenté à :**

John Mullins

Justine Pepin

Paulina Stevia Nouwou Mindom

**Remis le**

3 mars 2019

**Département de génie informatique et génie logiciel**

**Polytechnique Montréal**

1. **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.

Avec l’utilisation de plus en plus massive des réseaux sociaux, il est pertinent pour tout ingénieur informatique d’avoir une compréhension approfondie de leur fonctionnement. Or, les réseaux sociaux, dans leur fondement, sont tout simplement des graphes dont les sommets représentent des individus et dont les arcs représentent les relations entre ces individus. Savoir manipuler les graphes de connexions serait donc un atout non seulement en entreprise, mais aussi en environnement de recherche, puisque c’est un domaine aux multiples applications. Le jeu de société « Guess who ? », aussi connu sous le nom « Qui est-ce ? » en français, nous servira de moyen d’apprentissage pour la manipulation des graphes et l’application du raisonnement déductif. Il vous est donc demandé d’implémenter un agent pouvant jouer à ce jeu.

La théorie des graphes est une branche des mathématiques discrètes. Cette branche est entre autres utilisée dans le domaine des réseaux sociaux. En effet, les données étudiées comprennent un ensemble de nœuds et entre ceux-ci des liaisons que l’on nomme aussi arcs. Un nœud est simplement un objet, c’est à dire que pour un réseau social, il s’agira d’individu, mais en télécommunication il pourrait s’agir d’ordinateur connecté à un réseau.

Les travaux pratique ont pour objectif de mettre en pratique les notions théoriques acquisses en cours. Dans le cadre de ce travail pratique, les notions pratiquées se rapportent au second chapitre du cours soit sur la théorie des graphes.

Afin de savoir manipuler les graphes de connexions, on va créer une petite application qui jouera au jeu “Guess who?”. Le joueur(adversaire) va interagir avec l’agent(programme) qui effectue principalement deux taches soit : d’identifier les deux personnes mystères du joueur et de déterminer le lien le plus court entre ces deux individus mystères. De manière un peu plus détaillé, l’agent posera une série de question afin de cerner deux individus préalablement sélectionner par l’humain jouant au jeu et utilisera l’algorithme du chemin minimal de Dijkstra.

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.

1. **Présentation de nos 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.

Lors de l’élaboration de l’application, nous avons procédé au codage en utilisant une approche orientée objet. Cela a rendu la tâche plus simple pour mettre en évidence les fonctionnalités à implémenter. La solution contient donc les quatre classes suivantes : Vehicule, Graphe, Arc et Sommet.

**PRÉSENTÉ LES FICHIERS ICI**

* 1. **Diagramme de Classe**

1. **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.

Parmi les différentes méthodes à implémenter, la méthode plusCourtChemin fut définitivement la plus compliquée à implémenter, malgré le fait qu’il soit relativement simple de comprendre l’algorithme de Djikstra. Cela étant dû la récursivité de la méthode qui rend rapidement l’algorithme à trouver plus difficile. Bien sûr, il y eut aussi les multiples bugs tout au long du TP qui ont augmenté la durée du travail, malgré qu’un grand nombre de ces bugs aient été des erreurs d’inattention, réglées à l’aide de l’outil de débogage intégré à Visual Studio.

* 1. **Solutions apportées**

1. **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.

1. **Annexe**
   1. **Temps passé sur le TD**