Travail pratique #1

LOG2810 - Structures discrètes

Trimestre : Automne 2019

Alice GONG 1961605

Nu Chan Nhien TON 1961458

Kai Sen TRIEU 1963091

Groupe: 05

Présenté à :
Jean-Philippe Anctil

École Polytechnique Montréal
Remis le 5 novembre 2019

INTRODUCTION

Ce premier travail pratique avait pour objectif d'aider les étudiants du cours LOG2810 à mieux assimiler la matière enseignée dans le cadre de ce cours, à l'aide d'un exemple de mise en situation. Pour être plus précis, la problématique présentée dans l'énoncé s'agissait de trouver le chemin le plus efficace que pouvait entreprendre une flotte de drones pour ramasser des paquets. Ce cas hypothétique faisait donc allusion à la théorie des graphes, ainsi qu'à l'algorithme de Dijkstra.

Par ailleurs, l'étudiant était également demandé à implémenter un total de quatre composantes permettant de traiter différentes facettes de la problématique; elles seront décrites plus concrètement dans les paragraphes qui suivront. Certaines fonctions à implémenter devront se servir d'un fichier texte intitulé « entrepot.txt », dans lequel les sections de l'entrepôt a été représentés sous forme d'un graphe.

Nos démarches seront présentées dans les prochaines sections, notamment par l'entremise d'une explication de notre solution, ainsi que des difficultés rencontrées lors de ce laboratoire.

EXPLICATION DE LA SOLUTION

Comme mentionné antérieurement, quatre composantes principales (C1, C2, C3 et C4) ont dû être implémenté:

- C1 se composait d'une fonction permettant de créer le graphe modélisé dans le fichier texte " entrepot.txt " et d'une seconde ayant pour but d'afficher ce dernier;
- Quant à la deuxième composante, celle-ci devait permettre de prendre la commande de l'utilisateur, et de l'afficher par après;
- Pour C3, l'étudiant devait implémenter une fonction pour déterminer le chemin le plus optimal que peut prendre un certain drône pour ramasser une commande;
- Puis, C4 s'occupait de la partie interface du programme.

Pour compléter ces tâches, il nous semblait logique d'utiliser le concept d'orienté objet; nous avons donc décidé d'utiliser le langage Java pour notre programme. Aussi, puisque nous étions plus habiles avec l'environnement Intellij qu'avec Eclipse, nous nous sommes servies de ce dernier.

Qui dit programmation orientée-objet, dit l'usage de classes; les fonctions à implémenter de chaque composante ont donc été séparées sous forme de neuf classes, soit « Arc », « Commande », « Dijkstra », « Graphe », « RobotX », « RobotY », « RobotZ » et « Sommet », accompagnées d'une classe « Main » également. Le nom de chacune est plutôt explicite.

Le contenu (attributs et méthodes) de chaque classe s'illustre dans le diagramme de classes retrouvés ci-dessous.

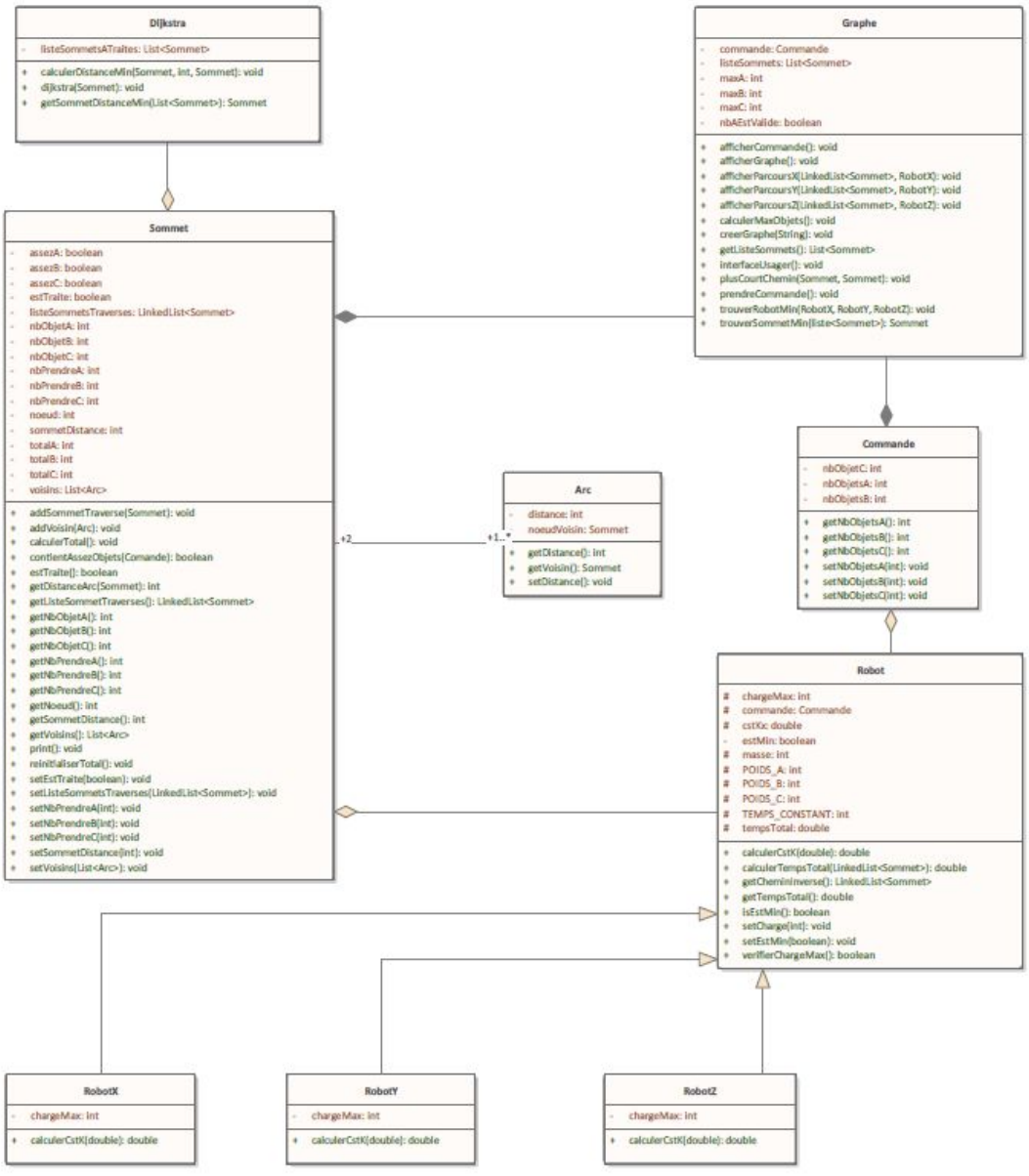

Figure 1. Diagramme de classes représentant notre programme.

Chaque Sommet possède une liste d'Arcs, qui contient le Sommet voisin associé à cet Arc, ainsi que la distance entre ces derniers. La classe Dijkstra permet d'effectuer les calculs pour déterminer les différentes distances minimum pour atteindre chaque Sommet. Notre classe Graphe est celle qui contient toutes les grosses méthodes des composantes, tel que l'affichage de l'interface, la création du graphe, l'affichage de ce dernier, la prise de la commande de l'usager, l'affichage de cette commande, l'affichage du chemin le plus court pour respecter la commande ainsi que la fonction quitter qui permet de fermer l'interface. Les méthodes nécessitant plus d'éclaircissement ont été brièvement décrites dans les sections suivantes.

Classe « Arc »

Un arc contient le Sommet voisin qui lui est associé ainsi que la distance entre le Sommet initial et son Sommet voisin.

Classe « Commande »

La classe Commande possède comme attribut le nombre d'objets A, B et C à ramasser pour l'usager. Elle possède également toutes les méthodes getters et setters associées à ces attributs.

Classe « Dijkstra »

Cette classe contient les méthodes permettant de déterminer, à partir d'un noeud d'arrivée, toutes les distances minimum pour se rendre à chacun des Sommets contenus dans un graphe. L'algorithme commence par traiter le noeud de départ et traite ensuite chacun de ses voisins, et ainsi de suite jusqu'à ce que tous les noeuds sont traités, et leur distance minimum mise à jour.

Classe « Robot »

Robot est une classe abstraite. Chaque robot possède une constante K, une charge maximum que ce robot peut transporter, une masse accumulée des objets qu'il a ramassés jusqu'à présent, ainsi qu'une certaine commande contenant les objets qu'il doit aller ramasser. Les différentes méthodes dans cette classe permettent de vérifier si la commande associée à ce robot respecte la charge maximum, ainsi que calculer le temps total que prendra le Robot pour parcourir un chemin donné.

Classes « RobotX », « RobotY », « RobotZ »

Ces trois classes héritent tous de la classe Robot et l'instancie. Ils diffèrent par leur charge maximum qu'ils peuvent tenir et par leur constante K qui est propre à chacun.

Classe « Sommet»

Un sommet contient une liste d'Arc qui lui est associé, comportant ainsi la liste de tous ses voisins ainsi que la distance correspondante. Il possède aussi comme attribut une LinkedList<Sommet>, qui regroupe le chemin des différents Sommet parcourus pour arriver ce ce Sommet selon l'algorithme de Dijkstra. Il contient également des attributs indiquant combien d'objets A, B et C, ainsi que combien de ces objets seront pris (s'il y a lieu) de ce Sommet lors du chemin parcouru. Les différents getters et setters de tous ces attributs sont implémentés dans la classe.

Classe « Graphe»

Finalement, la classe Graphe regroupe toutes les fonction en lien avec le Graphe traité lors du travail pratique, ainsi que l'interface. Elle possède comme attribut une liste de Sommets contenus dans le graphe complet, ainsi qu'une Commande, soit celle rentrée par l'usager. La première méthode créerGraphe() permet de prendre en paramètre un fichier texte contenu dans le même path que le reste des fichiers .java, et de le déchiffrer et de stocker toutes les données contenues à l'intérieur pour pouvoir afficher un Graphe. Nous avons utilisé la classe Scanner pour lire le fichier .txt et des pattern pour séparer les différents strings en des valeurs respectives. La classe contient également une méthode permettant d'afficher le graphe venant d'être créé à l'aide du fichier .txt, en affichant tous les noeuds ainsi que ses voisins et la distance entre eux. Une autre méthode permet de lire des inputs de l'usager et de prendre sa commande, cest-à-dire du nombre d'objets que le Robot doit récolter sur son chemin, ainsi que d'afficher cette commande. À chaque fois que prendreCommande() est appelée, tous les attributs étaient réinitialisées pour s'assurer qu'une ancienne Commande n'affecte pas la nouvelle. La méthode plusCourtChemin() utilise l'algorithme de Dijkstra pour trouver les distances minimum pour chaque Sommet, ainsi que d'initialiser tous les Robots avec la Commande de l'usager. Il finit ensuite par comparer les temps des différents Robots pour déterminer lequel est optimal pour cette Commande. Les différentes fonctions afficherParcours() permettent d'afficher le chemin optimal selon s'il est un RobotX, RobotY, ou RobotZ. Finalement la méthode interfaceUsager() permet d'afficher l'interface et de permettre à l'usager de décider selon 6 options : créer un graphe, afficher le graphe, prendre une

commande, afficher la commande, afficher le parcours optimal et finalement de quitter l'interface et de fermer le programme. Nous avons utilisé une switch case pour permettre de facilement différencier les choix de l'usager.

Finalement, dans le main, il suffit de créer un objet Graphe et d'appeler la méthode interfaceUsager() pour pouvoir afficher l'interface et permettre à l'usager de choisir l'option de son choix.

Nous avons décidé d'évaluer la distance minimum de chacun des noeuds à l'aide de l'algorithme de Dijkstra. Par la suite, nous avons calculé le nombre d'objets A, B et C accumulés au long de tous ces chemins parcourus. Notre stratégie était de parcourir le chemin sans rien ramasser et de tout ramasser lors du retour pour ainsi réduire la masse du robot pendant ce trajet et diminuer le temps total que prendra le robot pour faire ce chemin. Ensuite, nous avons comparé les différents temps des RobotX, RobotY et RobotZ pour déterminer lequel était le plus court. C'est alors celui qui était choisi et affiché à l'usager.

DIFFICULTÉS RENCONTRÉES LORS DU LABORATOIRE

Tout au long de ce laboratoire, plusieurs obstacles ont surgi.

D'abord, malgré le fait que nous avions opté pour le langage de programmation Java pour écrire notre programme, nous n'étions pas tout à fait familières avec ce dernier. Pour cela, l'existence de plusieurs classes existantes de Java nous étaient inconnues. Par exemple, lire un fichier texte nous posait problème, et nous avions dû effectuer un peu de recherche. Même si nous ne nous sommes pas acharnées sur ce problème pour longtemps, il s'agissait tout de même d'une première difficulté. Il fallait également trouver une façon pour créer des noeuds ainsi que des arcs à partir du fichier texte « entrepot.txt » qui nous était fourni. Pour que le code fonctionne, il faut que le fichier texte « entrepot.txt » se situe dans le même path que les autres fichers .java. Nous avons fini par utiliser la classe « Scanner » et la méthode « split() » pour extraire les différents strings du fichier texte et les convertir en int par la suite, classés dans des structures appropriés pour contenir l'information nécessaire aux calculs par la suite. Suite à ce travail, nous avons pu nous familiariser davantage avec le langage de programmation Java et de solidifier nos bases.

Par ailleurs, structurer notre programme sous forme de classes semblait comme un casse-tête aussi, dans le sens que nous n'étions pas sûres quelles méthodes devaient se retrouver dans quelle classe, et ainsi de suite. Ce n'est seulement qu'après avoir investi davantage de temps dans la conception de notre solution,

que nous avons compris la logique derrière cette structuration. La conception du diagramme de classes a également permis de mieux comprendre la structure du travail à accomplir et de mieux comprendre les différents liens entre toutes les classes.

Par contre, ces difficultés ne représentaient rien au côté de la méthode plusCourtChemin(). En effet, l'obstacle principal que nous avons rencontré s'agissait de son implémentation; la compréhension du fonctionnement de l'algorithme Dijkstra ne représentait pas un défi en tant que tel, mais la manière de l'appliquer dans le cas présenté l'était. En réalité, nous trouvions qu'il fallait optimiser le chemin utilisé par le robot sélectionné selon le temps nécessaire que ça lui prendrait pour y parcourir, plutôt que selon sa distance, soit l'utilité de l'algorithme de Dijkstra. Plusieurs composantes devaient alors être prises en compte, comme le nombre courant d'objets transportés par le robot, le nombre d'objets présents dans un certain noeud du graphe, ou bien le type de robot utilisé, etc. Le code est donc extrêmement long puisqu'il fallait prendre en compte de nombreux facteurs. Notre code ne permet malheureusement pas de générer le chemin optimal lors de tous les cas, mais nous avons décidé de se concentrer sur certains aspects et d'en laisser passer d'autres. Construire cette méthode s'avérait donc extrêmement laborieux. Malgré que la date de remise ait été déplacée pour plus tard que prévu, nous nous sommes tout de même senties pressées dans le temps, étant donné l'ampleur du travail à effectuer.

Une solution pour la prochaine fois serait évidemment de se prendre plus en avance; ce qui aurait été utile, c'est d'ériger un échéancier à respecter, ainsi, les tâches auraient été effectuées étape à la fois, à la place de tout d'affilée.

CONCLUSION

En conclusion, ce laboratoire nous a permis de nous familiariser avec la matière vue en classe, plus précisément, avec la théorie des graphes et l'algorithme de Dijkstra, à l'aide d'un scénario fictif. Non seulement cela, étant donné que nous avons développé notre code sans un code source fourni, contrairement à d'habitude, cela nous a permis de développer notre créativité et notre indépendance. C'était une belle expérience de pouvoir tout écrire depuis le début et de structurer nos classes et nos méthodes tel que nous le souhaitons, sans être contraintes par des classes prédéterminées. Nous étions également ravies que la date de remise ait été reportée puisque la complétion du travail semblait impossible en une aussi courte période de temps, sans oublier tous les contrôles! Si le laboratoire était à

refaire, il serait intéressant de l'effectuer à l'aide d'un autre algorithme traité en classe, tel que celui de Prim.