POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

Département de génie informatique et génie logiciel

LOG 2810 STRUCTURES DISCRETES

Rapport de laboratoire #1

TP1 : GRAPHES

éléments évalués	Points
Qualité du rapport : respect des exigences du rapport, qualité de la présentation des solutions	2
Qualité du programme	3
Composants implémentés	
C1	2
C2	2
C3 divisé en :	3
C3.1	1
C3.2	1
C3.3	1
C4	2
D1	1
D2	3
D3	2
Total	20

Soumis par

Alexandre THIMONIER 1782235

Freddy SOSSA 1784114

Timothée LABORDE 1782257

Le 2017-10-23

Introduction

Au cours de ce travail, l'objectif sera de mettre en place les notions vues en cours, notamment les graphes, les diagrammes (de Hasse) et les algorithmes tels que celui de Dijkstra. Pour cela nous devons mettre en place un projet comprenant deux parties : une première visant à implémenter un système de livraison par drone et une seconde déterminant un ordre et générant un diagramme de Hasse à partir d'un choix de déjeuners et de desserts.

Compréhension du sujet :

Partie I : Livraison de colis par drone

La première partie de ce projet vise plus particulièrement à déterminer les chemins empruntés par les drones et à optimiser ces parcours ainsi que le choix de drone pour obtenir un temps de livraison minimal, tel un logiciel de localisation et de géo-positionnement par satellite.

Pour cela nous disposons d'un choix parmi deux types de drones : les drones à moyennes autonomie avec des batteries de 3.3 Ampères et ceux à longue autonomie qui possèdent des batteries de 5.0 Ampères mais qui sont plus onéreux.

Chaque vol consomme de la batterie proportionnellement au poids du colis transporté, qui sera catégorisé en 3 types (plume, moyen et lourds), ainsi qu'au temps de vol. Ceci implique donc une norme de sécurité, en effet nous ne souhaitons pas causer d'accidents par manque d'énergie. Ainsi une seconde contrainte apparait (en plus d'un temps effectif minimal) : la charge de la batterie d'un drone ne doit jamais descendre en dessous de 20% de sa charge maximale.

Nous devons privilégier l'utilisation de drones 3.3 ampères mais si ceux-ci s'avèrent insuffisants, nous pouvons utiliser des drones 5.0 ampères. Le trajet pourrait aussi dévier pour passer par une recharge, rallongeant ainsi le temps de livraison mais l'assurant. Enfin si même le détour par une recharge puis l'utilisation d'un drone 5.0 ou les deux combinés ne remplissent pas la condition de sécurité, nous nous verrons dans l'obligation de refuser la livraison.

Afin de nous aider dans les calculs d'autonomie de chaque drone nous pouvons résumer les contraintes de batteries en fonction du type de colis dans le tableau suivant :

Type de drone / **Type de colis**	**Drone 3.3**		**Drone 5.0**	
	Perte de batterie par minutes	Autonomie Maximale en minutes	Perte de batterie par minutes	Autonomie Maximale en minutes
Plume	1%	80	1%	80
Moyen	2%	40	1.5%	53

Lourds	4%	53	2.5%	32

Figure [1] : Tableau des propriétés de la batterie en fonction du type de drone et du type de colis.

Nous avons dû prendre des décisions pour préciser ces contraintes de batteries : tout d'abords si nous passons par une recharge, on s'arrête forcément pour recharger pour éviter les accidents, de plus nous considérons qu'à chaque début de course le drone est chargé au maximum (batterie à 100%) et que nous n'avons pas à faire le retour du trajet.

Mais toutes ces contraintes ne servent pas si l'on ne peut obtenir le graphe des voies aériennes de la ville. Pour obtenir cela nous devons lire à partir d'un fichier texte une liste des nœuds de notre graphe, s'ils possèdent une station de recharge, s'ils ont une connexion vers un ou d'autres nœud et auquel cas les temps ou « poids » de ces arcs. Nous garderons pour cela le format du document fourni, soit la liste des nœuds et la présence d'une station de recharge (représenté par 1 sinon 0) et séparés par une virgule. S'en suit un saut de ligne et liste des connexions représentées par deux entiers (symbolisant les deux sommets) séparés par un virgule, suivi de la pondération, elle aussi séparée par une virgule.

Enfin il nous sera indispensable de produire une interface utilisateur, un « menu », à partir duquel l'usager pourra choisir de mettre à jour la carte en fournissant un fichier, de déterminer le chemin sécuritaire le plus court en entrant son sommet de départ et d'arrivée et enfin il pourra aussi naturellement quitter le programme. Cependant s'il choisit un index non valide, nous devons afficher une erreur et retourner au menu sans quitter.

Pour résumer nous devons implémenter un programme qui doit mettre à jour une carte à partir d'un fichier texte, émettre un chemin le plus court possible à partir d'un point de départ, un point d'arrivée et ladite carte, tout en respectant des contraintes de sécurité, de consommation de batterie, de type de drone et de type de colis.

Partie II : Déjeuner et Desserts

Dans cette partie du projet nous devons afficher un graphe orienté à partir d'un fichier. Pour cela il nous sera comme dans la première partie nécessaire de lire un fichier texte dont le nom nous est passé en paramètre. Nous allons garder la structure de celui-ci, soit d'abords une liste d'ingrédients accompagnés de leur noms et séparés par une virgule, puis une liste des recettes formés par plusieurs ingrédients séparés par une virgule.

Après avoir récupérer toutes ces informations, il nous faudra emmètre un diagramme de Hasse valide qui respecte toutes les contraintes vues en cours soit : respecter l'ordre et l'orientation des arcs, enlever les liens de réflexivité et de transitivité.

Enfin il faudra émettre un affichage en se basant sur celui fourni à l'annexe 2.d. De plus l'utilisateur pourra choisir son fichier et l'affichage au travers d'un menu basé sur des index à l'instar de la première partie.

Finalement nous devons créer un programme principal qui liera les deux parties grâce à un menu permettant de choisir l'une ou l'autre.

Présentation de nos travaux : Livraison de colis par drone

La première contrainte de cette partie que nous devons traiter et celle de lecture de fichier et du stockage des informations obtenues. Pour cela nous nous sommes dits qu'une classe Graphe serai la plus appropriée, car c'est effectivement dans un graphe que nous allons sauvegarder les nœuds. Une seconde classe apparaît alors en tandem, la classe Nœud, ou Node en anglais.

La classe Graphe comportera des nœuds (agrégation), nous commencerons donc par celle-ci. Nous savons qu'un nœud a besoin d'un numéro, soit un entier, d'une indication pour la station de recharge (celle-ci ne peut qu'exister ou non, nous choisissons donc un booléen) et potentiellement d'un nom (une string). À ces attributs nous devons ajouter des « getters » et des « setters » pour obtenir et changer les attributs. Une fois la classe nœud créée, nous passons à Graphe. De même nous savons qu'elle a besoin d'un tableau de nœuds, une matrice de poids (soit d'entiers) et un chemins parcouru (un autre tableau de nœud). Concernant les méthodes, on peut imaginer que les tableaux seront de tailles variables et il nous faudra donc allouer de la mémoire, grâce à une méthode allocate(). Ensuite le cœur de la classe, une méthode creerGraphe() qui comme son nom l'indique permettra de faire un graphe à partir d'un nom de fichier passé en paramètre, attention cependant,

ce fichier doit être placé dans le même dossier que celui contenant les fichiers .Java.

De plus afin de faciliter l'affichage par la suite il serai aviser de produire une méthode lireGraphe() qui affiche le graphe tel qu'il est présenté à l'annexe 2.a et elle utiliserai la méthode afficherVoisin() qui comme son nom l'indique affiche les voisins de chaque nœud.

Enfin nous pouvons créer la classe Dijkstra, qui a pour but d'exécuter l'algorithme du même nom. Celle-ci contiendra une matrice contenant les poids d'origine (et elle demeurera inchangée : matriceSource), une seconde matrice qui va évoluer au cour de l'exécution : matriceDijkstra, un nœud de début et un nœud de fin et enfin le parcours suivit. Pour exécuter l'algorithme nous allons avoir besoin d'initialiser (ce qui donne la méthode du même nom). Ensuite il faudra trouver la distance minimale et donc le nœud suivant, mettre à jour la matrice et enfin réaliser le tout jusqu'à atteindre l'arrivée. Pour s'assurer du bon fonctionnement nous sommes d'abords passé au travers du cours puis d'un pseudo code écrit à la main.

Par la suite nous avons pris conscience qu'il nous fallait établir un chemin optimisé (PlusCourtChemin()) qui doit prendre une décision en fonction des temps de parcours et des drones. Ceci nous ajoute donc une classe Drone qui va donc évidement contenir son type, son autonomie maximum, l'état de sa batterie et son type de colis. Nous allons devoir créer des accesseurs pour cette une nouvelle classe ainsi qu'une méthode de calcul de la batterie en fonction du temps. Temps qui va justement permettre de gérer notre prise de décision, il apparait donc clairement que nous avons besoin de variables de temps, temps maximal sans recharge au cours du parcours, temps total du parcours et temps depuis la dernière recharge.

Finalement nous pouvons obtenir le diagramme de classe suivant :

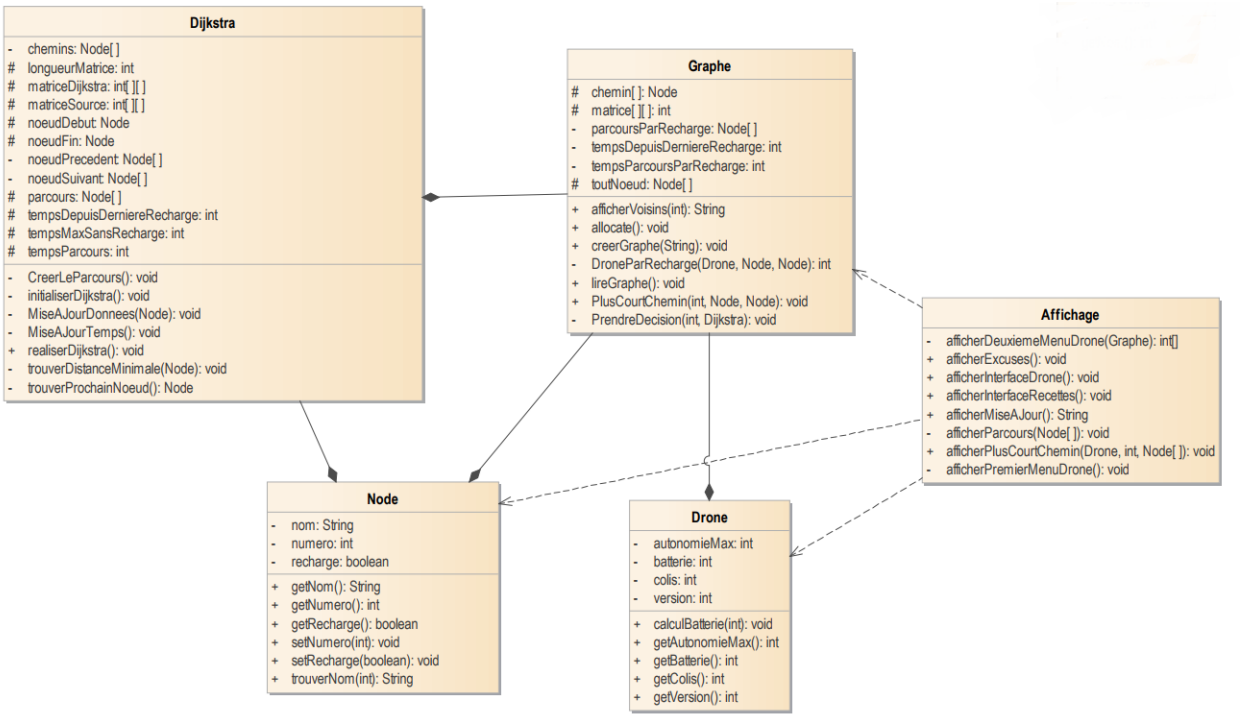

Figure [2] : Diagramme de classe de la partie livraison par drone

Présentation de nos travaux : Déjeuner et dessert

La première étape est de lire et d'enregistrer les données obtenues à partir de la lecture du fichier fourni pour initialiser les attributs. Afin de bien représenter les différents éléments à manipuler, il était nécessaire de créer des classes pour faciliter la gestion et l'encapsulation des données. A cet effet, nous avons créé deux classes : Ingredient et GrapheOriente.

La Classe Ingredient représente un Ingredient qui est caractérisé par deux informations : le nom de l'ingrédient et l'indice qui lui est associé. Quant à elle la classe GrapheOriente représente le Graphe en soit, contenant ainsi la matrice des arcs, les ingrédients utilisés, le nombre de chemins possibles sur le Diagramme de Hasse et la liste de tous les chemins possibles du diagramme de Hasse.

En effet, la matrice des arcs est en deux dimensions permet d'initialiser toutes les données. On enregistre donc tous les arcs entre les Nœuds. Si un arc existe entre deux nœuds (de i vers j), alors on insère à la position [i][j] la valeur 1 et -1 à la valeur [j][i] afin de respecter l'orientation des arcs. Cette matrice est la base pour réaliser toutes les tâches suivantes à savoir l'affichage du graphe Orienté et la génération du diagramme de Hasse.

Pour l'affichage du Graphe orienté, l'idée est assez simple. Il suffit de :

- Parcourir le tableau des ingrédients
- Récupérer chaque ingrédient
- Aller vérifier les ingrédients avec lesquels il est en relation et
- Afficher ces ingrédients dans le format demandé

La méthode CreerGrapheOriente de la classe GrapheOriente se base sur cet algorithme pour afficher le graphe Oriente des Ingredients.

En ce qui concerne l'affichage du diagramme du Hasse (méthode genererHasse( )) l'algorithme utilisé est le suivant :

- Identifier tous les minimaux du Diagramme du Hasse
- Parcourir chaque minimal et chercher les nœuds qui sont juste après lui. Cette partie est faite de façon récursive afin de réduire la longueur du code. Ainsi, sur chacun des nœuds suivants identifiés, on reprend la procédure jusqu'au maximaux tout en tenant compte des différentes règles du diagramme de Hasse
- Enregistrer au fur et à mesure qu'on parcourt les différents nœuds la liste des chemins
- Afficher les différents chemins obtenus au format demandé

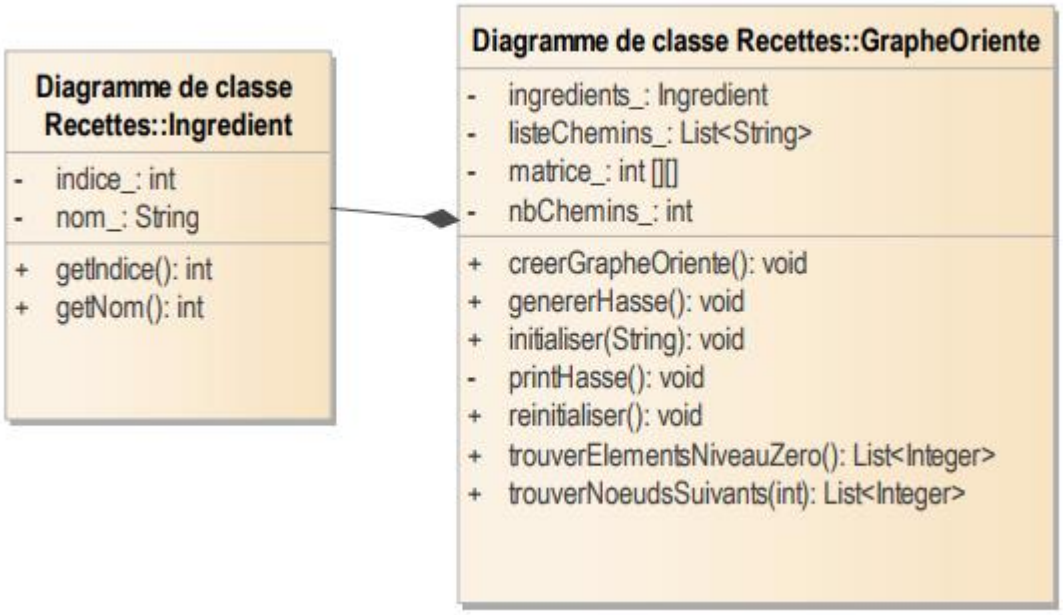

Figure [3] : Diagramme de classe de la partie Déjeuner et Desserts

Difficultés et solutions : Livraison de colis par drone

- Difficultés rencontrées
  - Nous avions mal anticipé la fin de cette partie, principalement la prise de décision
  - Remanier les calculs de temps et de batterie
  - Gêne de tableaux dynamiques avec allocation
  - Problème de lecture de fichiers
- Solutions potentielles
  - Remanier la prise de décisions par des cas dans la classe graphe plutôt que dans Dijkstra

- Utiliser la batterie plutôt que le temps, mais finalement nous avons décomposé en plusieurs variables avec des noms très descriptifs pour ne pas se perdre.

- Nous avons réalisé trop tard qu'en allouant en multipliant la capacité par deux posait des problèmes de calculs, donc nous avons optés pour une allocation de type capacité +1 à chaque fois mais nous aurions alors plus simplement utiliser des vecteurs.

- Remplacer les chemins absolus par des chemins relatifs

Difficultés et solutions : Déjeuner et dessert

- Difficultés rencontrées

  Pour la réalisation du diagramme de Hasse :

  - Intégration des règles de conception du diagramme de Hasse dans le code

  - Difficulté de débogage avec les fonctions récursives

- Solutions potentielles

  - Créer de nouvelles méthodes pour limiter le travail effectué par la fonction récursive

  - Intégration progressive des règles du diagramme de Hasse au code (une fonction traite juste la réflexivité et une autre la transitivité)

Présentation de nos travaux : Déjeuner et dessert

Au cours de ce projet nous avons pût mettre en place les notions de LOG2810 telles que l'algorithme de Dijkstra, diagramme de Hasse et les graphes mais aussi des notions de LOG1000 et de LOG2410. En effet pour la première fois nous partions de rien, nous n'avions pas de projet et de programmes fournis. C'était donc très satisfaisant de voir fonctionner notre projet alors qu'il est partit d'une feuille blanche.

De plus cela nous a permis de nous rappeler quelques notions de SSH, notamment celles de la gestion du temps et de la répartition du travail, quelques heures de préparations qui se sont avérées très utiles par la suite.

Concernant le temps alloué pour ce laboratoire, nous estimons avoir passé une soixantaine d'heures de travail en groupe accompagnées d'une dizaine d'heure personnelles. Ceci semble raisonnable pour une échéance de 3 trois semaines pour un groupe organisé.

Nos attentes concernant le prochain travail pratique sont une liberté de projet équivalente et un rapport temps disponible et temps de travail équivalent.