Département de génie informatique et de génie logiciel

LOG2810
Structures discrètes

TP1 - Graphes

Soumis par :
Chelsy Binet (1850411)
Jean-Simon Marrocco (1785835)
Sébastien Cadorette(1734603)

22 octobre 2017

Introduction:

La mise en situation de ce laboratoire est celle où un étudiant viendrait nous voir et nous demanderait de l'aide. Cet étudiant a déniché un stage dans une startup de drones et est en charge de déterminer quels chemins devront emprunter ces drones pour leurs livraisons. On doit donc implémenter des algorithmes se basant sur des notions de structures discrètes. Il faut être en mesure de créer un graphe, de l'afficher ainsi que de déterminer le plus court chemin pour livrer un colis. Plus tard, ce même étudiant revient nous voir avec une autre mission pour nous. Cette fois-ci on doit toujours pouvoir créer un graphe, mais on doit aussi être en mesure d'afficher un diagramme de Hasse et d'implémenter une interface. Les objectifs de ce TP sont donc de nous familiariser avec les notions théoriques de structures discrètes et en particulier sur les graphes.

Présentation de nos travaux:

Explication de la solution de la partie Drones:

Le but du premier du travail est de développer un algorithme permettant de trouver le chemin le plus rapide et approprié que devrait emprunter un drone selon certaines restrictions et certains critères. En d'autres mots, on doit prendre en compte le temps du trajet, le niveau des piles, le modèle de drone utilisé et le poids du colis lorsqu'on calcule le parcours. Si jamais le drone ne peut pas se rendre à la destination, plusieurs solutions s'offrent à lui. Tout d'abord, si jamais le drone ne peut se rendre à sa destination, il cherche s'il peut utiliser une borne de rechargement en chemin pour pouvoir finir le parcours. Par la suite, si cette solution n'est toujours pas faisable, nous devons vérifier avec un modèle de drone plus puissant. Si toujours, le drone plus puissant ne peut effectuer le parcours, la livraison demandée est refusée.

Pour résoudre ce problème, nous avons utilisé les étapes suivantes

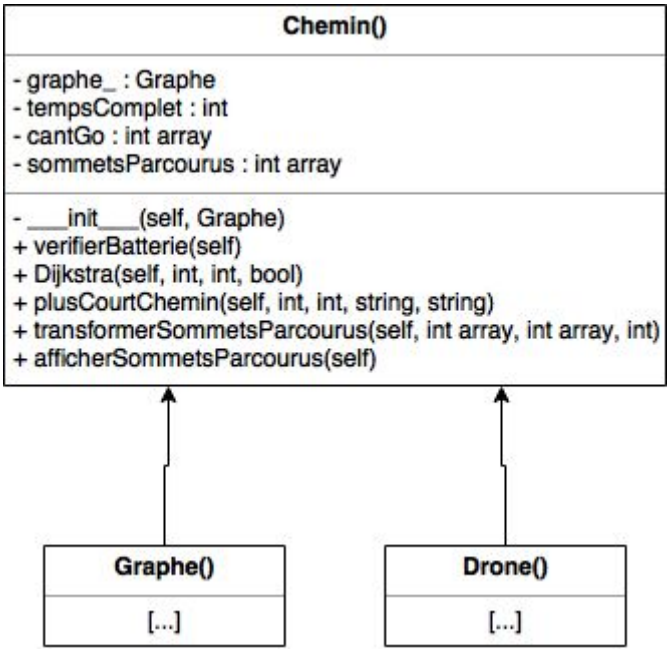
1. **creerGraphe()** : Premièrement, nous avons séparé l'information donnée pour créer des listes de nœuds (sommets) et d'arcs pour ensuite tout regrouper dans un graphique. Cela nous permet de traiter l'information beaucoup plus efficacement. Une des difficultés à cette étape était de bien interpréter l'information du fichier texte et de bien le diviser.
2. **dijkstra()** : Pour trouver le chemin le plus rapide, nous avons fait appel à l'algorithme de Dijkstra. En résumé, cet algorithme parcourt tous les liens à partir du point de départ juste qu'à la destination en gardant en mémoire le chemin qui prend le moins de temps. Le plus gros défi de cet algorithme était

de s'assurer qu'il choisit bien la meilleure option, car dans certains tests il trouvait le chemin avec le moins de nœuds, mais ce n'était pas nécessairement le plus efficace en prenant compte de l'utilisation de la pile du drone.

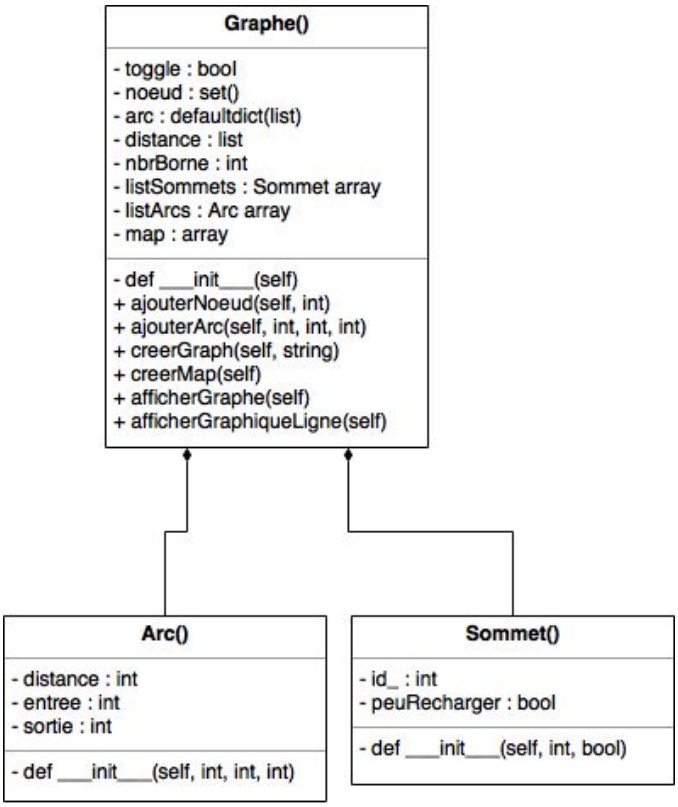
3. **plusCourtChemin()** : C'est dans cette méthode qu'on vérifie les restrictions et critères, ainsi que l'exécution des différentes solutions. Un défi qu'on a eu à surmonter a été de faire passer le drone par une borne de rechargement, dans le cas où le parcours jusqu'au point d'arrivé était trop demandant en terme d'énergie. Le problème était qu'au départ notre algorithme trouvait la borne de recharge la plus proche du point d'arrivé et non la borne la plus stratégique pour effectuer le parcours jusqu'au point d'arriver. En effet, au début de nos tests, le drone allait jusqu'à rallonger son parcours, simplement parce qu'il cherchait la borne de rechargement la plus près du point de départ uniquement, sans prendre en compte la position du point d'arriver. On a réglé ce problème en ajoutant des conditions lors de la recherche d'une borne de rechargement. Plutôt que de chercher la borne de rechargement la plus près du point de départ, on regarde à quel moment, lors du parcours vers le point d'arriver, le drone ira en-dessous d'un certain seuil de pourcentage de pile. Lors qu'il atteindra ce seuil, c'est à ce moment qu'il cherchera la borne de recharge, ayant pour conséquence de trouver une borne de rechargement plus près du point d'arriver.

Diagrammes de classes partie Drones:

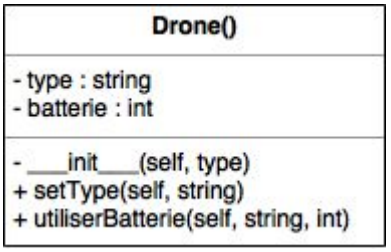
Classe Chemin



Classe Graphe



Classe Drone



Explication de la solution de la partie Recettes:

En ce qui concerne la deuxième partie du travail, c'est-à-dire celle concernant les déjeuners et les dessert, une approche orientée objet fut utilisée. Le but de cette partie était de lire un fichier pour ensuite afficher un graphe ainsi qu'un diagramme de Hasse, le tout regroupé sous une interface pour afficher un menu à l'utilisateur. Nous avons donc commencé par créer une classe LecteurFichier comportant les fonctions publiques suivantes:

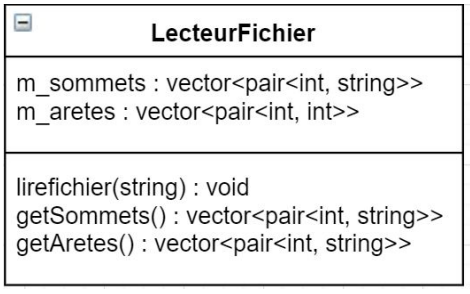
- **lireFichier()** : Retourne un void et qui prend en paramètre le nom du fichier. Cette fonction est implémentée dans le fichier LecteurFichier.cpp de façon triviale. Elle ouvre le fichier, le lit, et met dans le bon vecteur les données correspondantes.

- **getSommets()** : Retourne un vecteur de paire de *int* et de *string*. Le *int* étant le numéro associé à chaque ingrédients et la *string* étant le nom de l'ingrédient en question.
- **getAretes()** : Retourne un vecteur de paire de deux *int*. Le premier *int* étant le sommet de départ et le deuxième *int* étant le sommet d'arrivée.

C'est ensuite dans le main que l'on retrouve le prototypage des fonctions **creerGrapheOriente** ainsi que **genererHasse**. On retrouve aussi dans le **main** le code pour afficher le menu à l'utilisateur lui permettant de choisir entre l'option a, b ou c. La fonction **creerGrapheOriente** prend en paramètre le nom du fichier à lire. Le principe est très simple: une boucle *for* parcourt le vecteur de sommets et affiche voulu le graphe dans le format. Une autre boucle *for* parcourt le vecteur d'arêtes et compare le numéro de l'ingrédient du vecteur des sommets avec le numéro de l'ingrédient du vecteur des arêtes. Si ceux-ci correspondent, alors il faut afficher l'ingrédient à l'indice actuel. Le même principe est utilisé pour la fonction **genererHasse**. Une combinaison de boucle *for* et de condition *if* permettent de parcourir les vecteur tout en comparant les numéros à chaque indice et d'afficher dans le bon format le diagramme de Hasse.

Diagramme de classe partie Recettes:

Étant donné que, comme expliqué ci-haut, une seule classe fut nécessaire pour cette partie, le diagramme de classe est très simple. Il s'agit en fait simplement d'une classe mise sous forme de diagramme UML et non d'un diagramme de classe à proprement parler. Il peut toutefois être utile pour bien visualiser les attributs et méthodes de la classe.



Difficultés lors de la mise en commun des projets

Lorsque nous étions rendu au moment de mettre en commun les deux programmes, nous avons fait face à un problème. En effet, le programme 1 (Drones) a été développé en Python et le programme 2 (Recettes) en C++. et nous ne sommes pas parvenus à les combiner dans un menu principal commun. Toutefois, lors d'un échange avec le chargé de laboratoire, il nous a été mentionné que nous pouvions avoir les deux programmes séparés. Il a aussi été mentionné qu'un des deux programmes doit contenir le menu principal et que lorsque l'utilisateur choisirait

l'option d'utiliser l'autre programme, un message à l'écran afficherait d'exécuter le programme séparément.

Dans cette optique, nous avons développé le menu principal dans le programme Drones (Python). L'utilisateur peut donc directement utiliser le programme Drone à partir du menu principal, mais le programme Recettes doit être exécuté séparément. Tel que demandé, un message d'attention s'affiche lorsque l'utilisateur sélectionne l'option d'utiliser le programme Recettes à partir de Python.

Conclusion:

En conclusion, ce TP nous a permis d'en apprendre énormément sur le parcours des graphes. En effet, même si nous avions déjà des notions théoriques de bases sur ce sujet, ce TP nous a réellement permi de mettre en pratique ce que nous avions vu pendant les cours. Il s'agit là d'une bonne façon de voir concrètement l'utilité des mathématiques discrètes dans la vie de tous le jours et plus particulièrement des graphes. Nous nous attendons alors que le prochain laboratoire soit dans la même lignée, c'est-à-dire qu'il nous permette d'appliquer les notions vues en cours tout en nous permettant d'approfondir celles qui seraient moins claires.