LOG2810
TP1 : Graphes

Léandre Guertin (1841782)

Boubacar Telly Bah (1791077)

Abderahmane Bouziane (1842298)

Polytechnique Montréal

Devoir Remis à Foutse Khomh

Remis le 24 octobre 2017

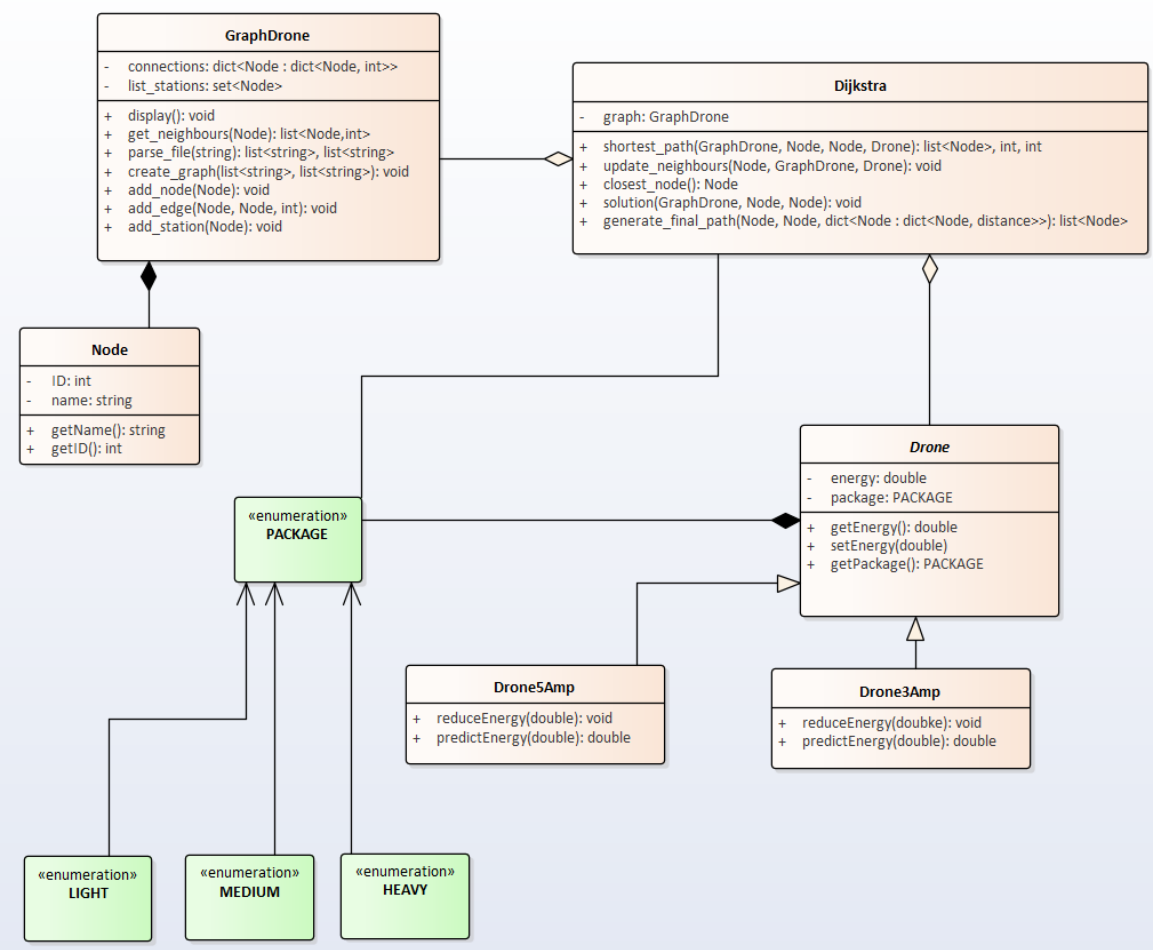
Introduction

L'objectif de ce laboratoire de LOG2810 est dans un premier temps de déterminer le plus court chemin que devait effectuer un drone allant d'un point A a un point B en prenant en compte plusieurs contraintes, soit le temps et la batterie du drone. Dans un second temps, nous avons à générer un diagramme de Hasse à partir d'un fichier texte donné. Nous allons donc, dans ce rapport, expliciter comment nous nous sommes organisés afin de réussir à résoudre ces deux problèmes. Les deux problèmes énumérés ci-dessus ont été séparé en deux problèmes concrets pour ensuite les rejoindre dans un seul projet via un interface console d'utilisateur. Cet interface permet de choisir entre le calcul de distance entre deux points à l'aide d'un drone ou bien faire la génération du diagramme. Dans ce rapport, nous allons tout d'abord vous présenter les diagrammes de classes des différents problèmes ainsi que la méthode résolution. Ensuite nous allons présenter les difficultés rencontrées lors de ce laboratoire.

Dijkstra

2.1 Diagramme de classe

Figure 2.1: Diagramme de classe du problème de recettes



2.2 Solution

2.2.1 Étape de résolution du problème

Afin de résoudre le problème, nous avons créé une classe Dijkstra qui cherche le plus cours chemin possible entre deux arrondissements selon le poids du packet transporté par le drône. Il essaye d'abord avec le drone de 3.3amp puis avec celui de 5amp si aucun chemin n'a été trouvé. Afin de trouver le chemin le plus court, nous avons implémenté notre version de l'algorithme de Dijkstra en appliquant plusieurs modifications

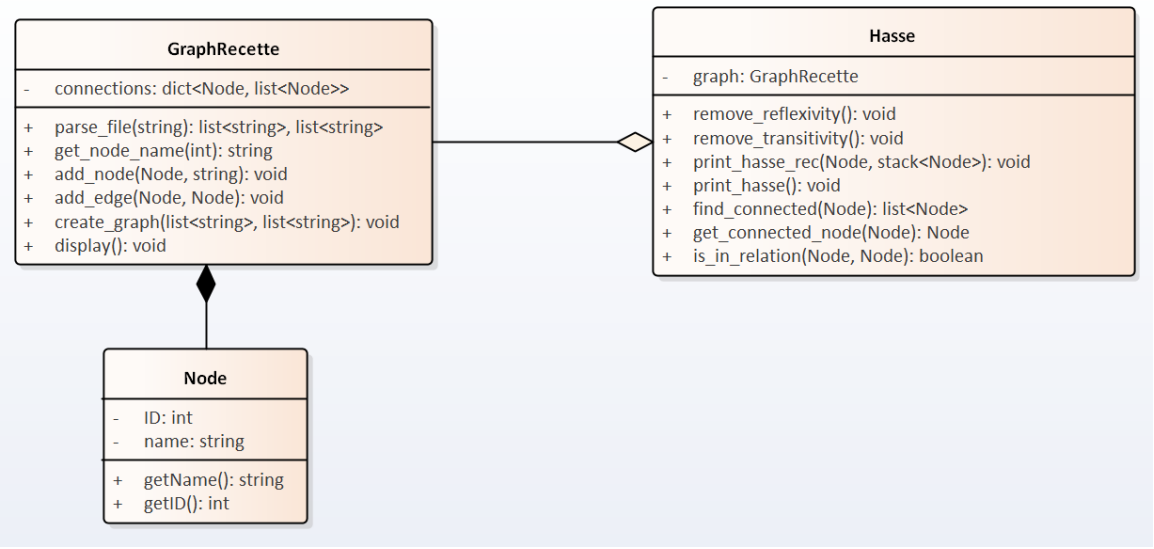
- L'algorithme de dijkstra de base va comme suis. Tout d'abord, on initialise un dictionnaire avec tout les noeuds comme clef et une distance par rapport au noeud initial infinie comme valeur ainsi que le noeud précédent emprumpté pour y arriver (None pour l'instant). On trouve ensuite le noeud ayant la distance la plus petite par rapport au noeud de départ parmis les noeuds dans ce dictionnaire. On parcourt alors ses voisins et on met à jour leur distance par rapport au noeud courant et leur noeud précédent seulement si la distance à partir du noeud courant est inférieur à la distance déja présente. On ajoute ensuite le noeud courant dans unn dictionnaire de noeuds visités et on recommence l'algorithme jusqu'a que ce que le noeud final soit dans les noeuds visités. On itère alors sur les noeuds précédents de noeud final jusqu'au noeud initial. C'est notre chemin.

- Les modifications qu'on à du apporté à l'algorithme pour qu'il fonctionne avec notre problème sont simples. On doit d'abord rajouter la notion d'énergie a notre algorithme. Pour ce faire on utilise un autre dictionnaire où l'on garde en mémoire l'énergie dépensée à chaque noeud. On met à jour cette énergie dans la méthode update neighbour. Si on passe par une station, on remet l'énergie dépensée à zéro. Autre chose, avant de mettre à jour les distance pour les voisins d'un noeud, on fait une prédiction de l'énergie restante si on décide de prendre le chemin avec ce voisin. Si l'énergie restante est inférieure à 20, on associe une distance infinie à ce noeud afin de l'éliminer de nos possibilités. On ajoute aussi 20 minutes au temps associé au chemin pour chacune des stations de recharges présentes dans le chemin.

Hasse Diagram

3.1 Diagramme de classe

Figure 3.1: Diagramme de classe du problème de recettes



- Noeud : Est la classe qui represente un noeud(dans notre cas une recette ou un ingredient) il possède un ID et un nom.
- GraphRecette : Est la classe qui représente le graph orienté contenant nos noeuds, Il contient un dictionnaire de noeuds où chaque clef représente un noeud et les clef des connections.
- Hasse : Est la classe qui va se charger de générer le diagramme de Hasse, il contient des méthodes qui permettent de supprimer la reflexivité, suprimer la transitivité et aussi d'afficher le diagramme de Hasse final.

3.2 Solution

3.2.1 Étape de résolution du problème

Afin de résoudre le problème, nous avons créé une classe Hasse qui va nous permettre de supprimer la réflexivité et la transitivité ainsi que l'affichage du diagramme de Hasse. Les différentes étapes qui ont été suivies afin de résoudre ce problème sont listées ci-dessous.

- La suppression de la réflexivité se fait plutôt facilement, on vérifie pour chacune clé dans notre graphe si elle est reliée à elle-même. Si elle l'est, alors nous supprimons la relation.

- La suppression de la transitivité, quant à elle, a été beaucoup plus difficile. Cette dernière se fait en plusieurs étapes. Dans un premier temps nous vérifions pour chaque nœud s'il possède des nœuds en commun avec ses voisins. Si oui dans ce cas nous supprimons la relation du nœud courant avec l'élément en commun.

- Pour l'affichage du graphe, nous allons dans un premier temps prendre les nœuds dans lesquels aucun autre noeud n'est connecté à celui-ci. Ensuite, nous voulons afficher toutes les liaisons possibles de ce noeud. Pour se faire, nous utilisons une pile. Pour chaque noeud pour lequel il est connecté, nous voulons l'ajouté dans la pile et lorsque nous rencontrons un noeud qui n'a pas de connection, nous affichons la pile. Puisque cette fonction est récursive, lorsque nous remontons aux fonctions plus élevés, nous retirons de la pile chaque élément, ce qui fait en sort que nous sommes capable d'y afficher toutes les possibilités de recette facilement.

Difficulté rencontrées

Tout au long du laboratoire, nous avons rencontré un grand nombre de difficultés, que ce soit au niveau du langage ou bien au niveau de la création des algorithmes.

Tout d'abord, le langage choisi fut une contrainte majeure pour deux des trois membres de l'équipe. Nous avons choisi de programmer ce laboratoire en Python3 puisque c'est ce qui est en très grande demande sur le marché à présent. Puisque deux des trois membres de l'équipe n'avaient pas beaucoup programmé en python au par avant, nous devions alors apprendre la sémantique du langage en même temps de faire les algorithmes. De ce fait, nous avons mit beaucoup de temps sur le site web "hackerrank.com" afin d'améliorer nos bases et nous permettre d'en apprendre d'avantage. Par chance, le troisième coéquipier avait déjà 2 ans d'expérience et nous aidait lors de problèmes rencontrés.

Par la suite, nous avions eu beaucoup de débat sur la structure du code. Nous avons commencer à essayer de faire un code qui allait être portatif sur les deux problèmes de notre laboratoire, mais nous avons réalisé par la suite que nous allions que nous rajouter du trouble pour des concepts différents. Nous avons donc dû changer tout ce que nous avions commencé et faire un refactoring pour séparer nos diagrammes de classe en deux problèmes tout à fait différent. Par exemple, la classe "Graph" contenait au début le graphe avec les drônes et le graphe avec les recettes. Mais après plusieurs réflexions, nous avons réaliser que nous allions que nous casser la tête à faire une classe qui gère bien les deux cas. Nous l'avons donc séparé en deux classes, soit la classe "GraphDrone" et "GraphRecette".

Ensuite, les algorithmes étaient aussi un grand problème rencontré. Même si nous avions déjà fait un algorithme de Dijkstra en INF1005C, nous n'avions pas poussé le concept aussi loin que dans ce laboratoire. En effet, implémenter les différentes contraintes pour le drône n'était pas une tâche facile. Nous avons par exemple eu la problématique de nd pas savoir que doit faire le drone lorsqu'il n'a pas assez d'énergie et qu'il ne peut pas se rendre plus loin.

Conclusion

Ce laboratoire nous a permis de mettre en application les connaissances théoriques acquises en LOG2810. La résolution des différents problèmes a fait appel à plusieurs notions apprises pendant le cours notamment l'utilisation de Graphe, la récursion , le parcours d'un graphe avec l'algorithme de Djikstra, etc. Malgré toutes les difficultés, nous avons vraiment apprécié cette expérience. Nous avons eu la chance, pour la plupart, d'apprendre un magnifique langage de programmation qui est concis et qui est fort utile à la réalisation d'algorithme. Elle nous a aussi mit dans un véritable contexte d'entreprise. Nous pouvons facilement imaginer que certaines entreprises font face à ce genre de problème aujourd'hui tout en ayant beaucoup plus de contraintes et d'étape que ce qu'il était présent dans le laboratoire.