LOG2810

Structures discrètes

TP1 : Graphes

Trimestre : Automne 2019

Équipe 61

Équipier 1 : Kevin Leumassi - 1892002

Équipier 2 : Merwane Chalabi - 1889346

Équipier 3 : Gabriel Thibaud – 1888604

Polytechnique Montréal Remis le 5 novembre 2019

Introduction :

Dans ce TP, il nous est demandé d'implémenter le système d'optimisation de trajets des robots pour une entreprise cherchant à diminuer leur coût de main d'œuvre. Nous devrons utiliser l'algorithme de Dijkstra en représentant l'entrepôt de l'entreprise comme un graphe. Chaque robot aura ses spécificités comme sa capacité maximale et sa vitesse, et chacun aura une liste d'objets à aller chercher.

Solution proposée :

- *Arc.java* : Nous avons commencé par créer la classe Arc qui possède deux nœuds de type Nœud et la distance entre les deux nœuds de type Integer (longueur de l'arc) comme attributs. Cette classe possède un constructeur par paramètres ainsi que des getters et des setters.

- *Nœud.java* : Nous avons alors implémenté la classe nœuds qui possède en tant qu'attributs le numéro du nœud (int), les quantités d'objets A, B et C (int). Cette classe possède des méthodes permettant de modifier les états (State) qu'elle contiendra durant l'exécution du code.

- *graphe.java* : Il s'agiras ici d'implémenter la classe Graphe afin de créer le graphe composé de nœuds et d'arcs. On créera une map pour les Arcs ainsi que pour les Nœuds pour ensuite créer les nœuds et les arcs à partir du fichier *entrepot.txt* et ensuite utiliser la fonction creerGraphe qui appelleras les fonctions creerNoeuds et creerArcs.

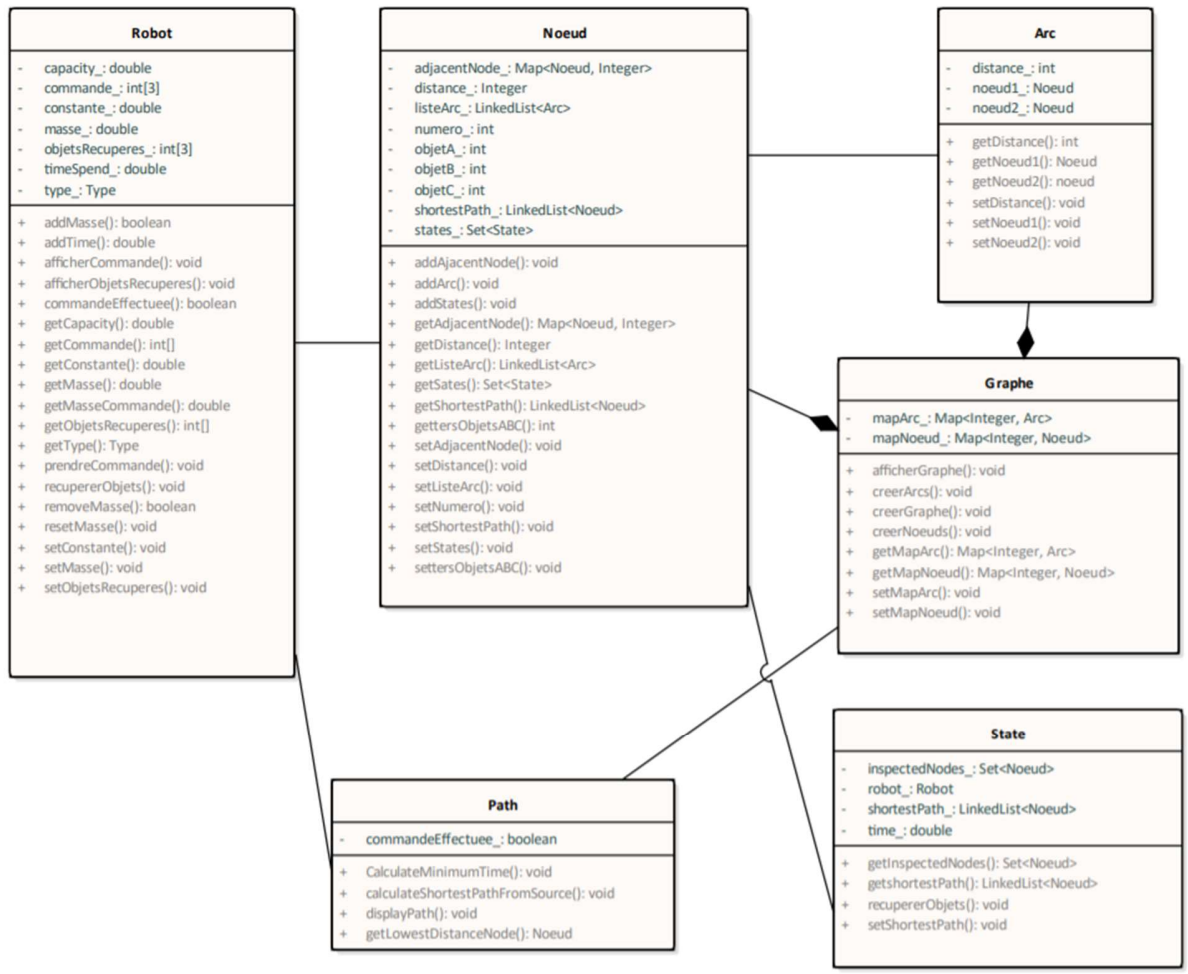
- *robot.java* : On implémente ici la classe robot permettant de différencier les robots de type X, Y et Z ayant différentes caractéristiques. On aura chaque type de robot qui aura une capacité maximale différente ainsi

qu’une constante définissant le temps mis pour parcourir une distance. Nous aurons des fonctions permettant d’ajouter ou de retirer la masse des objets que porterons les robots qui sont addMasse, setMasse, removeMasse ainsi que resetMasse. On aura aussi la fonction addTime qui retourneras le temps passé à parcourir un certain chemin. La fonction recupererObjets se chargeras d’ajouter des objets au robot en vérifiant si le robot n’a pas atteint sa capacité maximale et si sa commande n’a pas encore été effectuée.

- *State.java* : La classe state nous serviras à garder les chemins parcourus par les différents robots en mémoire avec tout les paramètres tels que le temps mis à parcourir un chemin et les différents nœuds parcourus. Nous avons mis en place ce système afin de déterminer le meilleur chemin possible pour nos robots.

- *Path.java* : La classe Path permettras de calculer le chemin le plus court via la méthode calculateShortestPathFromSource. On implémenteras dans ce même fichier la fonction getLowestDistanceNode qui va retourner le nœud le plus proche, CalculateMinimumTime qui sert à ajouter les états aux différents nœuds. On aura aussi notre main qui seras implémentée dans ce fichier.

Diagramme de classe



Difficultés rencontrées :

La première difficulté que nous avions rencontrée fut la compréhension du TP, c'est-à-dire les tâches qui nous sont demandées, les principaux concepts (que représentent nos classes actuelles) et le fonctionnement des composantes à implémenter. Nous avons donc commencé par un « brainstorming » en équipe, on a rapidement décidé de représenter les différents concepts par des classes telles que Robot, Nœud, Arc ou bien encore Graphe. La liberté, dans nos différentes implémentations dans ce TP était aussi une difficulté car il ne fallait pas trop se disperser afin de rester fidèle au travail attendu. Nous avons opté pour la création de plusieurs classes avec, pour certaines, beaucoup de méthodes dont nous aurons souvent besoin et dont le fonctionnement sera simple et intuitif afin de faciliter le débogage. Si un problème survient dans une fonction il est plus simple de le repérer et le résoudre si cette fonction est composée de plusieurs méthodes simples.

Ensuite, ce qui a sûrement été la plus importante difficulté pour notre équipe, fut l'implémentation du la fonction « plusCourtChemin() » que nous avons appelé « calculateShortestPathFromSource() » qui correspond à la fonction qui détermine le chemin le plus rapide pour effectuer la commande demandée. Afin d'implémenter au mieux cette fonction, nous nous sommes inspirés de l'algorithme de Dijkstra tout en adaptant ce dernier pour qu'il soit le plus efficace dans le contexte de notre TP. Par exemple, l'une des différences avec de l'algorithme de Dijkstra est que le nœud de départ et celui d'arrivé sont les mêmes (le nœud 0 dans notre cas). C'est-à-dire que lors de notre recherche du chemin le plus rapide, dès que notre robot s'éloigne du nœud 0 afin d'aller récupérer la commande, le robot s'éloigne aussi de sa destination (qui est aussi le nœud 0 une fois les objets récupérés).

De plus, le nombre de possibilités étant très important, chercher la possibilité la plus rapide est compliqué. Nous avons choisi d'assigner des états (classe State) aux nœuds. Ces états nous renseignent sur le robot

utilisé, la distance effectuée, les nœuds visités et le chemin le plus court jusqu'à présent (sous forme d'une liste liée de Nœud). Chaque nœud a aussi en mémoire les différentes possibilités pour qu'on y accède, il a donc aussi en mémoire les différents accès de ses voisins directs. Toute ses possibilités sont nombreuses et nous avions au début des problèmes de gestion de mémoire que nous avons amélioré en révisant et optimisant notre code.

Enfin, les différents types de robots qui possèdent des caractéristiques différentes, apportent une difficulté en plus au TP. Selon la commande, le type du robot influe sur la durée (le temps) pour compléter et effectuer la commande.

Conclusion :

Ce premier TP nous a permis de mettre en pratique les notions que nous avons étudiées en cours, principalement celles en lien avec les structures de données et la théorie des graphes en implémentant des algorithmes sur les structures discrètes abordées. Le TP permet aussi une approche plus compréhensible de l'algorithme de Dijkstra qui pouvait sembler un peu confus lors de sa lecture dans les notes de cours. Une fois en lien avec le TP, son fonctionnement parait plus logique et simple. De plus, toute notre implémentation a été effectuée en Java ce qui a consolidé notre maitrise de ce langage, nous avons volontairement décider de ne pas utiliser le C++ avec lequel nous sommes plus à l'aise.

Pour le prochain TP, nous espérons que ça sera une occasion de pratiquer les notions théoriques du cours (dans le cas du prochain TP, les états et les Automates à états finis) afin d'améliorer notre compréhension des concepts afin de mieux pouvoir les réutiliser plus tard.