LOG2810
Structures Discrètes

Travail de session sur
TP1 : Graphes

Présenté à :
Paulina Stevia Nouwou Mindom

Soumis par :
Guillaume Proulx -1899371
Marc-André Primeau Breton – 1856799
Adrien Lavigne - 2036856

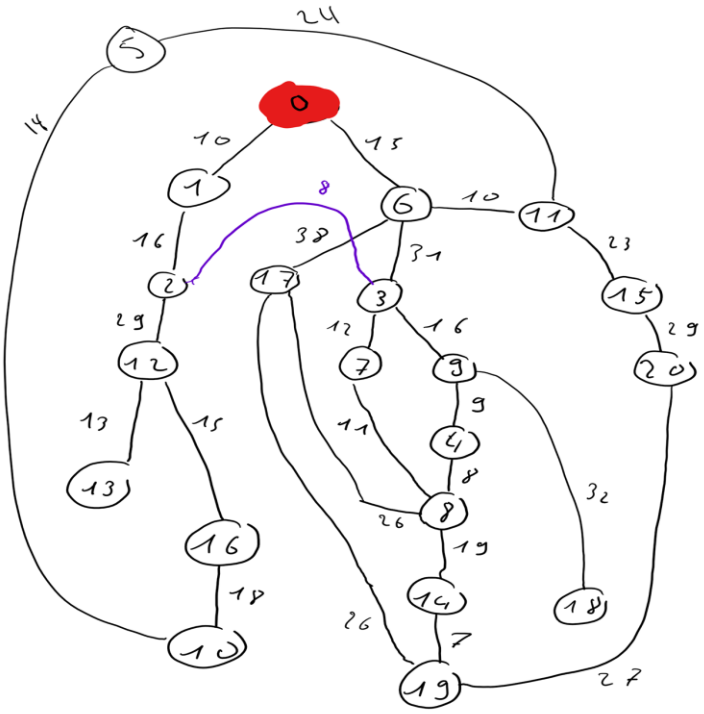
Section : 02

Le 5 novembre 2019

L'objectif de ce travail pratique est d'appliquer les notions théoriques sur les graphes sur un cas concret. La mise en situation est la suivante :

Une compagnie souhaite réduire le cout de la main-d'œuvre dans ses entrepôts en utilisant des robots à la place des humains et notre objectif est d'optimiser le trajet de ses robots à l'aide de l'algorithme de Dijkstra. De plus, nous avons à disposition plusieurs sortes de robots qui se déplacent plus ou moins vite ou qu'il peut transporter des charges plus lourdes.

Afin de bien représenter l'entrepôt de la compagnie, celui-ci sera représenté par le graphe suivant :



Le point 0 représente le point de départ et de retour des robots.

Notre Solution

Pour résoudre ce problème, nous avons décidé d'écrire notre solution en python. En effet, l'avantage d'utiliser Python est dû au fait qu'il existe des librairies qui permettent d'implanter certaines fonctions requises pour la réalisation de ce travail pratique. La première librairie utilisée est **NetworkX** qui permet de créer facilement des graphes et de les afficher à l'aide de simple méthode et la seconde librairie importée est la librairie **matplotlib. pyplot** qu'on utilise pour la l'affichage du graphique orienté. (À installer sur votre ordinateur pour le bon fonctionnement du code remis.)

- ***FileHandler.py*** : Contiens la méthode « **fillGraphFromFile** » qui permet de créer le graphique orienté à l'aide d'un fichier texte passé en argument et de la librairie NetworkX.

- ***Commande.py*** : Ce fichier permet d'initialiser la commande du robot. Elle contient les méthodes « **setter** », « **commandeValide** » et « **afficherCommande** » :

  Méthode « setter » : cette méthode permet d'initier la commande pour les objets A, B et C.

  Méthode « commandeValide » : cette méthode permet de vérifier le poids de la commande et de sélectionner le bon robot pour la réalisation de cette commande.

  Méthode « afficherCommande » : cette méthode permet d'afficher la commande rentrée.

- ***Robot.py*** : Ce fichier contient les conteneurs des différents colis que le robot va récupérer tout au long de son parcours c'est-à-dire « conteneurA » pour les colis de type A, « conteneurB » pour les colis de type B et « conteneurC pour les colis de type C. De plus, la méthode "calculConstVitesse" permet de calculer la vitesse du robot pour aller à chacun des points du graphe face au poids que celui-ci transporte sur lui. Finalement, la méthode "add" permet de rajouter les colis aux différents conteneurs du robot.

- ***Djiskstra_algo.py*** : Ce fichier permet de calculer la route la plus rapide pour pouvoir réaliser la commande le plus vite possible. Il contient les méthodes "**graph_to_lenght**", "**path_to_object**", "**finds_stops**", "**robot_actions**", "**find_way**", "**partialCommandFullfill**" et "**printCost**"

  Méthode "*graph_to_lenght" :* cette méthode permet de déterminer la distance et les différents chemins possibles pour se rendre aux différents points du graphe.

  Méthode "path_to_object" : Cette méthode permet de déterminer quel chemin peut réaliser la commande entrée avec les différents colis.

  Méthode "finds_stops" : Cette méthode permet de déterminer où le robot doit s'arrêter afin de ramasser tous les colis nécessaires pour la réalisation de la commande et le tout est stocké dans un tableau.

  Méthode "robot_actions" : Cette méthode permet de déterminer une série d'actions orchestrée par le robot pour ramasser les colis aux différents points et renvoie les actions déterminées dans un tableau pour le calcul du temps.

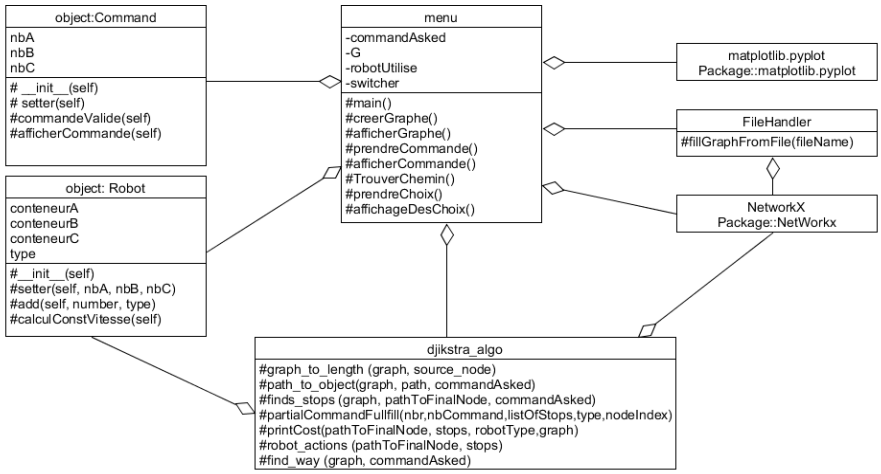
  Méthode "find_way" : Cette méthode regroupe les différentes méthodes pour la réalisation en un seul et unique bloque d'action qui renvoie le tableau d'action pour le calcul du temps pris par les robots.

  Méthode « partialCommandFullfill » : Cette méthode permet de savoir si les points d'un chemin permettent de remplir la commande envoyée.

  Méthode « printCost » : Cette méthode permet de calculer le temps total de déplacement du robot pendant tout le parcours de celui-ci.

- ***Menu.py :*** Ce fichier contient l'interface du projet pour l'interaction en l'utilisateur et le logiciel. Elle contient les fonctions permettant de créer le graphique, d'afficher le graphe, de prendre une commande, d'afficher la dernière commande rentrée et de trouver le chemin le plus court pour réaliser cette commande.

Diagramme de classes de notre solution :



Difficultés rencontrées

L'une des difficultés principales rencontrées lors de l'élaboration de ce travail pratique fut l'implémentation de la représentation et de la création du graphique, mais ceci fut rapidement résolu par la découverte de la librairie NetworkX qui permet de créer, d'associer et d'afficher des graphiques orientés de façon simple et efficace.

La deuxième difficulté rencontrée fut l'implémentation de l'algorithme de Dijkstra. En effet, malgré le caractère simpliste de cet algorithme, l'implémentation est une autre paire de manches, car on devait trouver une solution pour vérifier la quantité de colis de chaque type dans les nœuds du graphique.

Finalement, la dernière difficulté rencontrée fut l'implémentation de tous les détails du travail pratique tels que les différents robots ou les différents colis de l'entrepôt. Afin de ne pas oublier aucun détail, nous avons écrit un résumé qui excluait tous les détails inutiles à la réalisation de l'énoncé.

Conclusion

La réalisation de ce travail pratique nous a permis de mieux comprendre le fonctionnement de l'algorithme de Dijkstra et les mécaniques internes qui le déficient. De plus, on a eu un exemple concret de l'utilisation des graphiques orientés dans un élément quotidien de la vie. D'un autre côté, dû au fait que nous avons réalisé ce travail en python, nous avons eu l'opportunité d'améliorer ou d'agrandir nos connaissances à propos de ce langage. En plus de réaliser le fait qu'utiliser les librairies permet de sauver beaucoup de temps sur la conception de codes qui existent déjà. Finalement, nos attentes pour le prochain travail pratique sont l'application des graphiques orientés sur une plus grande échelle telle que sur le plan d'une ville comme Montréal où un automobiliste doit aller d'un point A à un point B.