Projet Modélisation 2018

1. Présentation du problème

Dans ce projet nous devions implémenter les algorithmes qui nous permettent de mettre en lumière les propriétés d’un graphe : connexité, cycle, cycle à valuations négatives. Un algorithme très pratique pour ce problème est le DFS. Il parcourt tout le graphe et note ainsi tous les arcs et leur direction. Il pourra donc détecter la présence d’un cycle et d’une composante connexe dans le graphe.

Grâce à ces propriétés, nous avons pu écrire les algorithmes de plus court chemin suivants : Dijkstra et Ford.

Dijkstra permet de trouver le plus court chemin (avec un coût par arc ou un temps, c’est au choix) grâce aux labels sur les arcs et sur les sommets. Cependant le graphe a pour contrainte de ne pas avoir de valuation négatives et pas de cycle non plus. C’est en cela que DFS est pratique. Un simple test permettra de savoir si l’utilisation de Dijkstra est possible ou non.

Dans la vie courante, ces algorithmes économiseraient beaucoup de temps et d’argent à ceux qui partent en voyage, par exemple, ou bien pour des sociétés de transports qui souhaitent livrer leur marchandise à moindre coût et/ou dans un laps de temps le plus petit possible.

1. Les choix liés à la programmation

Pour créer le graphe, nous avons réutilisé les classes que nous avons étudié dans le cours de Représentations des Connaissances. La structure d’un graphe est élémentaire : il est composé d’une liste de sommets et d’une liste d’arcs, tous deux héritant de la classe Maillon<> qui est une classe patron permettant de créer des chaînes.

Dans Maillon<> la donnée membre valeur est un pointeur sur l’élément de la liste qui est étudié (Sommet ou Arc). La donnée membre suivant est un pointeur sur l’élément qui vient juste après, qui est un maillon du même ordre que le reste de la liste.

Mais ces classes Graphe, Arcs et Sommets ne permettent pas de lire les fichiers gpr qui nous ont été fournis. C’est la classe ChargerGraphe qui fournit ce travail. Une instance de cette classe va lire le fichier fourni grâce à son chemin et les données récupérées ligne par ligne vont conduire à la création d’un arc ou d’un sommet, le tout donnant bien sûr le graphe résultant de cette analyse. Rien de bien compliqué jusque là.

Les choix que nous avons fait pour l’implémentation des algorithmes DFS, Dijkstra et Ford sont un peu plus poussés. Les algorithmes fournis en cours sont facile à faire sur le papier mais les transformer en code concordant avec les classes déjà existantes dans le projet demandaient un peu plus de réflexion.

1. Mise en place de Dijkstra

Afin de respecter l'algorithme de Dijkstra le plus fidèlement possible, j'ai décidé de mettre en place 2 conditions à appliquer sur l'ensemble des arcs et des sommets : Les valuations ne peuvent être négatives et l'entièreté des valuations doit être positive stricte (A la création des sommets à partir d'un .gpr, si la borne inf ou la borne sup ne sont pas précisées, elles sont initialisées à 0).

Si le graphe respecte ces conditions, j'initialise l'étiquette de tous les sommets à INT\_MAX (inclus dans limits.h) puis j'initialise celle du premier sommet à 0.

Je récupère le tout premier sommet passé en paramètre, et je l'ajoute à la liste des sommets marqués.

Je lance la boucle tant que la liste des arcs n'est pas vide ou bien que fin soit faux.

Dans cette boucle, je récupère tous les **arcs** **adjacents** à mon **sommet** **actuel**. Comme un arc est composé de deux sommets début et fin, je peux alors récupérer tous les **sommets** **adjacents** à partir de cet arc, et je les stocke dans une liste de sommets à traiter pour cette itération de la boucle.

J'initialise son étiquette avec le minimum entre son actuelle (INT\_MAX) ou celle du sommet précédent + la valuation de l'arc (0 + valuation de l'arc la toute première itération de la boucle).

Une fois tous les sommets mis à jour, j'avance dans la liste d'arcs et je regarde s'il y avait au moins un sommet à traiter. Si c'est le cas, je récupère celui qui avait l'étiquette la plus petite et ce sommet sera utilisé lors de la prochaine itération.

Un autre if vérifie que le sommet actuel a bien changé de valeur (donc étiquette différente de INT\_MAX), et s'il ne faut pas sortir de la boucle j'ajoute le sommet testé à la liste des sommets marqués.

Une fois sortie de la boucle, je retourne une liste de tous les sommets (du puits jusqu'à la source).

Il était nécessaire de créer parmi les paramètres de Dijkstra un pointeur de fonction car il n'est pas possible de simplement déterminer si l'utilisateur veut utiliser les coûts ou le temps (qui sont deux unités possibles pour Dijkstra).

Ce pointeur de fonction retournera soit le coût, soit le temps d'un arc.