Rapport TP1 - Graphes

Dans le cadre du cours de
Structures discrètes - LOG2810

Marc-André Jolicoeur - 1846304
Alexandre Boudreault - 1486776
Mathieu Bélanger - 1850591

Mardi, 24 octobre 2017

Introduction

Le travail pratique avait pour but de nous familiariser avec les notions de graphes vus en cours. La mise en contexte du travail est qu'un jeune étudiant a obtenu un stage dans une petite entreprise. Il n'a tout simplement pas les connaissances nécessaires pour effectuer son travail. Nous sommes donc appelés à le dépanner.

Le premier choix qui nous est confronté est le choix du langage de programmation. En discutant, nous avons réalisé que nos connaissances en Python étaient extrêmement limitées et que ce TP serait une excellente occasion pour les développer. Après plusieurs heures de débogage et de requêtes «Stack Overflow», nous avons réussi à terminer son projet.

L'application produite est une application console. Avec tout le charme et l'élégance de MS-DOS, la fenêtre de bienvenue offre à l'utilisateur 3 options : les Drones, les recettes, ou la sortie du programme.

L'option drone permet de déterminer s'il est possible d'effectuer la livraison d'un colis à l'aide d'un drone. L'application requiert un fichier texte ayant un format bien précis et demande 3 entrées de l'utilisateur : le point de départ de la livraison, la destination du colis et la taille du colis. L'application effectue ensuite les calculs et affiche les détails du trajet le plus efficace qui respecte les critères de sécurité.

L'option recette, quant à elle, permet de gérer quelques recettes en fonction de leurs ingrédients. La liste des recettes et des ingrédients est fournie dans un fichier texte, encore une fois, dans un format bien défini. L'application est munie d'un menu offrant la possibilité d'afficher la liste des recettes et ingrédients et d'afficher le diagramme de Hasse représentant les relations d'ordres dans le graphe.

Présentation de l'application produite et la démarche

Drones

Pour la première partie du TP, nous avions un fichier texte contenant toutes les stations de recharge du drone ainsi que toutes les liaisons entre les 19 arrondissements de Montréal. Nous avons opté pour l'utilisation d'une liste d'adjacence versus la matrice d'adjacence, car l'utilisation d'une liste permet de limiter la complexité d'un appel de dijsktra à $O(n)$, versus $O(n^2)$ pour la matrice d'adjacence. Ainsi l'utilisation de la *liste* sous forme d'un dictionnaire de dictionnaire (sommet1, (sommet2, distanceS1S2) est plus efficace et plus lisible dans le code.

Dans le contexte d'une utilisation normale de l'application, nous avons estimé qu'un nombre important de calculs de chemins seraient effectués entre chaque mise à jour de la carte. À cet effet, nous avons décidé de «pré-calculer» le chemin le plus cours pour toutes les combinaisons possibles (début, fin) de sommets lorsqu'une carte est lue par le programme. De cette manière, lorsqu'un utilisateur entre les paramètres de recherche d'une route, le programme ne fait qu'accéder à la case mémoire où l'information se trouve déjà.

L'implémentation de l'algorithme de Dijsktra est standard, aucune modification ne lui a été apportée. En prenant pour acquis que le programme effectuera surement un grand nombre de calculs de trajet sur une même carte, nous estimons qu'un calcul initialement lourd sera avantageux lorsqu'on le compare à un appel à Dijkstra par trajet. Après avoir décidé de produire une matrice des distances dès la lecture du fichier contenant les données, nous avons donc opté pour la variante de l'algorithme qui retourne *l'arbre* des chemins les plus cours d'un sommet donné à tous les autres du graphe. Ceci nous permet d'appeler qu'une seule fois la fonction Dijkstra par sommet.

Une fois la matrice des distances calculée, il s'agit simplement de déterminer le chemin optimal qui respecte les conditions.

- On a deux modèles de drones à notre disposition : les drones à 3.3A et les drones à 5A. Nous devons prioriser les drones 3.3A puisqu'ils sont significativement moins chers.
- La charge dans les piles ne peut pas descendre en dessous de 20%. Une pile à 20% de sa capacité maximale représente un danger public. Cette situation est donc à éviter à tout prix.
- La consommation des piles est en fonction du temps de trajet et varie selon le drone utilisé et la taille du colis :
    - Les drones 3.3A:
        - 10%/10min de trajet
        - 20%/10min de trajet
        - 40%/10min de trajet
    - Les drones 5A:
        - 10%/10min de trajet
        - 15%/10min de trajet
        - 25%/10min de trajet

La logique derrière la recherche de trajet est qu'il y a trois types de chemins possibles :

1. D'un sommet normal à un autre sommet;
2. D'un sommet normal à une station de chargement;
3. D'une station de chargement à une autre.

Le trajet le plus efficace serait effectué par le drone 3.3A et passerait par le chemin le moins couteux selon l'algorithme de Dijkstra. Nous notons un tel chemin le chemin direct.

Si ce chemin ne respecte pas les conditions de sécurité (à cause de la décharge de la pile selon la taille du colis), nous procédons à la recherche d'un chemin composé, soit un chemin qui commence au point de départ, qui fait un (des) arrêt(s) à une (quelques) station(s) de recharge, pour finalement se diriger vers sa destination. Ceci est représenté par une combinaison de plusieurs chemins de types 2 et 3.

Afin d'implémenter la recherche du meilleur chemin en fonction de la taille du paquet et des contraintes d'utilisation des drones nous avons procédé comme suit:

1. La route plus rapide et directe avec le drone 3.3A est-elle sécuritaire?
    a. Si oui, cette route est privilégiée.
    b. Sinon, la prochaine étape est d'essayer de construire un trajet empruntant des stations de recharge afin de compléter sécuritairement la livraison.
2. La route «composée» (de forme *départ --> station de recharge --> fin*) la plus courte est-elle règlementaire?
    a. Si oui, cette route est privilégiée.
    b. Sinon, essayer de nouveau de trouver une route «composée» passant par deux stations de recharge.
    c. S'il n'y a toujours pas de trajet, refaire une recherche de trajet en augmentant le nombre de stations de recharge à chaque fois jusqu'à ce qu'on puisse déterminer un trajet, ou que l'on ait atteint le nombre maximal de stations de recharge sur la carte.
3. Si un trajet sécuritaire demeure impossible, les étapes 1 et 2 sont appelées de nouveau afin de vérifier s'il existe un trajet sécuritaire en utilisant le drone 5A.

4. S'il n'existe toujours pas de trajet sécuritaire, la livraison est refusée puisqu'on ne peut pas garantir une livraison sécuritaire.

Lorsque nous obtenons un chemin direct qui respecte les conditions de sécurité, nous avons privilégié la diminution du temps de trajet plutôt que l'arrêt aux stations de recharge. Concrètement, ceci signifie que, lorsque possible, les drones ne s'arrêtent pas aux stations de chargement, même s'ils passent par cet arrondissement. Si nous avions voulu que les drones s'arrêtent à chaque fois qu'ils passent dans un arrondissement ayant une station de recharge, il serait relativement simple d'effectuer cette modification :

- Dans la fonction Dijkstra, lorsque la fonction détermine la newDistance, il serait possible d'ajouter une condition qui vérifie si le voisin étudié contient une station de recharge. Si c'est le cas, il faudrait ajouter un 20 minutes à la valeur de new distance.
- Dans ce cas, dans la fonction combineRoutes, il ne serait donc plus nécessaire d'ajouter un 20 minutes à chaque concaténation de trajet, puisque ce 20 minutes d'arrêt serait déjà pris en compte.
- Pour afficher le trajet, il ne serait donc plus nécessaire d'indiquer les stations auxquels le drone s'arrête pour se recharger puisque, par défaut, s'il y a une station de recharge sur le trajet, le drone s'arrêterait. La fonction combineRoutes aurait donc besoin d'éliminer la redondance des items qui se produit lorsque deux trajets sont unis.

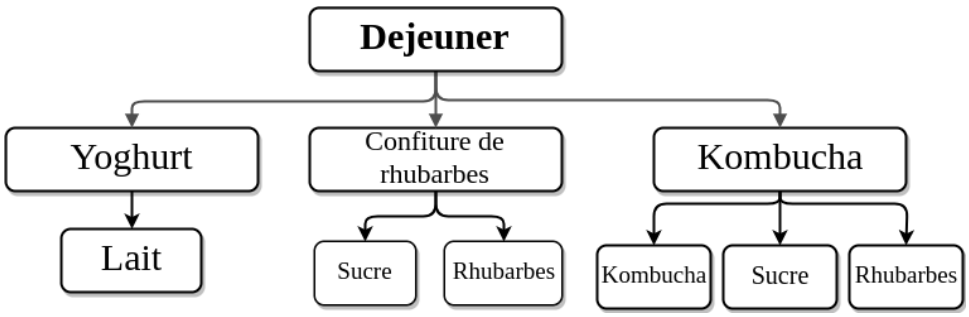
[début, sommet1,cStation] + [cStation,sommet2,fin] = [début,sommet1,cStation,cStation,sommet2,fin]

Recettes

Pour la deuxième partie du TP, nous disposions d'un fichier texte dans lequel chaque aliment est associé à un numéro. Nous avons créé un dictionnaire contenant le numéro (la clé) associé à chaque aliment (la valeur). Par exemple, 1 : "Déjeuner".

Le fichier texte indique que certains aliments sont composés de plusieurs ingrédients. Nous avons alors stocké dans un autre dictionnaire les recettes associées à chaque ingrédient (ex: 18 : [3, 11, 8]). Ainsi, sachant que 18 = sucre, 3 = confiture de rhubarbes, 11 = mousse au citron et 8 = kombucha, on sait que le sucre est un sous-ingrédient de la confiture de rhubarbes, du kombucha et de la mousse au citron. Puisque la confiture de rhubarbe et le kombucha sont des sous-ingrédients du 1 = déjeuner (3 : [1] et 8 : [1, 8]), on peut donc dire que le sucre est indirectement un sous-ingrédient du déjeuner. Une fois ces 2 dictionnaires construits, nous étions en mesure d'établir une hiérarchie de la liste d'ingrédients fournie.

Certains aliments contiennent plusieurs ingrédients, et parfois ces ingrédients contiennent eux-mêmes plusieurs ingrédients. Ainsi, pour trouver l'ensemble des recettes associées à un ingrédient, il faut parcourir tous les niveaux supérieurs du diagramme de Hasse. Un exemple facilite la compréhension :



Ainsi, d'après le diagramme, il est plus facile de constater que le sucre est un sous-ingrédient du kombucha et de la confiture de rhubarbes, mais aussi du déjeuner.

Notre fonction genererHasse est de type récursif. Elle vérifie d'abord si l'ingrédient choisi est contenu dans une recette. Si c'est le cas, la récursivité fait en sorte que l'on vérifie si cette recette est elle-même incluse dans une autre recette. Un problème est survenu lorsque la fonction récursive a rencontré le kombucha dû au fait que le kombucha contient du kombucha dans sa liste d'ingrédients. Ainsi, la fonction se retrouvait prise dans une boucle infinie. Il a donc fallu limiter à 1 fois la récursion dans le cas où l'aliment est inclus dans sa propre liste d'ingrédients.

Ayant fait notre projet en python, on n'utilise pas de classe, mais plutôt des fonctions et des dictionnaires. Notre premier dictionnaire est composé des couples **(clé) : "nom de l'aliment"** (ex: 1 : "Déjeuner") et le deuxième est composé de couples **(clé) : [recette$_1$, recette$_2$,...]** (ex: 18 : [3, 11, 8]). Nos fonctions interagissent avec ces 2 dictionnaires pour obtenir l'affichage voulu.

Difficultés rencontrées

Une difficulté fut le traitement des erreurs. Nous avions dû passer un certain temps à déterminer quelles erreurs risquent de se produire et comment procéder avec ces valeurs sans que l'application cesse de fonctionner.

C'est erreurs peuvent provenir d'une entrée utilisateur incohérente, un fichier non-standard, ou même des erreurs dans l'application elle-même.

Une autre difficulté est que nous n'avions pas accès à des données de validation. La seule manière de savoir si nous obtenons un résultat valide est de le calculer à la main. Avec 19, sommets, 2 drones et 3 tailles de colis, il est farfelu d'effectuer à la main le calcule pour chaque trajet possible. De plus, si un cas a bien fonctionné, comment peut-on valider que l'application fonctionnera dans un autre cas? Il faudrait répéter l'expérience sur tous les cas possibles, ou sur des cas représentatifs. Il serait pertinent de pouvoir accéder à une liste d'entrée et de sorties afin de pouvoir comparer les résultats calculés aux résultats attendus.

C'est certain que dans le cadre du stage de notre collègue, il est surement impossible d'obtenir un tel fichier. Mais dans le cadre d'un cours, la production d'un tel fichier serait surement envisageable.

En lien avec la difficulté précédente, l'affichage attendu dans le cadre du diagramme de Hasse n'était pas évident. En effet, nous avions mal interprété les listes attendues. Un exemple de liste aurait pu être fourni pour faciliter la compréhension.

Conclusion

Nous avons donc réussi à dépanner notre cher collègue. Notre application permet de déterminer la route la plus efficace entre deux points qui respecte les contraintes de sécurité établies par le problème. Une autre application permet de déterminer la hiérarchie parmi une liste d'ingrédients. Ayant effectué le travail en utilisant le langage Python, nous avons aussi acquis de nouvelles connaissances qui nous seront surement utiles dans de futurs projets.