POLYTECHNIQUE MONTRÉAL

LOG2810

Structures Discrètes

TP1

Graphes

Equipe 29

éléments évalués	Points
Qualité du rapport : respect des exigences du rapport, qualité de la présentation des solutions	2
Qualité du programme: compilation, structures de données, gestion adéquate des variables et de toute ressource (création, utilisation, libération), passage d'arguments, gestion des erreurs, documentation du code, etc.	2
Composants implémentés : respect des requis, logique de développement, etc.	
C1	3
C2	3
C3 Divisé comme suit	5
C3.1 Implémentation correcte de Dijkstra	2
C3.2 Choix de chemin selon le colis	2
C3.3 Choix du véhicule médicalisé/du refus selon le cas	1
C4	3
C5	2
Total de points	20

Elie Rouphael 1829529

Mourad Younes 1832387

Lynn Chararbsissy 1832027

I. Introduction

Dans le but de faciliter la vie des patients lors des transports médicaux (en assurant leur arrivée la plus rapide à un centre hospitalier) et d'économiser par rapport au coût de transport (éliminer le besoin de payer un chauffeur), les véhicules médicalisés autonomes constituent le centre du domaine de travail d'une start-up. Le but est alors de réaliser ce projet. Pour ce faire, il s'agit principalement de faire en sorte que ces véhicules-là puissent trouver, en fonction de la situation introduite en leur système, le trajet optimal qui leur permettra d'arriver à leur destination le plus rapidement possible.
Afin de pouvoir trouver le bon chemin à emprunter, il se doit de commencer par avoir à notre disposition une carte. Cette carte est alors représentée par un graphe connexe dont les sommets constituent les diverses stations CLSC et les arêtes quant à elles illustrent les distances entres ces points.

On dispose de deux sortes de véhicules : NI-MH et LI-ion. Plus encore, le véhicule devra choisir le trajet optimal en fonction de divers paramètres qui affecteront ce choix, notamment :
-Le point de départ
-Le point d'arrivée
- le niveau de risque du patient

De plus, afin d'optimiser les économies faites, on opte pour le NI-MH automatiquement (vu qu'il est moins cher et a un risque de surchauffe moins élevé) à moins que ce véhicule ne permette pas l'arrivée du patient.
En effet, vu que les véhicules sont autonomes, quelques points centraux de CLSC sont munis de bornes de recharge ou ceux-là pourront faire leur plein d'énergie. D'autre part, chaque type de véhicule autonome se caractérise par un facteur de perte d'énergie qui lui est propre. Plus précisément, facteur varie aussi en fonction du risque du patient.
De ce fait, le système ne considèrera pas les trajets qui feront passer les batteries du véhicule en dessous de 20% dans le but de maintenir un certain niveau sécuritaire au patient. Dans le cas où cette contrainte ne permet pas une arrivée rapide, on optera pour le véhicule à batterie en LI-ion. Si cette condition empêche aussi le déplacement via LI-ion, le patient se verra rejeter la demande et le transport lui sera refusé.
Sans oublier que chaque véhicule débute son trajet à 100% de charge et que la durée requise pour la recharge d'un véhicule, quel qu'il soit, est 120 min. En effet, cette durée devra être prise en considération lors du calcul et de la recherche du plus court chemin.
Les différents facteurs de perte d'énergie sont regroupés dans le tableau ci-dessous.

Tableau I. variation du facteur de perte d'énergie en fonction du véhicule et du risque du patient [1]

Véhicule / Risque	NI-MH	LI-ion
Faible	-6%/h	-5%/h
Moyen	-12%/h	-10%/h
Élevé	-48%/h	-30%/h

D'autre part, le système devra aussi être capable de lire un graphe puisque l'utilisateur est en mesure de changer la carte. Il faudra alors lire et extraire les données pour pouvoir afficher par la suite les sommets et distances obtenues, ainsi que les diverses liaisons entre les sommets.

De plus, le système devrait être capable d'extraire un sous-graphe qui, lui, contient le chemin le plus long qu'un véhicule peut parcourir sans avoir besoin de recharger sa batterie.

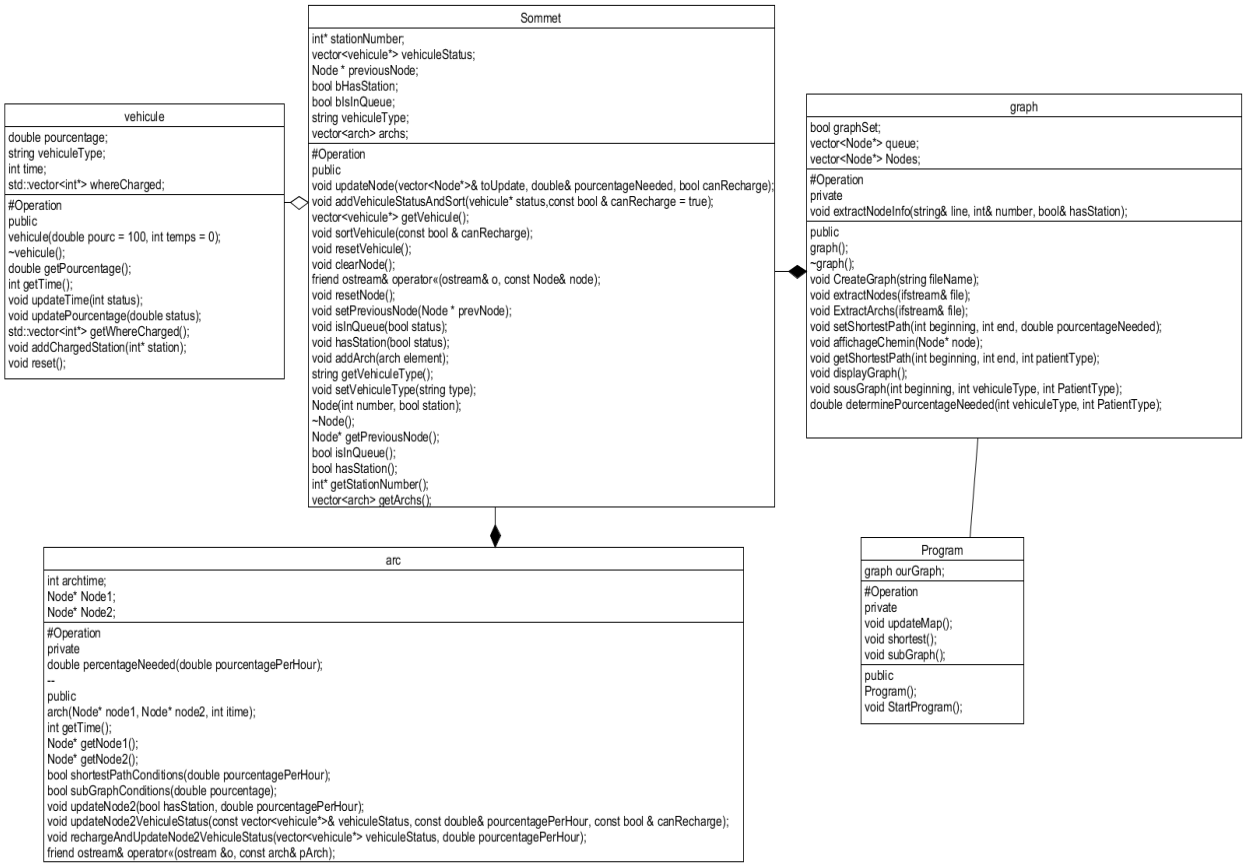
Cette fonctionnalité demandera de l'utilisateur le véhicule désiré, le risque du patient ainsi que le point de départ.

Finalement, un mode « quitter » sera développé pour arrêter le programme.

II. Présentation du travail

Dans le but de concevoir la solution demandée, on opte pour la programmation en C++ et l'on établit le diagramme de classe suivant :

Figure 1. Diagramme de classe adopté



Par conséquent, on a une classe représentant chaque nœud, une pour les arcs, une autre pour le graphe. Cette dernière comporte entre autres les méthodes permettant d'obtenir le plus court chemin et une autre qui extrait un sous-graphe.
On dispose aussi de la classe véhicule qui comporte le numéro du nœud où ce dernier a été chargé. De plus, un attribut qui décrit la durée requise pour arriver au nœud présent. Une classe Programme se charge de l'affichage.
Afin de parvenir à ces fins, on procède par fonctionnalité.

A. Mettre à jour la carte

Dans le cas où l'utilisateur choisit de changer de carte (et alors de graphe), le plus court chemin ainsi que le sous-graphe sont réinitialisés. De plus, la carte est lue (on lit le fichier en demandant de l'utilisateur le nom de ce dernier), les nœuds,

les arcs ainsi que les durées et la présence ou non d'une borne de recharge sont placés dans les bons attributs. Finalement, on affiche le graphe obtenu selon le format suivant :

(objet1, ((objet_voisin1_1, durée1_1), (objet_voisin2_1, durée2_1), ..., (objet_voisinn_1, duréen_1)))
(objet2, ((objet_voisin1_2, durée1_2), (objet_voisin2_2, durée2_2), ..., (objet_voisinn_2, duréen_2))) [1]

B. Plus court chemin

Pour trouver le plus court chemin, on attribue à chaque nœud un vecteur d'arcs qui contient les arcs comportant les sommets liés à ce nœud. Ainsi, à partir du point de départ, on commence par observer les nœuds aux alentours de cette position. On met à jour la distance ou durée séparant chaque nœud du point de départ. Tous les nœuds visités sont stockés dans la queue. D'autre part, on met à jour la queue de sorte qu'elle soit triée par ordre de distance. En d'autres termes, les nœuds les plus proches du point de départ seront placés en début. Cette opération est répétée à chaque nœud, (toujours prenant la distance à partir du point de départ) jusqu'à ce que tous les nœuds soient parcourus. Cependant, en avançant de nœud en nœud, on donne à l'attribut 'previousNode', la valeur du numéro du sommet à travers lequel il faut passer pour arriver à ce nœud le plus vite possible.

C. Extraire sous- graphe

Cette méthode fonctionne de manière similaire à celle qui trouve le plus court trajet :
On commence par demander de l'utilisateur le point de départ, le type de batteries désiré ainsi que le risque du patient. Pour ce qui suit, pour chaque nœud à partir du point de départ, on prend en considération les nœuds auxquels il est lié et on les ajoute à la queue tout en mettant à jour les distances par rapport au point de départ. Cependant, cette fois-ci on trie la queue par ordre décroissant de durée. De plus, le véhicule ne peut pas faire le plein d'énergie et doit de ce fait s'arrêter lorsque le nœud qui suit fera baisser son pourcentage d'énergie restant à moins que 20%. Ceci aboutira à la fin du parcours au chemin le plus long que le véhicule choisi dans les conditions de départ et risque du patient sans recharger.

D. Quitter

Cette fonction réinitialise, par le biais des destructeurs, le plus court trajet trouvé, le sous-graphe, la queue, ainsi que toutes les entités utilisées. Par la suite, on sort du programme.

E. Interface

On affiche le menu en demandant de l'utilisateur d'entrer les paramètres nécessaires à la fonction demandée. Pour chaque option existante, on appelle la méthode correspondante. Sans oublier de lancer des exceptions au cas ou l'entrée de l'utilisateur est invalide.

III. Difficultés rencontrées et solutions

Durant la conception de ce projet, on a fait face à divers problèmes. Notamment, au niveau de la comparaison des chemins possibles dans la fonction chargée de trouver le plus court chemin. En effet, l'élément de contrainte reposait sur le fait de sauvegarder plusieurs possibilités de charge dans le véhicule (s'il a chargé à la borne de recharge ou pas). Plus encore, il faut sauvegarder les valeurs prises par le pourcentage d'énergie restant dans la batterie dans plusieurs cas afin d'être en mesure de choisir plus tard, lorsque l'on est sur du nœud où il faut faire le plein d'énergie.
Afin de résoudre ce problème, on opte pour l'ajout d'un attribut « vehiculeStatus » constitué d'un vecteur de véhicules afin de pouvoir suivre la procédure sans avoir à recommencer du début à chaque fois que l'on arrive à un nœud où il est nécessaire d'avoir fait le plein d'énergie plus tôt dans le parcours pour être capable de poursuivre.

D'autre part, un autre problème qui pris beaucoup de temps à résoudre se résume par le fait que l'algorithme a été implémenté au départ sans prendre en considération les différentes contraintes imposées. Ceci fait qu'une fois ces dernières rajoutées, l'algorithme ne respectait plus celui de Dijkstra. Ce problème a causé un grand gaspillage de temps. Pour remédier à cela, de multiples tests sur papiers ont permis de trouver la manière adéquate de coder l'algorithme ainsi que de s'assurer de la fiabilité des résultats. Cette stratégie a été employée à plusieurs reprises afin d'accélérer le processus et d'être certains de la progression du travail.

Finalement, en ce qui concerne le sous-graphe, ce dernier a premièrement causé beaucoup de confusion quant au but réel de la fonction. Après une clarification du prof, le code fut conçu de telle façon qu'il était beaucoup trop similaire à la fonction chargée de trouver le plus court trajet. Mais cette méthode a induit le résultat de l'implémentation dans une boucle infinie. En effet, il existe des différences à considérer et des contraintes à ajouter.
Par conséquent, il a fallu rajouter la condition qui empêche le passage par le même nœud plus qu'une fois.

IV. Conclusion

Ce laboratoire a servi à développer nos capacités en programmation C++. Plus encore, cela nous a donné un avant-gout d'une situation dans la vie quotidienne d'un ingénieur informatique/logiciel. En effet, nous sommes arrivés au bout d'une semaine et demi à concevoir un système complet partant de la seule information qui est : la problématique à résoudre. D'autre part, ce projet nous a permis d'améliorer nos stratégies de travail en équipe, de gestion du temps et de répartition des tâches.

Quant aux nouvelles connaissances acquises, la plus importante serait : apprendre comment penser afin de prendre en charge la totalité d'un projet sans oublier une étape ni une considération.

En ce qui concerne le prochain laboratoire, on sera plus préparé au niveau de la méthode de travail et la répartition des tâches. Par conséquent, la productivité augmentera considérablement. De plus, on espère avoir davantage de temps à accorder au TP2 vu que nous avions beaucoup trop d'intra durant ces deux semaines. Le temps passé sur ce TP dépasse de peu 48h.

En somme, les véhicules médicalisés autonomes constituent une solution réelle applicable qui permettrait d'améliorer la situation des patients dans le monde. On pourrait peut-être commencer à l'adopter à Montréal.

Référence

[1] énoncé TP1