Travail Pratique #1 : Graphes

LOG2810 : Structures discrètes

Rapport remis par :
Mariam Sarwat (1928777)
Stéphanie Mansour (1935595)
Yasmina Abou-Nakkoul (1897266)

Équipe: 22
Groupe : 01

École Polytechnique de Montréal
Date de remise (03-03-2019)

1. Introduction

Dans le cadre du cours Structures Discrètes, un programme à compléter nous a été donné. L'objectif de ce programme était d'appliquer les notions de théorie de graphe en créant le jeu «Qui est-ce ?» . Ce jeu permet de poser une série de question à l'adversaire, soit l'utilisateur du programme, pour permettre à l'agent de deviner les deux individus mystères, à partir d'une liste d'individu contenue dans fichier texte donné, auxquelles l'adversaire pense. Tout au long du jeu, l'adversaire peut vérifier la liste des suspects restants selon l'agent. Après avoir réussi ou non à deviner les noms, l'agent demande à l'adversaire trois caractéristiques indésirables pour ensuite trouver le meilleur chemin (la meilleur amitié) entre ces deux individus mystères. Dans ce rapport est expliqué la façon dont nous avons abordé les différents tâches données, la difficulté à les compléter et une conclusion résumant l'utilité de ce laboratoire.

2. Présentation des travaux

Pour réaliser ce laboratoire, nous avons utilisé une approche orientée objet en C++. Pour mieux comprendre la façon dont nous avons abordé cette tâche, ci-dessous se trouve les différentes classes créer expliquant leurs rôles et les méthodes relier.

Individu

La première classe créée est celle d'Individu qui possède comme attribut le nom de la personne, la couleur de ses cheveux, de ses yeux et le département auquel elle appartient. Cette classe possède seulement les méthodes d'accès (getters) et de modifications (setters) des attributs énumérés ci-dessus.

Relation

La classe relation, quant à elle, contient trois attributs: deux Individus et un entier qui correspond au pourcentage de relation existant entre ces deux derniers. Tout comme la classe d'Individu, cette classe possède seulement les méthodes d'accès et de modification de ses attributs.

Reseau Social

La classe reseauSocial est essentielle puisqu'elle sauvegarde l'information sur tous les individus et leurs relations. En effet, elle contient les méthodes de lecture de fichier et les méthodes d'affichage:

creerReseauSocial(string fichierIndiv, string fichierRelat): cette fonction s'occupe de lire le fichier texte qui contient les informations sur les individus et leurs caractéristiques, ainsi que le fichier sur le pourcentage de relation entre les différents individus. Chaque fichier est lu ligne par ligne pour récupérer l'information nécessaire et la conserver dans un vecteur.

afficherReseauSocial(vector <Relation> relations): une méthode permettant d'afficher le réseau social créé à l'aide de la fonction ci-dessus.

afficherIndividus() et *afficherRelations()*: ces méthodes permettent d'afficher les données lus des fichier contenant les informations des individus et des relations, respectivement.

Guess who

Une des classes les plus importantes est celle « Guess Who » puisqu'elle possède toutes les méthodes pour la réalisation de la tâche principale du laboratoire: la création du jeu «Qui est-ce ?».

devinette(vector<Individu> vecteur): la méthode principale de la classe. Elle prend en charge la chaîne de question à poser à l'adversaire pour deviner les individus mystères. Cette fonction est composée d'un switch-case où chaque cas contient une question à poser à l'adversaire (ex: « Les individus mystère ont-ils les cheveux noirs? »). Ce dernier peut répondre de trois manière différentes : O (pour oui pour les deux), N (pour non pour les deux) ou U (oui pour une personne). Dépendamment de la réponse entrée, on procède au prochain cas.

yeux(string charact, ***bool*** *type)*, *departement(string charact,* ***bool*** *type)* et *cheveux(string charact,* ***bool*** *type)*: lorsque l'adversaire entre "O", on élimine tous les individus qui n'ont pas cette caractéristique. Si elle entre "N", on élimine tous les individus qui ont cette caractéristique. Si elle entre "U", on ne fait rien, mais on conserve la réponse. Ces trois fonctions accomplissent cette tâche pour chaque caractéristique.

demanderQuestion(string question): cette fonction est appelée à l'intérieur de *devinette(vector<Individu> vecteur)*, et est en charge de demander les différentes questions et de conserver la réponse. La question est demandé tant que l'adversaire n'entre pas une réponse valide (soit "O", "N" ou "U"). Elle garde aussi en mémoire le nombre de question que l'agent pose. La méthode *afficherListeRestant(string reponse)* s'y trouve aussi. Elle est appelée si l'adversaire répond "S" à une question, pour afficher la liste des suspects restants selon l'agent.

identifierIndividus(vector<Individu> vecteur): appelée après la fonction *devinette(vector<Individu> vecteur)*, elle pose les deux individus mystères retrouvés par l'agent. Par la suite, l'adversaire indique si c'est juste (soit "O", "N" ou "U"). Dans le cas où

l'adversaire répond "O", l'agent établit que son hypothèse est juste. Dans le cas "N", l'agent demande à l'adversaire d'entrée l'identité des deux individus. Finalement, si l'adversaire répond "U", l'agent demande lequel des deux hypothèses est faux et, par la suite, l'adversaire pourra entrer le bon nom. Par contre, si l'adversaire tente d'entrer un nom invalide, l'agent demandera à l'adversaire de rectifier sa réponse.

Il est important de noter qu'à la fin de la fonction devinette, il est possible que plus que deux individus aient toutes les caractéristiques indiquées par l'adversaire. Alors, pour rendre notre code plus simple, nous prenons les deux premiers individus de la liste restantes produites par devinette.

enleverArcsIndesirables(vector<Relation> vecteur): cette fonction est en charge de demander à l'adversaire trois caractéristiques qu'il juge indésirable. Après l'insertion de chaque caractéristique, chaque relation entre deux individus qui partagent l'un de ces traits est annulée, soit mis à zéro, sauf si un des individu est un individu mystère.

Chemin

minDistance(int dist[], bool sptSet[]) : cette méthode sert à chercher le lien entre deux nodes ayant le coût le plus bas, parmi celles non encore incluses dans la chaîne liant le premier individu mystère au deuxième.

trouverChaineContacts(vector<Individu> vecteur, vector<Relation> vecteur1, Individu mystere1, Individu mystere2) : dans cette méthode, nous commençons d'abord par créer la matrice du graphe des relations restantes entre tous les individus du fichier de départ. Nous définissons ensuite notre source comme étant notre premier individu mystère puis nous parcourons la matrice en faisant appel à la fonction *minDistance.* Nous ajoutons une relation et son coût à notre chaîne, que lorsque la relation a le coût le plus bas et que le noeud et sa relation n'ont pas déjà été parcourus. Cette fonction imprimera la chaîne de contacts recherchée.

Main

Dans le main on retrouve le menu principal du programme. Ce menu a été créé à l'aide d'un switch-case où chaque cas est une option du menu.

Cas 'a': permet de lire le fichier d'individu et celui de relation indiquer par l'adversaire, afin de créer le réseau social en faisant appel à la fonction *creerReseauSocial(string fichierIndiv, string fichierRelat).* Cette option doit absolument être exécutée avant l'option 'b', 'c' et 'd'.

Cas 'b': en sélectionnant cette option, le réseau social créé ainsi que le contenu des deux fichiers lus en 'a' sera affiché à l'aide des méthodes *afficherIndividus()*, *afficherRelations()* et *afficherReseauSocial(vector <Relation> relations)*.

Cas ‘c’: donne l’option à l’adversaire de jouer au jeu «Qui est-ce ?». C’est ici qu’on fait appel aux fonctions *devinette(vector<Individu> vecteur), identifierIndividus(vector<Individu> vecteur)* et *enleverArcsIndesirables(vector<Relation> vecteur)* expliquées ci-dessus.

Cas ‘d’: permet d’afficher tous les résultats obtenus durant l'exécution du cas ‘c’ en utilisant la fonction *afficherResultat()*, une méthode permettant à l’agent d’afficher les résultats de ses accomplissements. En d’autres mots, il affichera:

- le sous-graphe des caractéristiques désirables
- la meilleur chaîne entre les deux individus mystères
- le nombre de questions posées par l’agent durant le jeu «Qui est-ce ?»
- le nom des individus mystère trouvés
- le nom des individus mystère qui n’ont pas été devinés, dans certains cas
- les trois caractéristiques indésirables.

Cas ‘e’: cette option est affichée pour rendre possible à l’adversaire de quitter le programme.

Le programme réaffiche le menu tant que l’option ‘e’ n’a pas été choisie. De plus, le program affiche le menu de nouveau si l’utilisateur entre un index invalide.

Diagramme de classes

Ci-dessous se trouve le diagramme de classe présentant les relations entre les différentes classes de notre programme:

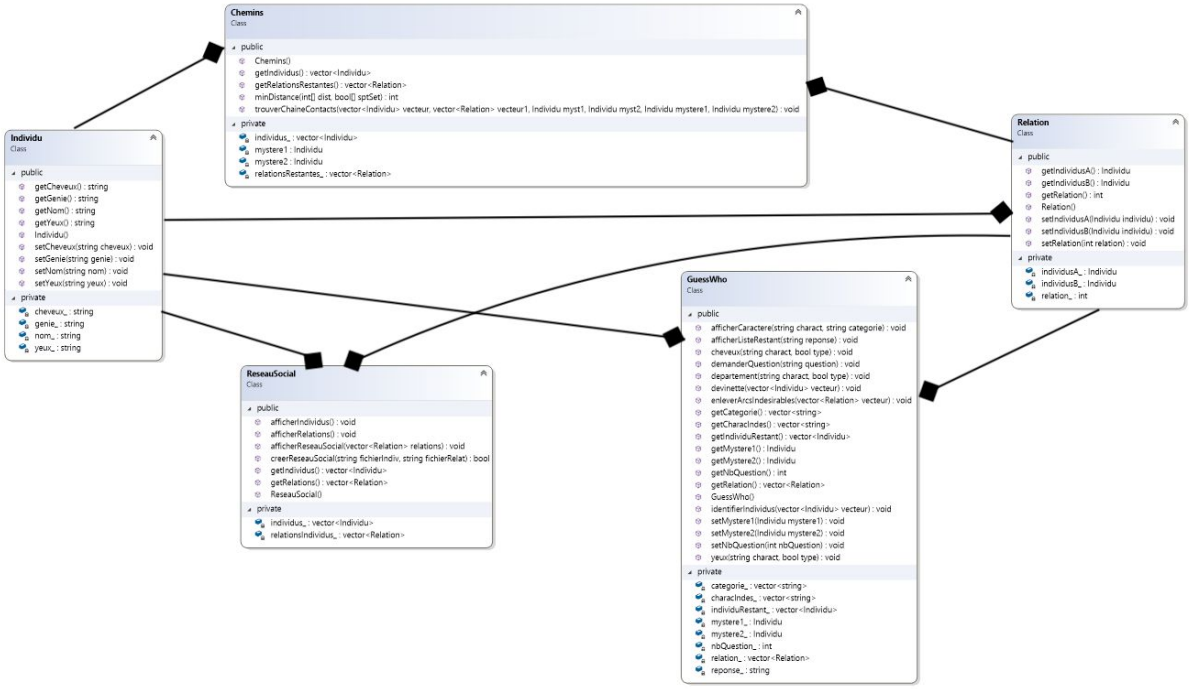

Figure 1.0 : Diagramme de classes

3. Difficultés rencontrées

Une des difficultés que nous avons rencontrés a été la conception du programme. En effet, nous avons reçu beaucoup d'information d'un coup et avons dû trouver un moyen approprié de tout séparer. Suite a une longue rencontre, nous avons réussi à catégoriser toutes les différentes tâches.

Parmi les différentes méthodes à implémenter, la méthode *trouverChaineContacts()* fut définitivement la plus compliquée. Malgré le fait que nous avons l'algorithme Dijkstra a notre disposition, l'adapter à notre situation a été un véritable défi et nous avions dû faire plusieurs recherche pour comprendre la complexité.

4. Conclusion

Ce laboratoire nous a été utile pour appliquer la théorie du cours, tel l'algorithme Dijkstra et la théorie des graphes. En effet, leurs application nous a permis de mieux comprendre la logique derrière ces derniers.

Par contre, la densité du travail de ce laboratoire a effectivement été lourde. Nous espérons avoir plus de temps pour compléter le prochain travail pratique, ou bien que la densité de travail sera moins lourde. De plus, nous aurions appréciés que l'horaire du cours soit adapté pour avoir une séance de TP, puisque ceci nous a désavantagé et nous a retardé dans la compréhension du laboratoire puisque nous nous trouvions a avoir plusieurs questions, ce qui a prolongé la réalisation du travail.

En résumé, nous espérons que le prochain laboratoire ait une charge proportionnel au temps accordé en classe pour la compréhension du travail exigé, tout en étant éducatif et nous permettant d'appliquer la théorie du cours comme celui-ci.