Paul Clas 1846912

Mazigh Ouanes 1721035

Simon Richtot 1637043

Rapport du TP1 : GRAPHES

LOG 2810 Structures Discrètes

Présenté à :

John Mullins

Justine Pepin

Paulina Stevia Nouwou Mindom

Remis le

3 mars 2019

Département de génie informatique et génie logiciel

Polytechnique Montréal

1. Introduction

Dans le cadre du cours, ce premier travail vise à faire le pont entre la théorie et la pratique en proposant une application informatique de la théorie des graphes ainsi qu'une implémentation de l'algorithme de Dijkstra. La mise en situation fournie présentait la problématique de savoir manipuler les graphes de connexions afin que notre logiciel puisse déterminer des individus mystères issus d'un fichier .txt que l'utilisateur aurait choisi.

La théorie des graphes est une branche des mathématiques discrètes. Cette branche est entre autres utilisée dans le domaine des réseaux sociaux. En effet, les données étudiées comprennent un ensemble de nœuds et entre ceux-ci des liaisons que l'on nomme aussi arcs. Un nœud est simplement un objet, c'est à dire que pour un réseau social, il s'agira d'individu, mais en télécommunication il pourrait s'agir d'ordinateur connecté à un réseau.

Afin de savoir manipuler les graphes de connexions, on va créer une petite application qui jouera au jeu "Guess who?". Le joueur(adversaire) va interagir avec l'agent(programme) qui effectue principalement deux taches soit : d'identifier les deux personnes mystères du joueur et de déterminer le lien le plus court entre ces deux individus mystères. De manière un peu plus détaillé, l'agent posera une série de question afin de cerner deux individus préalablement sélectionner par l'humain jouant au jeu et utilisera l'algorithme du chemin minimal de Dijkstra.

Pour aider cet étudiant, nous disposons d'un fichier texte qui modélise un graphe représentant les différentes caractéristiques (sommets) et le nombre d'individus qui les relies (arcs). Il nous était donc demandé de concevoir un programme permettant de générer un graphe à partir du contenu du fichier, de faire l'affichage de ce graphe en console et surtout, d'implémenter un algorithme permettant d'obtenir le plus court chemin à suivre par nos individus. L'application permet à l'utilisateur de faire un choix à partir d'un menu en console.

Les sections suivantes présentent les détails de la conception et l'architecture logicielle de notre solution, puis exposent les difficultés rencontrées et leurs solutions.

2. Présentation de nos travaux

Avant de développer une application, il est primordial d'en faire la conception afin d'éviter les erreurs et d'oublier des aspects importants. C'est une étape d'organisation qui nécessite un certain temps afin de minimiser les problèmes de programmation et de logique. Au départ, nous avons défini les principaux domaines à modéliser. Ensuite, nous avons étudié chacun des cas pour en définir le comportement et les relations. Le TP1 sur les graphes pourraient être divisé de façon simpliste en deux domaines : (1) jouer au jeu GuessWho et (2) trouver la meilleure chaîne de contacts entre les deux individus, c'est-à-dire celle qui sera de la meilleure qualité possible entre les deux individus trouvés.

Nous recommandons l'utilisation de IntelliJ pour consulter notre travail. Eclipse semble parfois modifier des fonctions et/ou arguments.

Lors de l'élaboration de l'application, nous avons procédé au codage en utilisant une approche orientée objet. Cela a rendu la tâche plus simple pour mettre en évidence les fonctionnalités à implémenter. La solution contient donc les six classes suivantes : Main, Agent, Reseau, Caracteristic, Matrix, F_C_arrays.

a. Diagramme de Classe

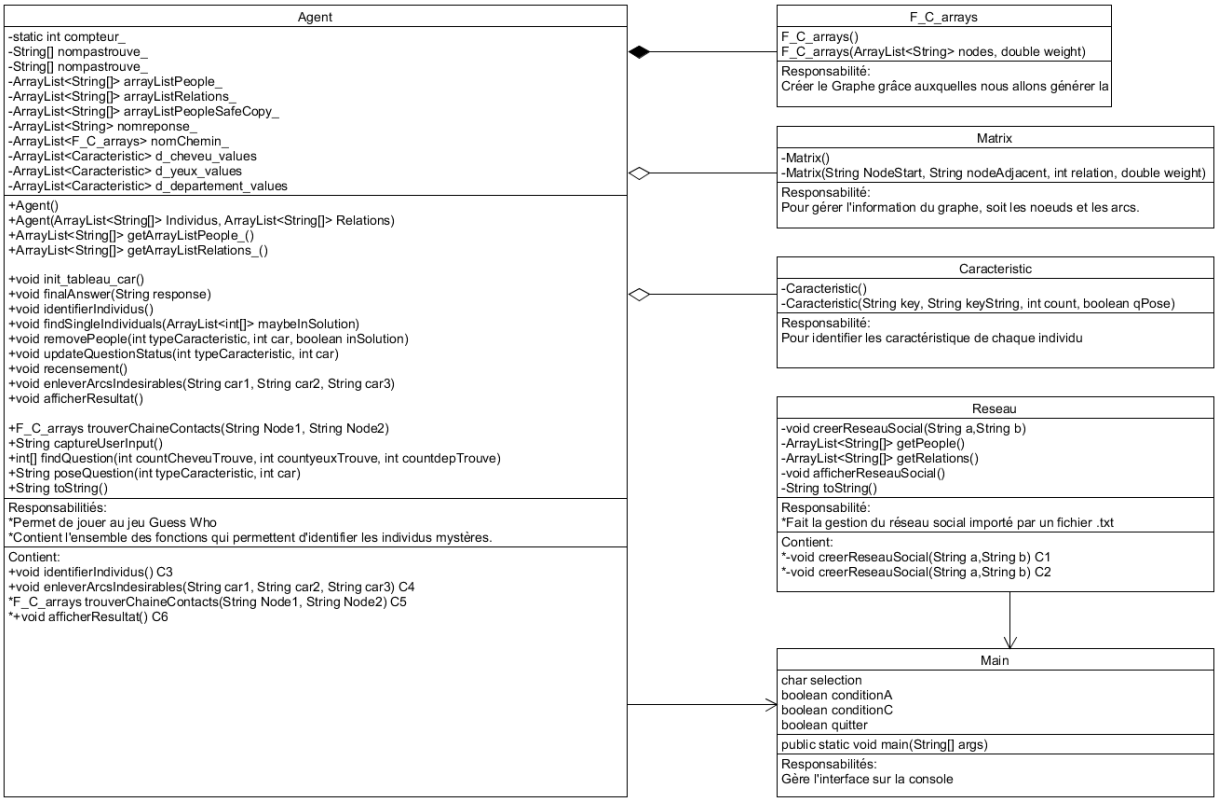


Diagramme de Classe de notre Package Tp1

NB : Java permet l'imbrication de classe dans un même fichier. Matrix et Caracteristic sont alors imbriqués dans le fichier Agent.java

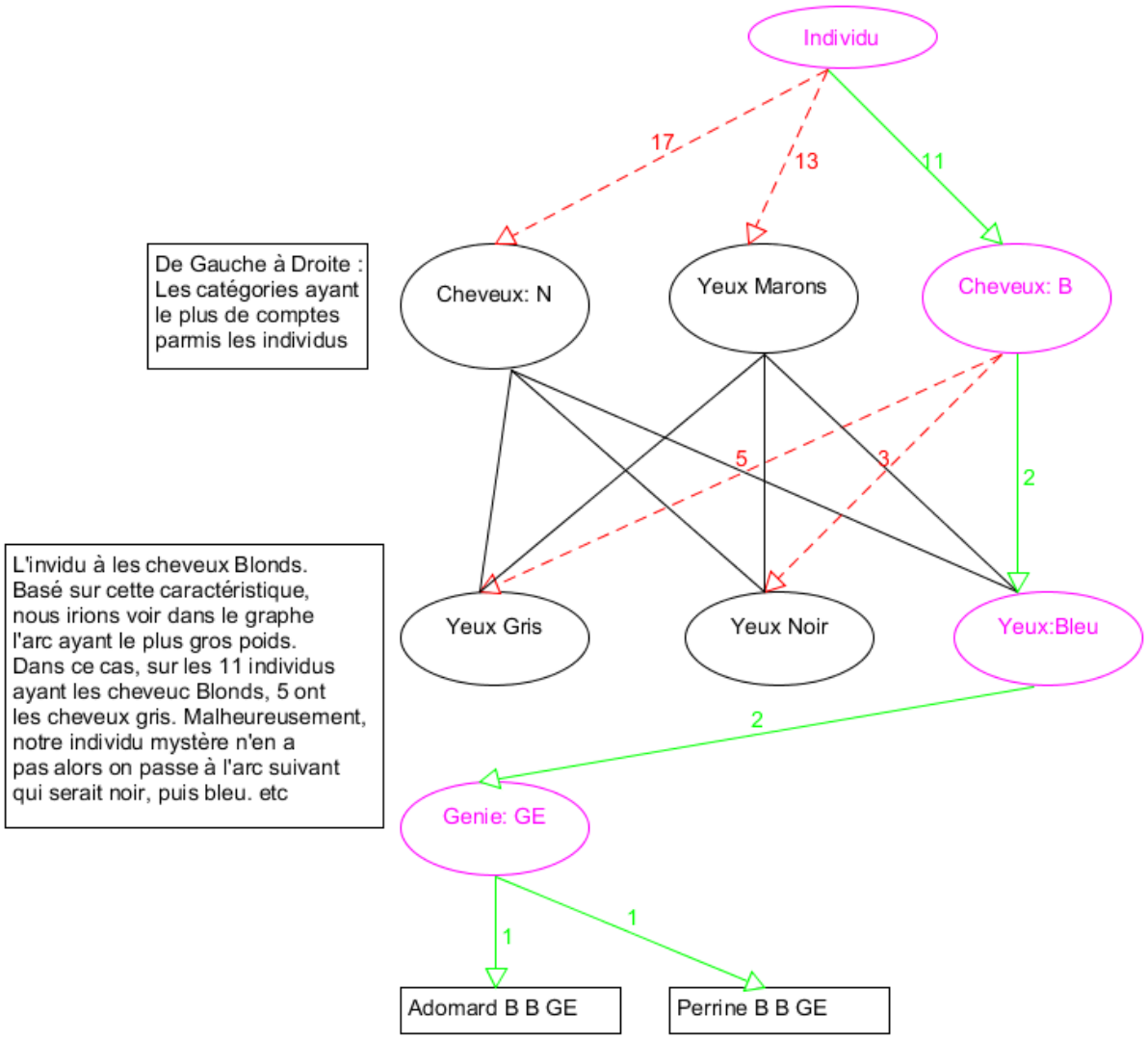
Pour résoudre le 1er problème du travail, nous avions successivement tenté et échoué à construire un graphe puis une succession de hashmap puis hashtable pour résoudre le problème le plus efficacement possible. Nous avons finalement décidé de créer un algorithme de question basé sur le nombre le plus grand de caractéristiques toujours pas encore identifié.

Total	Cheveux	Blond	Maron	Noir	Roux					
50	Nb	11	13	17	9					
100	Statistique	22	26	34	18					
	Yeux	Bleu	Noir	Gris	Marron	Vert				
50	Nb	17	12	13	1	7				
100	Statistique	34	24	26	2	14				
	Génie	GI	GE	GP	GC	GA	GM	GB	GInd	ER
50	Nb	6	8	4	6	6	7	5	5	3
100	Statistique	12	16	8	12	12	14	10	10	6

Tableau du poids des caractéristiques ayant le plus de poids parmi les individus

Cet algorithme va chercher toujours la caractéristique ayant le plus gros poids (compte d'individu ayant cette caractéristique) et pose la question à l'utilisateur. Nous avons élaboré cette stratégie de résolution du jeu Guess Who grâce aux articles « Spatial segregation in cities » et « The best strategy to play guess who » de Edwin Grappin sur le site https://probaperception.blogspot.com.

Toutefois, nous avons essayé de créer un graphe en représentant les premiers nœuds par des caractéristiques avec les arcs ayant le poids du compte de chacun des individus possédant cette caractéristique.



Graphe d'une indentification des individus

Nous n'avons pas réussi à recréer un graphe tel quel alors nous avons opté pour la solution plus facile de chercher dans une liste de tableau pour chercher les comptes de caractéristiques les plus présentes puis chercher dans la liste des noms ceux à qui correspondait la caractéristique choisie.

La deuxième partie du travail a été réalisé en utilisant une implémentation littéral de Dijkstra.

3. Difficultés rencontrées

Lors du processus de conception et de développement, certaines problématiques ont fait surface quant à l'élaboration du programme. Tout d'abord, dès la conception, réaliser l'architecture du logiciel était une tâche qui a été ardu.

Nous avons été très déçus de manquer de temps pour implémenter une solution au jeu Guess Who ? qui aurait utilisé les graphes, hashtable ou hashmap. Le manque de temps et une compréhension limité de Java au début du Tp nous ont limités pour résoudre ce problème grâce au graphe. certains membres de l'équipe maîtrisaient moins bien que d'autres langages (par exemple le C++). Nous avons donc dû faire des recherches sur le Java, ses classes ou librairies et ses façons de faire. Par exemple, faire la lecture d'entrées en console et la lecture du graphe à partir du fichier villes.txt a posé certains défis. Nous avons donc utilisé les classes BufferedReader pour obtenir les informations depuis le fichier texte et Scanner pour les entrées de la console. Or, l'utilisation du Scanner a nécessité une légère refactorisation du code du menu principal, puisque nous avions une fuite de mémoire à un certain point.

Parmi les différentes méthodes à implémenter, la méthode plusCourtChemin() fut définitivement la plus compliquée à implémenter, malgré le fait qu'il soit relativement simple de comprendre l'algorithme de Djikstra. Cela étant dû la récursivité de la méthode qui rend rapidement l'algorithme à trouver plus difficile.

Bien sûr, il y eut aussi les multiples bugs tout au long du TP qui ont augmenté la durée du travail, malgré qu'un grand nombre de ces bugs aient été des erreurs d'inattention, réglées à l'aide de l'outil de débogage intégré à IntelliJ.

Nous sommes conscients qu'il y a encore place à l'optimisation de notre algorithme. Il y a entre autres beaucoup de questions effectuées pour connaître les individus mystères. Une solution potentielle à cela aurait été d'implémenter un autre algorithme, celui de Yen par exemple, pour calculer plusieurs chemins potentiels, de vérifier la faisabilité selon l'essence et de prendre le plus court chemin parmi ceux-ci.

4. Conclusion

En bref, le développement de ce programme d'optimisation a fait en sorte de mettre en pratique les notions d'algorithmique et de théorie des graphes vus en cours, en appliquant un algorithme inspiré de Dijkstra à un problème d'optimisation basé sur le parcours d'un graphe. Fort utile, le travail pratique nous a donc permis de passer de la théorie à la pratique et d'apprendre à faire le traitement logiciel des graphes, en plus d'améliorer notre maîtrise du Java.

Nous avons trouvé la charge de travail raisonnable, malgré que l'algorithme du plus court chemin ait tout de même exigé beaucoup de temps de développement et de tests à cause des nombreuses contraintes à considérer.

Nous sommes toutefois très tristes de ne pas avoir eu les outils nécessaire, donné durant le cours, pour facilement implémenter des graphes de complexité variantes. À l'avenir, nous essayerons de faire davantage d'exercices du cours avec notre ordinateur et de faire plus de programmation en utilisant les savoirs appris durant le cours de mathématiques discrètes.

Ainsi, nous nous attendons à un travail pratique similaire pour le second laboratoire, soit qui nous permet d'appliquer un algorithme à une situation de la vie réelle.

5. Annexe

a. Temps passé sur le TD

1 semaine : Lundi 6h, Mardi, 6h, Mercredi 9h, Jeudi 6h, Vendredi 6h, Samedi 4h, Dimanche 7h.

Total : environ 40h

b. Bibliographie

Edwin Grappin, Spatial Segregation in cities, 15 Novembre 2012, http://probaperception.blogspot.com/2012/09/spatial-segregation-in-cities.html Consulté le 25 février.

Edwin Grappin, The best Strategy to play Guess Who, 25 Septembre 2012, https://probaperception.blogspot.com/2012/10/the-best-strategy-to-play-guess-who.html Consulté le 25 février

John Mullins, Note de cours LOG2810, Session Hiver 2019, Polytechnique Montréal