INFO-F203 - Projet Graphes : Banking

Quentin Ravau - 000361783 et Florentin Hennecker - 000382078

2013-2014

1 Introduction

Le projet consiste à résoudre les problèmes de dettes entre les différents cercles étudiants de l'ULB. Ce problème peut être symbolisé par un graphe où chaque noeud est un cercle avec son budget et chaque arête est une dette.

Détecter les cycles dans ce graphe est une priorité car cela revient à simplifier les dettes. Il faut ensuite exécuter les remboursements nécessaires afin d'éliminer le plus de dettes possibles.

2 Les classes Frat et Debt

On peut faire l'analogie entre Frat et noeud, et entre Debt et arête. Nous préférons ne pas nous arrêter trop longtemps sur ces deux classes qui ne sont qu'une simple implémentation d'un graphe. Ces deux classes ne sont pas du plus grand intérêt algorithmique.

3 La classe Graph

La classe Graph est composée de noeuds Frat que sont les cercles ainsi que de toutes les opérations agissant sur l'ensemble des cercles ou de leurs interactions. Ainsi, c'est Graph qui va s'occuper d'éliminer les cycles de dettes et effectuera les remboursements une fois que ces cycles auront été supprimés.

3.1 Créer le graphe

Le graphe doit prendre un fichier texte en paramètre sur lequel il se base pour sa création. Ce fichier texte doit suivre un format bien spécifique car il doit tout d'abord indiquer le nombre de cercles présents, puis indiquer chaque cercle et son budget et enfin signaler les dettes entre cercles.

Tout d'abord, le constructeur récupère le nombre de cercles afin de créer sa liste de cercles à la bonne taille.

Ensuite, il parcoure tout le fichier ligne par ligne, sépare chaque information de celle-ci (car ces informations sont séparées par des espaces) et stocke la ligne séparée dans une variable splitLine. Maintenant, grâce à la longueur de splitLine, la constructeur sait si il doit créer un nouveau noeud ou si il est face à une nouvelle dette.

Si la ligne était un nouveau cercle, le constructeur crée le cercle et le rajoute à la liste interne de Graph. Mais si il fait face à une nouvelle dette, le constructeur doit rechercher les cercles concernés par la dette dans cette liste interne et si et seulement si les deux existent dans cette liste peut-il créer la dette, autrement il affiche une erreur.

3.2 Détection des cycles

La détection des cycles se base sur un algorithme de *backtracking*, ou de recherche exhaustive. Cette recherche exhaustive est implémentée dans `findCycle()`. Cela nous permet, contrairement à l'algorithme trouvé dans le cours, de détecter tous les cycles, y compris les cycles imbriqués et les cycles adjacents.

Il faut remarquer qu'on lance l'algorithme sur tous les noeuds du graphe qui n'ont pas encore été visités. Ceci couvre le cas des graphes en plusieurs parties.

3.3 Suppression des cycles

La réduction des cycles est relativement simple : il suffit de trouver la dette la plus basse et de soustraire ce montant à toutes les dettes du cycle. Cependant, il faut vérifier si la plus petite dette qu'on vient de supprimer se trouve dans d'autres cycles que l'on n'a pas encore réduits. En effet, si ce lien est présent dans d'autres cycles et qu'on le casse, ces autres cycles n'en sont plus. Il faut donc les supprimer.

3.4 Le remboursement

Avant d'effectuer le remboursement, il est nécessaire que les cycles soient supprimés. A ce moment là, un ou plusieurs points de départ peuvent être définis. Ceux-ci correspondent à des cercles qui ne recevront pas d'argent ou à des cercles qui ne peuvent pas être remboursés car les cercles qui leur doivent de l'argent n'ont plus de budget.

Pour cela, la liste des cercles est copiée dans une variable start. Ensuite, on parcoure l'ensemble des cercles et de ses dettes. Si le cercle possède encore de l'argent, on enlève les créditeurs de ses dettes de start.

Ensuite, chaque cercle restant dans start rembourse ses dettes. Mais soit ce cercle a suffisamment d'argent pour rembourser sa dette soit il n'en a pas assez. Dans le premier cas, on rembourse la dette et on la supprime et dans le deuxième on rembourse ce que le budget permet et on met le budget à 0.

Enfin, il faut répéter toutes ces opérations tant que des nouveaux points de départ sont trouvés vu que les anciens points de départ auront soit plus de dette soit plus de budget, libérant ainsi les cercles qu'ils ont remboursés.

3.5 Créer une image

Pour illustrer notre graphe, il faut créer un fichier .dot dans lequel on insère au préalable l'en-tête "digraph G {\n".

Ensuite on parcours tous les noeuds du graphe et on crée leur noeud visuel en insérant leur nom et leur budget (les deux séparés par un \n pour faire plus joli).

Mais avant de passer au noeud suivant, toutes ses dettes doivent être illustrées aussi. Ainsi, on parcours aussi toutes les dettes du noeud en question et chaque dette est illustrée en insérant la ligne suivante : « info noeud -> info crediteur [label= « valeur dette »] ».

Et enfin, on termine le fichier .dot en fermant l'accolade ouverte dans l'entête sans oublier de refermer le fichier.

4 Conclusion

Pour conclure, ce projet fut très porté sur les graphes, concept important de l'informatique et son côté ludique rendit son travail plus aisé malgré une certaine difficulté dans l'algorithme de détection des cycles.

De plus, ce projet fut aussi notre première expérience en tant que travail de groupe, et ceci amena ses propres problèmes tout en se révélant positif au final.