Rapport du projet d'AAW : Réseau social en Java J2EE.

William Le Coroller & Romain Boutin

Mercredi 25 Février 2015

Table des matières

1 Introduction

Le but de ce projet est la réalisation d'un programme calculant les composantes fortement connexes d'un graphe. Il faut donc implémenter un algorithme de calcul de composantes fortement connexes d'un graphe orientée.

Le rapport est organisé en quatres parties principales :

- La partie **contexte** qui présente les notions et définition de la théorie des graphes ainsi que ce qui est attendu ainsi que les contraintes.
- La partie **théorie** qui explicite les deux algorithmes de calcul des composantes fortement connexes et qui les compare très succintement.
- La partie **technique & documentation** qui présente les choix techniques qui ont été effectué et fait le lien avec la documentation.
- La partie **résultats** qui présente les résultats des brefs tests de performances effectué sur les deux algorithmes implémentés.

2 Choix techniques & documentations

La documentation du projet est extrait des commentaires du code source puis formatté par doxygen pour le rendre plus lisible. Il est disponible dans le document PDF LaTeXci-joint (document « refman.pdf »). Il est également disponible sous un format web HTML « /doc/doxygen/html/index.html ».

Durant l'implémentation de ce projet, plusieurs choix techniques ont été fait. Nous tentons ici des les présenter et de les justifier lorsque cela est nécéssaire.

Le premier choix qui a été fait fut d'**utiliser le langage C/C++**. En effet, le Java aurait certainement été plus simple et plus rapide d'utilisation. Néanmoins en utilisant le C/C++ on avait déjà une bibliothèque de gestion des graphes toutes faites nous permettant un gain de temps substancielle.

Le second choix qui a été fait est de donner la possibilité à l'utilisateur de **sauvegarder le graphe généré aléatoirement dans un fichier de description** afin de pouvoir le re-entrer pour re-tester le graphe généré à plusieurs reprises. En effet, n'ayant pas implémenter la fonctionnalité d'input demandé, on a voulu donner à l'utilisateur la possibilité de re-tester le graphe généré.

Le troisième choix qui a été fait est d'**utiliser des « vector »** de la bibliothèque standard C/C++ afin de représenter les arêtes du graphe sous la forme d'une liste d'adjacence plutôt qu'une classe liste modifiée. Dans un premier temps, nous avions réalisé une classe implémentant la structure de donnée d'une liste chainée en utilisant des templates. Le soucis majeur de cette implémentation est qu'une telle structure ne pouvaient être détruite facilement dès lors qu'on lui passait une classe en type patron. N'ayant pas trouvé de solution **simple**, nous avons décider de nous en remettre à quelque chose de plus sûr. Les fichiers implémentant cette classe sont présent dans le dossier « /src/ » bien que non répertorié par doxygen.

Enfin, le quatrième et dernier choix est vis à vis de l'implémentation des algorithmes. **On a choisi d'implémenter les deux algorithmes** majeurs afin de pouvoir les comparer de façon grossière par curiosité. Les tests et résultats sont présentés dans la partie suivante.

On a, par ailleurs, vérifié en utilisant des outils externes tel que **gdb** ou **valgrind** qu'il n'y ai pas de fuite mémoire.

3 Conclusion

Du point de vue des résultats, il apparaît que l'algorithme de Tarjan semble plus rapide que l'algorithme de Kosaraju-Sharir.

On regrette de ne pas avoir implémenter une « vraie » liste ou une « vraie » pile ceci bien que la structure vector fonctionne très bien (probablement beaucoup mieux que n'importe quel liste ou pile de mon cru).

On regrette également de ne pas avoir davantage étoffé les tests effectué. On aurait pu leur donner un semblant d'approche scientifique.

On est néanmoins satisfait d'avoir implémenté les deux algorithmes principaux de calcul de composantes fortement connexes d'un graphe orienté et d'avoir mis en place un système simple afin de comparer grossièrement leurs performances.