Projet : Théorie des graphes

Chloé Bensoussan

Universite Nice Sophia-Antipolis
Master 1 IFI

1 Introduction

Ce projet consiste à implémenter trois algorithmes vus en cours permettant de trouver toutes les composantes fortement connexes d'un graphe orienté. Ce projet est composé globalement d'une classe `Graphe` et des algorithmes de Tarjan, Kosaraju et Gabow.

Dans ce rapport, je vais d'abord vous parler de mes choix d'implémentations choisis. Je continuerai par mon avis personnel et mes préférences entre les trois algorithmes pour terminer par un mode d'emploi et des performances d'exécution.

2 Implémentation

Pour la réalisation de ce projet, j'ai opter pour le langage `C++`. En effet, la programmation orientée objet m'a paru la plus évidente pour gérer facilement les structures de graphe.

D'un point de vue structurel du programme, j'ai crée une classe abstraite `Graphe` ayant des méthodes générales d'un graphe, comme par exemple les méthodes `DFS()` et `addEdge()`. J'ai ensuite créé deux classes héritant de celle-ci représentant deux structures différentes (graphe par liste et par matrice d'adjacence). Ainsi, les méthodes du graphe peuvent s'exécuter sans préciser la structure choisie. Ce choix est effectué dans le fichier `main.cpp` lors de la création du graphe. Il suffit simplement de choisir le constructeur correspondant à la structure voulue : `List()` ou `Matrix()`.

Ce programme possède de plus, trois classes représentant les algorithmes. Elles contiennent en effet les attributs nécessaires et la méthode de parcours du graphe.

Quant à la génération automatique de graphe, elle ajoute les arêtes entre deux sommets choisis aléatoirement. Il est donc nécessaire de vérifier l'existence de cet arête avant de l'ajouter dans le graphe, surtout pour la représentation par liste (il ne peut pas y avoir plusieurs fois la même arête dans la liste, alors que ce problème ne se pose pas dans la représentation matriciel).

3 Avis personnel

D'un point de vue conceptuel, j'ai une préférence pour l'algorithme de Kosaraju. Facilement applicable à la main, il n'utilise qu'une seule pile et ne demande pas de comparaison

entre des numéros attribué à chaque sommet. Néanmoins, cet algorithme n'a pas été facile lors de son implémentation. Par sa petite particularité dans la `DFS` (l'obtention de l'ordre de fermeture des sommets et non leur ouverture comme les deux autres algorithmes), cet algorithme a rendu son implémentation plus compliqué que ce que l'on pourrait imaginer uniquement par la recherche manuelle.

D'un autre côté, les algorithmes de Tarjan et Gabow n'ayant pas de démarches conceptuellement transparentes, furent facilement implémentable grâce aux pseudo-codes vus en cours. Les méthodes de parcours récursives réalisent efficacement une recherche en profondeur du graphe.

4 Mode d'emplois

Mon projet comporte trois répertoires et trois fichiers. Les codes se trouvent dans les répertoires `src/` et `header/`. Le répertoire `docs/` comportent les fichiers de données calculés par l'exécution du programme ainsi que les courbes de performances correspondantes.

Pour exécuter le programme, il suffit de lancer le script `script_run.sh`. Il effectue la compilation en utilisant le `makefile` et l'exécution du projet automatiquement. Puis, génère les courbes de performances avant de se terminer. Ces courbes seront disponibles dans le répertoire `docs/`.
Attention, la compilation doit être effectué sur un machine Linux.

La configuration des courbes est précisée dans le script `docs/script_gnuplot.sh`. Puisque ce script est appelé depuis le script précédent, toutes modifications doit s'effectuer avant la compilation du programme.

5 Performances

Voici les résultats d'exécution du programme. Ces tests ont été réalisés sur un graphe de 1000 sommets, faisant varier le nombre d'arêtes de 0 à 1 000 000 en utilisant la génération aléatoire. Les algorithmes sont exécutés séquentiellement à chaque tour de boucle.

La figure 1 représente le temps d'exécution des algorithmes en fonction du nombre d'arêtes. La structure utilisée ici est la représentation par liste. Nous remarquons que le temps varie proportionnellement au nombre d'arête pour les algorithmes de Tarjan et Gabow. C'est approximativement le cas pour l'algorithme de Kosaraju. Cependant nous remarquons que ce dernier possède un temps d'exécution très élevé comparant aux deux autres. Cela est dû par une double utilisation de la DFS.

Passons maintenant à la figure 2. J'ai séparé les courbes de Tarjan et Gabow avec celui de Kosaraju étant donné leur différence de grandeur. Ces graphes représentent la même exécution que la précédente en utilisant la structure matricielle. En effet, les algorithmes de Gabow et Tarjan possèdent toujours des temps proches, avec une augmentation presque linéaire. Néanmoins, la courbe de Kosaraju est plutôt constante.

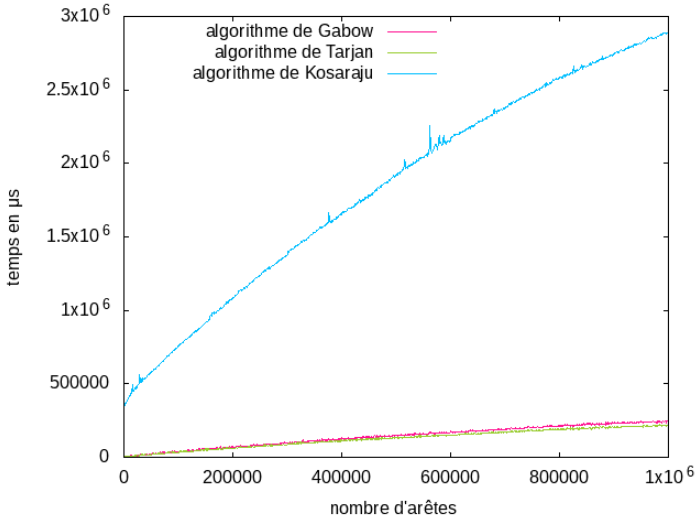

FIGURE 1 – Temps d'exécution d'un graphe de type liste.

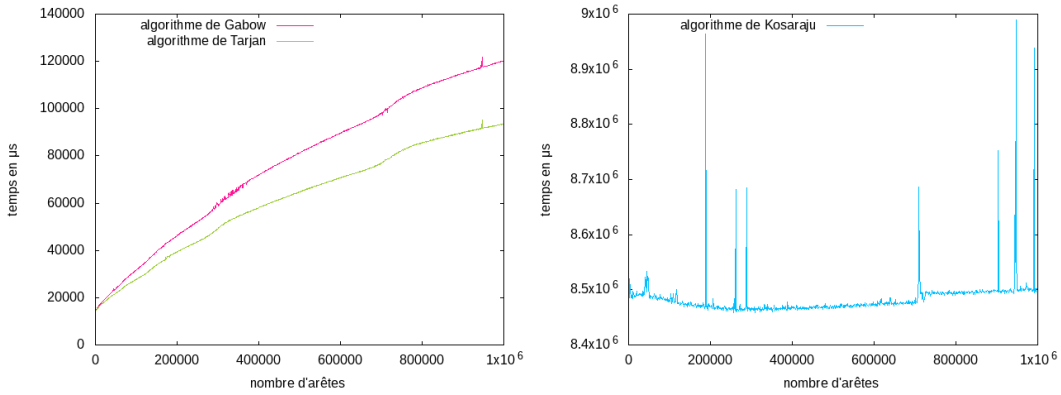

FIGURE 2 – Temps d'exécution d'un graphe de type matrice.

6 Conclusion

Pour conclure, mes préférences des algorithmes sont partagés selon le point du vue conceptuel et lors de la phase d'implémentation. L'algorithme apprécié n'est pas toujours la plus facilement implémentable. De plus, l'architecture du programme fait varier ses performances lors de son exécution. Il est donc important d'effectuer un bon choix d'algortihme ainsi que la structure des object à implémenter pour obtenir une exécution optimale. Concernant ce projet, la meilleure architecture est l'utilisation de graphe de type liste d'adjacence avec l'algorithme de Tarjan.