MASTER INFORMATIQUE FONDEMENTS ET INGENIERIE

Rapport du projet de résolution

Sujet :

Implémentation de Tarjan , Kosaraju et Gabow

Auteur : Julien Audibert(parcours IFI)

Résumé :
Ce rapport traite de la difficulté pour un développeur de choisir un algorithme pour résoudre un problème précis à la fois performant et facile à implémenter. En effet, la première impression d'un développeur sur un algorithme peut parfois se révéler trompeuse. Pour cela , nous étudierons dans un premier temps trois algorithmes permettant de trouver les composantes fortement connexes d'un graphe , puis par la suite nous comparerons nos premières impressions sur ces trois pseudo-codes et notre avis après les avoir développés.

Table des matières

Introduction

En théorie des graphes, les algorithmes de Tarjan , Kosaraju et Gabow permettent de déterminer les composantes fortement connexes d'un graphe orienté. La complexité de ces trois algorithmes est linéaire : $\mathcal{O}(V + E)$. Ces trois algorithmes prennent en entrée un graphe orienté et renvoient une partition des sommets du graphe correspondant à ses composantes fortement connexes. Mais en quoi sont-ils différents ?

1 Algorithmes

1.1 Tarjan

On lance un parcours depuis un sommet aléatoire que l'on ajoute dans une pile P et on lui attribue un numéro. A chaque sommet rencontré , si il n'est pas déjà numéroté , on lance un parcours depuis ce sommet (C'est un parcours du graphe en profondeur , une depth-first search (DFS)). Si il est numéroté ou si le parcours du graphe en profondeur est terminé , on y attribue un "numAccessible" représentant le sommet avec le plus petit numéro auquel il peut accéder (Son numéro accessible deviens le minimum entre son numéro accessible et celui de son successeur). Ensuite, on compare le numéro attribué au début du parcours avec le numéro accessible du parcours. Si les deux sont équivalents, le sommet V est une racine. On dépile la pile jusqu'à trouver le sommet V. Tous les sommets dépilés font partis de la même composante fortement connexe (CFC) que V, on ajoute la CFC à la partition. Enfin, on recommence avec un sommet du graphe qui n'a pas été visité.

1.2 Kosaraju

On lance un parcours en profondeur depuis un sommet aléatoire que l'on ajoute dans une pile P. Tous les sommets rencontrés sont mis dans la pile à la fin de leur visite. On relance un parcours en profondeur d'un autre sommet jusqu'à ce que tous les sommets du graphe soit visités. Ensuite, on inverse le sens des arrêtes du graphe (Transposée du graphe). Enfin , on dépile un sommet V de la pile. On effectue un parcours en profondeur depuis V , tous les sommets rencontrés font partis de la même composante fortement connexe que V, on ajoute la CFC à la partition. Et on supprime tous les sommets de la pile P et du graphe. On continue jusqu'à ce que P soit vide.

1.3 Gabow

On lance un parcours depuis un sommet aléatoire que l'on ajoute dans une pile P ainsi que dans une pile S et on lui attribue un numéro. A chaque sommet rencontré W , si il n'est pas déjà numéroté , on lance un parcours depuis ce sommet. Si il est numéroté et qu'il ne fait pas encore parti d'une CFC , on dépile P tant que le numéro du sommet de P est supérieur au numéro de W. Si le sommet de P est le sommet en cours de parcours appelé V , on dépile S jusqu'à trouver le sommet V. Tous les sommets dépilés de S font partis de la même composante fortement connexe que V, on ajoute la CFC à la partition. Puis on retire le sommet V de P. Enfin, on recommence avec un sommet du graphe qui n'a pas été visité.

1.4 Première impression

A première vue , Kosaraju semble l'algorithme le plus simple voir "magique". Deux DFS , une transposée du graphe et le tour est joué. Il est mon premier choix. Gabow est mon deuxième choix , utilisant simplement deux piles , il ne parait pas compliqué à implémenter. Enfin, Tarjan paraît le plus complexe, de nombreuses étapes sont nécessaires et il n'est pas aussi étonnant que les deux autres .

2 Implémentation

L'implémentation de ces trois algorithmes est faite en Java.

2.1 Générateur de graphes

Les graphes sont représentés par des chaînes de caractères composées comme ceci : "Sommet1 -> Sucesseur1 , Sucesseur2 ,.../ Sommet2 -> ... / ..." . Cette chaîne est parsée en entrée afin de créer le graphe. Les sommets sont représentés par des objets Sommet ayant pour attribut un nom , un numero , un numero accessible (pour Tarjan), un liste de sommets voisins (sucesseurs) , Une liste de sommets permettant de stocker la transposée (pour Gabow) , ainsi que deux booléens indiquant si le sommet est dans une pile ou dans une CFC. Le Graphe est une liste de sommets. La partition est une liste de CFC. Une CFC est une liste de sommets.
Les graphes sont donc représentés sous forme de listes d'adjacences.
Un générateur implémenté en python permet de créer des ensembles de graphe de façon aléatoire.

2.2 Tarjan

L'implémentation de Tarjan est minutieuse mais peu complexe. En effet , une première boucle permet de lancer le parcours sur l'ensemble des sommets du graphe n'ayant pas encore été visité. Le parcours est composé de trois parties principales. Une initialisation où le numéro est attribué au sommet , il est mis dans la pile , et son booléen pour la pile est passé à vrai. Ensuite, une DFS depuis ce sommet et une mise à jour de son numéro accessible. Et enfin, une recherche de la racine , le dépilement de la pile pour créer la CFC et l'ajouter à la partition.

2.2.1 Difficultés

Peu de difficultés se font ressentir , l'implémentation colle parfaitement au pseudo-code ce qui permet de finir assez rapidement et d'obtenir un code qui fonctionne de façon linéaire directement.

2.3 Kosaraju

L'implémentation de Kosaraju est complexe. En effet , une première boucle permet de lancer une DFS sur l'ensemble des sommets du graphe n'ayant pas encore été visité et les ajouter dans la pile. Ensuite , une transposée du graphe , ainsi une double boucle permet de transformer les prédécesseurs en successeurs et les successeurs en prédécesseurs. Enfin , une deuxième DFS qui cette fois ci ajoute les sommets rencontrés dans une CFC.

2.3.1 Difficultés

Lors de son implémentation quelques difficultés apparaissent, en effet , elle ne peut pas coller exactement au pseudo-code puisque celui-ci nécessite de supprimer les sommets rencontrés lors de la deuxième DFS du graphe ainsi que de la pile. Mais la suppression d'un élément précis d'une pile est "contre nature". Ainsi , ma première implémentation me demande un peu de réflexion. Assez rapidement , celle ci fonctionne mais sa complexité n'est pas linéaire. Après un petit profilage des différentes parties de mon code , le problème

apparaît. La suppression des sommets du graphe n'est pas linéaire , utilisant la fonction .remove() des listes java , sur les listes de successeurs de chaque sommet du graphe. La solution est trouvée dans la combinaison des deux booléens (isDansCFC et isDansP).

2.4 Gabow

L'implémentation de Gabow est assez simple. En effet , celle-ci est identique à Tarjan dans sa première partie , la seule différence est dans l'ajout d'un pile qui est remplie en même temps que la première. Lors de la DFS cette nouvelle pile est dépilée si le numéro du successeur est inférieur au sommet de la pile. La dernière partie est simplement composée d'une condition et d'une boucle qui consiste à dire que si le sommet de la deuxième pile est équivalent au sommet on dépile la première pile et on ajoute tout à une CFC tant que les sommets des deux piles ne sont pas équivalents. Puis on ajoute le sommet des piles à la CFC et on ajoute la CFC a la partition.

2.4.1 Difficultés

L'implémentation se fait sans difficultés mais quelques questions sur le "pourquoi cela fonctionne ?" apparaissent et me demandent quelques réflexions pour y répondre.

2.5 Avis après implémentation

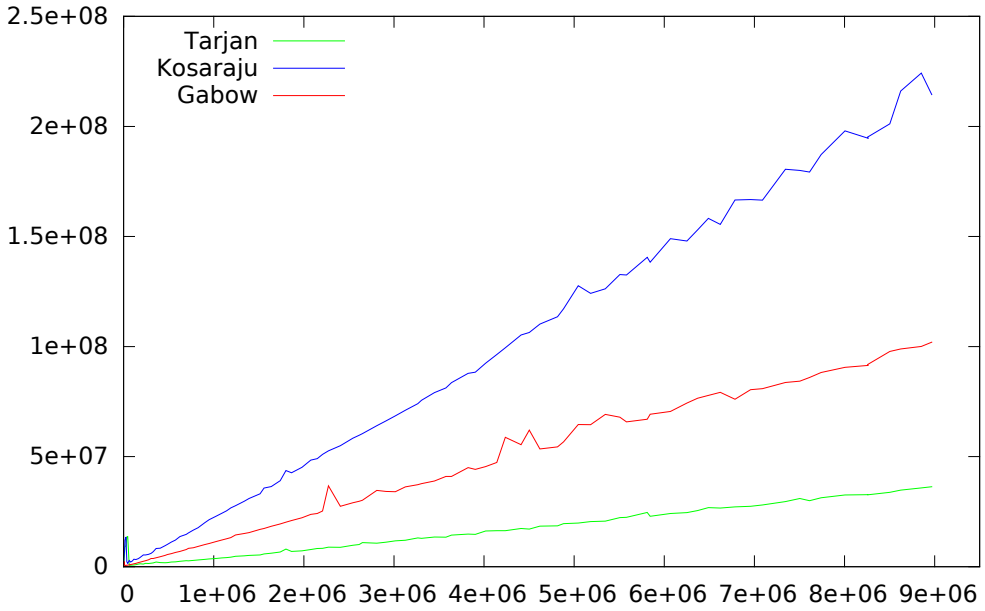
Contrairement à la première impression, Kosaraju est l'algorithme qui m'a demandé le plus de temps à implémenter et le plus de travail. Quand à Tarjan, il a été à l'inverse très rapide à développer , et n'a posé aucune difficulté. Enfin , Gabow colle à ma première impression , il est facile à implémenter et l'utilisation des deux piles est vraiment élégant.

3 Comparaison

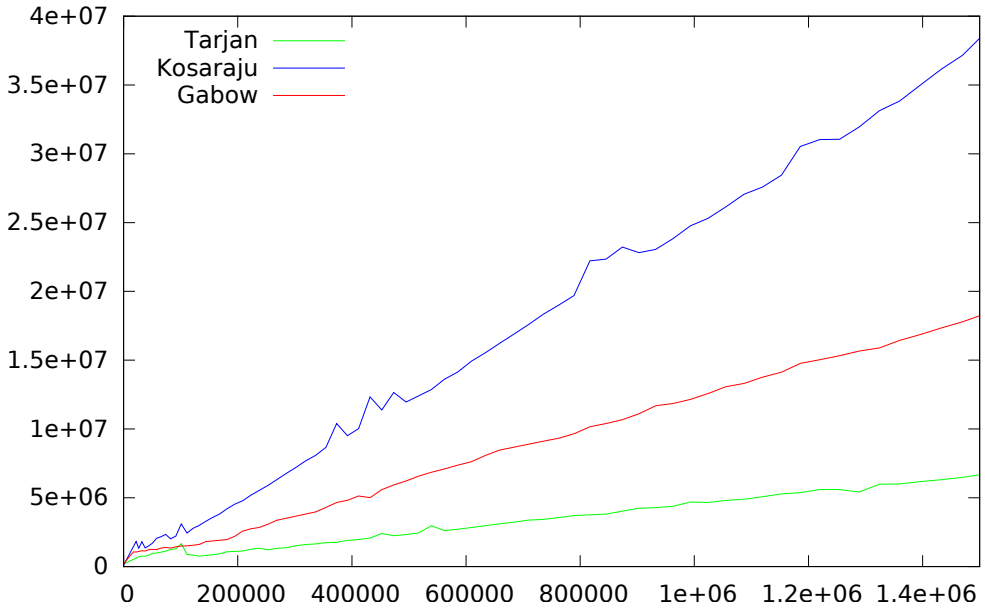
Pour comparer le temps d'exécution des trois algorithmes , je les ai lancé sur trois ensembles de graphes créés par le générateur. Le premier ensemble de graphes est composé de 100 graphes allant de 4 à 5000 sommets donc le nombre d'arêtes est compris entre 1 et le nombre de sommets - 2 pour chacun des sommets du graphe ce sont donc des graphes totalement aléatoires. Le deuxième ensemble de graphes est composé de 100 graphes allant de 4 à 5000 sommets donc le nombre d'arêtes est de 10% pour chacun des sommets du graphe ce sont donc des graphes dit creux. Enfin , Le troisième ensemble de graphes est composé de 100 graphes allant de 4 à 5000 sommets donc le nombre d'arêtes est compris entre 90% et 100% pour chacun des sommets du graphe ce sont donc des graphes dit denses.

Les trois graphiques si-dessous représentent le temps d'exécution en nano-secondes sur le nombre de sommets plus le nombre d'arêtes du graphe.

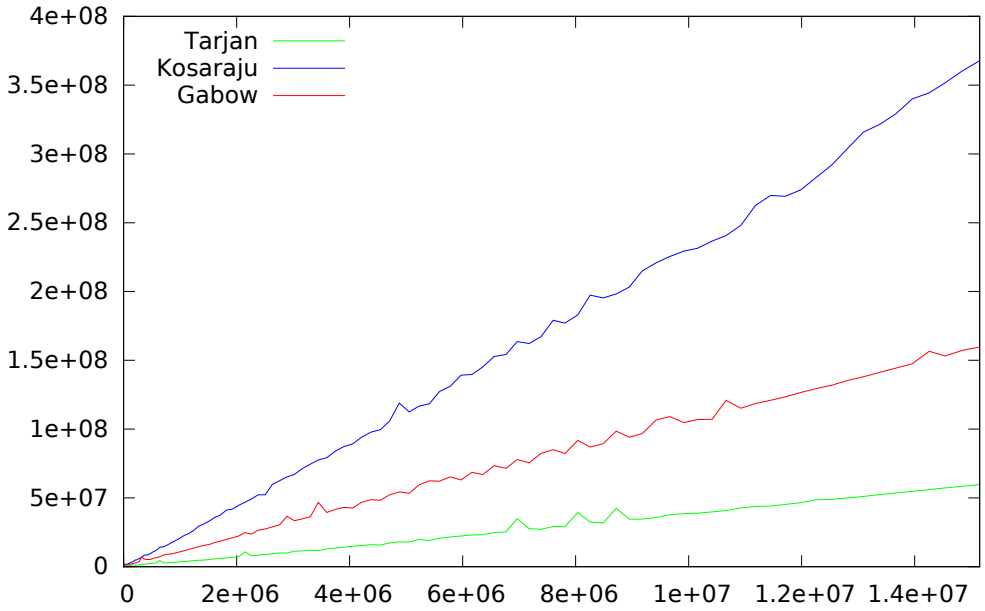
3.1 Graphes aléatoire



3.2 Graphes creux



3.3 Graphes denses



3.4 Analyse des Graphes

Sur l'ensemble de ces trois graphes , on peut voir que Kosaraju est bien plus lent que Gabow et Tarjan , ce qui est assez logique étant donné que celui-ci fait deux DFS dans le graphe tandis que Gabow et Tarjan n'en font qu'une. Cependant , les écarts en termes de vitesse d'exécution entre Gabow et Tarjan sont beaucoup moins triviale à expliquer , celles-ci peuvent trouver réponse dans les mécanismes internes au langage utilisé ainsi que dans des faiblesses dans l'implémentation. On peut également s'apercevoir que le temps est environ 10 fois supérieur sur les graphes denses par rapport aux graphes creux , ce qui bien au fait que les graphes creux ai environ 10 fois moins d'arêtes. Nous pouvons aussi remarquer que le temps d'exécution des algorithmes pourrait être beaucoup plus performant dans les graphes denses en remplaçant les DFS par des BFS . En effet , dans une clique la BFS atteindrait au premier tour tous les sommets du graphe.

Conclusion

Ce projet aura permis de nous apercevoir que la première impression d'un algorithme n'est pas toujours la bonne. Un algorithme simple à expliquer et à faire tourner sur papier n'est pas toujours un algorithme simple à implémenter. Ainsi, dans notre cas l'ordre de préférence de nos trois exemples aura été totalement renversé après les avoir mis au point. Ce projet soulève donc une question assez complexe à répondre. Comment choisir un algorithme si il est nécessaire de l'implémenter pour en connaître la réelle difficulté ?

Références

[REF] https ://en.wikipedia.org/wiki/Kosaraju's_algorithm/ **Kosaraju**.

[REF] https ://fr.wikipedia.org/wiki/Algorithme_de_Tarjan/ **Tarjan**.

[REF] An Extended Experimental Evaluation of SCC (Gabow's vs Kosaraju's) based on Adjacency List By Saleh Alshomrani Gulraiz Iqbal **Gabow VS Kosaraju analyse**.