République Algérienne Démocratique et Populaire
الجمهوريـــة الجـزائريـة الديمـقراطيـة الشـعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التـعليـم العـالي و البـحث العلـمي

ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
Ecole nationale Supérieure d'Informatique

Une Étude Comparative entre Les Algorithmes de Kosaraju et de Tarjan

Réalisé par :

MAHMAHI Anis

Promotion : 2019-2024

Introduction:

Composantes fortement connexes

En théorie des graphes, une composante fortement connexe d'un graphe orienté G est un sous-graphe maximal de G tel que pour toute paire de sommets U et V dans ce sous-graphe, il existe un chemin de U à V et un chemin de V à U.

Algorithmes de détermination des CFC d'un graphe orienté donné :

Plusieurs algorithmes permettent de calculer la décomposition en composantes fortement connexes d'un graphe en temps linéaire :

1. L'algorithme de Kosaraju, basé sur l'algorithme de parcours en profondeur, nécessite deux parcours du graphe.
2. L'algorithme de Tarjan ne nécessite qu'un seul parcours en profondeur.

Guide d'utilisation du code source :

Ce programme, je l'ai divisé en 3 fichier .c :

1. graphe.c : Contient la matrice d'adjacence du graphe.
2. tarjan.c : Contient le code source de l'algorithme du Tarjan.
3. kosaraju.c : Contient le code source de l'algorithme du Kosaraju

Afin d'exécuter ce programme il faut donc exécuter chaque fichier seul.

Etude Comparative :

L'algorithme de Tarjan :

Principe de fonctionnement:

Chaque nœud **V** se voit attribuer un entier unique **V.index**, qui numérote les nœuds consécutivement dans l'ordre dans lequel ils sont traversés. Il maintient également une valeur **V.low** qui représente le plus petit index de n'importe quel nœud dans la pile **W**, connu pour être accessible de **V** via le parcours en profondeur de **V** (y compris **V** lui-même).

```
V.low = min(V.low,W.index).
```

Par conséquent, **V** doit être laissé sur la pile si **V.low** < **V.index**, tandis que **V** doit être supprimé en tant que racine d'un composant fortement connexe si **V.low** == **V.index**.

La valeur **V.low** est mise à jour lors de la recherche en profondeur à partir de **V**, car on peut trouver les nœuds accessibles à partir de **V**.

Complexité temporelle:

L'algorithme est construit sur la base de DFS et par conséquent, chaque nœud est visité une fois et une seule. Ainsi, sa complexité est de O(V+E) où V est le nombre de sommets et E est le nombre d'arêtes.

Complexité Spatiale :

Dans le pire des cas, nous aurions le graphe entier sous la forme d'un grand CFC, et la pile devrait contenir tous les sommets. Alors la complexité spatiale sera O(V) où V = nombre total de sommets dans le graphe orienté donné.

L'algorithme de Kosaraju :

Principe de fonctionnement:

- Effectuer un DFS sur le graphe original : tout en gardant les nœuds traversés en utilisant une pile, lorsque le DFS se termine le premier nœud de chaque parcours sera au sommet de la pile.
- Inverser le graphe original.
- Refaire le DFS : effectuer DFS sur le graphe inversé, avec le nœud source est le sommet de la pile. Une fois que le DFS est terminé, tous les nœuds visités forment un composant fortement connexe. Si d'autres nœuds restent non visités, cela signifie qu'il y a plus qu'un composant fortement connexe, donc dépiler les sommets de la pile jusqu'à ce qu'un nœud non visité valide soit trouvé. Cette étape est répétée jusqu'à ce que tous les nœuds soient visités.

Complexité temporelle :

La complexité temporelle de cet algorithme dépend de la structure utilisée pour stocker le graphe donné, si on utilise un liste d'adjacence alors la complexité est de O(V+E), où V est le nombre de sommets et E est le nombre d'arêtes. Par contre si le graphe est représenté comme une matrice d'adjacence de taille V*V, l'algorithme nécessite alors un temps O(V^2).

Complexité Spatiale :

En plus de l'utilisation d'une pile, kosaraju crée de plus un graphe transposé du graphe de départ donc l'algorithme de Tarjan utilise moins de ressources (mémoire).

Expérimentation :

Après des expérimentations sur plusieurs graphes avec plusieurs sommets, on constate que les deux algorithmes ont presque la même performance lorsque le nombre des sommets et petits (< 1K), mais la différence apparaissent lorsque le nombre de sommets est tres grands (> 10K), et le temps d'exécution peut achever mois -40% en utilisant l'algorithme de Tarjan et ça revient au fait que Tarjan fait un seul parcours cependant Kosaraju fait 2 parcours du graphe.

Conclusion:

Les deux algorithmes ont la même complexité temporelle (linéaire), mais les techniques sont assez différentes.

Kosaraju est un algorithme efficace et le plus simple sur le plan conceptuel, mais il n'est pas aussi efficace en pratique que l'algorithme de Tarjan qui n'effectue qu'un seul parcours du graphe.

Références :

https://www.codingninjas.com/codestudio/library/time-space-complexities-of-graph-algo-3

https://iq.opengenus.org/kosarajus-algorithm-for-strongly-connected-components/

https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm

https://www.geeksforgeeks.org/comparision-between-tarjans-and-kosarajus-algorithm/