Comparaison d'algorithmes pour la détection de CFC

Arthur Finkelstein

1 Introduction

Les algorithmes sont une partie intégrante de la vie d'un développeur, ils en existent une quantité énormes ainsi pour résoudre certains problème un développeur se verra dans l'obligation de choisir lequel implémenté. Mais il faudra que le développeur choisissent celui qui sera le plus performant tout en restant facilement implementable.

1.1 Question

Ainsi notre problème est celui de trouver toutes les composantes connexes d'un graphe pour faire cela nous avons trois algorithme qui existent: celui de Tarjan, de Kosaraju et de Gabow. Tout trois partagent la même complexité $\mathcal{O}(V + E)$ donc seule la question de l'implémentation reste. La définition de Gabow m'a attiré par sa simplicité et son élégance, tandis que Kosaraju a un aspect magique et Tarjan parait plutôt quelconque.

2 Algorithme

2.1 Structure de données et langage

Pour le langage j'ai choisi Python 3 car notre but est de comparer la vitesse d'exécution des algorithmes entre eux et non les performances pur.

Les graphes vont être représenté en utilisant une liste de succession ainsi quand nous voulons trouver les voisins d'un nœuds nous avons une complexité en $\mathcal{O}(1)$ comparer a $\mathcal{O}(n)$ pour une matrice d'adjacence et une matrice d'incidence.

Toutes les informations stockés pour chaque nœuds sont placés dans des tableaux (qui seront également utilisé comme des piles grâce aux primitives append et pop de Python).

De plus pour les benchmarks Python 3 propose une primitive perf_counter() qui remplace le clock() qui est déprécié.

2.2 Tarjan

Implémentation

Pour Tarjan nous avons bien entendu une structure extrêmement semblable a une depth-first search, avec donc une structure récursive. L'algorithme ne comporte pas de difficultés, il suffit d'être méticuleux et bien comprendre quand mettre le nœud que l'on visite ou son voisin.

Préférence

Tarjan était deuxième dans ma liste de préférence mais quand j'ai du l'implémenter après Kosaraju j'ai mis un certains a comprendre que sa structure était une celle de la dfs. De plus sa définition est bien différente du code que nous devons écrire.

2.3 Kosaraju

Implémentation

Pour Kosaraju nous avons un algorithme qui lui même n'as pas de ressemblance à une dfs mais va en utilisé, une sur le graphe tel quel et un certains nombre sur le graphe inversé. Les difficultés viennent du fait qu'il faut faire attention durant la phase sur le graphe inversé de bien gardé un tableau des nœuds visités afin de ne pas repasser sur des nœuds déjà visité. De plus l'algorithme est bien sur dépendant de la dfs codé.

Préférence

Kosaraju était le dernier sur ma liste de préférence mais lors de son implémentation sa définition peut être directement transposé en code ce qui facilite grandement l'implémentation et qui le fait passer devant Tarjan.

2.4 Gabow

Implémentation

Pour Gabow nous reprenons une structure semblable a une dfs, cependant cette fois ci l'implémentation demande l'utilisation de 2 piles. Une fois de plus il faut être méticuleux et le code donnera le résultat attendu.

Préférence

Gabow qui était premier sur ma liste de préférence et qui est resté premier même lors de l'implémentation car c'était le dernier algorithme que j'ai implémenté et sa structure qui est quasiment une copie carbone de l'algorithme de Tarjan ont fait que son implémentation était extrêmement facilité.

3 Comparaison

Pour comparer les algorithmes entre eux, j'ai simplement lancé chacun des algorithmes sur les mêmes graphes générés automatiquement, les graphes ont entre 10 et 1500 nœuds et le nombre d'arêtes par nœuds varie entre 10% et 80%.
Et ensuite on fait passer les résultats dans un script Gnuplot pour afficher des courbes.

3.1 Graphe creux

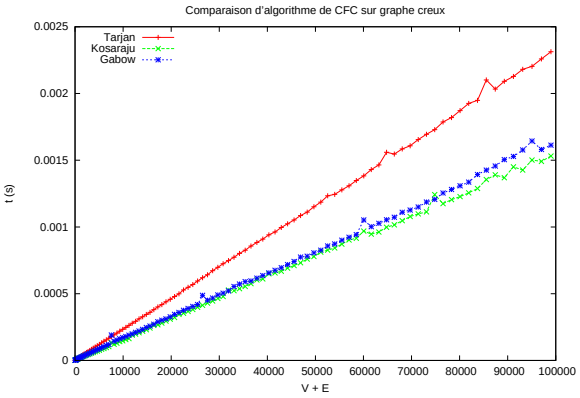

Figure 1: Comparaison sur graphe creux

Nous remarquons bien que l'implémentation de l'algorithme de Kosaraju et de Gabow ont un temps d'exécution très proche tandis que l'algorithme de Tarjan prends 1.5 fois plus de temps. Cela peut être du à un problème d'implémentation qui augmente le coefficient de la complexité de Tarjan.

3.2 Graphe dense

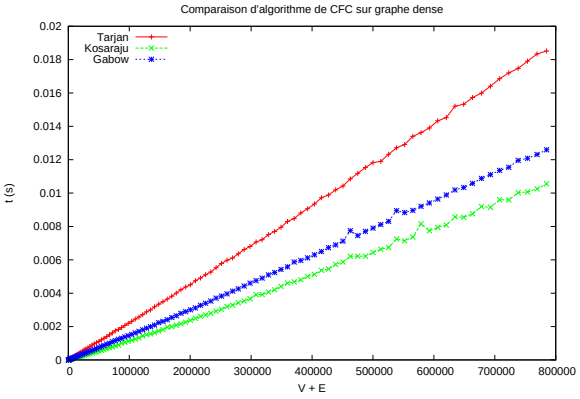

Figure 2: Comparaison sur graphe dense

Les résultats restent les mêmes avec l'implémentation de Tarjan qui est la plus lente mais à la place d'avoir ici Gabow et Kosaraju avec quasiment les même temps, nous remarquons que Kosaraju est l'implémentation la plus rapide. Encore une fois nous avons sûrement un problème d'implémentation.

3.3 Graphe à densité quelconque

Encore une fois les résultats restent extrêmement sensible même avec des graphes qui ont un nombre variable d'arête par sommet.

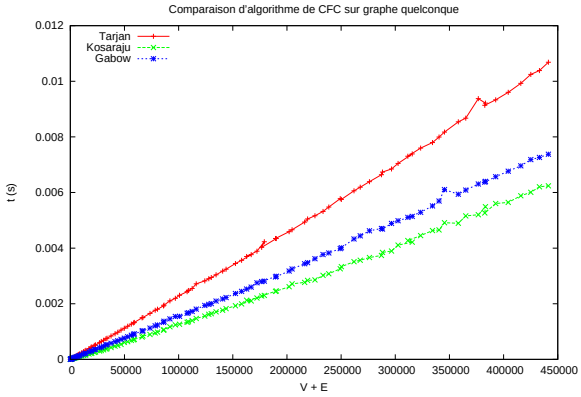

Figure 3: Comparaison sur graphe quelconque

4 Conclusion

Nous avons pu remarquer grâce à cet exercice que la définition d'un algorithme et son implémentation, en gardant la complexité voulu, est un exercice assez difficile car Kosaraju et Gabow sont des algorithme élégant mais ils paraissaient assez compliqué a codé et leur complexité pouvaient en souffrir cependant nous remarquons que c'est Tarjan l'algorithme le plus "basique" qui a souffert de l'implémentation et qui prends un facteur 2 dans sa complexité.