Dauphine | PSL
UNIVERSITÉ PARIS

Place du Maréchal de Lattre de Tassigny
75775 Paris Cedex 16

dauphine.psl.eu

Nathan Zafizara-Benetrix – L3 MIAGE App
Directed Feedback Vertex Set

1 – Approche

Le cœur de mon approche se base sur le théorème suivant :

> **Théorème 2.2** *Soit $G = (V, A)$ un graphe orienté. Alors les 4 propriétés suivantes sont équivalentes :*
>
> 1. *G admet un ordre topologique ;*
> 2. *G est sans circuit ;*
> 3. *Il n'y a pas d'arc arrière dans un parcours DFS de G (quel que soit le parcours) ;*
> 4. *L'ordre suffixe inverse d'un parcours DFS de G est un ordre topologique.*

Plus précisément, *G est sans circuit si et seulement si il n'y a pas d'arc arrière dans un parcours DFS de G*.

Mon algorithme consiste donc à supprimer l'ensemble des arcs arrière.

Avant de pouvoir les supprimer, il fallait d'abord les identifier. Comme vu en cours, l'identification s'effectue par une recherche en profondeur, puis en fonction des valeurs des préfixes et suffixes. J'avais créé au début un $EdgeTypeEnum$ qui listait l'ensemble des types d'arcs possible (avant, arrière, croisé, d'arbre) mais que j'ai finalement supprimé car un booleen mis à $true$ si l'arc est arrière et à $false$ sinon était suffisant, il n'était pas nécessaire de connaitre le type des autres arcs.

Pour supprimer ces arcs, une solution brutale est de supprimer tous les sommets présents dans les arcs arrière, cependant cette solution n'est pas performante car supprime beaucoup trop de sommets.

Il m'est donc venu immédiatement l'idée de supprimer qu'un seul des sommets de chaque arc arrière, mais lequel d'entre les deux ?

Supposons que nous avons deux arcs arrière (5,1) et (2,5). 5 étant présent dans les deux arcs arrière, c'est lui qu'il faut supprimer.

Ainsi l'idée de supprimer à chaque fois le sommet le plus présent dans l'ensemble des arcs arrière m'est apparue. Cela a donné la création du tableau $List < Integer >$

Dauphine | PSL
UNIVERSITÉ PARIS

Place du Maréchal de Lattre de Tassigny
75775 Paris Cedex 16

dauphine.psl.eu

$countArcArriere$ qui à l'indice $i - 1$ donne le nombre de fois que le sommet i est dans un arc arrière.

Lors de la première version de mon algorithme, la boucle principale était un $while(il\ y\ a\ des\ arcs\ arrière)$, et consistait à supprimer le sommet donnant la valeur maximum de $countArcArriere$. Une fois supprimé, une nouvelle recherche des arcs arrière était lancée, et la liste $countArcArriere$ était actualisée. Étant donné qu'il fallait à chaque itération supprimer un sommet, supprimer tous les arcs associés, refaire une recherche en profondeur et actualiser $countArcArriere$, cet algorithme était très chronophage et n'était finalement pas si performant.

Ne sachant, à ce moment, pas utilisé le SIGTERM, une vérification du dépassement du temps était effectuée à chaque passage dans la boucle tant que, ce qui a donné $while(il\ y\ a\ des\ arcs\ arrière\ ET\ le\ temps\ limite\ n'est\ pas\ depassé)$. Il était impossible de fixer le temps limite à 10 minutes, car le passage dans le $while$ pouvait être après le dépassement de temps (par exemple à 10min07sec). J'avais donc fixé le temps limite à 9 minutes.
Malgré cette limite de temps, il restait quelques dépassements de temps dans le test Optil, surement dus au fait que le passage dans le while était trop tard. Je ne pouvais pas baisser encore plus le temps, au risque de limiter encore plus la performance pour les graphes moyens, donc j'avais fixé une limite de sommets ou d'arcs. Le programme n'exécutait pas l'algorithme si le graphe dépassait 30 000 sommets ou 2 000 000 d'arcs et donnait pour solution tous les sommets ayant un degré sortant strictement positif (car un sommet ayant un degré sortant nul n'est nécessairement pas dans un cycle). Cette technique m'a permis d'avoir des résultats sur tous les 100 graphes du test, mais des résultats pas convaincants.

Il fallait donc réfléchir à une approche plus performante. Pour cela, j'ai fait le constat suivant : le théorème dit « *G est sans circuit si et seulement si il n'y a pas d'arc arrière dans un parcours DFS de G »,* ce qu'il veut dire qu'il faut nécessairement supprimer tous les arcs arrières. Une nouvelle approche consistait donc à itérer non pas sur les sommets, mais sur les arcs. J'ai voulu conserver ce fameux tableau $countArcArriere$ qui me semblait vraiment pertinent, donc mon algorithme est le suivant : pour chaque arc arrière, je supprime le sommet ayant la valeur de $countArcArriere$ la plus grande, et si un des deux sommets était déjà supprimé, alors cet arc était passé car n'existait en réalité plus. Cet algo, qui est relativement simple, m'a donc permis de supprimer au plus un sommet par arc arrière.

Les résultats des tests étaient sans appel, cet algorithme était bien plus performant que la première approche car supprime beaucoup moins de sommets, et plus rapidement du au fait qu'un seul DFS était effectué, et qu'il n'était plus nécessaire de supprimer des sommets et arcs du graphe pour retester.

Cependant il restait encore quelques graphes donnant dépassement de temps limite, j'ai au début remis les conditions de dépassement, s'il y a plus de 2 250 000 (valeur trouvée à tâtons) et effectué une vérification de temps tous les x arcs visités. Trouver la valeur optimale de ce x était assez compliqué car s'il est petit, alors le temps sera vérifié souvent

Dauphine | PSL
UNIVERSITÉ PARIS

Place du Maréchal de Lattre de Tassigny
75775 Paris Cedex 16

dauphine.psl.eu

donc sera plus précis, mais cela requiert beaucoup de calculs. Là où un x grand permettait un gain de performance mais une vérification assez imprécise. J'ai essayé plusieurs valeurs comme 50, 100, 250 ou 500, mais je n'ai jamais su laquelle était la meilleure.

J'ai pu, heureusement, abandonner cette idée car un camarade de classe m'a expliqué l'utilisation du SIGTERM, qui m'a permis d'exploiter pleinement les 10 minutes qui me sont alloués, et de ne plus passer les graphes trop gros.

Il fallait juste trouver la solution à envoyer en cas de SIGTERM, et j'ai décidé d'utiliser la solution brutale énoncé au début de cette partie, donc de supprimer l'ensemble des sommets présents dans au moins un arc arrière.
Les résultats avec l'implémentation de SIGTERM sont les meilleurs que j'ai pu avoir sur Optil.

2 – Algorithmes et complexité

Soit $G = (V, A)$ une graphe orienté quelconque, on pose $|V| = n$ et $|A| = m$.

Le programme est composé de deux méthodes principales : $DFS\_Iterative()$ et $findVerticesToDelete\_V2()$ (V2 car comme expliqué précédemment, j'ai eu 2 approches pour ce projet).

$DFS\_Iterative()$, l'algorithme de recherche en profondeur nous donnant les valeurs de préfixes et suffixes, a été récupéré en pseudo-code sur le cours d'algorithmes dans les graphes des L3 IM2D.

Voici une capture d'écran du polycopié :

```
ALGORITHME DFS_itératif
lire G, n
pour i allant de 1 à n
   pref[i] <- 0
   suff[i] <- 0
P <- PileVide()
A <- ensemble vide
p <- 1
s <- 1
Pour i allant de 1 à n
    Si pref[i]==0 Alors
        empiler(P,i)
        pref[i] <- p
        p <- p+1
        Tant que P non vide faire
            v <- sommetPile(P)
            Si existe un successeur j de v t.q pref[j]==0 alors
                empiler(P,j)
                pref[j] <- p
                p <- p+1
                A <- A + (v,j)
            Sinon
                i <- depiler(P)
                suff[i] <- s
                s <- s+1
```

L'algorithme implémenté en Java est essentiellement la même chose, je n'ai juste pas conservé l'ensemble des Arcs A, car celui-ci est rempli dans la lecture du graphe.

Il a une complexité de $O(n + m)$ car on parcours tous les sommets et on passe dans le pire des cas par tous les arcs.

Dauphine | PSL
UNIVERSITÉ PARIS

Place du Maréchal de Lattre de Tassigny
75775 Paris Cedex 16

dauphine.psl.eu

$List < Integer > \ findVerticesToDelete\_V2()$ est l'algorithme permettant de trouver les sommets à supprimer. Il implémente donc l'approche expliquée ci-dessus, donc de supprimer pour chaque arc arrière le sommet étant le plus de fois présent dans l'ensemble de tous les arcs arrière.

Il a une complexité de $O(m * n)$ car il parcours tous les arcs du graphe et pour chaque arc il parcourt (2 fois) la liste $countArcArriere$ qui est de taille n.

3 – Conclusion

J'ai vraiment apprécié faire ce projet, le fait d'être en compétition avec les camarades de classes me donnait vraiment envie de me surpasser à chaque fois pour faire la meilleure performance.
Je trouve que je m'en suis sorti avec un algorithme finalement assez simple, pour des résultats et un classement plus que convenable à mon goût.

Ce projet était aussi un défi personnel, celui de le coder en Java. Je n'ai jamais vu le Java de manière théorique, mais je le pratique au quotidien depuis le début de l'année en entreprise. Étant un langage que j'aime bien, et que je préfère de loin à Python, je me suis donc mis au défi de partir de zéro (là ou en entreprise toutes les fondations sont déjà faites) afin de réaliser ce projet. J'ai beaucoup appris sur ce langage durant ce projet, et je pense que j'ai réussi le défi que je me suis lancé.