DOUBABI Mustapha
L3 MIAGE Apprentissage

Rapport du projet

Algorithme dans les graphes

Vue d'ensemble

Mon algorithme traite la problématique de 2 manières différentes selon la taille des graphes.

Ainsi, le programme est divisé en 2 parties : une qui traite les graphes dont le nombre de nœuds est inférieur à 3000 et une qui traite les graphes dont le nombre de nœuds est supérieur à 3000. Ce nombre a été établi selon les performances respectives de ces 2 algorithmes.

Néanmoins, l'idée générale reste la même : on fait des parcours DFS, on détecte des arcs arrières et on traite ces informations pour rendre le graphe acyclique. Dans les 2 cas, les programmes retournent le graphe entier si 10 minutes se sont écoulées.

Explication du premier algorithme (noeuds < 3000)

Tout d'abord, je supprime tous les nœuds dont le degré sortant est nul à l'aide des fonctions Nettoyer_Graphe( ) et Supprimer_Noeuds( ) dans le but d'alléger les graphes. Ces nœuds supprimés (dont l'ensemble sera noté I) n'ont aucune utilité puisque le parcours en profondeur ne renvoie aucun autre nœud et on ne peut pas trouver d'arcs arrière (uv) tel u ou v appartiennent à I.

Par la suite, j'applique la fonction ArcA_Bis( ) sur chaque nœud du graphe, et ce afin d'obtenir tous les "seconds" nœuds de tous les arcs arrières de tous les parcours en profondeur.

Je classe ces nœuds selon leur redondance (dans le sens décroissant) et je les supprime un à un dans une boucle qui teste si le graphe est toujours cyclique (à l'aide d'une fonction IsThereACycle( )) après chaque suppression de nœud.

Le programme s'arrête dès que le graphe est acyclique et je retourne l'ensemble des sommets qui ont été supprimés depuis le lancement de l'algorithme (sans compter l'ensemble I).

Complexité : **O (n^2 +nm)** tel que n = |V| et m = |A| car le DFS de complexité O(n+m) est appliqué sur tous les nœuds du graphe (y compris ceux dont le degré est nul mais considérons ce nombre comme peu important).

Explication du second algorithme (noeuds >= 3000)

Pour ce programme, je ne supprime pas de sommets dont le degré sortant est nul par souci de temps. Sa structure consiste seulement à appliquer la fonction ArcA ( ) pour déterminer les arcs arrières dans un graphe (plus précisément le 'premier' nœud de l'arc). J'expliquerai donc seulement comment fonctionne ArcA ( ).

La variable stack est une liste utilisée pour stocker les nœuds parcourus au cours de la recherche. La variable discovery_time est un dictionnaire qui enregistre le temps (noté t) à laquelle chaque nœud a été découvert.

L'algorithme commence par examiner chaque nœud du graphe : si un nœud n'a pas encore été découvert (ou dans un arc arrière), il est ajouté à la variable stack et à discovery_time.

Ensuite, une boucle while est utilisée pour continuer la recherche tant que la stack n'est pas vide. Pour chaque voisin de chaque nœud, le code vérifie s'il a déjà été découvert. Si le voisin n'a pas été découvert, il est ajouté à stack, à discovery_time et on commence à l'explorer à son tour. Sinon, si le voisin se trouve dans la stack, cela signifie qu'un arc arrière a été trouvé, alors on l'ajoute à la variable back_edges.

Enfin, si un nœud a été exploré, il est retiré de la stack.

Complexité : **O (n + m)** tel que n = |V| et m = |A|.

Conclusion sur le projet

Ce projet s'est révélé très enrichissant dans la mesure où il m'a permis de me confronter aux difficultés de développement d'un algorithme complexe. Le fait de nous avoir mis en compétition avec les autres élèves nous a tous poussé à donner du meilleur de nous-mêmes.