Rapport du travail effectue sur les TP1 et 2 d'algorithmique gnrale

Arthur Zucker

08/02/2019

Rapport du travail effectué sur les TP1 et 2 d'algorithmique générale

J'ai choisi pour ce projet de créer une forge pour pouvoir travailler chez moi et à l'école. De plus vu la taille du programme, j'ai chois de programmer de façon modulaire afin que les temps de compilation soient réduits. J'ai donc écris un `MakeFile` que vous trouverez en annexe.

Partie A : Préliminaire

J'ai choisi comme structure d'implémenter un graphe sous ses trois formes : une matrice d'adjacencen, une liste pi et alpha et une liste de successeurs.

```
typedef struct Graphe{
    int nb_aretes;
    int nb_sommets;
    int oriente;
    int *pi;
    int *alpha;
    int **liste_adjacence;
    int **liste_successeurs;
}Graphe;
```

La première partie du TP était assez simple à implémenter, jusqu'au dfs, qui lui m'a pris beaucoup de temps. De plus la question sur la connexité a du être retravaillée pour répondre aux attentes du sujet.

Question 1 :

La fonction `Graphe* create_from_file(char nom_de_fichier[])` permet d'implémenter ma structure graphique à partir d'un graphe décrit en fichier texte. Cette donction implémente tout d'abord la matrice d'adjacence, qui est plus facile à manipuler. Ensuite la fonctiuon `void initialize_all(Graphe *graph)` initializse les valeurs de pi, alpha et de la matrice des successeurs. Lorsque l'on execute le mian, on peut entrer à l'aide d'un scanf, le nom du graphe que l'on souhaite étudier.

Question 2 :

Afin de répondre à cette question, j'ai utilisé la matrice d'adjacence, qui me paraissait plus simple à utiliser. A partir de celle-ci, je crée un fichier .dot en utilisant la syntaxe universelle dot. Je pourrais pour améliorer l'efficacité utiliser la liste d'adjacence.

Question 3 :

J'ai écris 5 fonctions de dfs qui ont chacune une utilité particulière. Cependant la forme des ces 5 fonctions est toujours la même : la fonction `void dfs(Graphe *graphe,int sommet_depart ,liste_ordre **ordre_par_sommet, int **sommet_marque_dugraphe)` que j'ai baptisé "le gros dfs" utilise la fonction `void dfss(Graphe *graphe,int sommet, int *sommet_marque, liste_ordre *ordre)` que j'ai surnommée "le petit dfs". Le petit dfs prend en entrée un sommet, dont il va parcourir le premier successeur en profondeur pour ensuite parcourir ses autres successeurs aussi en profondeur. Pour chaque sommet parcouru, le sommet est marqué et le sommet est aussi ajouté à l'ordre de parcours du dfs. De plus, l'un des dfs permet de créer l'arbre dfs, ou l'arbre de parcour suprimant les arretes non utilisée par le parcour. Sur les 5 dfs codé, 4 utilisent la matrice pi et alpha, qui permettent d'avoir la meilleure implémentation possible. L'un utilise la matrice d'adjacence qui est nécessaire pour l'orientation forte d'un graphe non orienté.

Question 4 :

Afin de teste si un graphe est connexe, j'ai juste à utiliser la fonction de dfs que j'ai implémentée précédement. Seulement si le graphe d'entrée est orienté, je dois le "dés-orienter" le rendre non orienté car l'ordre n'est pas important pour la connexité. Ainsi, il ne me reste plus qu'à lancer un dfs et si le tableau des ordre de parcour renvoyé par mon dfs contient deux ordre de parcours, cela veut dire que à partir d'un des sommet, le grand dfs ne parvient pas à marquer un autres des sommet et doit donc relancer le petit dfs sur celui ci! La condition de connexité est donc très simple.

Question 5 :

L'inversion du graphe orienté se fait en utilisant la matrice d'adjacence et la fonction `void initialize_all(Graphe *graph)` pour ensuite mettre à jour le graph obtenu.

Partie B : Composantes fortement connexes

Cette partie du TD à été la plus longue et laborieuse : c'est nottement pour cette partie que j'ai du écrire plusieurs dfs.

Question 1 & 3:

Pour déterminer les composantes fortement connexes j'ai utilisé l'algorithme de kosaraju, qui utilise un premier dfs (ordre post parcour) puis un deuxième dfs sur l'inverse du graphe en utilisant l'ordre de paracour inverse obtenue par le premier dfs. Le tableau d'ordre retourné par ce deuxième dfs contient les composantes fortement connexes de ce graphe. A partir de ce tableau, je crée un deuxième graphe, contenant en sous graphe les composantes fortement connexes.

Question 2 :

La complexité de cette algorithme est :

Question 3 :

cf question 1

Figure 1: sorbonne

Partie C : Orientation forte d'un graphe

Partie assez facile seulement, la dernière question n'est pas implémentée de la bonne manière.

Question 1 :

Question 2 :

Question 3 :