Finding Strongly Connected Components
Projet 8 INF442

Abdessamad Ed-dahmouni, El Mehdi Achour

June 4, 2017

1. Question 1

Pour trouver les composantes fortement connexes d'un graphe orienté nous nous sommes inspirés de l'algorithme de Kosaraju en le modifiant pour l'adapter au format des fichiers texte représesntant les graphes que nous prenons comme input. Nous utilisons donc des listes d'ajacence pour représenter nos graphes sous forme de tables de hashage puisque les tableaux ne sont pas adaptés au format de certains fichiers textes récupérés depuis le SNAP, comme le Wiki-vote.txt ou certains sommets ont des numéros qui dépassent le nombre de sommets. Les étapes de l'algorithme sont les suivantes:
1) Nous créons une pile vide et nous faisons un parcours en profondeur du graphe, dans ce parcours, après avoir appelé récursivement la fonction qui permet le parcours en profondeur sur les voisins d'un sommet v nous ajoutons ce sommet à la pile.
2) Créer le graphe transposé en inversant les directions de tous les arcs de notre graphe.
3) Retirer les éléments de la pile un à un jusqu'à ce que celle-ci soit vide. Pour un élément retiré v, nous prenons v comme source et nous faisons un parcours en profondeur. Ce parcours en profondeur permettra d'avoir à travers la composante connexe contenant v.

Nous avons pour ce faire écrit un nouveau constructeur qui permet de générer notre graphe à partir d'une chaine de caractères comportant le nom du fichier texte à traiter, ainsi que deux fonctions l'une pour remplir la pile et la seconde pour le second parcours du graphe, ainsi qu'une fonction qui nous permet de créer le graphe transposé. Nous avons aussi choisi d'écrire les composantes connexes chacune dans une ligne dans un fichier texte que génére le programme.

Comme nous effectuons 2 parcours en profondeur d'un graphe plus une transposition. La complexité en temps est de $\mathcal{O}(n+m)$ ou n est le nombre de sommets et m le nombre d'arcs.

Nous avons testé notre programme sur quelques fichiers du SNAP ainsi que le graphe aléatoire généré dans la question 2.

2. Question 2

Nous avons écrit un nouveau constructeur qui prend en argument un entier n représentant le nombre de sommets ainsi qu'un réel p représentant la probabilité. Et pour tout couple de sommets (u,v) nous générons une variable aléatoire uniforme appartenant à $[0,1]$ grâce à la fonction $rand()$ que nous renormalisons et si celle-ci est inférieur à p nous ajoutons un arc de u vers v.

En fonction du nombre de sommets, la probabilité p permet de générer en moyenne des graphes conduisant à des composantes connexes non toutes isolées à partir d'un certain seuil. Par exemple, pour $n = 15$, le seuil est à peu près $0,17$.

3. Question 3

On vérifie facilement que pour une répartition du travail la plus équitable sur P processeurs : (n modulo P) des processeurs doivent prendre en charge $\lfloor n/P \rfloor + 1$ sommets chacun, et les autres processeurs prendront en charge $\lfloor n/P \rfloor$ sommets chacun. Ainsi chaque processeur s'occupe de générer une liste d'adjacence pour un certain ensemble de sommets en parallèle. Nous utilisons MPI pour la parallèlisation.

4. Question 4

Pour cette question, nous avons écrit le programme de façon séquentielle puis nous avons essayé de le parallèliser, ainsi nous avons écrit une fonction $SCCpar()$ qui prend en argument $vertices$ l'ensemble des sommets du sous-graphe à traiter, le numéro du processeur $myid$, la somme n des tailles des composantes connexes calculées jusqu'à présent ainsi qu'un tableau $p[nprocs]$ de 0 et 1 indiquant si chaque processeur a été utilisé ou pas encore, ainsi que des paramètres que nous ne changeons pas durant tous les appels à cette fonction. Cette fonction active la case du processeur $myid$ dans p, calcule les ensembles de successeurs et prédecesseurs d'un sommet v choisi aléatoirement parmi $vertices$, puis en déduit les 4 ensembles définis dans l'énoncé en utilisant des librairies adaptées ($< set >$ et $< algorithm >$ principalement). Ensuite le processeur éxécutant cette fonction écrit la composante connexe S_1 dans la console et modifie ainsi n,puis il envoie P, n et respectivement les ensembles S_2, S_3 et S_4 si ils sont non vides à 3 processeurs $3 * myid + 1$, $3 * myid + 2$ et $3 * myid + 3$ qui à leurs tours dès réception du message éxécutent la fonction $SCCpar()$ sur ces ensembles de sommets. Cette fonction est donc récursive et la condition de terminaison est que la taille de l'ensemble des sommets à traiter soit nulle.

Le tableau p a été utilisé afin d'envoyer un message aux processus qui restent en attente d'un message pour terminer le programme. ceci est effectué par le processus qui calcule la dernière composante (on sait qu'on a calculé la dernière composante connexe quand la condition $(n == ntotal)$ devient vérifiée, où $ntotal$ est le nombre de sommets du graphe g).