Ecole Polytechnique

Cours : "Parallélisme"
Travaux dirigés
E. Goubault & M. Martel

TD 3

21 janvier 2002

1 Composantes connexes d'un graphe

Soit $G = (V, E)$ un graphe *non*-orienté: V est un ensemble de noeuds et $E \subseteq V \times V$ est l'ensemble d'arêtes du graphe G (en tant que relation, on supposera E symmétrique). On rappelle qu'un noeud x est *voisin immédiat* de y dans G si et seulement si xEy. Deux noeuds x et y sont *reliés* si et seulement si il existe une suite finie $(x_0, \cdots, x_n)$ de noeuds de G, avec $x_0 = x$, $x_n = y$ et (x_i, x_{i+1}) immédiatement voisins (pour $i = 0, \cdots, n-1$). La relation "être relié à" est une relation d'équivalence sur les noeuds de G. On appelle *ensemble de composantes connexes* de G l'ensemble des classes d'équivalence de noeuds de G pour cette relation d'équivalence. On dira qu'un noeud v de G appartient à la composante n de G si sa classe d'équivalence est la $n$ième parmi les composantes connexes de G.

Question 1

Soit u le nombre de noeuds d'un graphe G et v son nombre d'arêtes. Quelle est la complexité asymptotique, au pire, de l'algorithme séquentiel décrit précédemment (on ne cherchera pas à prouver ce résultat)?

Corrigé Le principe de l'algorithme séquentiel est de faire un parcours en profondeur d'abord, et de marquer chaque noeud rencontré en cours d'un parcours provenant initialement d'un sommet donné, par le numéro de ce sommet.

D'où une complexité de $O(u+v)$ ($O(u^2)$ au pire): pour chacun des u sommets, en série, on parcourt les arêtes non déjà parcourues (en tout au plus v).

On appelle pseudo-forêt tout graphe *orienté* (c'est-à-dire que les arêtes ont maintenant une direction: la relation de voisinage n'est plus forcément symétrique) tel que tout noeud ait au plus un voisin immédiat. Dit de façon plus formelle, pour tout noeud n de la pseudo-forêt $G = (V, E)$, le cardinal de $\{n' \mid (n, n') \in E\}$ est inférieur ou égal à un: le nombre d'arcs sortant de tout noeud est au plus un. A la figure 1 est donné un exemple de pseudo-forêt, à la figure 2 on trouvera un graphe qui n'est pas une pseudo-forêt.

On appelle arbre tout graphe *orienté* avec une seule racine (c'est-à-dire un seul point qui n'a pas de voisin immédiat), et un unique chemin entre tout noeud du graphe et cette racine. Remarquez bien que l'on définit les arbres avec une convention d'orientation des arcs exactement à l'envers de ce que l'on a fait par exemple en tronc-commun. On dit qu'un arbre est étoilé si chacun de ses noeuds est *directement* relié à la racine.

Un pseudo-arbre est un arbre auquel on a rajouté un arc allant de la racine à un autre noeud (par exemple à la figure 1). Un pseudo-arbre étoilé est un arbre étoilé auquel on a rajouté une boucle de la racine vers elle-même (on appelera encore ce noeud, la racine du pseudo-arbre étoilé).

On appelle graphe fonctionnel G tout graphe fabriqué à partir d'une fonction $f : V \to V$, où V est un ensemble fini, de la façon suivante: $G = (V, E)$ (le même V) où $E = \{(v, f(v)) \mid v \in V\}$.

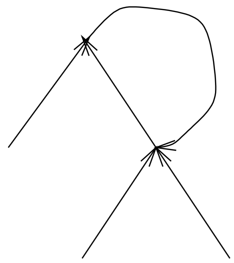

Figure 1: Une pseudo-forêt.

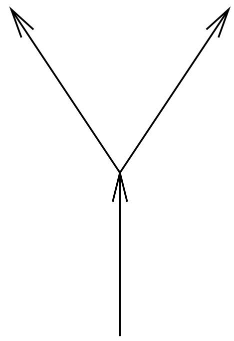

Figure 2: Un graphe qui n'est pas une pseudo-forêt.

Question 2

Est-ce que tous les graphes sont des graphes fonctionnels[1]? Montrer que les graphes fonctionnels sont des pseudo-forêts, que les pseudo-forêts sont des unions disjointes d'arbres et de pseudo-arbres et que les pseudo-forêts qui sont des graphes fonctionnels sont des unions de pseudo-arbres.

Corrigé Le graphe de la figure 2 n'est pas un graphe fonctionnel: le sommet intermédiaire aurait deux images par une fonction dans ce cas!

Un graphe fonctionnel est tel que chaque sommet v a un unique arc sortant: c'est $(v, f(v))$.

Soit G une pseudo-forêt. Considérons T une (quelconque) des composantes connexes de G. Remarquons déjà que T ne peut avoir qu'une composante fortement connexe ayant un cardinal (nombre de sommets) strictement supérieur à un, au plus. En effet, supposons que l'on ait par exemple 2 composantes fortement connexes C_1 et C_2 de cardinal strictement supérieur à un dans T connexe, alors il existe un arc a de $u \in C_1$ vers $v \in C_2$ (ou le contraire, peu importe). Mais il existe forcément un arc sortant de u, restant dans C_1, donc distinct de a, donc u a au moins deux arcs sortant, impossible car T est un sous-graphe d'une pseudo-forêt.

Si T n'a aucune composante fortement connexe de cardinal strictement supérieur à un, cela veut dire que T ne comporte aucun cycle. Dans ce cas, il ne peut y avoir deux chemins d'un sommet u vers un sommet v dans T, car sinon un sommet intermédiaire aurait au moins deux arcs sortant. Donc T est un arbre.

Supposons maintenant que T a une composante fortement connexe C non réduite à un point. Alors C est forcément un cycle simple (c'est à dire un chemin dont les deux extrémités sont identifiées) - sinon il y aurait plus d'un arc sortant dans cette composante fortement connexe. Soit r l'un quelconque des sommets. Enlevons l'unique arc sortant b de r. Alors $C \backslash b$ est connexe non fortement connexe, et $T \backslash b$ est une pseudo-forêt connexe, dont toutes les composantes fortement connexes sont réduites à un point. Par ce que l'on vient de voir, c'est un arbre: résultat, T est un arbre auquel on a adjoint un arc de la racine vers un sommet de T.

Enfin, un arbre ne peut pas être un graphe fonctionnel, car il n'y aurait pas d'image de la racine.

[1]On pourra soit donner une preuve si oui, soit donner un contrexemple, si non.

On va maintenant pouvoir définir un algorithme parallèle de recherche de composantes connexes sur une machine *PRAM*, dont on ne spécifie pas pour le moment le type (EREW, CREW ou CRCW). On suppose disposer au début de l'algorithme, d'une représentation d'un graphe $G = (V, E)$ par une matrice d'adjacence A, c'est-à-dire si on suppose $V = \{1, \cdots, n\}$, A est une matrice $n \times n$ telle que $A_{i,j} = 1$ si $(i, j) \in E$ (et 0 sinon).

On définit maintenant la fonction $C : V \to V$ avec

$$C(v) = \begin{cases} v & \text{si } v \text{ est un noeud isolé} \\ min\{u \mid A_{u,v} = 1\} & \text{sinon} \end{cases}$$

Question 3

Indiquer, selon le type de la PRAM utilisée, à quelle complexité on peut arriver pour le procédé qui, étant donné `A`, calcule le tableau qui à l'entrée v associe $C(v)$.

Corrigé

- EREW, ERCW, CREW: chaque $C(v)$ est calculé en parallèle indépendament. Reste à calculer un min sur n valeurs sur un EREW ou CREW, cours: $O(log(n))$. Donc $O(log(n))$ en tout.
- CRCW mode calcul de consistant (cours): un processeur par $A(u, v)$, lectures indépendantes. Sur chaque processeur (i, j): si valeur locale égale à un, écrire $min(C(j), i)$ sur $C(j)$. Complexité en $O(1)$.

Question 4

Soit G_C le graphe fonctionnel de C. On sait par la question 2 que G_C est une union de pseudo-arbres $G_i = (V_i, E_i)$ $(i = 1, \cdots, s)$. Montrer que tous les noeuds de chaque V_i sont dans la même composante connexe du graphe G de départ.

Corrigé Si V_i est de cardinal 1, alors c'est évident.

On suppose maintenant que le cardinal de V_i est strictement supérieur à 1. Soient u et v deux sommets distincts de V_i. Alors, $(u, C(u))$ appartient au pseudo-arbre G_i; de plus $(u, C(u))$ correspond à une arête de G. De même, pour $(C(u), C(C(u)))$ et ainsi de suite jusqu'à arriver à la racine r de G_i. On fait de même pour v. Donc il y a un chemin de u à v passant par r dans G.

On peut prouver de plus (mais on ne demande pas de le faire) que tout cycle de G_C est soit une boucle triviale (de la forme (v, v) pour un noeud v) soit contient exactement deux arcs. On sait aussi par la question 2 que chaque cycle des pseudo-arbres G_i contient la racine de G_i.

Question 5

On cherche à transformer les pseudo-arbres G_i de la pseudo-forêt G_C de la question 4 en pseudo-arbres étoilés T_i ou en points isolés.

Programmer un algorithme PRAM CREW (on le simulera par un programme JAVA multi-threadé) en $O(\lceil log(n) \rceil)$ (où n est le nombre de noeuds du pseudo-arbre considéré) effectuant cette transformation.

Corrigé Pour chaque sommet u d'un G_i, on cherche à lui associer en parallèle (comme unique arc sortant) le sommet de G_i atteignable depuis u de numéro minimum. C'est la technique de saut de pointeur du cours (qui nécessite des lectures parallèles) en $O(\lceil log(n) \rceil)$. On obtient ainsi un pseudo-arbre étoilé, de racine le sommet de numéro minimal dans G_i. Tous les G_i peuvent être traités en parallèle.

Remarquez que les T_i forment une pseudo-forêt qui est le graphe fonctionnel d'une fonction $D : V \to V$.

Soient r_i les racines des pseudo-arbres étoilés T_i $(i = 1, \cdots, s)$.

Grâce à la question 4, on sait que les r_i définissent un sur-ensemble des composantes connexes du graphe G de départ. Mais en fait, certains des r_i sont reliés par un chemin dans le graphe G à d'autres r_j. On acceptera sans démonstration le fait que r_i est relié par un chemin à r_j dans G si et seulement si il existe $v \in V_i$, $w \in V_j$ tels que $(v, w) \in E$.

Question 6

En utilisant la fonction D définissant les pseudo-arbres étoilés T_i, programmer une machine PRAM CRCW (en mode consistant) qui calcule efficacement le graphe des "super-noeuds" r_i par une matrice d'adjacence A', dans la mémoire partagée, tel que r_i est relié à r_j si il existe un chemin dans G qui relie r_i à r_j.

Corrigé On associe un processeur à chaque arête (x, y) de G. En parallèle, on écrit 1 dans $A'(D(x), D(y))$ (il faut le mode consistant). Cela est fait en $O(1)$.

Question 7

Indiquer *sans programmer* quelle serait *a priori* la complexité du calcul de la question 6 si on avait une machine PRAM CREW, et si on avait une machine PRAM EREW.

Corrigé Par le théorème de simulation (cours) pour CREW: $O(log(n^2)) = O(log(n))$ (autant de processeurs que d'arêtes, d'ordre au plus n^2). Pour passer de CREW à EREW, on n'a pas besoin de toutes façons de lectures en parallèle pour l'algorithme: également $O(log(n))$.

On définit un algorithme qui, partant de $k = 0$, $A_0 = A$ de taille $n_0 \times n_0$, fait les étapes suivantes, tant que $n_k > 0$ (n_k étant la taille de A_k):

- calculer G_C et les T_i par la méthode de la question 4 et 5,
- calculer $A_k = (A_{k-1})'$ de taille $n_k \times n_k$ par la méthode de la question 6.

Programmer cet algorithme.

Question 8

Prouver que $n_k \leq n_{k-1}/2$. On pourrait prouver, *mais on ne le demande pas ici*, que l'algorithme sur une machine PRAM CRCW en mode consistant prendrait un temps $O(log^2(n))$ pour un total de $O(n^2)$ opérations, où n est le nombre de noeuds dans le graphe G de départ. Quelle est l'efficacité de l'algorithme?

Corrigé A la fin de l'itération k, certains des n_{k-1} pseudo-arbres étoilés seront réduits à des cycles triviaux, contenant uniquement un sommet: disons n'_{k-1} de ces n_{k-1} pseudo-arbres. Ceux qui restent, en nombre de $n_{k-1} - n'_{k-1}$, seront contenus dans les n_k pseudo-arbres non triviaux. Donc $n_k \leq (n_{k-1} - n'_{k-1})/2$, parce-que tout pseudo-arbre non-trivial contient au moins deux sommets. Donc $n_k \leq n_{k-1}/2$.

Pour ce qui est de l'efficacité, il nous faut de l'ordre de n^2 processeurs pour implémenter un algorithme séquentiel qui prendrait un temps de l'ordre de $O(n^2)$, en un temps de $O(log^2(n))$. Soit une efficacité de $O(1/log^2(n))$.