GRI – Cours 2

Christophe Prieur Clémence Magnien

22 janvier 2014

Connexité et calcul des distances

Le test de la connexité et le calcul des distances entre deux nœuds se font par un parcours en largeur.

Parcours en largeur

Algorithm 1: Parcours en largeur

```
Argument : sommet de départ v_0;
F <- file vide;
ajouter v_0 dans F;
pe e <- tableau de taille   , initialisé à -1;
pe e[v_0] <- v_0;
while F non vide do
    u <- defiler(F);
    for chaque voisin v de u do
        if pere[v] == -1 then
            ajouter v dans F;
            pere[v] = u;
        end
    end
end
```

Pour chaque sommet v, on stocke le père de v dans l'arbre de parcours. Si `pere[v] == -1`, cela signifie que v n'a pas été visité. Par ailleurs, cette information sera utile par la suite (dans les prochains cours et TP, lorsqu'on aura besoin de calculer la longueur des chemins).

Implémentation d'une file FIFO

Pour implémenter le parcours en largeur, on a besoin d'une file. On peut l'implémenter de la manière suivante.

On prend un tableau de taille , et deux pointeurs, in et out. Le tableau est comme un tube, où on ajoute des valeurs vers la droite et où on les prend de la gauche. Lorsque les pointeurs in et out sont au même point, c'est que le tube est vide, sinon in est situé à droite de out et le contenu du tube est entre les deux.

```
fifo_create(taille) :
 fifo = malloc(sizeof(*fifo));
 fifo->tab = malloc(n * sizeof(* fifo->tab));
 fifo_init(fifo);
 return fifo;

fifo_init(fifo) :
 fifo->in = fifo->out = fifo->tab;

fifo_add(fifo, v) :
 *(fifo->in++) = v;

fifo_take(fifo) :
 return *(fifo->out++);
```

Attention, à chaque fois qu'on veut réutiliser le tube pour un nouveau parcours, on doit le réinitialiser (c'est à dire remettre les pointeurs au début du tableau).

Connexité, composantes connexes

Un parcours en largeur va visiter tous les sommets de la composante connexe du sommets de départ, et uniquement ces sommets.

Pour visiter tous les sommets, l'algorithme est le suivant : tant qu'il existe un sommet v non visité, faire BFS(v).

Ceci peut être fait de la façon suivante : pour v0 allant de 0 à n-1, faire si pere[v0] == -1, alors BFS(v0).

Si l'on veut connaître le nombre de composantes connexes et leurs tailles, on peut stocker un plus un tableau `comp` qui contient un identifiant de la composante à laquelle appartient un noeud.

Par exemple

```
cc_id = 0;
Pour v0 de 0 à n-1, faire
  si pere[v] == -1, alors
    BFS(v) // BFS modifié pour ajouter une ligne
           // comp[v] = cc_id quand on ajoute v dans la file
    cc_id = cc_id + 1
```

Distance

La distance entre deux nœuds n'est définie que pour les nœuds appartenant à une même composante connexe.

On s'intéressera donc au calcul de la distance dans la plus grande composante connexe.

Le parcours en largeur permet de calculer les distances entre le nœud source et tous les nœuds de sa composante.

Il faut pour cela modifier le parcours en largeur.

On stocke les distances dans un tableau de taille `n`.

Ensuite :

— Si v0 est le nœud source, alors $d(v0, v0) = 0$
— Si v est un voisin de u trouvé dans le BFS, alors $d(v0, v) = d(v0, u) + 1$

Observations en pratique

Observation la plupart du temps d'une composante connexe géante.

En pratique, la distribution des distances est homogène. La distance moyenne est faible par rapport au nombre de nœuds (de l'ordre de log)

Cœur et périphérie

Notion de périphérie : parties arborescentes de la plus grande composante connexe. On peut supprimer la périphérie en supprimant itérativement les nœuds de degré 1 jusqu'à ce que tous les nœuds aient degré au moins 2. Les nœuds qui restent forment le cœur.

Définition formelle du cœur : sous graphe $H = (C, E(C))$ de G tel que, pour tout $v \in C$, $d^\circ{}_H(v) \geq 2$.

Remarque : le cœur est nécéssairement connexe.

Comment calculer les nœuds qui sont dans la périphérie (c'est une variante du parcours en largeur) :

— Calculer les composantes connexes et leurs tailles
— Pour tout nœud u faire `periph[u] = 0`
  (le tableau periph stocke si un nœud est dans la périphérie ou non)
— Recopier les degrés des nœuds dans un tableau `degretmp`
— Initialiser une file vide
— Pour chaque nœud v de la plus grande composante connexe :
  — si degretmp[v] == 1 alors
    — mettre v dans la file
    — periph[v] = 1
— Tant que la file est non vide, faire :
  — récuperer un sommet v dans la file
  — Pour chaque voisin u de v faire :
    — `degretmp[u]- -`
    — si degretmp[u] == 1
      — mettre u dans la file
      — periph[u] = 1

À la fin, les nœuds pour lesquels periph[u] == 1 sont dans la périphérie de la plus grande composante connexe. On peut les compter en faisant une passe sur le tableau.