Option Informatique
Complexité et arbres

Corrigé

21 octobre 2007

1 Parcours en escalier dans un tableau

Question 1 *Décrire un algorithme de remplissage du tableau à deux dimensions représentant la matrice M.*

On remplit d'abord la première colonne avec les puissances de 2.
On déduit ensuite chaque colonne de la précédente en la multipliant par 3.

```
let remplir n =
  let m= make_matrix (n+1) (n+1) 1 in
  for i=1 to n do
      m.(i).(0) <- 2 * m.(i-1).(0)
  done ;
  for j=1 to n do
      for i=0 to n do
          m.(i).(j) <- 3 * m.(i).(j-1)
      done
  done ;
  m ; ;
```

Question 2 *Quel est le nombre total de multiplications de votre algorithme ?*

Le nombre de multiplications est égal à $(n+1)^2 - 1 = n^2 + 2n$.

Question 3 *Ecrlre la fonction* `pgcd` *qui a prend pour arguments une matrice M (telle que $m_{i,j} = 2^i 3^j$ pour $0 \leq i \leq n$ et $0 \leq j \leq n$) et quatre indices i, j, k, l dans $[0, n]$ et retourne pour résultat la valeur du plus grand commun diviseur de $m_{i,j}$ et de $m_{k,l}$.*

On utilise la règle donnant le PGCD lorsque les deux nombres sont factorisés en produit de nombres premiers.

```
let pgcd m i j k l =
   let p = min i k and q = min j l in m.(p).(q) ; ;
```

Question 4 *On dispose d'une matrice M telle que $m_{i,j} = 2^i 3^j$ pour $0 \leq i \leq n$ et $0 \leq j \leq n$, décrire un algorithme qui étant donné un entier p,$1 \leq p \leq 2^n 3^n$, calcule les indices i, j du plus grand des minorants de p de la forme $2^i 3^j$. On s'efforcera dans cette question de minimiser le nombre de comparaisons en exploitant la remarque qui suit.*

On cherche le plus grand élément inférieur ou égal à p sur chaque ligne (s'il existe).

Sur la ligne d'indice 0, on part de la droite, jusquà trouver un élément $m_{0,j_0} \leq p$.

A l'entrée de la première boucle et à la sortie de chaque boucle indexée par i_0, (i, j) est la position du plus grand des éléments $\leq p$ situés sur les lignes d'indice allant de 0 à i_0 (invariant de boucle).

Quand on passe de (i_0, j_0) à $(i_0 + 1, j_0 - 1)$, il est inutile de de comparer m_{i_0+1,j_0-1} à $m_{i,j}$:
en effet $m_{i_0+1,j_0-1} = \frac{2}{3} m_{i_0,j_0} \leq \frac{2}{3} m_{i,j} < m_{i,j}$, ce qui économise un test.

Lorsqu'on est dans la colonne d'indice 0 et que l'étape suivante conduit à $j = -1$, il n'y a alors plus rien faire dans les lignes suivantes.

```
let pgm m p =
  let n =(vect_length m) -1 in
  let i = ref(0) and j = ref(n) and j0 = ref(n) in
  while m.(0).(!j0) > p do j0 :=!j0 - 1 done;
     j :=!j0;
     for i0=1 to n do
       if!j0 >= 0  && m.(i0).(!j0) > p  then
         j0 :=!j0 -1
       else if!j0 >= 0 &&  m.(!i).(!j) < m.(i0).(!j0) then
         begin
         i := i0; j :=!j0
         end
      done;
      (!i,!j,  m.(!i).(!j));;
```

Question 5 *Combien de comparaisons mettant en jeu un éliment de la matrice M effectue votre algorithme ?*

Appelons curseur le couple (i_0, j_0) : celui-ci a trois sortes de déplacements possibles :

$\longleftarrow$ sur la première ligne d'indice 0, $\swarrow$ ou $\downarrow$ ensuite.

Le déplacement vertical utilise deux comparaisons, les deux autres une seule.

- S'il n'y a pas de déplacement horizontal sur la première ligne, alors à chaque boucle on descend si possible d'une ligne, ce qui donne au plus $2\,n$ comparaisons (majorant atteint lorsque $p = m_{n,n}$).
- S'il n'y a eu d'abord p déplacements horizontaux sur la première ligne, alors ensuite, chaque déplacement vertical du curseur est automatiquement suivi d'un déplacement oblique car :

si $m_{i_0,j_0} \leq p$ et $m_{i_0,j_0+1} > p$ et $m_{i_0+1,j_0} \leq p$, alors $m_{i_0+2,j_0} = 4\, m_{i_0,j_0} > 3\, m_{i_0,j_0} = m_{i_0,j_0+1} > p$.

Il y a donc au plus $n - p$ déplacements obliques et $\lceil \frac{n-p}{2} \rceil$ déplacements verticaux du curseur, d'où un nombre de comparaisons majoré par : $p + (n - p) + 2 \lceil \frac{n-p}{2} \rceil = n + (n - p + 1) \leq 2\, n$ car $p \leq 1$.

Au total, il y a au plus $2\, n$ comparaisons mettant en jeu un élément de la matrice M.

2 Parcours en largeur d'abord

Question 6 *Donner le type d'un arbre binaire strict correspondant à cette définition.*

```
type arbre = feuille of int
  | noeud of arbre * float * arbre;;
```

Question 7 *Que doit valoir L initialement ?*

```
L := [T];;
```

Question 8 *Que doit-on faire pour le traitement, la réactualisation de L et la poursuite du calcul en largeur d'abord ? Proposer une structure plus adéquate pour L qu'une liste.*

Si la tête de L est une feuille, on lui applique `fonc_f`. Dans le cas contraire, la tête de L est de la forme (G, x, D). il faut alors appliquer `fonc_n` à x

et rajouter G et D en queue de liste.

On remarque que l'ajout en queue d'une liste est en complexité linéaire en taille de la liste. La structure de données permettant la lecture en tête et l'ajout en queue la plus efficace est la file de priorité (pile *FIFO*).

Question 9 *Quand arrête-t-on le traitement ?*

On arrête le traitement quand la liste L est vide.

Question 10 *En déduire une fonction* `traite_L fonc_f fonc_n` *qui effectue le traitement en largeur d'abord des arbres de L. On utilisera les fonctions élémentaires sur les arbres.*

```
let rec traite_L fonc_f fonc_n = function
  | [] → () ;
  | (feuille a) : :q → begin
      (fonc_f a) ;
      (traite_L fonc_f fonc_n q) ;
    end ;
  | (noeud (g,x,d)) : :q → begin
      (fonc_n x) ;
      (traite_L fonc_f fonc_n (q@[g ;d])) ;
    end ; ;
```

Question 11 *En déduire une fonction* `largeur fonc_f fonc_n T` *qui effectue le traitement en largeur d'abord de T.*

```
let rec largeur fonc_f fonc_n T =
  traite_L fonc_f fonc_n [T] ; ;
```

Question 12 *Tester votre fonction en prenant pour* `fonc_f` *et* `fonc_n` *des simples foncitons d'affichage, de façon à imprimer les étiquettes des noeuds de l'arbre en largeur d'abord.*

```
let affiche_f n =
  begin
    print_string (string_of_int n) ;
    print_newline () ;
  end ; ;

let affiche_n n =
  begin
    print_string (string_of_float n) ;
    print_newline () ;
  end ; ;

let t1 = noeud (
   noeud ( feuille 2, 3.5, feuille 5),
   4.2,
   feuille 8) ; ;

largeur affiche_f affiche_n t1 ; ;
```

Question 13 *Quelle est la complexité de la fonction* `largeur`*, en nombre d'opérations élémentaires sur la liste L (insertion, lecture en tête) ? Quelle serait sa complexité si la liste L était remplacée par une file de priorité (où l'insertion en queue est faite en temps constant) ?*

Chaque noeud de l'arbre est extrait exactement une fois de la liste L, et il y a une insertion dans L pour chaque sous-arbre de `T`, c'est à dire pour chaque noeud de `T`. Si l'insertion en queue dans `L` prend un temps linéaire, la complexité de la fonction `largeur` est donc $O(n^2)$ tandis que si elle prend un temps constant, sa complexité est $O(n)$.

Question 14 *Donner un encadrement du nombre de noeuds n d'un arbre quasi-complet en fonction de sa hauteur h.*

$$2^h \leq n \leq 2^{h+1} - 1$$

Question 15 *Exprimer le rang du k-ième sommet du d-ième étage dans un parcours en largeur d'abord. Montrer qu'il y a $i-1$ sommets entre un sommet d'indice i et son fils gauche.*

Le d-ième étage comprend 2^{d-1} sommets (sauf le dernier étage), donc le k-ième sommet du d-ième étage est d'indice :

$$2^0 + 2^1 + \ldots 2^{d-2} + k = 2^{d-1} + (k-1)$$

Les sommets entre un sommet d'indice $i = 2^{d-1} + (k-1)$ et son fils gauche (s'il existe) sont donc les $2^{d-1} - k$ successeurs de l'étage d, puis $2(k-1)$ sommets de l'étage $d+1$ (les fils des prédécesseurs de l'étage d). Cela fait en tout $i-1$ sommets.

Question 16 *Quel est l'indice des fils (gauche, droit) du sommet d'indice i ? Quel est celui de son parent ?*

Le fils gauche a, d'après la question précédente, pour indice $2i$ et le fils droit a pour indice $2i+1$. On en tire aisément que le parent d'un sommet d'indice i a pour indice $\lfloor i/2 \rfloor$.

Question 17 *Étant donné le tableau D représentant le parcour sen largeur d'un rabre T, écrire une fonction* `tree_of_array` *qui renvoie l'arbre T, de type* `'n arbre`*.*

```
let rec aux d i =
  let n = (vect_length d) in
  if (2*i > (n-1)) then
    noeud (vide, d.(i),vide)
  else let gauche = (aux d (2*i)) in
  if (2*i+1 > n) then
    noeud (gauche,d.(i), vide)
  else
    let droite = (aux d (2*i+1)) in
    noeud (gauche, d.(i), droite) ; ;
```