Option Informatique
Complexité et arbres

Corrigé

20 octobre 2006

1 Le retour de Fibonacci

Question 1 *Quelle valeur suffit-il de donner initialement à n et p pour avoir H vraie ?*

$n = \max(a, b)$ et $p = \min(a, b)$.

Question 2 *On suppose $p = 0$. Quel est, dans chaque cas, le PGCD de n et p ?*

Si $p = 0$, alors $n \wedge p = n$.

Question 4 *En déduire une écriture itérative puis récursive multiple de la fonction* `pgcd(a,b)`*. Justifier de son arrêt et de sa validité.*

```
let gcd a b =
  let n = ref (max a b)
    and p = ref (min a b) and r = ref 0 in
  if (!p=0) then
    n
  else begin
    while !p<>0 do
      r := (!n mod !p);
      n := !p;
      p := !r
    done;
    n;
  end;;
```

```
let rec gcd_rec a b =
  if a < b then
    (gcd_rec b a)
  else
    if b = 0 then
      a
    else
      (gcd_rec b (a mod b));;
```

Question 5 *On suppose que $a \geq b > 0$ Établir que si* `pgcd(a,b)` *fait n appels récursifs avec $n \geq 1$ alors $a \geq f_n$ et $b \geq f_{n-1}$.*

Montrons que si $a \geq b > 0$ et si `gcd_rec(a,b)` fait $n \geq 1$ appels récursifs, alors $a \geq f_n$ et $b \geq f_{k-1}$ par récurrence sur n. On fonde l'induction pour $n = 1$ en disant qu'alors $b \geq 1 = f_0$ et $a \geq 1 = f_1$.

Comme $b > (a \texttt{ mod } b)$, dans chaque appel récursif l'hypothèse selon laquelle $a \geq b$ est conservée.

Supposons que la propriété est vraie si $n - 1$ appels sont effectués : puisque $n > 0$ on a $b \geq 1$ et `gcd_rec(a,b)` appelle `gcd_rec(b, a mod b)` récursivement, qui effectue à son tour $n - 1$ appels récursifs. On a donc $b \geq f_n$ par hypothèse de récurrence, et $a \texttt{ mod } b \geq f_{n-1}$. On a $b + (a \texttt{ mod } b) = b + (a - \lfloor a/b \rfloor b) \leq a$.

On en tire ainsi $a \geq f_{k-1} + f_k$, d'où le résultat.

Question 6 *Montrer que $f_n \geq (\frac{1+\sqrt{5}}{2})^{n-1}$*

Ceci se montre sans difficulté par récurrence sur n. On rappelle la formle de Binet :

$$f_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}$$

Question 7 *En déduire la complexité de la version récursive est en $O(\ln b)$. Que dire de la complexité de la version itérative ?*

On tire le résultat des deux questions précédentes. La version itérative a la même complexité que la version récursive, si l'on considère le nombre d'affectations à la place du nombre d'appels récursifs comme mesure de complexité.

2 Parcours en largeur d'abord

Question 8 *Donner le type d'un arbre binaire strict correspondant à cette définition.*

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

Question 9 *Que doit valoir L initialement ?*

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

Question 10 *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 11 *Quand arrête-t-on le traitement ?*

On arrête le traitement quand la liste L est vide.

Question 12 *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 13 *En déduire une fonction* `largeur fonc_f fonc_n T` *qui effecture le traitement en largeur d'abord de T.*

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

Question 14 *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 15 *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 16 *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 17 *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 18 *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 19 *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) ; ;
```