

TD de *Logique et Circuits* n° 11
(Correction)

Induction sur les arbres

I) Schéma d'induction

Exercice 1 On considère les arbres binaires de type I étiquetés sur les entiers, définis par :

`type arbre = F | N of int * arbre * arbre;;`

Le schéma de définition inductive sur ces arbres vu en cours est :

$$f(a) = g(a, f(\text{fils_gauche}(a)), f(\text{fils_droit}(a)))$$

Pour chaque fonction Ocaml de la liste suivante, donner l'équation correspondante:

1. `let rec taille a =
 match a with
 F -> 0
 |N(n,g,d)-> 1 + taille g + taille d;;`
2. `let rec hauteur a =
 match a with
 F -> 0
 |N(n,g,d)-> 1 + max (hauteur g) (hauteur d);;`
3. `let rec chemins a =
 let rec ajoute x l = match l with
 []-> []
 |h::k-> (x::h):: (ajoute x k) in
 match a with
 F -> [[]]
 |N(n,g,d)->
 let a_gauche = ajoute 'g' (chemins g)
 and a_droite = ajoute 'd' (chemins d) in
 a_gauche @ a_droite;;`

(commencer par executer à la main la fonction `chemins`, par exemple sur `N(N(F,F),F)`).

Correction :

1. $g(a,n,m) = 1 + n + m$
 $f(F) = 0$
2. $g(a,n,m) = 1 + \max(n,m)$
 $f(F) = 0$
3. $g(a,l_1,l_2) = f_1(l_1) @ f_2(l_2)$
 $f_1(l) = g_1(l, f_1(h_1(l)))$
 $g_1(h :: l', l'') = ('g' :: h) :: l''$
 $h_1(h :: l) = l$
 $f_1([]) = []$
(idem pour f_2)
 $f(F) = [[]]$

II) Arbres binaires de recherche

Un arbre binaire de recherche (ABR) est un élément du type **arbre**, a , tel que, pour tout noeud $Arb(n,g,d)$ de a , toutes les étiquettes de g sont plus petites ou égales à n et toutes celles de d , sont plus grandes que n .

L'intérêt principal de cette définition est que la complexité du problème de la recherche d'une étiquette dans un ABR a de taille n est $O(\log_2 n)$, à condition que a soit *équilibré*. Intuitivement, un ABR est d'autant plus équilibré que le rapport entre sa taille et sa hauteur est grand. Il existe plusieurs définitions non équivalentes d'arbre équilibré. Dans la suite on s'intéresse aux 3 définitions suivantes. Un arbre binaire a est équilibré si:

1. Pour toute paire c_1, c_2 de chemins dans a , **longueur** $c_1 \leq 2$ **longueur** c_2 .
2. Pour tout noeud $Arb(e, a_1, a_2)$ de a , $|\mathbf{hauteur} a_1 - \mathbf{hauteur} a_2| \leq 1$.
3. Pour tout chemin c dans a , **longueur** $c = \mathbf{hauteur} a$ ou **longueur** $c = \mathbf{hauteur} a - 1$.

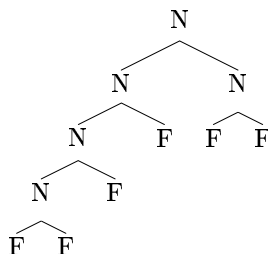
Pour $i = 1, 2, 3$ on dit qu'un arbre est i -équilibré s'il satisfait la contrainte i ci-dessus. Le but de cet exercice est de montrer que tout arbre $(i + 1)$ -équilibré est aussi i -équilibré, et que ces trois notions donnent, dans un ABR, la même complexité.

Exercice 2

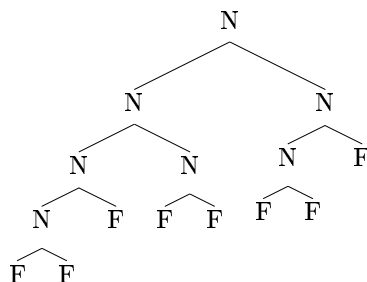
Montrer que les trois notions sont distinctes, c'est-à-dire, pour $i = 1, 2$, donner un exemple d'arbre binaire i -équilibré mais pas $(i + 1)$ -équilibré.

Correction :

1 et non 2:



2 et non 3:



Correction :

Exercice 3 Montrer par induction que si un arbre 2-équilibré a est de hauteur h , alors pour tout chemin c dans a on a **longueur** $c \geq h/2$. En déduire que si un arbre est 2-équilibré, alors il est 1-équilibré.

Indication. Pour $a = N(e, g, d)$, on pourra sans perte de généralité supposer $c = 'g' \cdot c'$, et considérer les deux cas **hauteur**(g) = $h - 1$ et **hauteur**(g) = $h - 2$.

Correction : Soit a 2-éq., s'il est F ok. S'il est $N(i,g,d)$, g et d sont 2-éq., et donc pour tout chemin c dans g $long(c) \geq h(g)/2$.

Soit $'g'c$ un chemin dans a , avec c chemin dans g . Vu que a est 2-éq., $h(g) = h - 1$ ou $h(g) = h - 2$.

Si $h(g) = h - 1$ alors, alors

$$long('g'c) = long(c) + 1 \geq (h - 1)/2 + 1 \geq h/2$$

Si $h(g) = h - 2$ alors, alors

$$long('g'c) = long(c) + 1 \geq (h - 2)/2 + 1 = h/2$$

Donc si a est 2-éq. et c, c' sont deux chemins dans a , alors $h(a)/2 \leq long(c), long(c') \leq h(a)$, donc $long(c) \leq 2long(c')$, i.e. a est 1-éq.

Exercice 4 Montrer que si un arbre a est 3-équilibré, alors il est 2-équilibré. Raisonner par l'absurde, en supposant qu'un noeud de a contredit la propriété 2.

Correction : Supposons que a soit 3-éq. et qu'il existe un noeud $a' = N(i,g,d)$ de a tel que $|h(g) - h(d)| > 1$. Sans perte de généralité, supposons $h(g) < h(d) - 1$. Soit c le chemin de a' dans a , c' un chemin quelconque dans g et c'' un chemin dans d tel que $long(c'') = h(d)$. On a $long(cc') \leq long(c) + h(g) < long(c) + h(d) - 1 = long(cc'') - 1$, i.e. $|long(cc') - long(cc'')| > 1$.

Exercice 5 Montrer que si tous les chemins d'un arbre ont même longueur h , alors cet arbre est de taille $2^h - 1$. En déduire que si un arbre 3-équilibré est de hauteur h , alors sa taille n est comprise entre les bornes suivantes :

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

En déduire que la complexité de la recherche dans un ABR 3-équilibré de taille n est en $O(\log_2 n)$.

Correction : 1. On prouve par induction sur h que l'arbre complet de hauteur h a 2^h feuilles et $2^h - 1$ noeud.

2. On remarque que un arbre 3-équilibré de hauteur h est complet jusqu'à $h - 1$, et a au moins un noeud supplémentaire a hauteur $h - 1$.

3. On a $h - 1 \leq \log_2 n \leq h$, donc $h \leq \log_2 n + 1$. Comme la recherche dans un ABR coûte au plus h , on en déduit la complexité $O(\log_2 n)$.

Exercice 6 Déduire de l'exercice précédent et de la définition 1 que si un arbre a 1-équilibré est de hauteur h , alors sa taille n est comprise entre les bornes suivantes :

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

En déduire que la complexité de la recherche dans un ABR 1-équilibré de taille n est en $O(\log_2 n)$. Conclure.

Correction :

Vu que a est 1-éq., tous ses chemins ont longueur au moins $h/2$, donc a est complet jusqu'au niveau $h/2$, et il a au moins un chemins de longueur h . Au total, $2^{h/2} - 1 + h/2 \leq \text{taille } a$.

On prouve que $hauteur(a) \leq 2\log_2(\text{taille}(a) + 1)$. Suppose $hauteur(a) > 2\log_2(\text{taille}(a) + 1)$ alors $taille(a) \geq 2^{hauteur(a)/2} - 1 + hauteur(a)/2 > 2^{\log_2(\text{taille}(a)+1)} - 1 + \log_2(\text{taille}(a) + 1) = \text{taille}(a) + \log_2(\text{taille}(a) + 1) \geq \text{taille}(a)$

Le nombre de comparaisons nécessaires pour vérifier si un entier appartient à l'ensemble des étiquettes d'un ABR 1-éq de taille n est au plus $hauteur(a) \leq 2\log_2(n + 1)$, la complexité de la recherche est donc $O(\log_2(n))$.

Conclusion: Les trois criteres sont équivalents du point de vue de la complexité de la recherche (donc il faudrait preferer 1 qui est plus faible).