Feuille de TD n°4 : arbres binaires

MPSI Lycée Clemenceau : option informatique

Mai 2025

Exercice 1 : Ecrire une fonction `test_squelette` qui teste l'égalité du "squelette" de deux arbres binaires.

Correction : On considère donc le type suivant :

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

Il suffit alors récursivement les fils d'un noeud avec égalité si on a deux feuilles.

```
rec test_squelette a b = match (a,b) with
    | F , F -> true
    | F , _ -> false
    | _ , F -> false
    | N (g1,d1) , N (g2,d2) -> (test_squelette g1 g2) && (test_squelette d1 d2);;
```

Exercice 2 : La numérotation de Sosa-Stradonitz d'un arbre binaire strict, utilisée notamment en généalogie, consiste à attribuer un numéro à chaque noeud ou feuille d'un arbre binaire strict, en suivant les règles suivantes :

- le numéro de la racine est égal à 1 ;
- si un noeud est de numéro n, son fils gauche se verra attribuer le numéro 2n et son fils droit le numéro 2n + 1.

On définit le type suivant :

```
type 'a arbre = Feuille of 'a | Noeud of 'a * 'a arbre * 'a arbre ;;
```

Ecrire une fonction de type `'a arbre -> (int *'a) arbre` qui ajoute à chaque noeud ou feuille son numéro.

Correction : on utilise simplement une fonction auxiliaire qui remplace les étiquettes avec la règle donnée.

```
let numero a =
    let rec aux a n = match a with
         | Feuille i -> Feuille (n,i)
         | Noeud( i , fg, fd) -> Noeud( (n,i), (aux fg (2*n)), (aux fd (2*n+1)));
    in aux a 1;;
```

Remarquons au passage que le parcours hiérarchique d'un arbre correspond à un parcours par numérotation de Sosa croissante

Exercice 3 : On définit le type suivant :

```
type arbre = Nil | Noeud of int * arbre * arbre ;;
```

Écrire une fonction de type `arbre -> int -> int list` qui, à partir d'un arbre binaire et d'un entier n, calcule la liste de toutes les étiquettes des noeuds de la profondeur n de l'arbre.

Correction : voici une première version

```
let rec profondeur a n  = match (a,n) with
    | Nil ,  _ -> []
    | (Noeud (k, _, _)) , 0 -> [k]
    | (Noeud (_, fils_g , fils_d)), n -> let lst_g = profondeur fils_g (n-1)
                                         and lst_d = profondeur fils_d (n-1)
                                         in lst_g @ lst_d ;;
```

mais cette méthode à l'inconvénient d'être coûteuse, ceci à cause de l'utilisation répétée de l'opérateur de concaténation des listes, **qui n'est pas de coût constant**.

Considérons par exemple le cas d'un arbre binaire complet (si la profondeur est h alors le nombre de feuilles est 2^h) : le nombre de noeuds à la profondeur h est égal à $n = 2^h$, et le nombre c_h d'insertion en tête de liste qu'effectue la fonction précédente vérifie la relation : $c_h = 2c_{h-1} + 2^{h-1}$. Cette équation se résout en : $c_h = (h + 2c_0)2^{h-1} = (h + 2)2^{h-1}$.

Ainsi, le nombre d'insertion réalisées pour former cette liste de n éléments est équivalent à $\frac{1}{2}n\log_2(n)$.

Il est possible de n'utiliser que n insertions en utilisant un accumulateur. On commence par le fils droit afin d'avoir les éléments dans la liste dans le bon ordre :

```
let profondeur a n  =
      let rec aux acc a n  = match (a,n) with
           | Nil, _ -> acc
           | (Noeud (k, _, _)), 0 -> k::acc
           | (Noeud (_, fils_g , fils_d)), n -> let lst_d = aux acc fils_d (n-1)
                                                in aux lst_d fils_g (n-1)
      in aux [] a n ;;
```

Exercice 4 : Le nombre de Strahler d'un arbre binaire strict est une mesure de la complexité de cet arbre ; il est défini de la manière suivante :

- le nombre de Strahler d'une feuille est égal à 1 ;
- si i et j désignent les nombres de Strahler des fils gauche et droit d'un noeud, alors le nombre de Stralher de ce dernier sera égal à $max(i, j)$ si $i \neq j$, et à $i + 1$ si $i = j$.

On définit le type suivant : `type arbre = Feuille | Noeud of arbre * arbre ;;`

1) Définir une fonction de type `arbre -> int` qui détermine le nombre de Strahler d'un arbre binaire strict.

Correction : il suffit de suivre la définition du nombre de Strahler

```
let rec nb_strahler ar = match ar with
                | Feuille -> 1
                | Noeud(a,b) -> let i = nb_strahler a and j = nb_strahler b in

                                  if i = j then i+1 else max i j;;
```

2) Quels sont, pour une hauteur donnée, les arbres de complexité maximale ? de complexité minimale ?

Correction : Commençons par prouver qu'un arbre de hauteur h a un nombre de Strahler n inférieur ou égal à $h+1$, avec égalité si et seulement s'il est complet :
- le cas $h = 0$ est immédiat : l'arbre n'est constitué que d'une feuille, donc $n = 1$, et cet arbre est complet.
- Si $h > 1$, supposons le résultat acquis pour tout arbre de hauteur inférieure ou égale à $h-1$, et considérons un arbre de hauteur h.
Sa racine possède deux fils, l'un de hauteur $h-1$ et l'autre de hauteur inférieure ou égale à $h-1$. Par hypothèse de récurrence, les nombres de Strahler des deux fils sont inférieurs ou égaux à $(h-1)+1 = h$, donc le nombre de Strahler de la racine est inférieure ou égale à $h+1$.
De plus, pour qu'il y ait égalité, il est nécessaire que les deux fils aient un nombre de Strahler égal à h, ce qui, par hypothèse de récurrence, impose qu'ils soient tous deux complets et de hauteur $h-1$. On en déduit que l'arbre initial est lui aussi complet.
Montrons maintenant qu'un arbre de hauteur $h > 1$ a un nombre de Strahler supérieur ou égal à 2, avec égalité si et seulement si tous les noeuds à l'exception d'un seul possèdent une feuille et un noeud pour fils :
- le cas $h = 1$ ne concerne qu'un seul arbre, et son nombre de Strahler est égal à 2
- Si $h > 2$, supposons le résultat acquis au rang $h-1$, et considérons un arbre de hauteur h. L'un de ses deux fils est de hauteur $h-1$ donc par hypothèse de récurrence a un nombre de Strahler supérieur ou égal à 2 ; il en est donc de même de l'arbre initial. De plus, pour qu'il y ait égalité, il faut que ce fils ait un nombre de Strahler égal à 2, ce qui impose par hypothèse de récurrence que tous ses descendants à l'exception d'un seul aient un noeud et une feuille pour fils, et il faut aussi que l'autre fils ait un nombre de Strahler égal à 1, c'est à dire soit une feuille. Ceci achève de prouver le résultat demandé.

3) Les arbres de complexité minimale sont appelés des arbres filiformes.
Combien y-en-a-t'il pour une hauteur donnée ?
Écrire une fonction de type `int -> arbre list` qui détermine la liste de tous les arbres filiformes de hauteur donnée.

Correction : À tout arbre filiforme de hauteur h on peut associer un mot de $h-1$ lettres égales à G ou D : la première lettre indique si le fils gauche (G) ou droit (D) est un noeud, et les lettres suivantes forment le mot associé à ce fils. La correspondance étant bijective, il y a 2^{h-1} arbres filiformes de hauteur h. On génère la liste des arbres filiformes de hauteur h en suivant le principe énoncé ci-dessus.

```
let rec filiforme n = match n with
         | 0 -> []
         | 1 -> [Noeud (Feuille , Feuille)]
         | h -> let lst = filiforme (h-1) in
                let lst1 = List.map ( function arb -> Noeud (Feuille , arb)) lst
                and lst2 = List.map ( function arb -> Noeud (arb, Feuille)) lst in
                      lst1 @ lst2 ;;
```

Autre fonction pour éviter l'utilisation de la concaténation couteuse et l'utilisation de la fonction `List.map` qui répète l'utilisation d'une fonction sur tous les éléments d'une liste.

```
let  filiforme n = let rec iter  lst = match lst with
                        | [] -> []
                        | a::q -> Noeud(Feuille,a)::Noeud(a,Feuille)::(iter q) in
                         let rec aux n =  match n with
                                     | 0 -> []
                                     | 1 -> [Noeud(Feuille,Feuille)]
```

```
                                    | n -> let ll = aux (n-1) in iter ll
                        in aux n;;
```

Exercice 5 : On définit le type d'arbre binaire strict suivant :

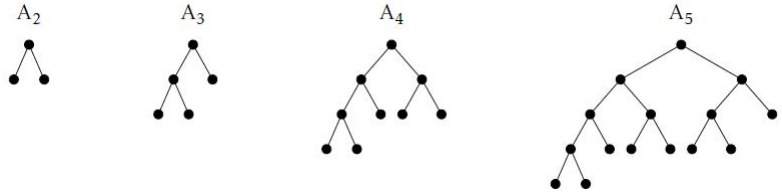
```
type arbre = Feuille | Noeud of arbre * arbre ;;
```

et on définit par récurrence la suite des arbres de Fibonacci en convenant que :

— A_0 et A_1 sont des feuilles ;

— si $n > 2$, A_n est l'arbre binaire dont le sous-arbre gauche est A_{n-1} et le sous-arbre droit, A_{n-2}.

a) Dessiner les arbres A_2, A_3, A_4 et A_5, puis définir une fonction Caml qui génère les arbres de Fibonacci.



Ceux-ci peuvent être générés à l'aide de la fonction suivante même si celle-ci n'est pas performante pour les raisons qu'on connait de complexité :

```
let rec fibonacci n = match n with
              | 0 -> Feuille
              | 1 -> Feuille
              | n -> let fils_g = fibonacci (n-1) and fils_d = fibonacci (n-2)
                     in Noeud (fils_g , fils_d) ;;
```

b) Quelle est la hauteur de A_n ?

Montrer que les arbres de Fibonacci sont H-équilibrés, c'est à dire qu'en tout noeud de l'arbre les hauteurs des sous-arbres gauche et droit différent au plus de 1.

c) Déterminer le nombre de feuilles et de noeuds de A_n à l'aide des nombres de la suite de Fibonacci.

d) Quel est le nombre de Strahler de l'arbre A_n ?

Correction : Montrons par récurrence sur $n > 1$ que A_n est un arbre de hauteur $n-1$:
- c'est le cas pour $n = 1$ et $n = 2$;
- supposons $n > 3$ et le résultat acquis aux rangs $n-1$ et $n-2$. Par définition de A_n, sa hauteur est égale au maximum des hauteurs de A_{n-1} et A_{n-2} plus 1, c'est à dire : $max(n-2; n-3)+1 = n-1$, ce qui prouve le résultat au rang n.

Montrons alors, toujours par récurrence sur $n > 1$, que A_n est un arbre H-équilibré :
- c'est bien le cas des arbres A_1 et A_2 ;
- supposons $n > 3$ et le résultat acquis aux rangs $n-1$ et $n-2$. Par hypothèse de récurrence, tous les noeuds hormis peut-être la racine ont des sous-arbres gauches et droits dont les hauteurs diffèrent au plus de 1. Il reste à examiner la racine, dont le fils gauche, A_{n-1}, est de hauteur $n-2$ et le fils droit, A_{n-2}, de hauteur $n-3$. L'arbre A_n est donc bien H-équilibré.

Notons f_n le nombre de feuilles de l'arbre A_n. Nous avons $f_0 = f_1 = 1$ et $f_n = f_{n-1} + f_{n-2}$ pour $n > 2$, donc f_n est égal au n^e nombre de Fibonacci. Et sachant qu'il s'agit d'arbres binaires stricts, le nombre de noeuds de A_n est égal à $f_n - 1$.

En utilisant les fonctions strahler et fibonacci il est facile de postuler que le nombre de Strahler de A_n est égal à $\lfloor \frac{n}{2} \rfloor + 1$. Prouvons-le par récurrence :

- le nombre de Strahler de A_0 et A_1 est égal à 1, ce qui est conforme à la formule que l'on cherche à prouver ;
- supposons donc $n > 2$ et le résultat acquis aux rangs $n-1$ et $n-2$. Deux cas sont à envisager : - si $n = 2p$ est pair, les nombres de Strahler de A_{n-1} et de A_{n-2} sont égaux à p, donc celui de A_n à $p + 1 = \lfloor \frac{n}{2} \rfloor + 1$.

- si $n = 2p + 1$ est impair, le nombre de Strahler de A_{n-1} est p+1 et celui de A_{n-2} est p, donc celui de A_n est $p + 1 = \lfloor \frac{n}{2} \rfloor + 1$.

Dans les deux cas, l'hypothèse de récurrence est vérifiée.