Chapitre 3

Correction des exercices

Exercice 1

```
let rec genere_complet = function
  | (-1) -> Nil
  | n    -> let f = genere_complet (n-1) in Node (f, f) ;;
```

```
let cheminement a =
  let rec aux acc = function
    | Nil             -> 0
    | Node (Nil, Nil) -> acc
    | Node (fg, fd)   -> aux (acc + 1) fg + aux (acc + 1) fd
  in aux 0 a ;;
```

La fonction **aux** utilise un accumulateur qui conserve la profondeur du nœud auquel il s'applique ; ainsi, arrivé à une feuille, on est en mesure de connaitre la longueur du chemin qui mène de la racine à cette feuille.

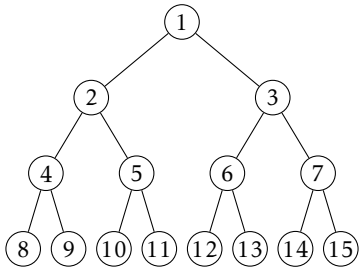
Un arbre complet de hauteur n possède 2^n feuilles, toutes situées à la profondeur n. Le cheminement est donc égal à $n2^n$.

Exercice 2 *La numérotation de Sosa-Stradonitz est une méthode de numérotation des ancêtres pour les généalogies ascendantes. Elle fut mise au point par le généalogiste espagnol Jerónimo de Sosa en 1676, reprenant en cela la méthode publiée à Cologne en 1590 par Michael Eyzinger. Cette méthode fut définitivement popularisée en 1898 par Stephan Kekulé von Stradonitz, célèbre pour son ouvrage recensant les tableaux d'ascendance des souverains européens de son époque.*

```
let sosa a =
  let rec aux k = function
    | Feuille a         -> Feuille (k, a)
    | Noeud (a, fg, fd) -> Noeud ((k, a), aux (2*k) fg, aux (2*k+1) fd)
  in aux 1 a ;;
```

La fonction auxiliaire **aux** prend deux arguments : un entier k et un arbre, l'entier k indiquant le numéro associé à la racine de l'arbre.

On peut observer que le parcours hiérarchique d'un arbre correspond à un parcours par numérotation de Sosa croissante :



Exercice 3 Voici une première solution :

```
let rec profondeur a n = match a with
  | Nil                        -> []
  | Noeud (k, _, _) when n = 0 -> [k]
  | Noeud (_, fg, fd)          -> let lg = profondeur fg (n-1)
                                  and ld = profondeur fd (n-1)
                                  in lg @ ld ;;
```

mais cette démarche à l'inconvénient d'être coûteuse, 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 : le nombre de nœuds 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 : $c_h = 2c_{h-1} + 2^{h-1}$, équation qui 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* :

```
let profondeur a n =
  let rec aux acc n = function
    | Nil                          -> acc
    | Noeud (k, _, _) when n = 0 -> k::acc
    | Noeud (_, fg, fd)            -> let acc1 = aux acc fd (n-1)
                                      in aux acc1 fg (n-1)
  in aux [] n a ;;
```

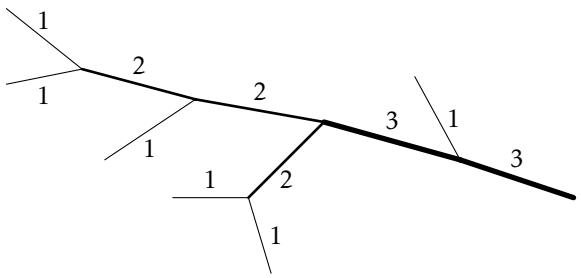
Exercice 4 Les feuilles correspondent aux nœuds dont la liste des fils est vide. D'où la fonction :

```
let rec nb_feuilles = function
  | Noeud (_, [])  -> 1
  | Noeud (_, lst) -> it_list (fun a b -> a + (nb_feuilles b)) 0 lst ;;
```

Le calcul de la hauteur se déroule de la même façon :

```
let rec hauteur = function
  | Node (_, [])  -> 0
  | Node (_, lst) -> 1 + it_list (fun a b -> max a (hauteur b)) 0 lst ;;
```

Exercice 5 *Arthur Newell Strahler est un professeur de sciences de la Terre de l'université Columbia qui a développé l'indicateur qui porte son nom pour classer un réseau hydrographique en fonction de la puissance de ses affluents. Cet indicateur est aussi utilisé en théorie de la compilation pour calculer le nombre de registres nécessaires au calcul d'une expression arithmétique.*
Un exemple de réseau hydrographique suivant la classification de Strahler (dans ce contexte, ce sont les arêtes plutôt que les sommets qui sont étiquetés) :




```
let rec strahler = function
  | Feuille        -> 1
  | Noeud (fg, fd) -> let (i, j) = (strahler fg, strahler fd) in
                      if i = j then i + 1 else max i j ;;
```

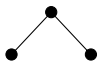
Commençons par prouver qu'un arbre de hauteur h a un nombre de Strahler n inférieur ou égal à $h+1$, avec égalité si et seulement s'il est complet :

- le cas $h = 0$ est évident : l'arbre n'est constitué que d'une feuille, donc $n = 1$, et cet arbre est complet.
- Si $h \geqslant 1$, 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 $h+1$.

En outre, 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 $h-1$. En conséquence de quoi l'arbre initial est lui aussi complet.

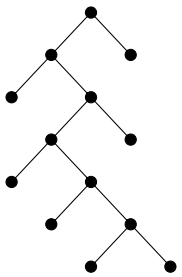
Montrons maintenant qu'un arbre de hauteur $h \geqslant 1$ a un nombre de Strahler supérieur ou égal à 2, avec égalité si et seulement si tous les nœuds à l'exception d'un seul possèdent une feuille et un nœud pour fils :

- le cas $h = 1$ ne concerne qu'un seul arbre, et son nombre de Strahler est égal à 2 :



- Si $h \geqslant 2$, 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 nœud 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.

Un exemple d'arbre filiforme :



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 nœud, et les lettres suivantes forment le mot associé à ce fils. Par exemple, à l'arbre représenté ci-dessus est associé le mot GDGDD. 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.

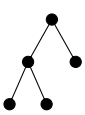
```
let rec filiforme = function
  | 0 -> []
  | 1 -> [Noeud (Feuille, Feuille)]
  | h -> let lst = filiforme (h-1) in
         let lst1 = map (function arb -> Noeud (Feuille, arb)) lst
         and lst2 = map (function arb -> Noeud (arb, Feuille)) lst in
         lst1 @ lst2 ;;
```

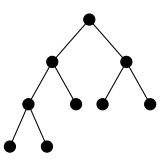
Même si cela ne s'impose pas ici, les amateurs pourront préférer utiliser la fonctionnelle **`list_it`** :

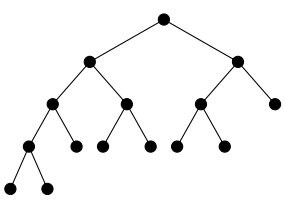
```
let rec filiforme = function
  | 0 -> []
  | 1 -> [Noeud (Feuille, Feuille)]
  | h -> let lst = filiforme (h-1) in
         list_it (fun a b -> (Noeud (Feuille, a))::(Noeud (a, Feuille))::b) lst [] ;;
```

Exercice 6 Les premiers arbres de Fibonacci sont les suivants :

A_2 

A_3 

A_4 

A_5 

Ceux-ci peuvent être générés à l'aide de la fonction suivante (peut efficace, mais qu'on ne cherchera pas à améliorer pour l'instant) :

```
let rec fibonacci = function
  | 0 | 1 -> Feuille
  | n     -> let fg = fibonacci (n-1) and fd = fibonacci (n-2)
             in Noeud (fg, fd) ;;
```

Montrons par récurrence sur $n \geqslant 1$ que A_n est un arbre de hauteur $n-1$:

- c'est le cas pour $n = 1$ et $n = 2$;
- supposons $n \geqslant 3$ 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 $n \geqslant 1$, que A_n est un arbre H-équilibré :

- c'est bien le cas des arbres A_1 et A_2 ;
- supposons $n \geqslant 3$ et le résultat acquis aux rangs $n-1$ et $n-2$. Par hypothèse de récurrence, tous les nœuds 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 $n \geqslant 2$, donc f_n est égal au n^{e} nombre de Fibonacci [1]. Et sachant qu'il s'agit d'arbres binaires stricts, le nombre de nœuds 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 $\left\lfloor \frac{n}{2} \right\rfloor + 1$. Prouvons-le par récurrence :

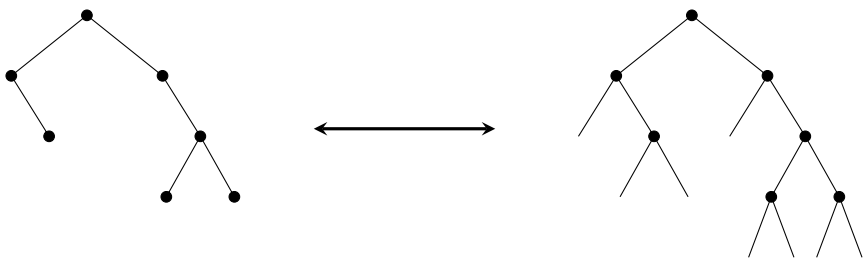
- le nombre de Strahler de A_0 et A_1 est égal à 1, ce qui est conforme à la formule que l'on cherche à prouver ;
- supposons donc $n \geqslant 2$ et le résultat acquis aux rangs $n-1$ et $n-2$. Deux cas sont à envisager :
  - si $n = 2p$ est pair, les nombres de Strahler de A_{n-1} et de A_{n-2} sont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

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

Exercice 7 Un arbre binaire à $n+1$ nœuds possède une racine, un fils gauche ayant k nœuds, et un fils droit ayant $n-k$ nœuds, avec $k \in [\![0, n]\!]$. L'entier k étant fixé, il existe $c_k c_{n-k}$ arbres binaires de ce type, et donc :

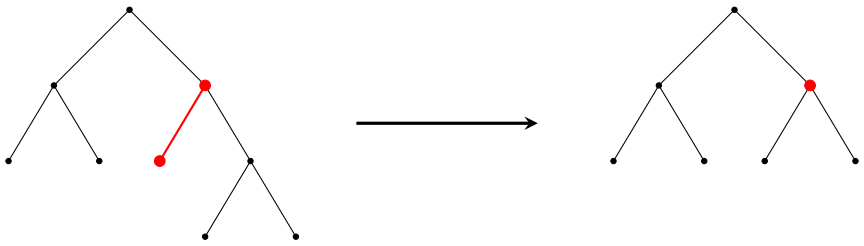
$$c_{n+1} = \sum_{k=0}^{n} c_k c_{n-k}.$$

Tout arbre binaire peut être complété en un arbre binaire strict en ajoutant des feuilles à tous les nœuds n'ayant pas déjà deux fils :

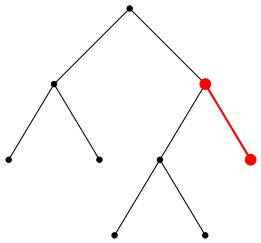


1. Ou au $(n+1)^{\text{e}}$ suivant la convention choisie pour définir cette suite.

On obtient ainsi un arbre binaire strict ayant toujours n nœuds, et donc $n+1$ feuilles.
Réciproquement, tout arbre binaire strict à $n+1$ feuilles, une fois « effeuillé », n'a plus que n nœuds. On établit donc ainsi une correspondance bijective entre les arbres binaires à n nœuds et les arbres binaires stricts à $n+1$ feuilles.
Considérons maintenant l'une de ces $n+1$ feuilles, et supprimons-là ainsi que son père, le second fils prenant la place du père :



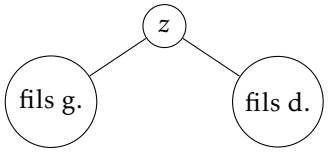
On établit ainsi une application entre l'ensemble $\mathscr{A}$ des couples formés d'un arbre strict à $n+1$ feuilles et l'une de celles-ci et l'ensemble $\mathscr{B}$ des couples formés d'un arbre strict à n feuilles et l'un de ses $2n-1$ sommets (nœud ou feuille).
Nous avons $|\mathscr{A}| = (n+1)c_n$ et $|\mathscr{B}| = (2n-1)c_{n-1}$.
Il reste à observer que tout élément de $\mathscr{B}$ possède deux antécédents par cette transformation, suivant que la feuille supprimée soit le fils gauche ou droite. Par exemple, l'arbre représenté plus haut possède comme second antécédent l'arbre suivant :



On peut donc affirmer que $|\mathscr{A}| = 2|\mathscr{B}|$, soit $(n+1)c_n = 2(2n-1)c_{n-1}$. Une fois cette égalité établie, prouver par récurrence que $c_n = \dfrac{1}{n+1}\binom{2n}{n}$ ne présente plus de difficulté.

Exercice 8 Considérons le premier ancêtre commun z à x et à y, et envisageons plusieurs cas :

- si z est différent de x et de y alors ces deux nœuds appartiennent, l'un au sous-arbre issu du fils gauche de z, et l'autre au sous-arbre issu du fils droit :



  et l'ordre d'apparition relative de x et de y est identique pour le parcours préfixe comme pour le parcours suffixe, ce qui n'est pas le cas ;
- si $z = y$, alors y précède x suivant l'ordre préfixe et lui succède suivant l'ordre suffixe, ce qui n'est pas non plus le cas.

On en déduit que $z = x$, ce qui prouve que x est bien un ascendant de y.

♦ ♦
♦