TD : Arbres

Exercice 1. *Arbre sans étiquette*. On définit le type arbre suivant : `type arbre = Vide | N of arbre * arbre ;;`

1. Rédiger une fonction `genere_complet` de type `int -> arbre` qui retourne un arbre complet de hauteur h (on convient que `Vide` a hauteur 0). Un arbre est complet si tous ses niveaux sont remplis au maximum. Quelle est sa complexité ?

```
# genere_complet 2 ;;
- : arbre = N (N (Vide, Vide), N (Vide, Vide))
```

2. On appelle *longueur de cheminement* d'un arbre la somme des profondeurs de chacune de ses feuilles. Rédiger une fonction `lg_cheminement` de type `arbre -> int` qui calcule la longueur de cheminement d'un arbre quelconque. À quoi est égal la longueur de cheminement d'un arbre complet de hauteur h ?

Exercice 2. La numérotation de *Sosa-Stradonitz* d'un arbre binaire entier, utilisée notamment en généalogie, consiste à attribuer un numéro à chaque nœud (interne ou feuille) d'un arbre binaire strict, en suivant les règles suivantes :

— le numéro de la racine est égal à 1 ;

— si un nœud est de numéro n, on attribue les numéros $2n$ et $2n + 1$ à ses fils gauche et droit.

On définit le type suivant : `type 'a arbre = F of 'a | N of 'a * 'a arbre * 'a arbre ;;`
Écrire une fonction `numerote_sosa` de type `'a arbre -> (int * 'a) arbre` qui ajoute à chaque nœud son numéro.

```
# numerote_sosa (N(8,N(5,N(4,F 2,F 0), N(6,N(3,F 1, F 7),F 10)), F 12)) ;;
- : (int * int) arbre =
N ((1, 8),
 N ((2, 5), N ((4, 4), F (8, 2), F (9, 0)),
  N ((5, 6), N ((10, 3), F (20, 1), F (21, 7)), F (11, 10))),
 F (3, 12))
```

Exercice 3. *Étiquette d'un arbre à une profondeur donnée*. On représente les arbres binaires à étiquettes entières comme suit : `type arbre = Vide | N of int * arbre * arbre ;;`.
Écrire une fonction `etiquettes_p` de type `arbre -> int -> int list` qui, à partir d'un arbre binaire et d'un entier p, calcule la liste de toutes les étiquettes des nœuds de la profondeur p de l'arbre.

```
# let a=N(8,N(5,N(4,Vide,Vide), N(6,N(3,Vide, Vide),Vide)), Vide) in etiquettes a 1, etiquettes a 2 ;;
- : int list * int list = ([5], [4; 6])
```

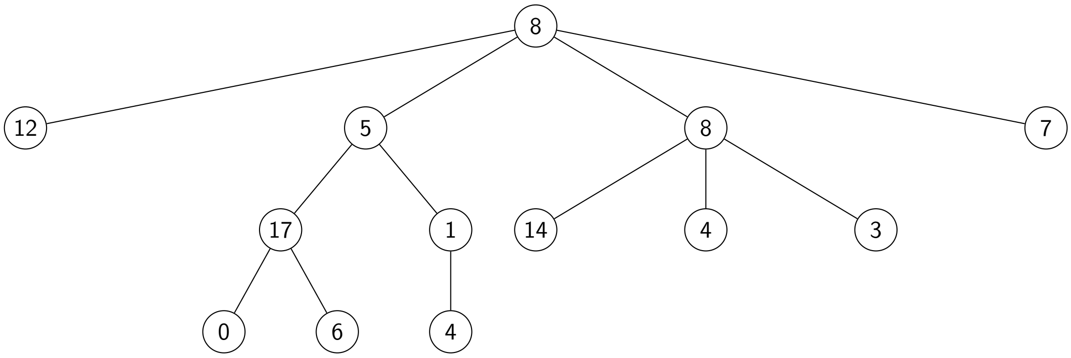
Rappel : `@` est la concaténation de listes.

Exercice 4. *Arbres généraux*. On représente les arbres généraux par le type suivant :
`type 'a arbre = N of 'a * ('a arbre list) ;;`
Rédiger les fonctions `nb_feuilles` et `hauteur` de type `'a arbre -> int` qui calculent respectivement le nombre de feuilles et la hauteur d'un tel arbre (un nœud est une feuille si la liste de ses fils est vide). On fera deux méthodes :

— avec `List.map` et une fonction annexe qui fait la somme (resp. calcule le maximum) d'une liste d'entiers ;

— avec deux fonctions récursives imbriquées.

Par exemple avec `a` l'arbre représenté comme suit :



Qu'on déclare ainsi :

```
let a=N (8,
   [N (12, []);
    N (5, [N (17, [N (0, []); N (6, [])]); N (1, [N (4, [])])]);
    N (8, [N (14, []); N (4, []); N (3, [])]);
    N (7, [])])
;;
```

On a :

```
# nb_feuilles a ;;
- : int = 8
# hauteur a ;;
- : int = 3
```

Exercice 5. *Nombre de Strahler d'un arbre.* Le nombre de Strahler d'un arbre binaire entier 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 s_g et s_d désignent les nombres de Strahler des fils gauche et droit d'un nœud, alors le nombre de Stralher de ce dernier sera égal à $\max(s_g, s_d)$ si $s_g \neq s_d$, et à $s_g + 1$ sinon.

On définit le type suivant : `type arbre = F | N of arbre * arbre ;;`

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

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

3. Les arbres de complexité minimale sont appelés des arbres filiformes. Combien y en-a-t'il pour une hauteur donnée ?

4. Écrire une fonction `enumere_fili` de type `int -> arbre list` qui détermine la liste de tous les arbres filiformes de hauteur donnée (on convient que `F` a hauteur 0).

```
# enumere_fili 3 ;;
- : arbre list =
[N (F, N (F, N (F, F))); N (F, N (N (F, F), F)); N (N (F, N (F, F)), F);
 N (N (N (F, F), F), F)]
```

Exercice 6. Soient x et y deux nœuds d'un même arbre. Montrer que y est un descandant de x (c'est-à-dire qu'il se trouve dans le sous-arbre enraciné en x) si et seulement si x est avant y dans le parcours préfixe de l'arbre et après dans le parcours suffixe (postfixe).