Arbres binaires en OCaml. Exercices en cours

Exercice 1. On utilise ici le type :

```
type 'a arbre = Nil | Node of ('a arbre * 'a * 'a arbre);;
```

Il modélise des arbres binaires. On rappelle qu'un *nœud* d'un arbre A est un sous-arbres non vide, une *feuille* un nœud dont les fils sont vides et un *nœud interne* un nœud qui n'est pas une feuille. Une *branche* est un chemin de la racine aux feuilles. La *profondeur* d'un nœud est la distance qui le sépare de la racine, celle d'un arbre est la plus grand profondeur de feuille. La profondeur de la racine étant zéro. La *taille* est le nombre de nœuds.

1. Représenter :

```
let a = let b = Node (Nil, 1, Nil) and c = Node (Nil, 2, Nil)
      and d = Node (Nil, 4, Nil) in let g = Node (b,4,c) and f=
      Node (Nil, 5, Node(d,3,Nil)) in Node(g,6,f);;
```

2. Implanter les grands classiques que sont **taille a** et
   hauteur a qui donne la plus grande profondeur de feuille.
   Donner la complexité de **hauteur**
3. **nb_feuilles** qui donne le nombre de feuilles.
4. **nb_internes** qui donne le nombre de nœuds. internes.
5. Donner la fonction **applique a m** de signature **'a arbre -> ('a -> 'a -> 'a) -> 'a** où a est un arbre et m un opérateur binaire associatif et commutatif. La fonction retourne le résultat de m appliqué à toutes les étiquettes de l'arbre (il faut donc que l'arbre ait au moins deux nœuds). Par exemple m peut être l'opérateur **max**.

Exercice 2. Avec le type d'arbres du cours, implanter la fonction **miroir** qui prend en paramètre un arbre et renvoie l'arbre miroir : l'arbre initial dont les fils gauches des nœuds deviennent des fils droits et réciproquement.

```
let bt = Node(Node(Nil,1,Node(Nil,2,Nil)),3,Node(Nil,4,Nil))
in miroir bt;;
  - : int arbre =
Node (Node (Nil, 4, Nil), 3, Node (Node (Nil, 2, Nil), 1, Nil))
```

Exercice 3. Dans cet exercice, on se donne le type :

```
type 'a tree = F of 'a | N of 'a tree * 'a * 'a tree;;
```

1. Peut-on représenter ainsi l'arbre vide ?
2. A quelle catégorie appartiennent les arbres ainsi modélisés ?

3. Écrire les fonctions **size : 'a tree -> int** et **height : 'a tree -> int** dont les noms se passent de commentaires.
4. Définissons le *nombre de Strahler* d'un arbre récursivement :
   — Si l'arbre est réduit à une feuille, son nombre de Strahler est 1 ;
   — Sinon
      — Si les deux fils ont des nombres de Strahler s, s' avec $s < s'$ alors son nombre de Strahler est s'.
      — Sinon si les deux fils ont même nombre de Strahler s alors son nombre de Strahler est $s + 1$.

   (a) Écrire le fonction **strahler : 'a tree -> int** qui calcule le nombre de Strahler d'un arbre.

   (b) Déterminer le nombre de Strahler d'un arbre parfait.

Exercice 4. Ecrire la fonction **path** de type **'a arbre -> 'a -> 'a list** qui renvoie la liste des étiquettes d'un chemin allant de la racine de **a** à un nœud d'étiquette **x**.

Avec :

```
# let a = let b = Node (Nil, 1, Nil) and c = Node (Nil, 2, Nil)
        and d = Node (Nil, 3, Nil) in let g = Node (b,4,c) and f=
        Node (Nil, 5, Node(d,6,Nil)) in Node(g,7,f) in
        path a 8;;
```

on obtient :

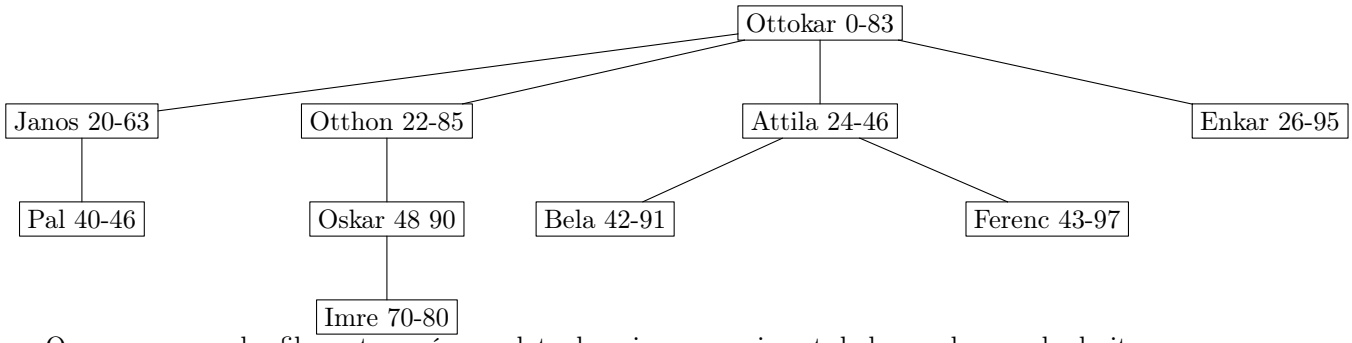
```
# path a 6;;
- : int list = [7; 5; 6]
# path a 8;;
- : int list = []
```

Exercice 5. Dans cet exercice, les arbres ne sont pas binaires. Il sont représentés par le type

```
type king = C of string * int *int * (king list);;
```

Un arbre représente tous les descendants mâles directs de la descendance directe du roi de Syldavie. Chaque étiquette est un triplet constitué du prénom, de la date de naissance et de celle de décès de ce descendant.

Par exemple considérons



On suppose que les fils sont rangés par date de naissance croissant de la gauche vers la droite. On suppose aussi qu'un fils né toujours avant la mort de son père (les rois de Syldavie ont la santé solide).

Écrire la fonction **rois** qui prend en paramètre un arbre généalogique et imprime dans l'ordre les noms des dépositaires du titre.

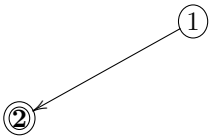
Par exemple, avec l'arbre ci-dessus, on doit obtenir Ottokar, Othon, oska, Bela et Ferrenc.

Un fils aîné, s'il n'est pas déjà mort au décès de son père doit régner et ses descendants sont prioritaires par rapport aux autres fils du roi et leurs descendants. En clair, on explore prioritairement les branches gauches.

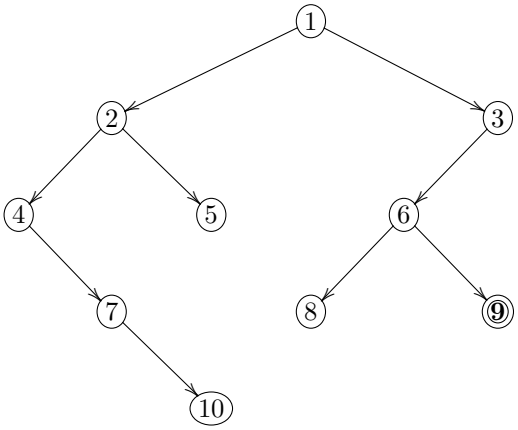
Exercice 6. Dans cet exercice on appelle « feuille la plus à droite » la feuille que l'on atteint en prenant systématiquement à droite lorsque cette option est possible.

Dans les exemples ci-dessous (nommés **arbrel** et **arbrell**) on a mis en gras l'étiquette de la feuille la plus à droite :

Arbre I



Arbre II

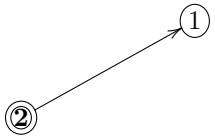


1. Écrire la fonction **rightLeaf (t :'a tree) : 'a** qui renvoie la feuille la plus à droite d'un arbre.

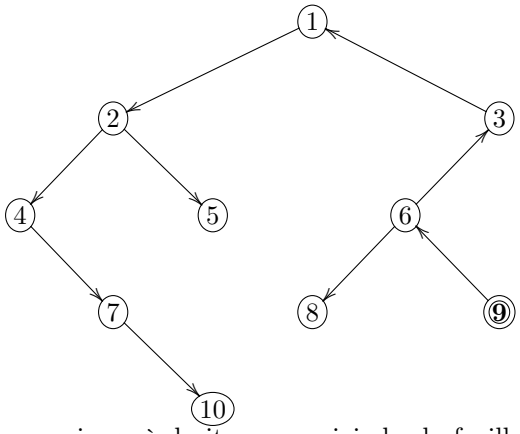
```
# let arbreII=
  let quatre = Node(Nil,4,Node(Nil,7,Node(Nil,10,Nil))) and
  six = Node(Node(Nil,8,Nil),6,Node(Nil,9,Nil)) in
  let trois = Node(six,3,Nil) and deux = Node(quatre,2,Node(Nil,5,Nil))
  in Node(deux,1,trois)
in rightLeaf arbreII;;
- : int = 9
# let arbreI = Node(Node(Nil,2,Nil),1,Nil) in
rightLeaf arbreI;;
  - : int = 2
```

2. On veut maintenant « renverser » l'arbre. Il s'agit de prendre pour nouvelle racine la feuille la plus à droite et d'inverser certaines flèches en conséquence. Il s'agit donc d'inverser certaines flèches dans l'arbre initial. Les arbres **arbrel** et **arbrell** deviennent :

Arbre I



Arbre II



Dans le cas pathologique où il n'y aurait aucun virage à droite pour rejoindre la feuille la plus à droite (cf **arbre I**), on convient que les nœuds du nouvel arbre ne contiennent que des descendants gauches.

Écrire la fonction **reverse (t :'a tree) :'a tree** qui renverse un arbre selon le principe ci-dessus.

```
# let arbreI = Node(Node(Nil,2,Nil),1,Nil) in
reverse arbreI;;
  - : int tree = Node (Node (Nil, 1, Nil), 2, Nil)
# let arbreII=
  let quatre = Node(Nil,4,Node(Nil,7,Node(Nil,10,Nil))) and
  six = Node(Node(Nil,8,Nil),6,Node(Nil,9,Nil)) in
  let trois = Node(six,3,Nil) and deux = Node(quatre,2,Node(Nil,5,Nil))
  in Node(deux,1,trois)
in
reverse arbreII;;
  - : int tree =
Node
 (Node (Node (Nil, 8, Nil), 6,
   Node
    (Node
      (Node (Node (Nil, 4, Node (Nil, 7, Node (Nil, 10, Nil))), 2,
        Node (Nil, 5, Nil)),
      1, Nil),
    3, Nil)),
 9, Nil)
```