université de BORDEAUX	**Année 2017/2018** — L2 Info/MI, UE 4TIN402U Algorithmique des structures de données arborescentes Mercredi 13 juin 2018, 9h–10h30 — Durée: 1h30 Documents: **non autorisés**	**Collège Sciences et Technologies**

Exercice 1 *Insertion à la racine dans les ABR*
On considère le type `'a tree` suivant :

```
type 'a tree = Empty | Bin of 'a * 'a tree * 'a tree
```

1. Écrire une fonction en `Ocaml` `bst_cut` qui prend en argument un élément `x` de type `'a` et un arbre binaire de recherche `t` de type `'a tree` telle que `bst_cut x t` retourne un couple composé de **deux** arbres binaires de recherche :
   - le premier contenant tous les éléments de `t` qui sont strictement plus petits que `x`,
   - le second contenant tous les éléments de `t` qui sont strictement plus grands que `x`.

   Ainsi par exemple, si on construit l'arbre `t` de la façon suivante :

   ```
   let leaf x = Bin(x, Empty, Empty)
   let left = Bin(10, leaf 5, Bin(15, leaf 12, Empty))
   let right = Bin(30, leaf 25, Bin(35, Empty, leaf 42))
   let t = Bin(20, left, right)
   ```

   alors après `let t1,t2 = bst_cut 27 t`, on pourra avoir comme valeur de `t1` :

   ```
   Bin (20,
    Bin (10, Bin (5, Empty, Empty), Bin (15, Bin (12, Empty, Empty), Empty)),
    Bin (25, Empty, Empty))
   ```

   et comme valeur de `t2` :

   ```
   Bin (30, Empty, Bin (35, Empty, Bin (42, Empty, Empty)))
   ```

2. En utilisant la fonction `bst_cut`, écrire une fonction `bst_insert` qui prend en argument
   - un arbre binaire de recherche `t` de type `'a tree`,
   - une valeur `x` de type `'a` qui n'apparaît pas comme clé dans l'arbre `t`,

   et qui renvoie l'arbre obtenu en insérant la clé `x` à la racine de l'arbre `t`, tout en respectant le fait que l'arbre renvoyé reste un arbre binaire de recherche. La clé de la racine de l'arbre renvoyé doit être `x`.

   Note. Vous pouvez faire cette question même si vous n'avez pas écrit la fonction `bst_cut`.

Exercice 2 *Arbres rouges et noirs*
Quel est l'arbre rouge et noir produit par l'algorithme d'insertion vu en cours lorsqu'on insère les entiers 7, 6, 5, 3, 4, 2, 1 dans cet ordre, à partir de l'arbre constitué d'une unique feuille ? Détailler chaque étape d'insertion.
Rappel. Les feuilles sont noires et n'ont pas de valeur. Vous pouvez donc, au choix, les représenter ou non.
Note. Lisez attentivement la suite d'entiers : elle n'est pas décroissante à cause de la position de 3 et 4.

Exercice 3 *Arbres équilibrés*
Pour un entier $h \geqslant 0$, on note $\mathsf{m}(h)$ le nombre ***minimal*** de nœuds dans un arbre AVL de hauteur h.

1. Calculer $\mathsf{m}(0)$, $\mathsf{m}(1)$, $\mathsf{m}(2)$ et $\mathsf{m}(3)$.
2. Trouver une relation de récurrence liant $\mathsf{m}(h+2)$, $\mathsf{m}(h+1)$ et $\mathsf{m}(h)$.
3. On définit la suite $(u_h)_{h\in\mathbb{N}}$ par $u_h = 1 + \mathsf{m}(h)$. Quelle relation de récurrence la suite u_h satisfait-elle ?
4. En déduire qu'il existe une constante $\alpha > 0$ telle que $\mathsf{m}(h) > \alpha(\sqrt{2})^h$.

Exercice 4 *Codage et parcours*
On définit le type suivant pour représenter des squelettes d'arbres binaires **pleins**.

```
type stree =  Leaf   |   Bin of stree * stree
```

À tout arbre `t` de type `stree`, on associe une liste de 0 et de 1 définie ainsi :

- `encode(t) = [1]`, si `t` est une feuille (`Leaf`).
- `encode(t) = [0] @ encode(t1) @ encode(t2)`, si `t` a pour fils gauche `t1` et pour fils droit `t2`.

1. Écrire la fonction `encode : stree -> int list` en `OCaml`. Cette fonction doit retourner une liste de 0 et de 1 codant l'arbre de type `stree` passé en argument.
2. Écrire en `OCaml` la fonction `decode : int list -> stree` qui, étant donnée une liste de 0 et 1, retourne l'arbre binaire de type `stree` qu'elle code, ou indique une erreur si la liste ne code pas un arbre (utiliser `failwith "Error"` dans ce dernier cas).

   On pourra s'aider d'une fonction auxiliaire récursive qui s'applique à une liste `binlist` de 0 et 1, et qui retourne un couple composé des deux valeurs suivantes :

   - un arbre de type `stree` dont le code est une liste `debut`, par laquelle commence la liste `binlist`,
   - la fin de `binlist` qui reste à décoder, c'est-à-dire la liste `fin` telle que `binlist = debut @ fin`.

Le code suivant présente un exemple des fonctions `encode` et `decode`.

```
# t ; ;
- : stree = Bin (Bin (Bin (Leaf, Leaf), Bin (Leaf, Leaf)), Leaf)
# encode t ; ;
- : int list = [0 ; 0 ; 0 ; 1 ; 1 ; 0 ; 1 ; 1 ; 1]
# decode [0 ; 0 ; 0 ; 1 ; 1 ; 0 ; 1 ; 1 ; 1] ; ;
- : stree = Bin (Bin (Bin (Leaf, Leaf), Bin (Leaf, Leaf)), Leaf)
```