université de BORDEAUX	**Année 2018/2019** — L2 Info/MI, UE 4TIN402U Algorithmique des structures de données arborescentes Lundi 6 mai 2019, 9h–10h30 — Durée : 1h30 Documents : **non autorisés** Le barème est donné à titre **indicatif**	**Collège Sciences et Technologies**

Exercice 1 *Bijections (2 points)*
Soit $n = 5$.

1. Dessiner **un** squelette d'arbre binaire S de taille n. Vous pouvez choisir celui que vous souhaitez.
2. Dessiner l'image de S par la bijection vue en cours entre l'ensemble des squelettes d'arbres binaires de taille n et l'ensemble des squelettes d'arbres binaires complets de taille $2n + 1$.
3. Dessiner P, l'image de S par la bijection vue en cours entre les squelettes d'arbres binaires de taille n et l'ensemble des squelettes d'arbres planaires de taille $n + 1$.
4. Donner le mot de Dyck associé à P par la bijection vue en cours.

Exercice 2 *Arbres AVL (5 points)*
Soit T, l'arbre binaire de recherche (ABR) illustré sur la Fig. 1.

1. L'arbre T est-il un arbre AVL ? Justifier votre réponse.
2. Dessiner T_1, l'ABR obtenu de T par suppression de 30 avec l'algorithme vu en cours.
3. L'arbre T_1 est-il un AVL ? Si oui, on note $T_2 = T_1$. Sinon, dessiner T_2, un AVL obtenu de T_1 par équilibrage en expliquant les rotations effectuées.
4. Dessiner T_3, l'ABR obtenu de T_2 par suppression de 45.
5. L'arbre T_3 est-il un AVL ? Sinon, dessiner T_4, un AVL obtenu de T_3 par équilibrage en expliquant les rotations effectuées.
6. Combien de rotations peut entraîner une suppression dans un arbre AVL dans le pire des cas ? Donner un ordre de grandeur en fonction de n, la taille de l'arbre AVL.

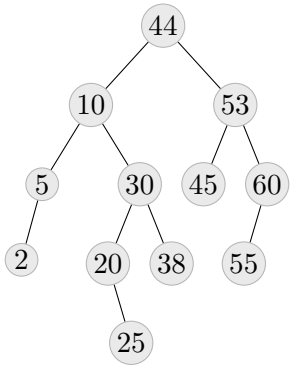
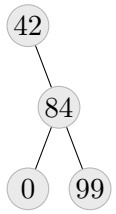


FIGURE 1 – T - arbre binaire de recherche

Exercice 3 *Hauteur gauche (3 points)*

On dit qu'une arête dans un arbre binaire est une ***arête gauche*** lorsqu'elle relie un sommet à son fils gauche. Sur l'arbre `Bin(42, Empty, Bin(84, Bin(0, Empty, Empty), Bin(99, Empty, Empty)))` représenté ci-dessous, $84 \to 0$ est la seule arête gauche (les arêtes $42 \to 84$ et $84 \to 99$ sont des arêtes droites).



On définit la ***hauteur gauche d'une branche*** comme son nombre d'arêtes gauches. On définit la ***hauteur gauche d'un arbre*** comme le maximum de toutes les hauteurs gauches de ses branches. La hauteur gauche de l'arbre vide est, par convention, -1.

Par exemple, la hauteur gauche de l'arbre ci-dessus est 1 car la branche $42 \longrightarrow 84 \longrightarrow 0$ a pour hauteur gauche 1, et la seule autre branche, $42 \longrightarrow 84 \longrightarrow 99$, a pour hauteur gauche 0.

1. Quelle est la hauteur gauche de l'arbre symétrique de celui de l'exemple ci-dessus ?
2. Exprimer la hauteur gauche d'un arbre binaire non vide en fonction de celle de son sous-arbre gauche et de celle de son sous-arbre droit. On demande une ***justification claire***.
3. En utilisant le type `type 'a bin = Empty | Bin of 'a * 'a bin * 'a bin`, écrire une fonction retournant la hauteur gauche d'un arbre.

Exercice 4 *Mots de Dewey (6 points)*

Dans un squelette d'arbre binaire, on étiquette une arête gauche par 0 et une arête droite par 1. À chaque nœud, on associe un *mot de Dewey*, qui est la liste des étiquettes des arêtes rencontrées en parcourant la branche de la racine jusqu'à ce nœud. En particulier, le mot de Dewey associé à la racine est la liste vide.

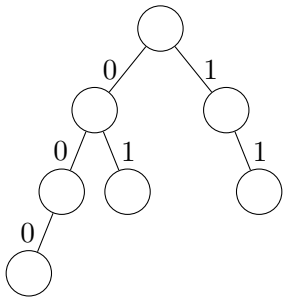


FIGURE 2 – Squelette d'arbre et ensemble de mots associés `[[]; [0]; [0;0]; [0;0;0]; [0;1]; [1]; [1;1]]`

1. En utilisant le type `type skel = Empty | Bin of skel * skel`, écrire une fonction `dewey_read` de type `int list -> skel -> skel` telle que `dewey_read word t` retourne le sous-arbre atteint en suivant la liste `word` (composée de 0 et de 1) sur l'arbre `t` si `word` correspond à un nœud de l'arbre, et `Empty` sinon. Par exemple, si `t` est l'arbre ci-dessus, alors :

```
# dewey_read [1;1] t;;
- : char skel = Bin (Empty, Empty)
# dewey_read [0;0] t;;
- : char skel = Bin (Bin (Empty, Empty), Empty)
# dewey_read [0;0;1] t;;
- : char skel = Empty
```

2. En utilisant le même type, écrire une fonction `dewey_words` de type `skel -> int list list` telle que `dewey_words t` retourne l'ensemble des mots de Dewey associés à tous les nœuds de l'arbre binaire `t`. Par exemple, si `t` est l'arbre ci-dessus,

```
# dewey_words t;;
- : int list list = [[]; [0]; [0; 0]; [0; 0; 0]; [0; 1]; [1]; [1; 1]]
#  dewey_words (Bin(Empty, Empty));;
- : int list list = [[]]
# dewey_words Empty;;
- : int list list = []
```

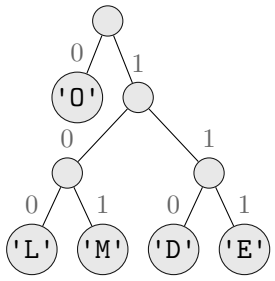
Exercice 5 *Codage de Huffman (4 points)*
On utilise le type suivant pour représenter les arbres de code de Huffman :

```
type huffman_tree = Single of char | Double of huffman_tree * huffman_tree
```

Par exemple, un arbre de Huffman `huff_tree` construit à partir de la chaîne `MOODLE` est illustré en Fig. 3(a) et est défini ainsi :

```
val huff_tree : huffman_tree =
  Double (Single 'O',
  Double (Double (Single 'L', Single 'M'), Double (Single 'D', Single 'E')))
```

Écrire une fonction `decompress` de type `huffman_tree -> int list -> char list` qui à partir d'un arbre de Huffman t et d'une liste ℓ_{in} de 0 et 1, produit la liste de caractères ℓ_{out} correspondant à ℓ_{in} selon le codage défini par t (on supposera que la liste ℓ_{in} correspond bien à un message complet).
Par exemple, l'appel `decompress huff_tree [1;0;1;0;0;1;1;0;1;0;0;1;1;1]` produit la liste de caractères `['M';'O';'O';'D';'L';'E']`.



(a)

Code	Caractère
0	'O'
100	'L'
101	'M'
110	'D'
111	'E'

(b)

ℓ_{in}	[1;0;1;0;0;1;1;0;1;0;0;1;1;1]
ℓ_{out}	['M';'O';'O';'D';'L';'E']

(c)

FIGURE 3 – Exemple d'un codage de Huffman : (a) Arbre de Huffman (b) Liste des codes (c) Décompression