université de BORDEAUX	**Année 2016/2017** L2 Info/MI, UE 4TIN402U Algorithmique des structures de données arborescentes Jeudi 9 mars 2017 Durée : 1h20 Documents : **non autorisés** Le barème est donné à titre **indicatif**	**Collège Sciences et Technologies**

Exercice 1 *Arbre binaire de recherche (3pts)*

1. Dessiner l'*arbre binaire de recherche* construit par des ajouts successifs des éléments de la suite $1, 9, 2, 8, 3, 7, 4, 6, 5$ en utilisant la méthode vue en cours (adjonction aux feuilles).
2. Quel est le nombre maximal de comparaisons effectuées pour la recherche d'un élément dans cet arbre ?
3. De manière générale, exprimer à l'aide de la notation $O()$ la complexité, dans le pire des cas, de l'algorithme de recherche dans un *arbre binaire de recherche* contenant $n \geqslant 0$ éléments.

Pour la suite du devoir, on définit le type suivant permettant de représenter les arbres binaires contenant un entier dans chaque nœud.

```
type  btree =
    Empty
  | Node of  btree * int * btree
```

Exercice 2 *Arbre binaire (2pts)*

Écrire une fonction en `OCAML` ou pseudo-code qui calcule la hauteur d'un arbre. Par convention et par commodité, on définit la hauteur d'un arbre vide égale à -1.
Exemples :

```
# btree_height Empty;;
- : int = -1
# btree_height (Node(Empty, 1, Empty));;
- : int = 0
# btree_height (Node(Empty, 1, Node(Node(Empty, 3, Empty), 4, Empty)));;
- : int = 2
```

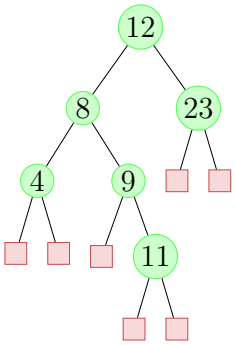
Exercice 3 *Arbre binaire de recherche (4pts)*

Un *arbre binaire de recherche* non vide A est dit *de domaine plus petit* qu'un *arbre binaire de recherche* non vide B si

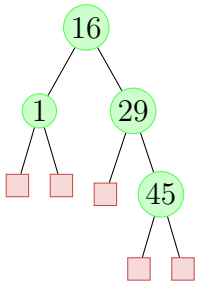
— le plus petit élément de A est supérieur ou égal au plus petit élément de B, **et**

— le plus grand élément de A est inférieur ou égal au plus grand élément de B.

L'illustration est donnée sur la Fig. 1.
Écrire une fonction en `OCAML` ou pseudo-code `smaller` qui prend en arguments deux arbres binaires de recherche `a` et `b` de type `btree`, et telle que `smaller a b` teste si l'arbre binaire de recherche `a` est de domaine plus petit que `b`.



(a) Arbre A



(b) Arbre B

FIGURE 1 – L'arbre binaire de recherche A a un domaine plus petit que l'arbre binaire de recherche B

Exercice 4 *Arbre binaire (4pts)*

On dit qu'un arbre binaire est *parfait* si toutes ses branches ont la même longueur.

Écrire une fonction en `OCAML` ou pseudo-code `perfect` qui s'applique à un arbre binaire `bst` de type `btree` et qui retourne `true` si l'arbre `bst` est parfait, et `false` sinon. On pourra s'aider d'une fonction auxiliaire récursive qui s'applique à un arbre binaire de type `btree` et qui retourne un couple composé de **deux** valeurs :

— un booléen indiquant si l'arbre est parfait ou pas,

— la hauteur de l'arbre.

Aide : On aura éventuellement besoin de la fonction `OCAML fst` qui retourne le premier élément d'un couple. Alternativement, on peut nommer un couple par `let (a,b) = ...`

Exemples :

```
# perfect Empty;;
- : bool = true
# perfect (Node(Empty, 1, Empty));;
- : bool = true
# perfect (Node(Empty, 1, Node(Empty, 2, Empty)));;
- : bool = false
# perfect (Node(Node(Empty, 2, Empty), 1, Node(Empty, 3, Empty)));;
- : bool = true
```

Exercice 5 *Arbre 234 (2pts)*

Parmi les affirmations suivantes indiquer si elles sont vraies ou fausses s'il s'agit des *arbres 2-3-4*. Justifier votre réponse.

1. Le nombre de fils d'un nœud est égal au nombre de valeurs contenues dans ce nœud + 1.
2. Si n est le nombre de nœuds alors sa hauteur est exactement $\log_2(n)$.
3. La différence de hauteur entre les sous-arbres qui ont pour racines les fils d'un même nœud est d'au plus 1.
4. L'insertion aboutit toujours à la création d'une feuille.

Exercice 6 *Arbre 234 (3pts)*

1. Dessiner les *arbres 2-3-4* construits successivement par les ajouts des éléments $1, 9, 2, 8, 3, 7, 4, 6, 5$.
2. Quel est le nombre maximal de comparaisons effectuées pour la recherche d'un élément dans cet arbre ?
3. De manière générale, exprimer à l'aide de la notation $O()$ la complexité, dans le pire des cas, de l'algorithme de recherche dans un *arbre 2-3-4* contenant $n \geqslant 0$ éléments.

Exercice 7 *Arbre 234 (2pts)*

Soit l'*arbre 2-3-4* donné sur la Fig. 2. Détailler les étapes de l'insertion de 13 avec éclatements à la descente.

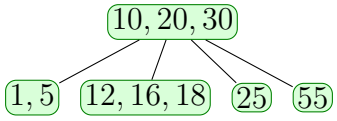


FIGURE 2 – Arbre 2-3-4

Exercice 8 *Arbre binaire de recherche (3pts)*

Écrire une fonction en `OCAML` ou pseudo-code `bst_cut` qui s'applique à un élément `e` et un arbre binaire de recherche `bst` (de type `btree` défini plus haut) et telle que `bst_cut e bst` retourne un couple composé de **deux** arbres binaires de recherche :

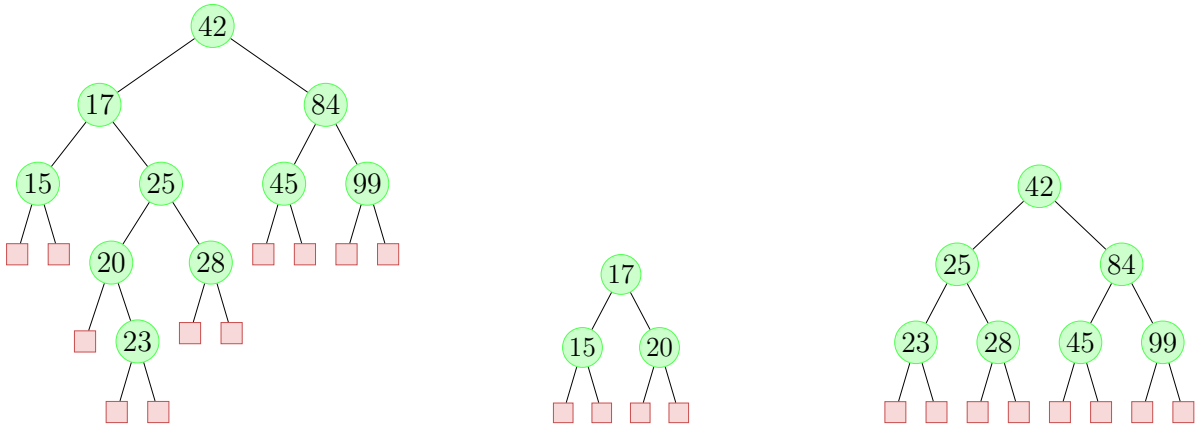
— le premier contenant tous les éléments de `bst` qui sont strictement plus petits que `e`,

— le second contenant tous les éléments de `bst` qui sont strictement plus grands que `e`.

La coupure d'un arbre binaire de recherche est illustrée sur la Fig. 3.

Exemples :

```
# bst_cut 1 (Node(Empty, 2, Empty));;
- : int btree * int btree = (Empty, Node (Empty, 2, Empty))
# bst_cut 2 (Node(Empty, 2, Empty));;
- : int btree * int btree = (Empty, Empty)
# bst_cut 3 (Node(Empty, 2, Empty));;
- : int btree * int btree = (Node (Empty, 2, Empty), Empty)
# bst4;;
- : int btree =
Node
 (Node (Node (Empty, 1, Empty), 4,
   Node (Node (Empty, 5, Node (Empty, 7, Empty)), 9, Node (Empty, 11, Empty))),
 15, Node (Empty, 16, Empty))
# bst_cut 8 bst4;;
- : int btree * int btree =
(Node (Node (Empty, 1, Empty), 4, Node (Empty, 5, Node (Empty, 7, Empty))),
 Node (Node (Empty, 9, Node (Empty, 11, Empty)), 15, Node (Empty, 16, Empty)))
# bst_cut 9 bst4;;
- : int btree * int btree =
(Node (Node (Empty, 1, Empty), 4, Node (Empty, 5, Node (Empty, 7, Empty))),
 Node (Node (Empty, 11, Empty), 15, Node (Empty, 16, Empty)))
```



(a) T arbre binaire de recherche (b) T_{inf22} (c) T_{sup22}

FIGURE 3 – Coupure de T par 22 en deux arbres binaires de recherche, T_{inf22} avec des éléments inférieurs à 22, et T_{sup22} avec des éléments supérieurs à 22