Algorithmics
Final Exam #2 (P2)

Undergraduate 1st year (S2)
EPITA

6 June 2016 - 10 : 00 *(D.S. 307430.1 BW)*

Instructions (read it) :

- ☐ You must answer on **the answer sheets provided.**
  - No other sheet will be picked up. Keep your rough drafts.
  - Answer within the provided space. **Answers outside will not be marked**: Use your drafts!
  - Do not separate the sheets unless they can be re-stapled before handing in.
  - Penciled answers will not be marked.
- ☐ The presentation is negatively marked, which means that you are marked out of 20 points and the presentation points (maximum of 2) are taken off this grade.
- ☐ **Code:**
  - All code must be written in the language `Python` (no C, CAML, ALGO or anything else).
  - **Any `Python` code not indented will not be marked.**
  - All that you need (types, routines) is indicated in the **appendix** (last page)!
  - Your functions must follow the given examples of application.
- ☐ Duration : 2h

Exercise 1 (Leonardo trees – *5 points*)

In this exercise we will study some properties of a certain type of trees: the Fibonacci trees. Those are defined recursively as follows:

$$\begin{cases} A_0 = \ EmptyTree \\ A_1 = < o, EmptyTree, EmptyTree > \\ A_n = < o, A_{n-1}, A_{n-2} > \ if \ n \geqslant 2 \end{cases}$$

1. Give a graphical representation of the Fibonacci tree A_5.
2. For each tree A_n with $0 \leqslant n \leqslant 6$, give the values of: the height H_n, the size T_n, the number of leaves F_n and the Fibonacci number Fib_n (with $Fib_0 = 0$ et $Fib_1 = 1$).
3. Give, as functions of $n \geqslant 2$ and potentially the Fibonacci number Fib_n: the height H_n, the size T_n and the number of leaves F_n of the tree A_n.
4. Prove that the tree A_n is height-balanced.

Exercise 2 (BST and mystery – *5 points*)

```python
def bstMystery(x, B):

 # first part
    P = None
    while B != None and x != B.key:
        if x < B.key:
            P = B
            B = B.left
        else:
            B = B.right
    if B == None:
        return None

 # second part
    if B.right == None:
        return P
    else:
        B = B.right
        while B.left != None:
            B = B.left
        return B
```

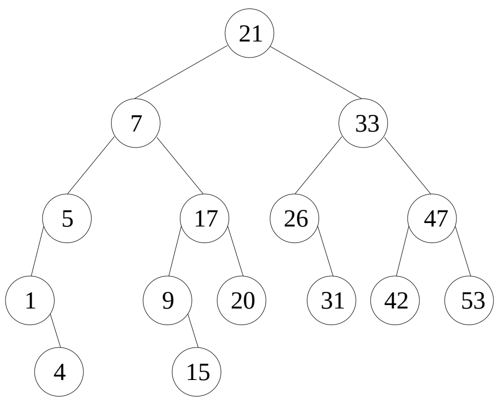

FIGURE 1 – tree B_1

```python
def call(x, B):
    p = bstMystery(x, B)
    if p == None:
        return None
    else:
        return p.key
```

1. Let B_1 be the tree in figure 1. What are the results of each of the following calls?
   (a) `call(25,` B_1`)`
   (b) `call(21,` B_1`)`
   (c) `call(20,` B_1`)`
   (d) `call(9,` B_1`)`
   (e) `call(53,` B_1`)`
2. `bstMystery(x, B)` is called with `B` any binary search tree, where all elements are different.
   During execution, at the end of part 1:
   (a) What does `B` represent?
   (b) What does `P` represent?
3. What does the fonction `call(x, B)` do?

In the two following exercises, we use a new implementation of binary trees where each node contains the size of the tree it is root of.

```python
class BinTreeSize:
    """
    """
def newBinTreeSize(key, left, right, size):
    B = BinTreeSize()
    B.key = key
    B.left = left
    B.right = right
    B.size = size    # size of the tree
    return B
```

Exercise 3 (Add the size – *4 points*)

Write the function `copyWithSize(`B`)` that takes a "classic" binary tree B (`BinTree()` without the size) as parameter and returns an equivalent tree (containing same values at same places) but with the size specified in each node (`BinTreeSize()`).

Exercise 4 (Median – *7 points*)

We will study the research of the node that contains the median value in a binary search tree. That is, the value at the rank $size(B) + 1\ div\ 2$ in the list of elements in increasing order.

For this, we want to write the function `nthBST(`B`, `k`)` that returns the node that contains the k^{th} element of the tree B. For example, the call `nthBST(`B_1`, 3)` with $B - 1$ the tree in figure 1 will return the node that contains the value 5.

1. **A little help for the rest:**

   Let B be a binary search tree with n elements. If the k^{th} element (with $1 \leq k \leq n$) is in the root, how many elements do the two subtrees of B contain?

2. **Abstract study:**

   The *size* operation, defined as follows, is added to the abstract definition of binary trees (given in appendix):

   OPERATIONS

   $size$: BinaryTree → Integer

   AXIOMS

   $size$ (emptytree) = 0

   $size$ (<o, L, R>) = 1 + $size$ (L) + $size$ (R)

   Give an abstract definition of the operation *nth* (that has to use the operation *size*): complete the given definitions.

3. **Implementation:**

   The functions you have to write use binary trees with the size in each node (`BinTreeSize()`).

   - Write the function `nthBST(`B`, `k`)` that returns the tree with the k^{th} element as root. We suppose that this element always exists: $1 \leq k \leq size(B)$.
   - Write the function `median(`B`)` that returns the median value of the binary search tree B if non empty.

Appendix

Binary Tree Algebraic Abstract Type

TYPES
BinaryTree
USES
Node, Element
OPERATIONS

emptytree	:	$\rightarrow$ BinaryTree
<_, _, _>	:	Node $\times$ BinaryTree $\times$ BinaryTree $\rightarrow$ BinaryTree
root	:	BinaryTree $\rightarrow$ Node
l	:	BinaryTree $\rightarrow$ BinaryTree
r	:	BinaryTree $\rightarrow$ BinaryTree
content	:	Node $\rightarrow$ Element

PRECONDITIONS

$root(B)$ **is defined if-and-only-if** $B \neq emptytree$
$l(B)$ **is defined if-and-only-if** $B \neq emptytree$
$r(B)$ **is defined if-and-only-if** $B \neq emptytree$

AXIOMS

$root(<o, L, R>) = o$
$l(<o, L, R>) = L$
$r(<o, L, R>) = R$

WITH

L, R	:	BinaryTree
o	:	Node