Algorithmics
Correction Final Exam #2 (P2)

Undergraduate 1^{st} year (S2) – Epita

6 June 2016 - 10:00

Solution 1 **(Leonardo trees –** ***5 points*****)**

1. The Fibonacci tree A_5 is the one in figure 1 with each node containing its balance factor value.

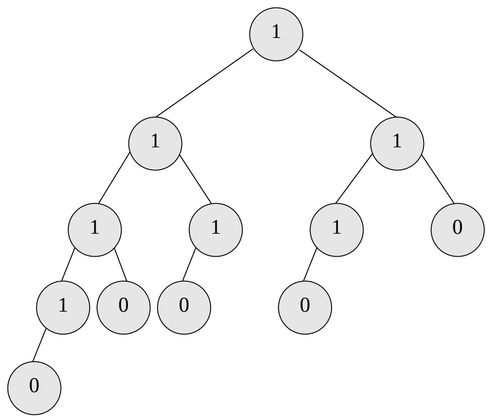

Figure 1: The Fibonacci tree A_5

2. Table of H_n, T_n, F_n and Fib_n :

n	H_n	T_n	F_n	Fib_n
0	_	0	0	0
1	0	1	1	1
2	1	2	1	1
3	2	4	2	2
4	3	7	3	3
5	4	12	5	5
6	5	20	8	8

3. $n \geq 2$:
   - $H_n = n - 1$
   - $T_n = Fib_{n+2} - 1$ as $T_n = T_{n-1} + T_{n-2} + 1$
   - $F_n = Fib_n = Fib_{n-1} + Fib_{n-2}$

4. A_0 is a leaf, A_1 has a single node at its left, nothing at its right : these trees are height-balanced.
   With $n \geq 2$, A_n height is $n - 1$. Its subtrees are A_{n-1} of height $n - 2$ and A_{n-2} of height $n - 3$.
   Thus, the balance factor of the root of A_n is 1 $(n - 2 - (n - 3))$.
   All internal nodes of a Fibonacci tree have a balance factor of 1 : it is an height-balanced tree.

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

1. *Returned results?*
   (a) `call(25, B)` : None
   (b) `call(21, B)` : 26
   (c) `call(20, B)` : 21
   (d) `call(9, B)` : 15
   (e) `call(53, B)` : None

2. `bst_mystery(x, B)` (`B` any BST, with distinct elements).
   At the end of part 1:
   (a) `B` is the tree that contains x in its root, None if x is not in the tree.
   (b) On the search path, `P` is the tree which root is the last encounter node before descending on the left (it stays None if we never go to the left).

3. `call(x, B)`: if x is found in B and is not the greatest value, the function returns the value just after x in B. Otherwise it returns `None`.

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

```
        def addSize(B):
            if B == None:
                return(None, 0)
            else:
                C = BinTreeSize()
                C.key = B.key
                (C.left, size1) = addSize(B.left)
                (C.right, size2) = addSize(B.right)
                C.size = 1 + size1 + size2
                return (C, C.size)

# another version

        def addSize2(B):
            if B == None:
                return(None, 0)
            else:
                (left, size1) = addSize2(B.left)
                (right, size2) = addSize2(B.right)
                size = 1 + size1 + size2
                return (newBinTreeSize(B.key, left, right, size), size)
```

```
        def copyWithSize(B):
            (C, size) = addSize(B)
            return C
```

Solution 4 **(Median – *7 points*)**

1. B BST with n elements such that the k^{th} element ($1 \leq k \leq n$) is in the root:
   - size(l(B)) $= k - 1$
   - size(r(B)) $= n - k$

2. *Abstract definition of the operation nth (median was given):*

   AXIOMS

   k = *size*(G)+1 ⇒ *nth* (<r, G, D>, k) = r
   k ≤ *size* (G) ⇒ *nth* (<r, G, D>, k) = *nth* (G, k)
   k > *size* (G) +1 ⇒ *nth* (<r, G, D>, k) = *nth* (D, k - *size* (G)-1)

3. **Specifications:**
   The function `nthBST(`B`, `k`)` with B a nonempty BST and $1 \leq k \leq size(B)$ returns the tree with the k^{th} element of B as root.

```
def nthBST(B, k):

    if B.left == None:
        leftSize = 0
    else:
        leftSize = B.left.size

    if leftSize == k - 1:
        return B
    elif k <= leftSize:
        return nthBST(B.left, k)
    else:
        return nthBST(B.right, k - leftSize - 1)


def nthBST2(B, k):

    if B.left == None:
        if k == 1:
            return B
        else:
            return nthBST2(B.right, k - 1)

    else:
        if k == B.left.size + 1:
            return B
        elif k <= B.left.size:
            return nthBST2(B.left, k)
        else:
            return nthBST2(B.right, k - B.left.size - 1)
```

Specifications:
The function `median(`B`)` returns the median value of the binary search tree B if it is non empty. Otherwise, it returns `None`.

```
def median(B):
    if B != None:
        return nthBST(B, (B.size+1) // 2).key
    else:
        return None
```