Algorithmics
Correction Final Exam #2 (P2)

Undergraduate 1^{st} year S2# – Epita

January, 7th 2020 - 13h-15h

Solution 1 **(Leonardo trees – *3 points*)**

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

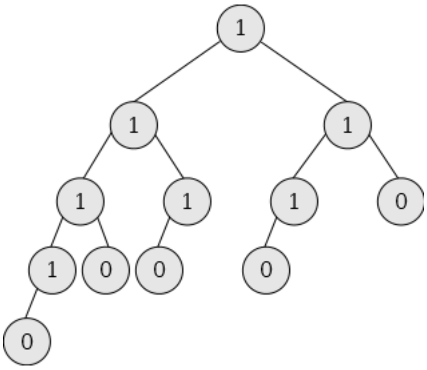


Figure 1: The Fibonacci tree A_5

2. (a) $h_n = n - 1$

   (b) 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 (Leonardo Trees, again – *4 points*)

Specifications:
The function `leonard_tree`(n) builds the Fibonacci tree A_n.

```
def leonardo_tree(n):
    if n == 0:
        return None
    elif n == 1:
        return BinTree(1, None, None)
    else:
        G = leonardo_tree(n-1)
        D = leonardo_tree(n-2)
        key = G.key
        if D != None:
            key += D.key

        return BinTree(key, G, D)
```

Solution 3 (Deletion)

1. **Specifications:**
La fonction `maxBST`(B) retourne la valeur maximale de l'arbre binaire de recherche non vide B.

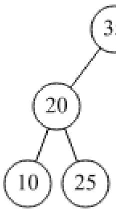
```
def maxBST(B):
  while B.right != None:
      B = B.right
  return B.key
```

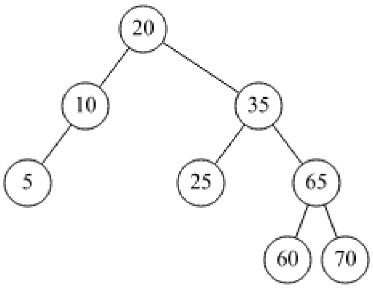
2. **Specifications:**
The function `delBST`(B, x) deletes the element x from the binary search tree B and returns the tree.

```
def delBST(B, x):
    if B == None:
        return None
    else:
        if x == B.key:
            if B.left == None:
                return B.right
            elif B.right == None:
                return B.left
            else:
                B.key = maxBST(B.left)
                B.left = del_bst(B.left, B.key)
                return B
        else:
            if x < B.key:
                B.left = delBST(B.left, x)
            else:
                B.right = delBST(B.right, x)
            return B
```

Solution 4 (AVL – *3 points*)

Final AVL from the list $[25, 60, 35, 10, 20, 5, 70, 65]$.

Tree built by insertions of 25, 60, 35, 10, 20:

 	
	Rotations:
	`rlr(25) rdg(25)`
	`lrr(25) rgd(25)`

Tree after insertions of 5, 70, 65:

 	
	Rotations:
	`rr(35)  rd(35)`
	`rlr(60) rdg(60)`

Solution 5 (What is this? – *3 points*)

1. *Results for* (a) `test(`B_2`)`: True
   (b) `test(`B_3`)`: False

2. `test(`B`)` checks if the binary tree B is height-balanced.

3. To optimize this function: if the boolean of the first call is false, it is possible to avoid the second by directly returning `(?,False)`.