Algorithmics
Correction Final Exam #2 (P2)

Undergraduate 1^{st} year S2 – Epita

May, 22th 2019

Solution 1 **(Add the size – *4 points*)**

Specifications:

The function `copyWithSize(`B`)` with B a "classic" binary tree (`BinTree`) returns a copy of B with the size specified in each node (`BinTreeSize`).

```
# __copySize(B) returns the pair (copy of B: BinTreeSize, its size: int)
        def __copySize(B):
            if B == None:
                return(None, 0)
            else:
                (left, size1) = __copySize(B.left)
                (right, size2) = __copySize(B.right)
                size = 1 + size1 + size2
                return (BinTreeSize(B.key, left, right, size), size)

    # another version
        def __copySize2(B):
           if B == None:
               return(None, 0)
           else:
               C = BinTreeSize(B.key, None, None, 1)
               (C.left, size1) = __copySize2(B.left)
               (C.right, size2) = __copySize2(B.right)
               C.size += size1 + size2
               return (C, C.size)
```

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

Solution 2 **(Insertion with size update)**

Specifications:

The function `addwithsize(`B`,` x`)` adds x in leaf in the binary search tree B (`BinTreeSize`) unless it is already present. It returns a pair: (the tree after a potential insertion, a boolean that indicates if the insertion occurred).

```
        def addBSTSize(x, A):
            if A == None:
                A = BinTreeSize(x, None, None, 1)
                return (A, True)
            else:
                if x < A.key:
                    (A.left, insert) = addBSTSize(x, A.left)
                elif x > A.key:
                    (A.right, insert) = addBSTSize(x, A.right)
                else:
                    insert= False
                if insert:
                    A.size += 1
                return (A, insert)
```

***Solution 3* (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 non empty BST and $1 \leq k \leq size(B)$ returns the tree with the k^{th} element of B as root.

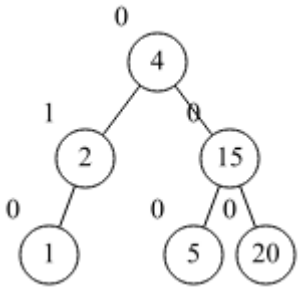
```python
def nthBST(B, k):

    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`.

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

***Solution 4* (AVL – *3 points*)**

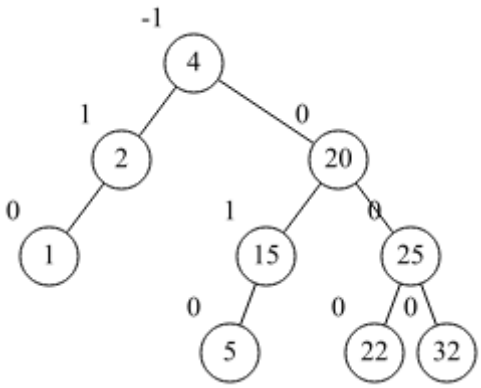
Tree built by insertions of 5, 15, 20, 2, 4, 1



Rotations

```
lr(rg) 5
lrr(rgd) 5
rr(rd) 15
```

Tree after insertions of 32, 25, 22



Rotations

```
rlr(rdg) 20
rlr(rdg) 15
```

***Solution 5* (AVL - Re-balancing – *3 points*)**

Specifications:

The function `rebalancing(`A`)` takes the non empty AVL A whose root has its balance factor in $[-2, 2]$. If necessary, it performs a rotation to re-balance A. It returns a pair: the possibly modified tree and a boolean indicating if the tree height has changed.

```
1         def rebalancing(A):
2             if abs(A.bal) < 2:
3                 return (A, False)
4             if A.bal == 2:
5                 if A.left.bal == 1:
6                     return (rr(A), True)
7                 elif A.left.bal == 0:
8                     return (rr(A), False)
9                 else:
10                    return (lrr(A), True)
11            else:    # A.bal == -2
12                if A.right.bal == -1:
13                    return (lr(A), True)
14                elif A.right.bal == 0:
15                    return (lr(A), False)
16                else:
17                    return (rlr(A), True)
```