Algorithmics
Correction Final Exam #2 (P2)

UNDERGRADUATE 1^{st} YEAR (S2) – EPITA

29 May 2017 - 13h45

Solution 1 **(2-4 trees ... – *6 points*)**

1. The successive insertions of values $\{Q, U, E, S, T, I, O, N, B, A, Z, Y, K\}$, give the 2-4 tree in figure 1.

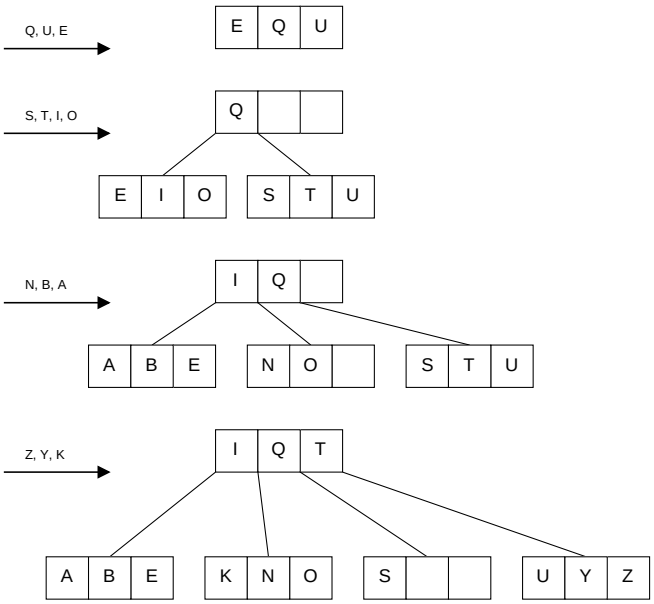


Figure 1: 2-4 tree after insertions of values $\{Q, U, E, S, T, I, O, N, B, A, Z, Y, K\}$.

2. The Red-black tree associated with the 2-4 tree of the previous question is the tree in figure 2.

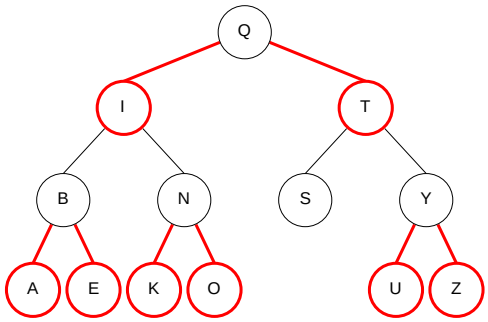
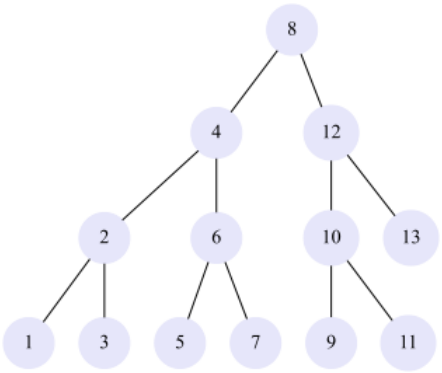


Figure 2: Red-black tree associated with the 2-4 tree of figure 1.

3. Three properties of a 2-4 tree could be:
   - A 2-4 tree is a search tree,
   - The nodes of a 2-4 tree are of three types : 2-node, 3-node or 4-node,
   - All the external nodes (leaves) of a 2-4 tree are at the same depth,
   - A 2-4 tree is balanced.
4. Three properties of a red-black tree could be:
   - A red-black tree is a binary search tree,
   - The nodes of a red-black tree are either red or black,
   - The root node of a red-black tree is always black,
   - In the red-black tree, a child node that contains a twin element of the one contained in the parent node is red,
   - In the red-black tree, the branches have a number of black links equal to the height of the corresponding 2-4 tree,
   - A red-black tree is balanced
5. The ***simple*** *method, using the red-black tree that represents it, allows one to determine the size of a 2-4 tree:* Count all the black nodes of the red-black tree.

Solution 2 **(Trees and mystery –** ***3 points*****)**

1. *Tree built by* `makeTree(13)`*:*



2. *Properties of the tree built by* `makeTree(n)` *(*$n > 0$*)*:
   (a) Complete tree
   (b) Binary Search Tree

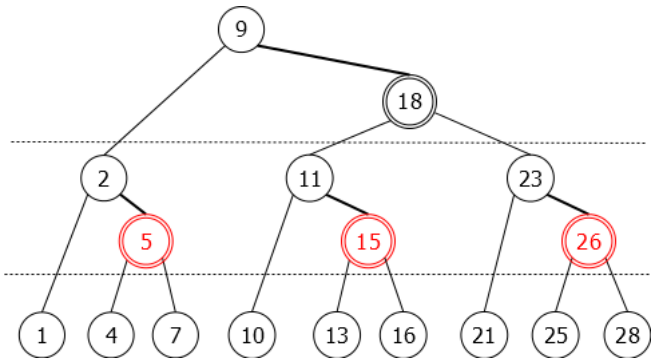
***Solution 3* (BST → AVL – *5 points*)**

Specifications:
The function `madeAVLfromBST(`B`)` builds a copy of the binary tree B with the balance factors specified in each node.

```
            def BST2AVL(B):
                if B == None:
                    return(None, -1)

                else:
                    A = avl.AVL(B.key, None, None, 0)
                    (A.left, hl) = BST2AVL(B.left)
                    (A.right, hr) = BST2AVL(B.right)

                    A.bal = hl - hr
                    return (A, 1 + max(hl, hr))


            def MakeAVL(B):
                (A, h) = BST2AVL(B)
                return A
```

***Solution 4* (AA Trees – *6 points*)**

1. *AA tree obtained after insertion of 4 in the tree in figure 6.*



2. **Specifications:**
The function `insertAA(`x`, A)` inserts x in the AA tree A unless x is already in A. It returns the resulting tree.

```
def insertAA(x, A):
    if A == None:
        return AAtree(x, None, None, 1)

    else:
        if x < A.key:
            A.left = insertAA(x, A.left)
            if A.left.level == A.level:
                A = skew(A)
                if A.right.right != None and A.right.right.level == A.level:
                    A = split(A)

        elif x > A.key:
            A.right = insertAA(x, A.right)
            if A.right.right != None and A.right.right.level == A.level:
                A = split(A)

        return A
```