CS5800: Algorithms — Iraklis Tsekourakis

Homework 6 Name: Avinash R. Arutla

1. (20 points)

Solution:

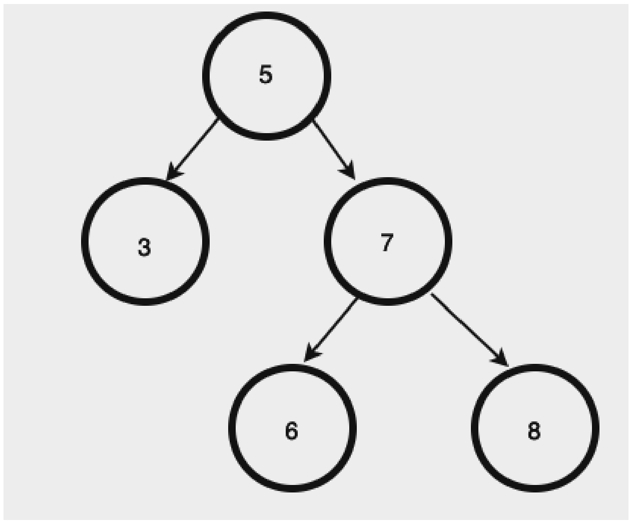
```
TREE-INSERT(root,key):

    if root is None:
        root = CreateNewNode(key)
        return root

    if key < root.value:
        if root.left == None:
            root.left = CreateNewNode(key)
        else:
            TREE-INSERT(root.left,key)

    else if key>root.value:
        if root.right == None:
            root.right = CreateNewNode(key)
        else:
            TREE-INSERT(root.right,key)

    return root
```

2. (20 points)

Solution:

The operation of deleting nodes in BST is not commutative. For example, deleting node x and then node y, may not result in the same tree that deleting node y and deleting node x gives out to be. This is due to the fact that when the first operation of deletion is carried out, there are chances that the tree would be altered differently than its counterpart operation. So thats why BST is not commutative necessarily.

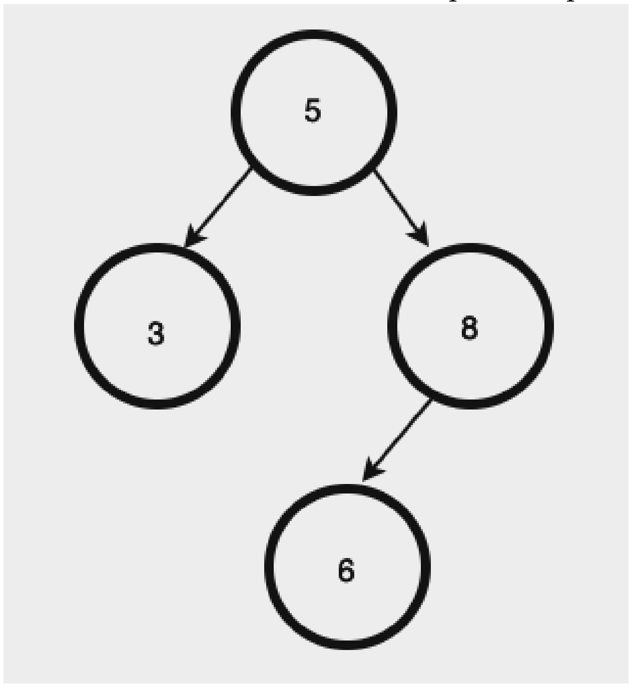
Lets consider a BST with the following structure



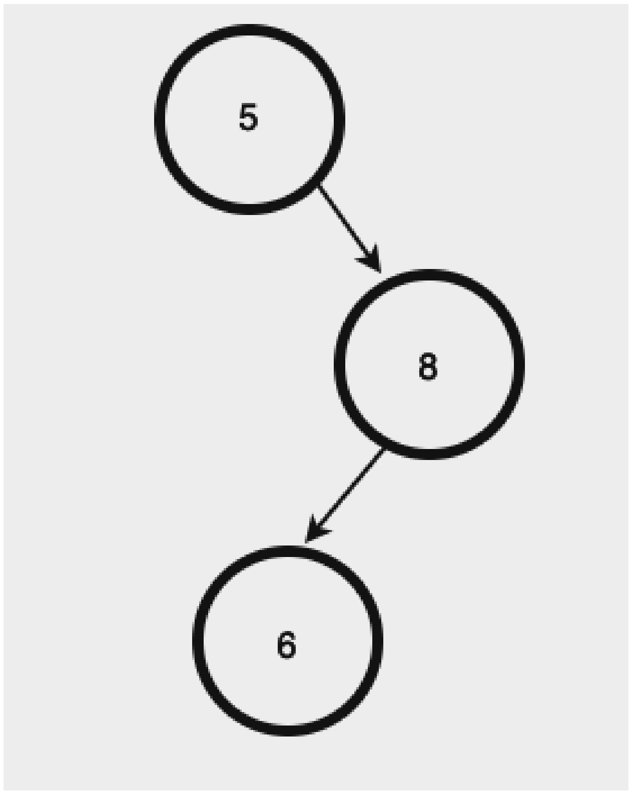
There are two cases here that are used to better understand the commutative property

a. Deleting 3 and then 7

When we first delete 7, element 8 replaces 7 spot as the successor. The resulting tree is as follows

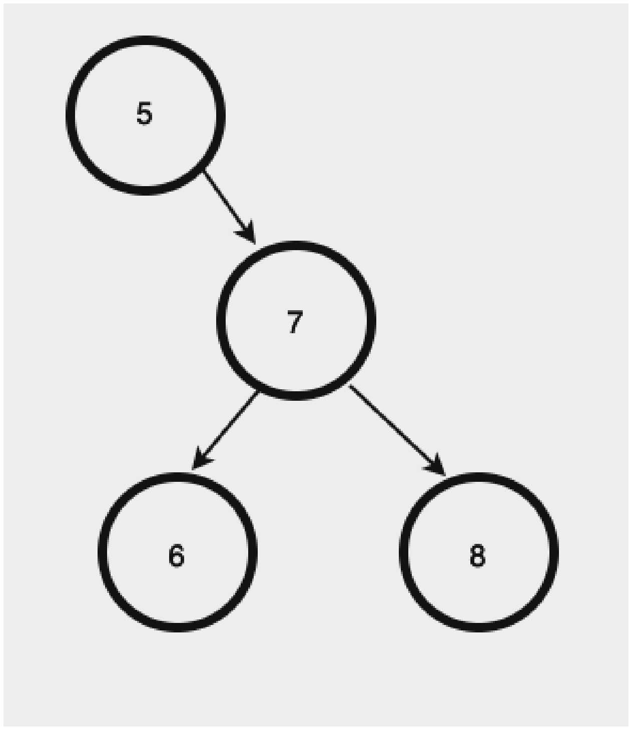


When we delete 3, since it has no children, its simply removed

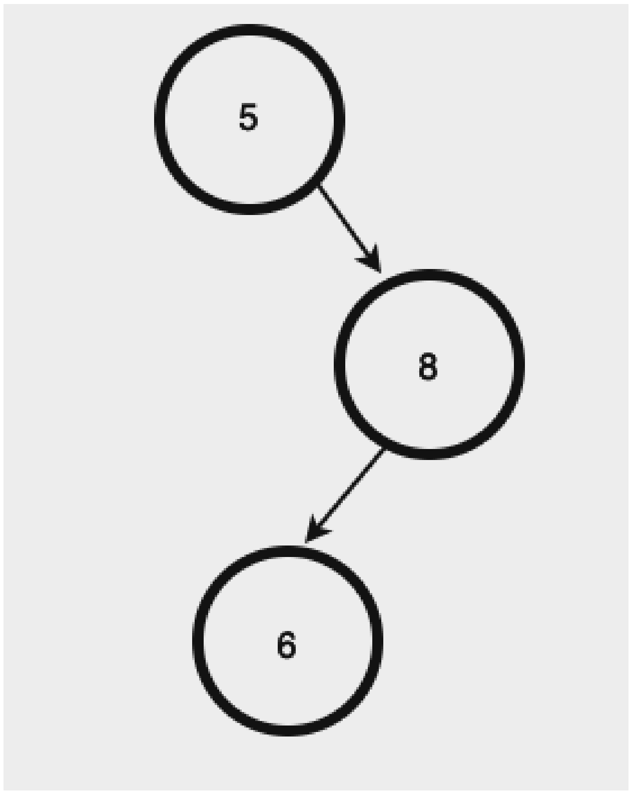


b. Deleting 7 and then 3

First we remove 3, because it has no children, it is just deleted



Then, we replace 7 with 8 when we delete 7

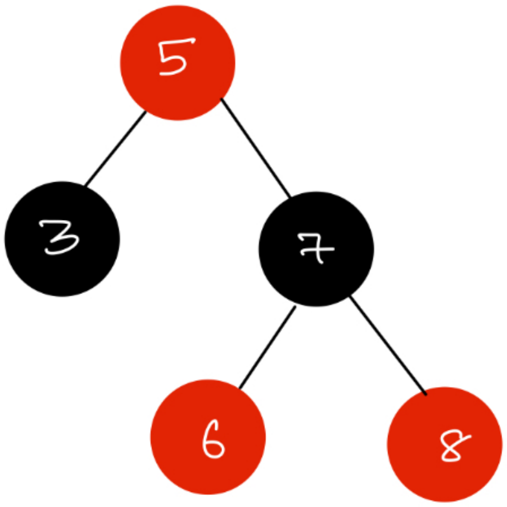


Here we can see that even though the sequence was different, the output was same. However this is usually not the case

3. (10 points)

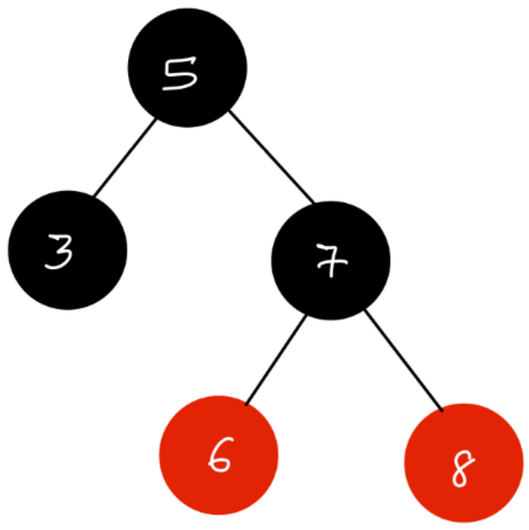
Solution:

The below diagram shows a simple red black tree



The above tree follows rules 1,3,4,5.

Now as described by the question, we make the root element 5 as black



Now lets compare the rules of RBT to check whether the modified version still adheres to the rules.

1. Here the rule 1 hasnt been violated since every node is colored either red or black.

2. Since the color of the root node is black, rule 2 hasnt been violated.

3. Here since there are no further color changes or new nodes that were created, there is no change and hence no red nodes have red children.

4. Every path has equal number of black nodes, so this rule hasnt been violated here.

5. Here all the nil nodes are still black, so property 5 hasnt been violated too.

As we can see from the above description of rules, even though the color of the root node is changed from red to black, there are no observable differences with the rules. So rules have been preserved.

4. (15 points)

Solution:

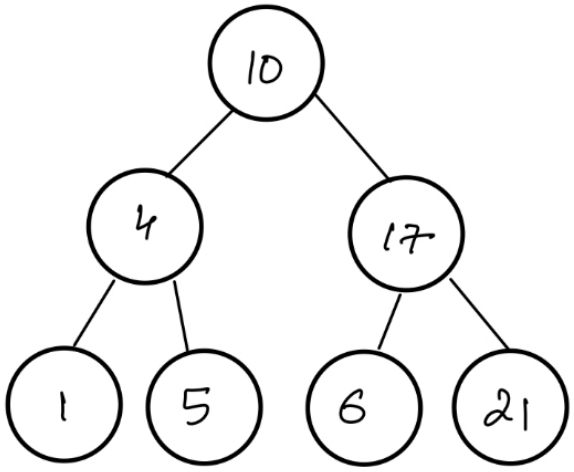
```
RIGHT-ROTATE(T,x):
```

```
if x.left!= NIL:
    y = x.left
    x.left = y.right

    if y.right!=NIL:
        y.right.parent = x

    y.parent = x.parent

    if x.parent == NIL:
        T.root = y

    else if x == x.parent.right:
        x.parent.right = y
    else:
        x.parent.left = y

    y.right = x
    x.parent = y
```

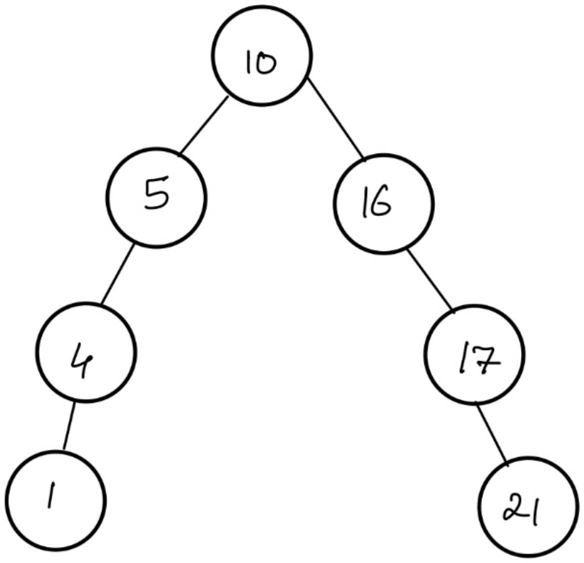
5. (15 points)

Solution:

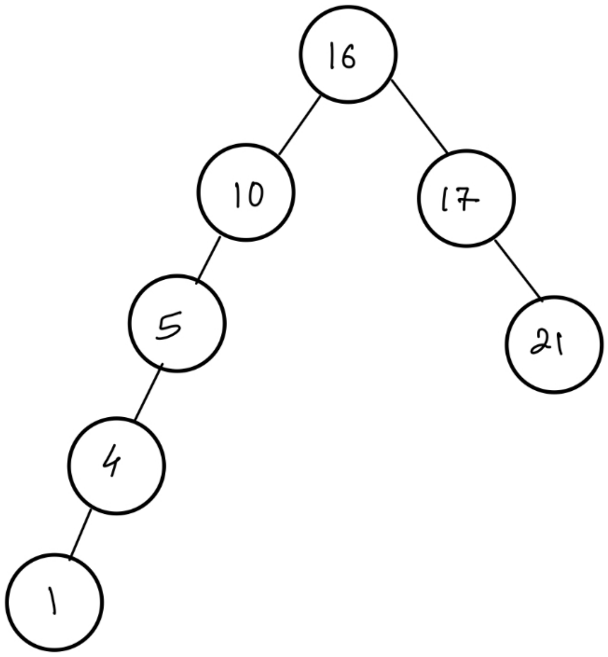
Height 2



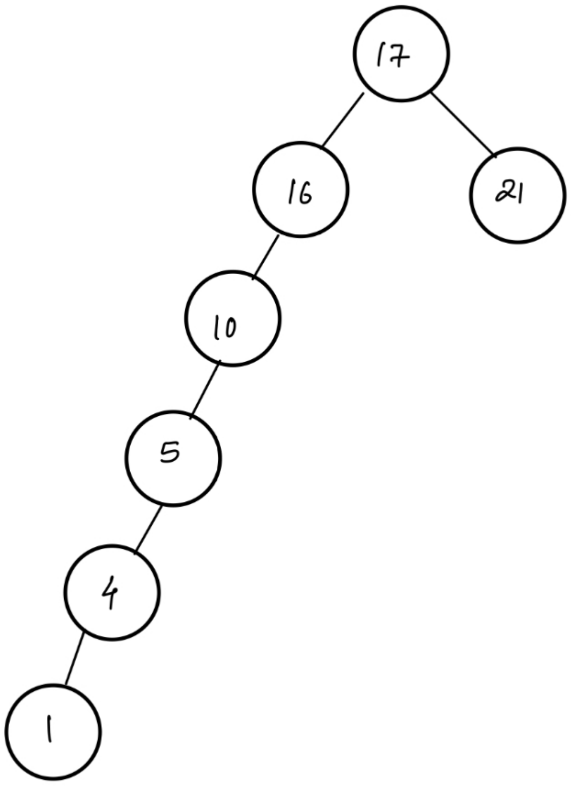
Height 3



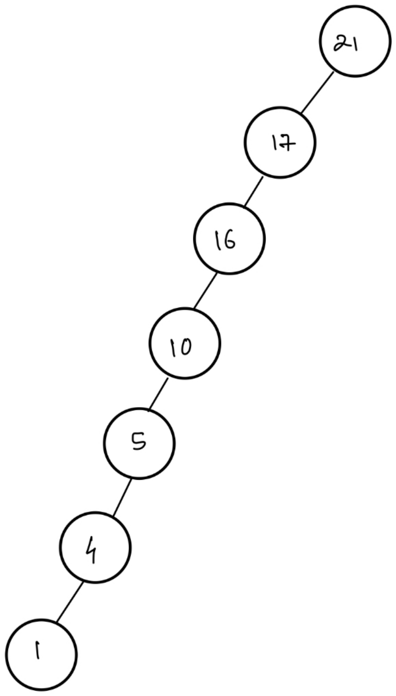
Height 4



Height 5



Height 6



6. (10 points)

Solution:

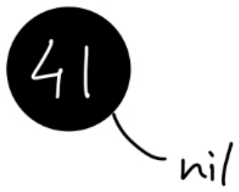
Setting the newly inserted node *z* as black would immediately impact the balance of the red black tree.

This is because the property 5 would be violated, where all the paths leading from the root to the leaves have the same number of black nodes. However this could be adjusted, by rotating the tree and doing color changes, but would make the whole operation more complex and time consuming, indirectly affecting more parts of the tree.

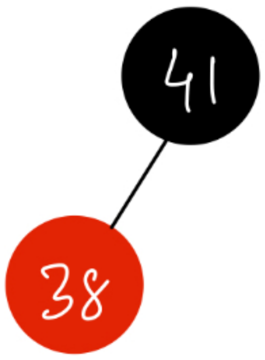
7. (18 points)

Solution:

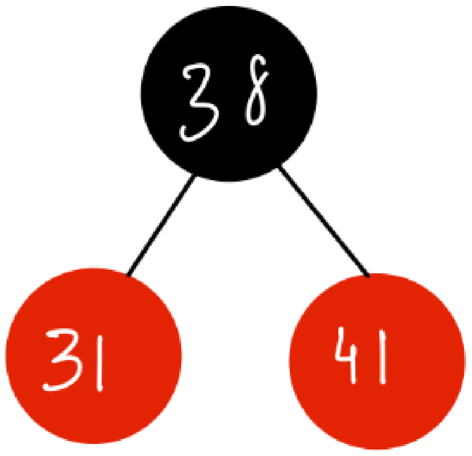
RBT after inserting 4



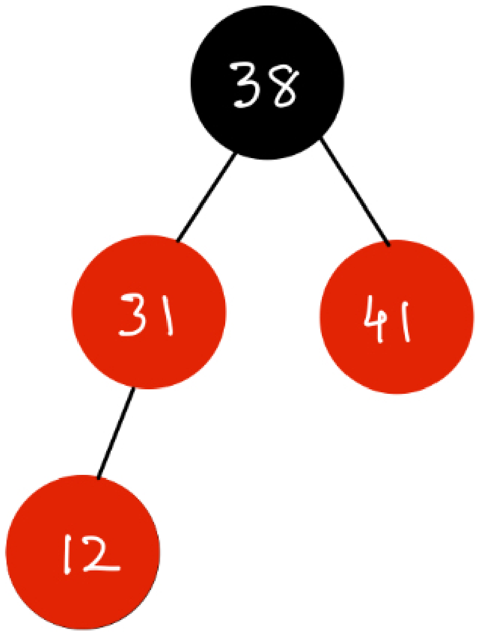
RBT after inserting 38



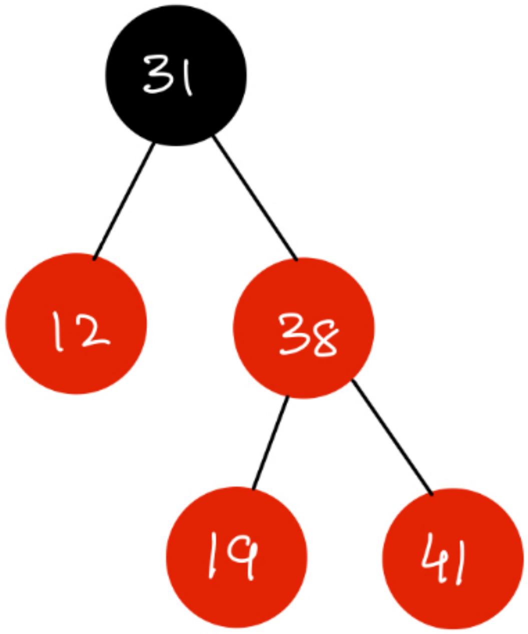
RBT after inserting 31



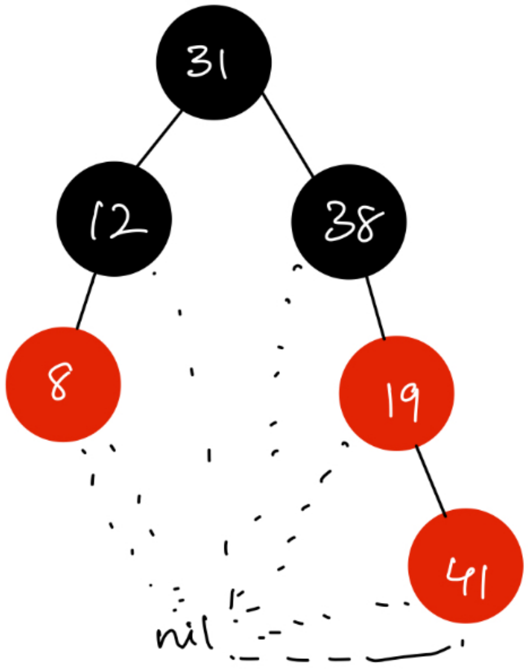
RBT after inserting 12



RBT after inserting 19



RBT after inserting 8


31
12
38
8
19
41
nil