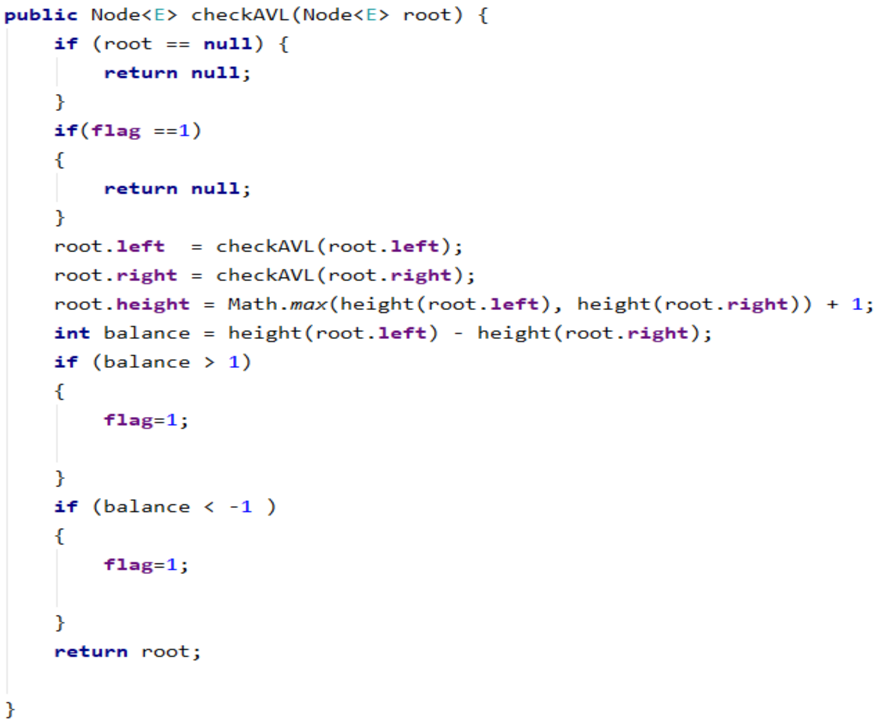
GTU Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework 7
Part 2 Report

Refik Orkun Arslan
151044063

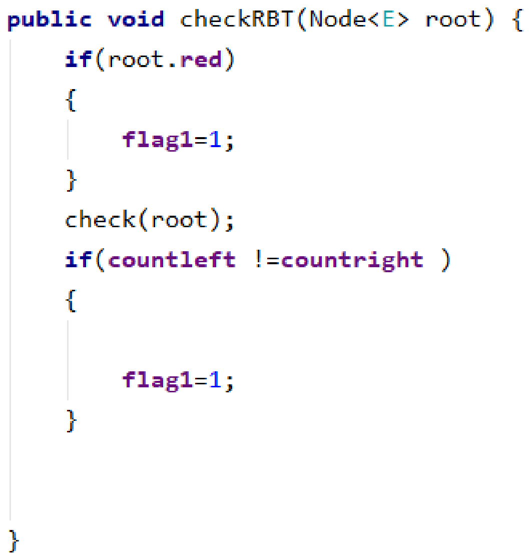
Detailed system requirements



```
public Node<E> checkAVL(Node<E> root) {
    if (root == null) {
        return null;
    }
    if(flag ==1)
    {
        return null;
    }
    root.left  = checkAVL(root.left);
    root.right = checkAVL(root.right);
    root.height = Math.max(height(root.left), height(root.right)) + 1;
    int balance = height(root.left) - height(root.right);
    if (balance > 1)
    {
        flag=1;

    }
    if (balance < -1 )
    {
        flag=1;

    }
    return root;

}
```

avl tree control method

Calculated the balance. When the balance is more than 1 and -1, we can say that the balance is broken, not avl tree.

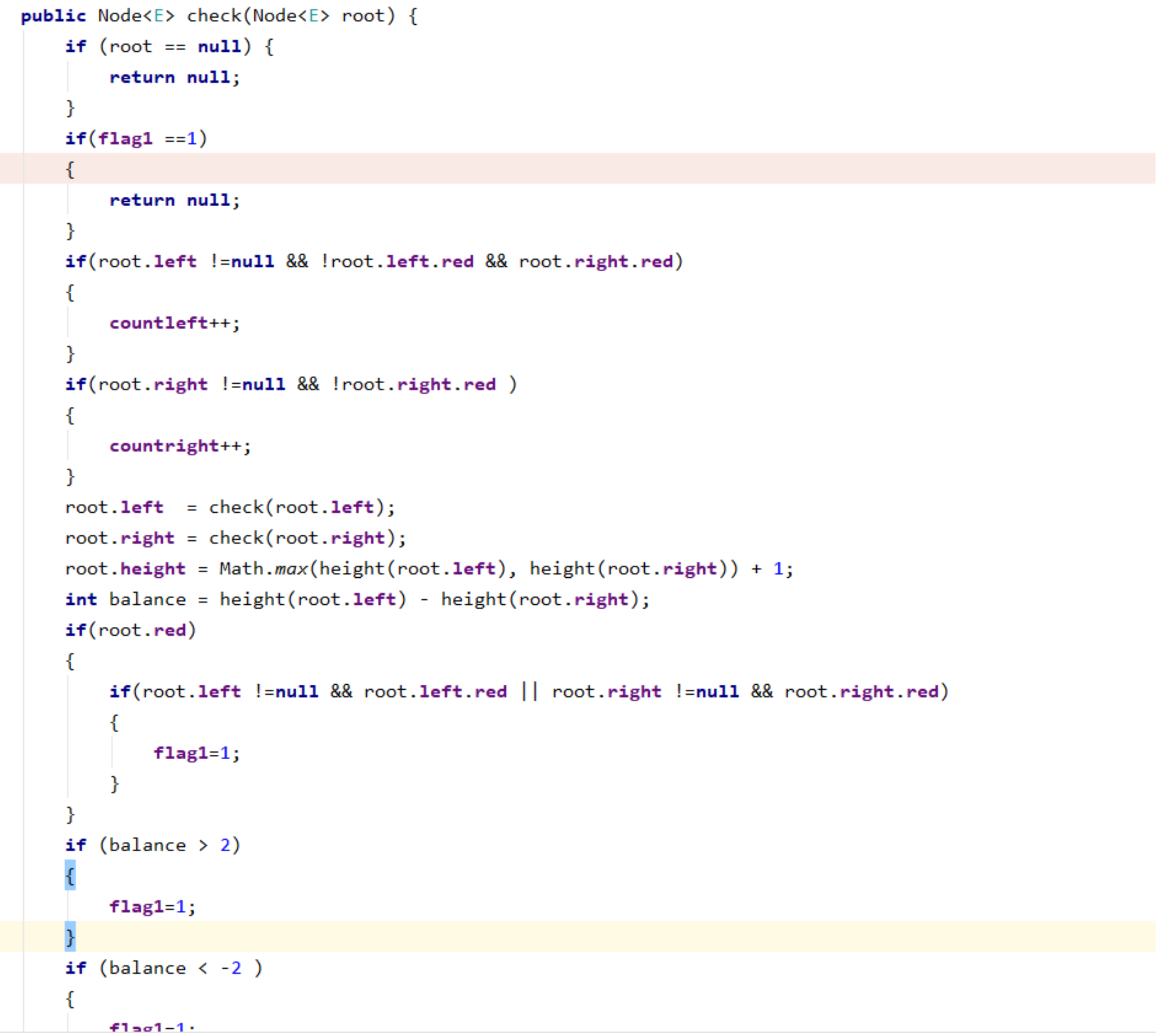


```
public void checkRBT(Node<E> root) {
    if(root.red)
    {
        flag1=1;
    }
    check(root);
    if(countleft !=countright )
    {

        flag1=1;
    }



}
```

first we check the root red is wrong

call check function

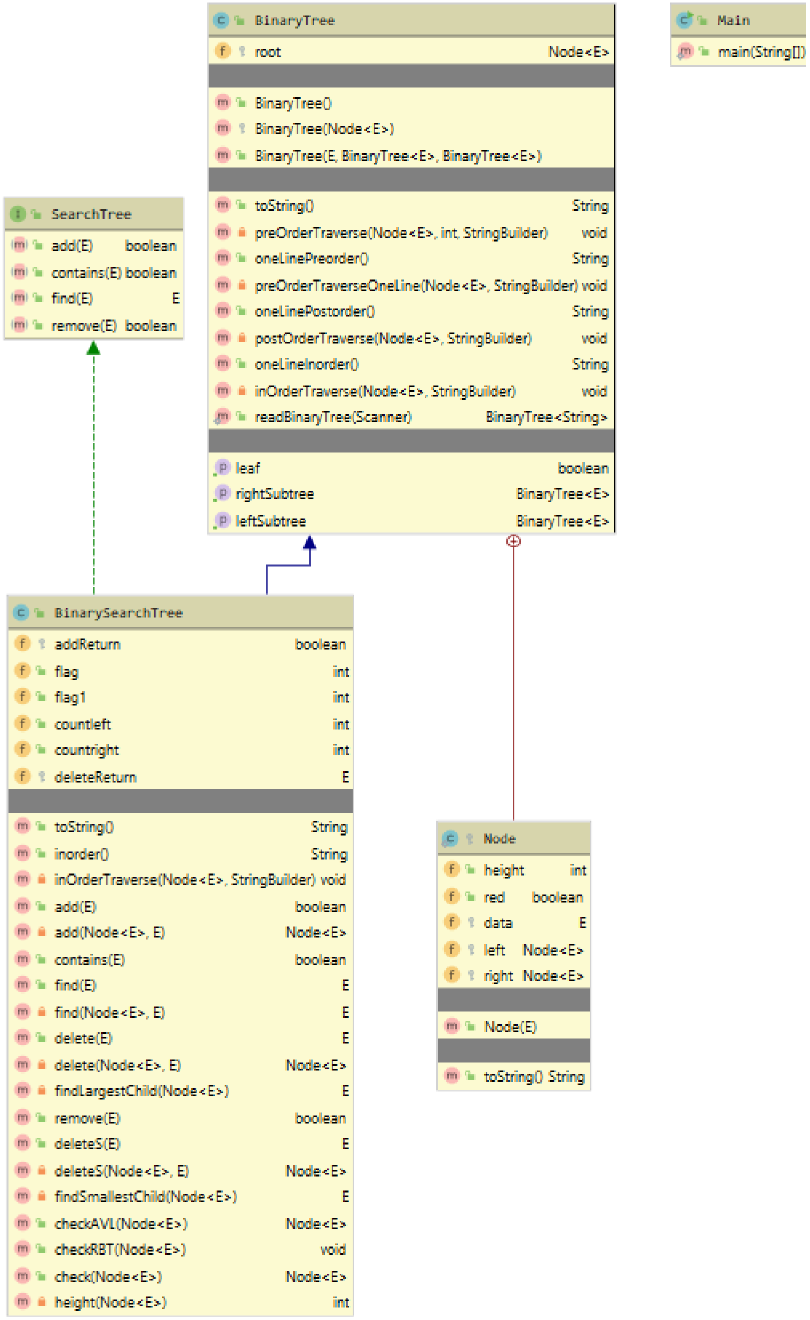
then we check if the black nodes are equal on each path



```
public Node<E> check(Node<E> root) {
    if (root == null) {
        return null;
    }
    if(flag1 ==1)
    {
        return null;
    }
    if(root.left !=null && !root.left.red && root.right.red)
    {
        countleft++;
    }
    if(root.right !=null && !root.right.red )
    {
        countright++;
    }
    root.left  = check(root.left);
    root.right = check(root.right);
    root.height = Math.max(height(root.left), height(root.right)) + 1;
    int balance = height(root.left) - height(root.right);
    if(root.red)
    {
        if(root.left !=null && root.left.red || root.right !=null && root.right.red)
        {
            flag1=1;
        }
    }
    if (balance > 2)
    {
        flag1=1;
    }
    if (balance < -2 )
    {
        flag1=1;
```

countleft and countright count black node in the path
Calculated the balance. When the balance is more than 2 and -2, we can say that the balance is broken, red black tree.

CLASS DİAGRAM



Problem solutions approach

1.The root node has zero, one or two child nodes.
2.Each child node has zero, one or two child nodes, and so on.
3.Each node has up to two children.
4.For each node, its left descendants are less than the current node, which is less than the right descendants.
That rules Avl tree. Implementation that complies with these conditions

Rules That Every Red-Black Tree Follows:

1.Every node has a colour either red or black.

2.The root of the tree is always black.

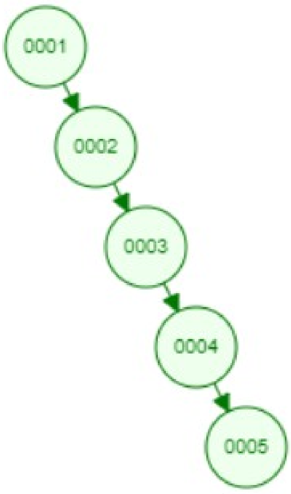
3.There are no two adjacent red nodes (A red node cannot have a red parent or red child).

4.Every path from a node (including root) to any of its descendants NULL nodes has the same number of black nodes.

That rules red-black tree. Implementation that complies with these conditions

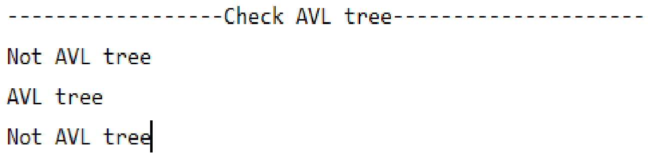
Test cases ,Running command and results

AVL TREE

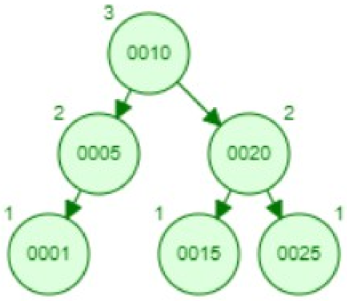


```
bb.add(1);
bb.add(2);
bb.add(3);
bb.add(4);
bb.add(5);
```

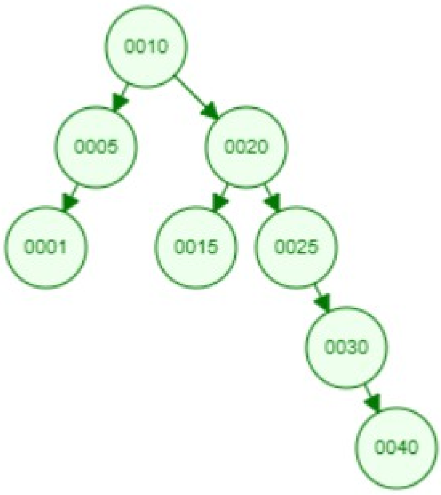
OUTPUT



```
------------------Check AVL tree--------------------
Not AVL tree
AVL tree
Not AVL tree
```

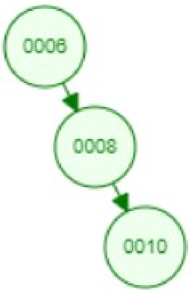


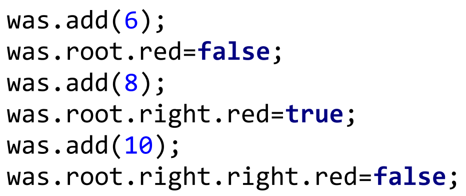
```
tt.add(10);

tt.add(20);
tt.add(5);
tt.add(25);
tt.add(15);
tt.add(1);
```



```
ta.add(10);
ta.add(20);
ta.add(5);
ta.add(25);
ta.add(15);
ta.add(1);
ta.add(30);
ta.add(40);
```

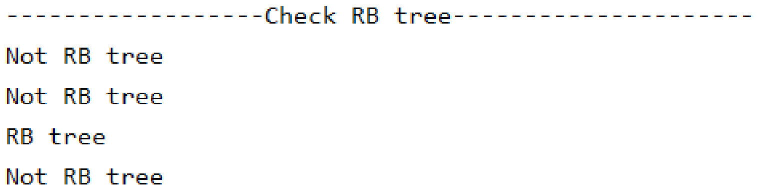
RB TREE



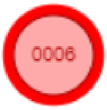


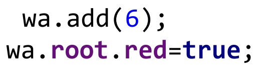
```
was.add(6);
was.root.red=false;
was.add(8);
was.root.right.red=true;
was.add(10);
was.root.right.right.red=false;
```

OUTPUT

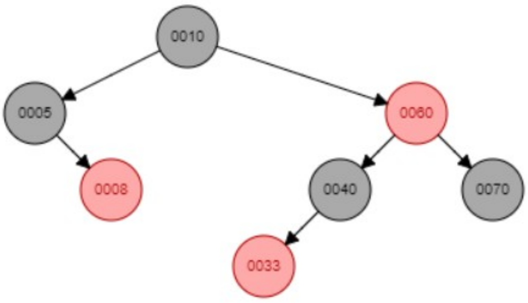


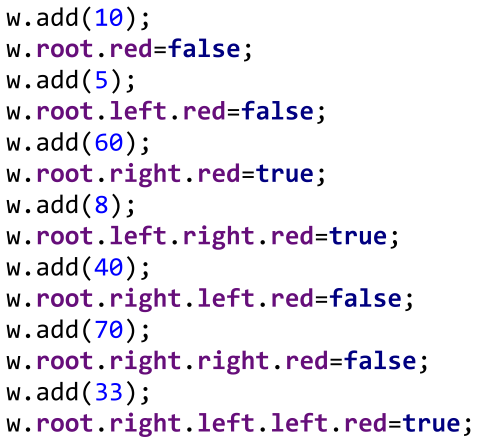
```
------------------Check RB tree---------------------
Not RB tree
Not RB tree
RB tree
Not RB tree
```



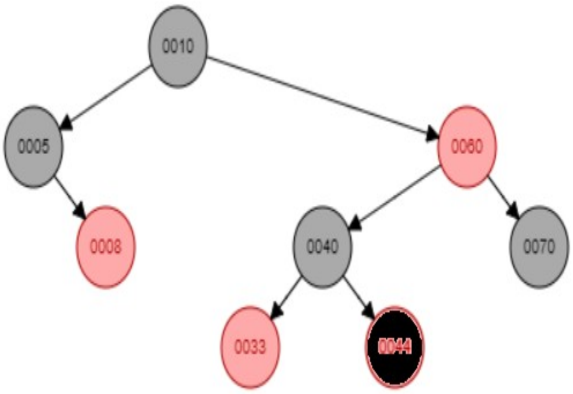


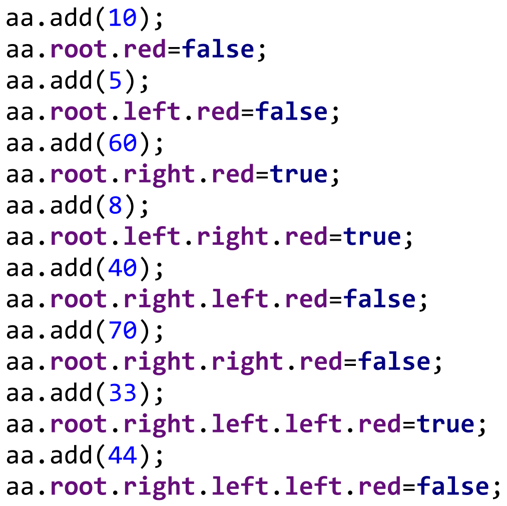
```
 wa.add(6);
wa.root.red=true;
```





```
w.add(10);
w.root.red=false;
w.add(5);
w.root.left.red=false;
w.add(60);
w.root.right.red=true;
w.add(8);
w.root.left.right.red=true;
w.add(40);
w.root.right.left.red=false;
w.add(70);
w.root.right.right.red=false;
w.add(33);
w.root.right.left.left.red=true;
```





```
aa.add(10);
aa.root.red=false;
aa.add(5);
aa.root.left.red=false;
aa.add(60);
aa.root.right.red=true;
aa.add(8);
aa.root.left.right.red=true;
aa.add(40);
aa.root.right.left.red=false;
aa.add(70);
aa.root.right.right.red=false;
aa.add(33);
aa.root.right.left.left.red=true;
aa.add(44);
aa.root.right.left.left.red=false;
```