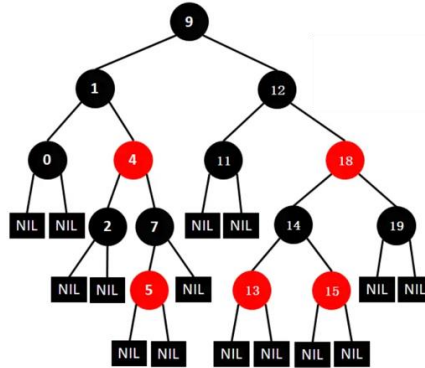



Assignment-2

Due: 11:59 pm on Wednesday (03/16/2022)

Implement insert and delete functions in red black tree based on the pseudo code in textbook. Choose any language as you like. Copy your code in this report (add comments to important instructions). Print out the tree structures when you do any changes to the tree (e.g., rotation and recoloring). The tree structures should be generated by your code and shown by screenshots.

1. Insert 16
2. Delete 1



Note – I have executed deletion operation after insertion operation. I have mentioned all comments in code section. I have tried to print each operation in graphical way using some online resources and reference book, I hope you like it. I have attached code at the end of this file. And, please ignore this symbol -  from output images.

Code –

```
import java.io.*;
```

```
import java.util.*;
```

```
class LineForNode
```

$$\{$$

```
LineForNode prev;
```

String str;

LineForNode(LineForNode prev, String str)

```
this.prev = prev;
```

```
this.str = str;
```

$$\}$$

```
class Node {
```

```
public int data;  
public Node right;  
public Node left;  
public Node parent;  
public int color;
```

```
public Node(int data) {  
    this.left = null;  
    this.right = null;  
    this.parent = null;  
    this.data = data;  
    this.color = 0;  
}  
}
```

```
class Main {
```

```
    Node root;  
    Node NIL;
```

```
public Main() {  
    Node nilNode = new Node(0);  
    nilNode.color = 1;  
    this.NIL = nilNode;  
    this.root = this.NIL;  
}
```

```
public void leftRotation(Node x){  
    Node y = x.right;
```

```
    // turn y's left subtree into x's right subtree  
    x.right = y.left;
```

```
// if y's left subtree is not empty, then x become parent of the subtree's root
```

```
if(y.left != this.NIL) {
```

```
    y.left.parent = x;
```

```
}
```

```
// x's parent becomes y's parent
```

```
y.parent = x.parent;
```

```
// if x was root, then y becomes the root
```

```
if(x.parent == this.NIL) {
```

```
    this.root = y;
```

```
}
```

```
// if x was left child, then y becomes the left child
```

```
else if(x == x.parent.left) {
```

```
    x.parent.left = y;
```

```
}
```

```
// if x was right child, then y becomes the right child
```

```
else {
```

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

```
}
```

```
// Make x becomes y's left child and make x's parent to y
```

```
y.left = x;
```

```
x.parent = y;
```

```
}
```

```
public void rightRotation(Node x) {
```

```
    Node y = x.left;
```

```
// turn y's right subtree into x's left subtree
```

```
x.left = y.right;
```

```
// if y's right subtree is not empty, then x become parent of the subtree's root
```

```
if(y.right != this.NIL) {
```

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

```
}
```

```
// x's parent becomes y's parent
```

```
y.parent = x.parent;
```

```
// if x was root, then y becomes the root
```

```
if(x.parent == this.NIL) {
```

```
    this.root = y;
```

```
}
```

```
// if x was right child, then y becomes the right child
```

```
else if(x == x.parent.right) {
```

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

```
}
```

```
// if x was left child, then y becomes the left child
```

```
else {
```

```
    x.parent.left = y;
```

```
}
```

```
// Make x becomes y's right child and make x's parent to y
```

```
y.right = x;
```

```
x.parent = y;
```

```
}
```

```
public void insertFixup(Node z) {
```

```
    while(z.parent.color == 0) {
```

```
        // check z's parent a left child
```

```
        if(z.parent == z.parent.parent.left) {
```

```
// make y to z's uncle
```

```
Node y = z.parent.parent.right;
```

```
// case 1 - if z's "uncle" (y) is red, then change parent of z to black, change uncle color to black and change grandparent to red and make grandparent as current node
```

```
if(y.color == 0) {  
    z.parent.color = 1;  
    y.color = 1;  
    z.parent.parent.color = 0;  
    z = z.parent.parent;  
    System.out.println("\n");  
    System.out.println("\n");  
    System.out.println("case 1 - if z's "uncle" (y) is red");  
    System.out.println("\n");  
    System.out.println("\n");  
    printTree(z, null, false);  
    System.out.println("\n");  
    System.out.println("\n");  
}
```

```
else {
```

```
// case 2 - if z's "uncle" (y) is black and z is a right child, then perform left rotation
```

```
if(z == z.parent.right) {  
    z = z.parent;  
    leftRotation(z);  
    System.out.println("\n");  
    System.out.println("\n");  
    System.out.println("case 2 - if z's "uncle" (y) is black and z is a right child");  
    System.out.println("\n");  
    System.out.println("\n");  
    printTree(z.parent.parent, null, false);  
    System.out.println("\n");  
    System.out.println("\n");  
}
```

```
}
```

```
// case 3 - if z's "uncle" (y) is black and z is a left child, then change color of parent to black and  
grandparent color to red, then perform right rotation
```

```
z.parent.color = 1;
```

```
z.parent.parent.color = 0;
```

```
rightRotation(z.parent.parent);
```

```
System.out.println("\n");
```

```
System.out.println("\n");
```

```
System.out.println("case 3 - if z's "uncle" (y) is black and z is a left child");
```

```
System.out.println("\n");
```

```
System.out.println("\n");
```

```
printTree(z.parent, null, false);
```

```
System.out.println("\n");
```

```
System.out.println("\n");
```

```
}
```

```
}
```

```
// same step as above, this one for left child
```

```
else {
```

```
Node y = z.parent.parent.left;
```

```
if(y.color == 0) {
```

```
z.parent.color = 1;
```

```
y.color = 1;
```

```
z.parent.parent.color = 0;
```

```
z = z.parent.parent;
```

```
System.out.println("\n");
```

```
System.out.println("\n");
```

```
System.out.println("case 1 - if z's "uncle" (y) is red");
```

```
System.out.println("\n");
```

```
System.out.println("\n");
```

```
printTree(z, null, false);
```

```
System.out.println("\n");
```

```

    System.out.println("\n");
}
else {
    if(z == z.parent.left) {
        z = z.parent;
        rightRotation(z);
        System.out.println("\n");
        System.out.println("\n");
        System.out.println("case 2 - if z's “uncle” (y) is black and z is a right child");
        System.out.println("\n");
        System.out.println("\n");
        printTree(z.parent.parent, null, false);
        System.out.println("\n");
        System.out.println("\n");
    }
    z.parent.color = 1;
    z.parent.parent.color = 0;
    leftRotation(z.parent.parent);
    System.out.println("\n");
    System.out.println("\n");
    System.out.println("case 3 - if z's “uncle” (y) is black and z is a left child");
    System.out.println("\n");
    System.out.println("\n");
    printTree(z.parent, null, false);
    System.out.println("\n");
    System.out.println("\n");
}
}
}
this.root.color = 1;
}

```

```

public void insert(Node z) {
    //variable for the parent of the added node
    Node y = this.NIL;
    Node temp = this.root;

    while(temp != this.NIL) {
        y = temp;
        if(z.data < temp.data)
            temp = temp.left;
        else
            temp = temp.right;
    }
    z.parent = y;

    // if no node is available in tree, then new node will be a tree
    if(y == this.NIL) {
        this.root = z;
    }

    // if data of child is less than its parent, then new node will be left child
    else if(z.data < y.data)
        y.left = z;

    // if data of child is more than its parent, then new node will be right child
    else
        y.right = z;

    z.right = this.NIL;
    z.left = this.NIL;

    // after insertion, cal fixup to follow red black tree rules
    insertFixup(z);
}

```



```

public void delete(Node z) {
    Node y = z;
    Node x;
    int yOriginalColor = y.color;
    if(z.left == this.NIL) {
        x = z.right;
        // replace z by its right child
        rbTransplant(z, z.right);
    }
    else if(z.right == this.NIL) {
        x = z.left;
        // replace z by its left child
        rbTransplant(z, z.left);
    }
    else {
        // make y is z's successor. minimum function will return next leftmost child
        y = minimum(z.right);
        yOriginalColor = y.color;
        x = y.right;
        // check if y is null, child is not available
        if(y.parent == z) {
            x.parent = z;
        }
        // check if y is farther down the tree
        else {
            // replace y by its right child
            rbTransplant(y, y.right);
            // z's right child becomes y's right child
            y.right = z.right;
            y.right.parent = y;
        }
    }
}

```

```

        // replace z by its successor y and give z's left child to y
rbTransplant(z, y);
y.left = z.left;
y.left.parent = y;
y.color = z.color;
}
if(yOriginalColor == 1)
    // after deletion, we need to fix tree
    deleteFixup(x);
}

```

```

public void rbTransplant(Node u, Node v) {
    if(u.parent == this.NIL)
        this.root = v;
    else if(u == u.parent.left)
        u.parent.left = v;
    else
        u.parent.right = v;
    v.parent = u.parent;
}

```

```

public Node minimum(Node x) {
    while(x.left != this.NIL)
        x = x.left;
    return x;
}

```

```

public void deleteFixup(Node x) {
    while(x != this.root && x.color == 1) {
        // check if x is left child
        if(x == x.parent.left) {
            // assign w to x's sibling

```

```
Node w = x.parent.right;
```

```
// Case 1 - if x's sibling w is red, then switch the colors of w and x.p and perform leftrotation
```

```
if(w.color == 0) {
```

```
    w.color = 1;
```

```
    x.parent.color = 0;
```

```
    leftRotation(x.parent);
```

```
    w = x.parent.right;
```

```
    System.out.println("\n");
```

```
    System.out.println("\n");
```

```
    System.out.println("Case 1 - if x's sibling w is red");
```

```
    System.out.println("\n");
```

```
    System.out.println("\n");
```

```
    printTree(x.parent.parent, null, false);
```

```
    System.out.println("\n");
```

```
    System.out.println("\n");
```

```
}
```

```
// Case 2 - if x's sibling w is black, and both of w's children are black, then change color of parent  
and sibling
```

```
if(w.left.color == 1 && w.right.color == 1) {
```

```
    w.color = 0;
```

```
    x = x.parent;
```

```
    System.out.println("\n");
```

```
    System.out.println("\n");
```

```
    System.out.println("Case 2 - if x's sibling w is black");
```

```
    System.out.println("\n");
```

```
    System.out.println("\n");
```

```
    printTree(x, null, false);
```

```
    System.out.println("\n");
```

```
    System.out.println("\n");
```

```
}
```

```
// Case 3 - if x's sibling w is black, w's left child is red, and w's right child is black, then switch the  
colors of w and its left child w.left and perform right rotation
```

```

else {
    if(w.right.color == 1) {
        w.left.color = 1;
        w.color = 0;
        rightRotation(w);
        w = x.parent.right;
        System.out.println("\n");
        System.out.println("\n");
        System.out.println("Case 3 - if x's sibling w is black, w's left child is red, and w's right child is
black");
        System.out.println("\n");
        System.out.println("\n");
        printTree(w.parent, null, false);
        System.out.println("\n");
        System.out.println("\n");
    }

    // Case 4 - if x's sibling w is black, and w's right child is red, then perform left rotation on x.p
    w.color = x.parent.color;
    x.parent.color = 1;
    w.right.color = 1;
    leftRotation(x.parent);
    x = this.root;
    System.out.println("\n");
    System.out.println("\n");
    System.out.println("Case 4 - if x's sibling w is black, and w's right child is red");
    System.out.println("\n");
    System.out.println("\n");
    printTree(x, null, false);
    System.out.println("\n");
    System.out.println("\n");
}
}

```

```

// Repeat above steps for left subtree
else {
    Node w = x.parent.left;
    if(w.color == 0) {
        w.color = 1;
        x.parent.color = 0;
        rightRotation(x.parent);
        w = x.parent.left;
        System.out.println("\n");
        System.out.println("\n");
        System.out.println("Case 1 - if x's sibling w is red");
        System.out.println("\n");
        System.out.println("\n");
        printTree(x.parent.parent, null, false);
        System.out.println("\n");
        System.out.println("\n");
    }
    if(w.right.color == 1 && w.left.color == 1) {
        w.color = 0;
        x = x.parent;
        System.out.println("\n");
        System.out.println("\n");
        System.out.println("Case 2 - if x's sibling w is black");
        System.out.println("\n");
        System.out.println("\n");
        printTree(x, null, false);
        System.out.println("\n");
        System.out.println("\n");
    }
    else {
        if(w.left.color == 1) {
            w.right.color = 1;

```

```

    w.color = 0;
    leftRotation(w);
    w = x.parent.left;
    System.out.println("\n");
    System.out.println("\n");
    System.out.println("Case 3 - if x's sibling w is black, w's left child is red, and w's right child is
black");
    System.out.println("\n");
    System.out.println("\n");
    printTree(w.parent, null, false);
    System.out.println("\n");
    System.out.println("\n");
}
w.color = x.parent.color;
x.parent.color = 1;
w.left.color = 1;
rightRotation(x.parent);
x = this.root;
System.out.println("\n");
System.out.println("\n");
System.out.println("Case 4 - if x's sibling w is black, and w's right child is red");
System.out.println("\n");
System.out.println("\n");
printTree(x, null, false);
System.out.println("\n");
System.out.println("\n");
}
}
}
x.color = 1;
}

```

// below function is used to print tree in graphical format

```
public static void showLineForNodes(LineForNode p)
```

```
{
```

```
    if (p == null) {
```

```
        return;
```

```
    }
```

```
    // print line for node in recursive way
```

```
    showLineForNodes(p.prev);
```

```
    System.out.print(p.str);
```

```
}
```

```
public static void printTree(Node root, LineForNode prev, boolean isLeft)
```

```
{
```

```
    if (root == null) {
```

```
        return;
```

```
    }
```

```
    String prev_str = "  ";
```

```
    LineForNode LineForNode = new LineForNode(prev, prev_str);
```

```
    printTree(root.right, LineForNode, true);
```

```
    if (prev == null) {
```

```
        LineForNode.str = "———";
```

```
    }
```

```
    else if (isLeft) {
```

```
        LineForNode.str = ".———";
```

```
        prev_str = "  |";
```

```
    }
```

```
    else {
```

```
        LineForNode.str = "^———";
```

```
        prev.str = prev_str;
```

```
}
```

```
showLineForNodes(LineForNode);
```

```
if(root.color == 0)
```

```
{
```

```
    System.out.println(" " + root.data+"(R)");
```

```
}
```

```
else
```

```
{
```

```
    System.out.println(" " + root.data+"(B)");
```

```
}
```

```
if (prev != null) {
```

```
    prev.str = prev_str;
```

```
}
```

```
LineForNode.str = "  |";
```

```
printTree(root.left, LineForNode, false);
```

```
}
```

```
public static void main(String[] args) {
```

```
    Main t = new Main();
```

```
    // create a tree as per question format
```

```
    Node a = new Node(9);
```

```
    Node b = new Node(1);
```

```
    Node c = new Node(12);
```

```
    Node d = new Node(0);
```

```
    Node e = new Node(4);
```

```
    Node f = new Node(11);
```

```
    Node g = new Node(18);
```

```
    Node h = new Node(2);
```



```
Node i = new Node(7);  
Node j = new Node(14);  
Node k = new Node(19);  
Node l = new Node(5);  
Node m = new Node(13);  
Node n = new Node(15);
```

```
t.insert(a);  
t.insert(b);  
t.insert(c);  
t.insert(d);  
t.insert(e);  
t.insert(f);  
t.insert(g);  
t.insert(h);  
t.insert(i);  
t.insert(j);  
t.insert(k);  
t.insert(l);  
t.insert(m);  
t.insert(n);
```

```
System.out.println("Our Input Tree");  
System.out.println("\n");  
System.out.println("\n");  
t.printTree(t.root, null, false);  
System.out.println("\n");  
System.out.println("\n");
```

```
System.out.println("Operation 1");
```

```
System.out.println("\n");  
Node o = new Node(16);  
t.insert(o);  
System.out.println("After inserting node 16");  
t.printTree(t.root, null, false);  
System.out.println("\n");  
System.out.println("\n");
```

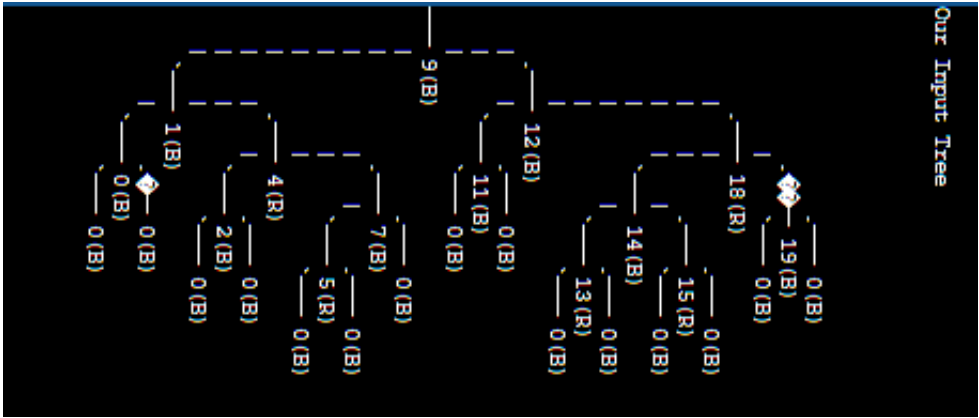
```
System.out.println("Operation 2");  
System.out.println("\n");  
t.delete(b);  
System.out.println("After deleting node 1");  
t.printTree(t.root, null, false);
```

```
}
```

```
}
```

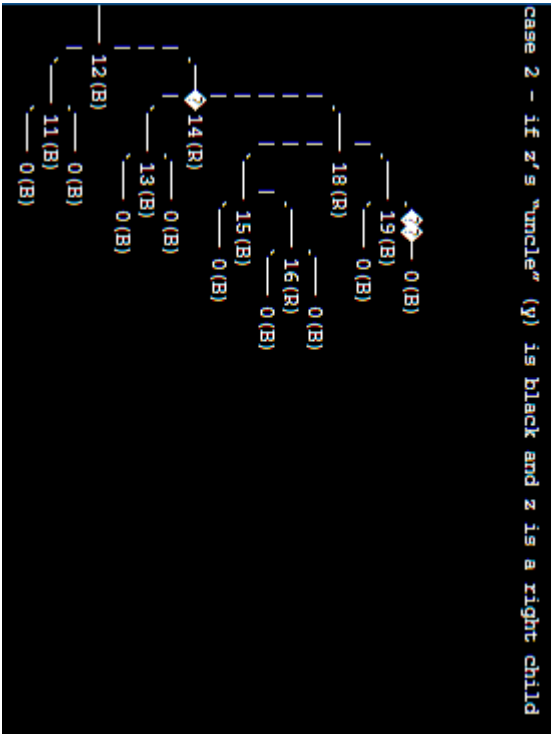
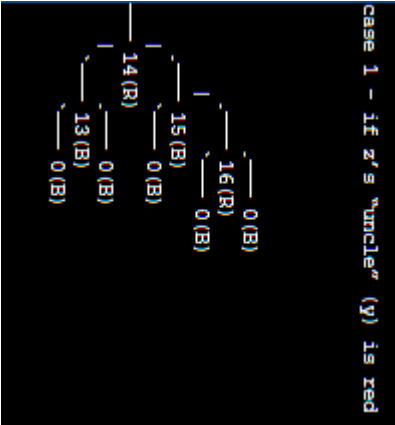
Output –

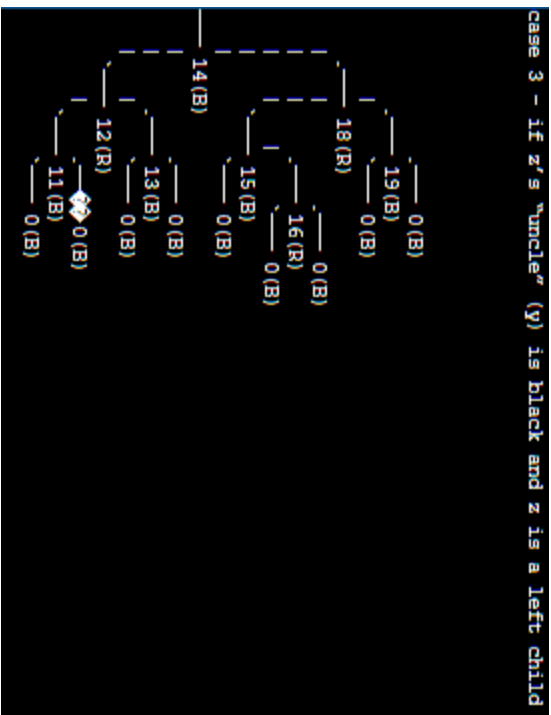
Input Tree -



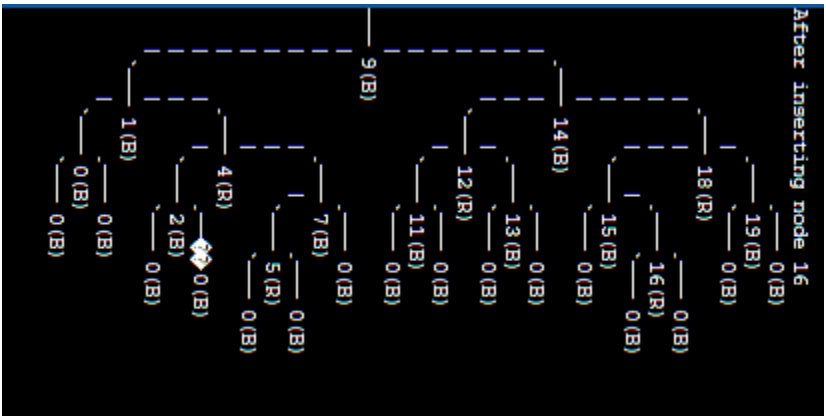
Operation 1 –

(Below images shows sequence of execution occurs in insertion process such as which case is executed like case1 or case2.)



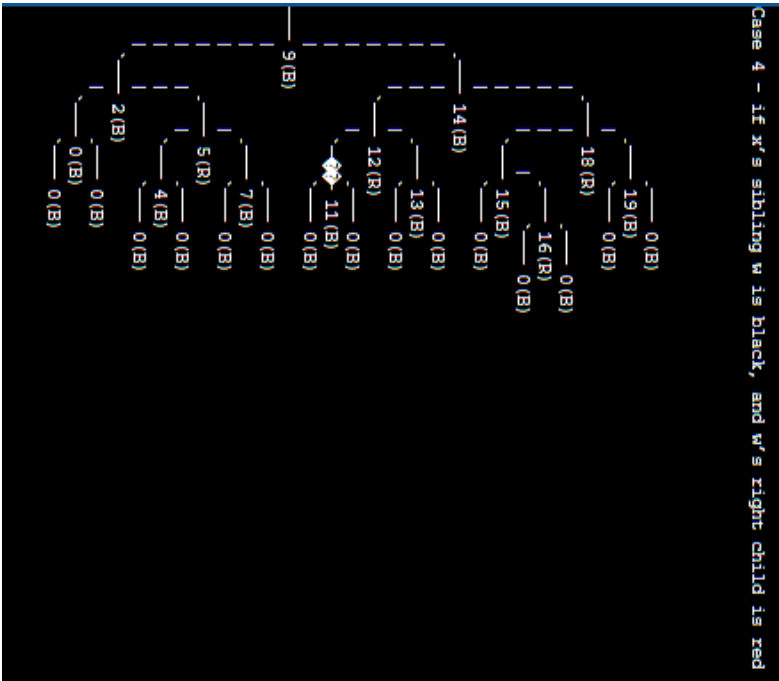


Final Output after insertion operation –



Operation 2 – (Note – I have executed deletion operation after insertion operation)

(Below images shows sequence of execution occurs in deletion process such as which case is executed like case1 or case2.)



Final Output after deletion operation –

