import java.util.\*;

enum Color {

Red, Black

}

class TreeNode {

public int data;

public TreeNode right;

public TreeNode left;

public TreeNode parent;

public Color color;

public TreeNode(int data) {

this.left = null;

this.right = null;

this.parent = null;

this.data = data;

this.color = Color.Red;

}

}

class RedBlackTree {

TreeNode root;

TreeNode NIL;

public RedBlackTree() {

TreeNode nilNode = new TreeNode(0);

nilNode.color = Color.Black;

this.NIL = nilNode;

this.root = this.NIL;

}

public void leftRotate(TreeNode x) {

TreeNode y = x.right;

x.right = y.left;

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

y.left.parent = x;

}

y.parent = x.parent;

if (x.parent == this.NIL) { // x is root

this.root = y;

} else if (x == x.parent.left) { // x is left child

x.parent.left = y;

} else { // x is right child

x.parent.right = y;

}

y.left = x;

x.parent = y;

}

public void rightRotate(TreeNode x) {

TreeNode y = x.left;

x.left = y.right;

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

y.right.parent = x;

}

y.parent = x.parent;

if (x.parent == this.NIL) { // x is root

this.root = y;

} else if (x == x.parent.right) { // x is left child

x.parent.right = y;

} else { // x is right child

x.parent.left = y;

}

y.right = x;

x.parent = y;

}

public void insertFixup(TreeNode z) {

while (z.parent.color == Color.Red) {

if (z.parent == z.parent.parent.left) { // z.parent is the left child

TreeNode y = z.parent.parent.right; // uncle of z

if (y.color == Color.Red) { // case 1

z.parent.color = Color.Black;

y.color = Color.Black;

z.parent.parent.color = Color.Red;

z = z.parent.parent;

} else { // case2 or case3

if (z == z.parent.right) { // case2

z = z.parent; // marked z.parent as new z

leftRotate(z);

}

// case3

z.parent.color = Color.Black; // made parent black

z.parent.parent.color = Color.Red; // made parent red

rightRotate(z.parent.parent);

}

} else { // z.parent is the right child

TreeNode y = z.parent.parent.left; // uncle of z

if (y.color == Color.Red) {

z.parent.color = Color.Black;

y.color = Color.Black;

z.parent.parent.color = Color.Red;

z = z.parent.parent;

} else {

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

z = z.parent; // marked z.parent as new z

rightRotate(z);

}

z.parent.color = Color.Black; // made parent black

z.parent.parent.color = Color.Red; // made parent red

leftRotate(z.parent.parent);

}

}

}

this.root.color = Color.Black;

}

public void insert(TreeNode z) {

TreeNode y = this.NIL; // variable for the parent of the added node

TreeNode 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 (y == this.NIL) { // newly added node is root

this.root = z;

} else if (z.data < y.data) // data of child is less than its parent, left child

y.left = z;

else

y.right = z;

z.right = this.NIL;

z.left = this.NIL;

insertFixup(z);

}

public void rbTransplant(TreeNode u, TreeNode 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 TreeNode minimum(TreeNode x) {

while (x.left != this.NIL)

x = x.left;

return x;

}

public void deleteFixup(TreeNode x) {

while (x != this.root && x.color == Color.Black) {

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

TreeNode w = x.parent.right;

if (w.color == Color.Red) {

w.color = Color.Black;

x.parent.color = Color.Red;

leftRotate(x.parent);

w = x.parent.right;

}

if (w.left.color == Color.Black && w.right.color == Color.Black) {

w.color = Color.Red;

x = x.parent;

} else {

if (w.right.color == Color.Black) {

w.left.color = Color.Black;

w.color = Color.Red;

rightRotate(w);

w = x.parent.right;

}

w.color = x.parent.color;

x.parent.color = Color.Black;

w.right.color = Color.Black;

leftRotate(x.parent);

x = this.root;

}

} else {

TreeNode w = x.parent.left;

if (w.color == Color.Red) {

w.color = Color.Black;

x.parent.color = Color.Red;

rightRotate(x.parent);

w = x.parent.left;

}

if (w.right.color == Color.Black && w.left.color == Color.Black) {

w.color = Color.Red;

x = x.parent;

} else {

if (w.left.color == Color.Black) {

w.right.color = Color.Black;

w.color = Color.Red;

leftRotate(w);

w = x.parent.left;

}

w.color = x.parent.color;

x.parent.color = Color.Black;

w.left.color = Color.Black;

rightRotate(x.parent);

x = this.root;

}

}

}

x.color = Color.Black;

}

public void rbDelete(TreeNode z) {

TreeNode y = z;

TreeNode x;

Color yOrignalColor = y.color;

if (z.left == this.NIL) {

x = z.right;

rbTransplant(z, z.right);

} else if (z.right == this.NIL) {

x = z.left;

rbTransplant(z, z.left);

} else {

y = minimum(z.right);

yOrignalColor = y.color;

x = y.right;

if (y.parent == z) {

x.parent = z;

} else {

rbTransplant(y, y.right);

y.right = z.right;

y.right.parent = y;

}

rbTransplant(z, y);

y.left = z.left;

y.left.parent = y;

y.color = z.color;

}

if (yOrignalColor == Color.Black)

deleteFixup(x);

}

public void inorder(TreeNode n) {

if (n != this.NIL) {

inorder(n.left);

System.out.print(n.data + " ");

inorder(n.right);

}

}

private TreeNode search(TreeNode node, int value) {

TreeNode returnNode = NIL;

while ((node != NIL) && returnNode == NIL) {

int nodeValue = node.data;

if (value < nodeValue)

node = node.left;

else if (value > nodeValue)

node = node.right;

else {

returnNode = node;

break;

}

returnNode = search(node, value);

}

return returnNode;

}

private boolean delete(TreeNode node, int value) {

TreeNode deleteNode = search(node, value);

if (deleteNode != NIL) {

rbDelete(deleteNode);

return true;

} else {

return false;

}

}

public static void main(String[] args) {

RedBlackTree tree = new RedBlackTree();

// menu

System.out.println("1 - Insert");

System.out.println("2 - Delete");

System.out.println("3 - Search");

System.out.println("4 - Display");

System.out.println("5 - Exit");

int choice;

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.print("Enter your choice : ");

choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.println("Inserting elements into Red Black Tree");

System.out.print("Enter number of elements: ");

int n = scanner.nextInt();

System.out.println("Insert elements");

for (int i = 0; i < n; i++)

tree.insert(new TreeNode(scanner.nextInt()));

System.out.println("Elements inserted successfully");

break;

case 2:

System.out.print("Delete Node: ");

int delete\_val = scanner.nextInt();

boolean isDeleted = tree.delete(tree.root, delete\_val);

if (isDeleted)

System.out.println("Node " + delete\_val + " deleted successfully");

else

System.out.println("Node " + delete\_val + " doesn't exist");

break;

case 3:

System.out.print("Search Node: ");

int search\_val = scanner.nextInt();

TreeNode searchNode = tree.search(tree.root, search\_val);

if (searchNode != tree.NIL)

System.out.println("Node " + search\_val + " found");

else

System.out.println("Node " + search\_val + " not found");

break;

case 4:

System.out.println("Inorder traversal of tree is : ");

tree.inorder(tree.root);

System.out.println("");

break;

case 5:

System.exit(0);

default:

System.out.println("Choice is incorrect, Enter a correct choice");

}

}

}

}

// input: 10 20 30 100 90 40 50 60 70 80 150 110 120

// delete: 10

// delete: 120

// inorder: 20 30 40 50 60 70 80 90 100 110 150

Output:

PS Y:\development\ds-java\red-black-tree> java RedBlackTree

1 - Insert

2 - Delete

3 - Search

4 - Display

5 - Exit

Enter your choice : 1

Inserting elements into Red Black Tree

Enter number of elements: 13

Insert elements

10

20

30

100

90

40

50

60

70

80

150

110

120

Elements inserted successfully

Enter your choice : 4

Inorder traversal of tree is :

10 20 30 40 50 60 70 80 90 100 110 120 150

Enter your choice : 3

Search Node: 20

Node 20 found

Enter your choice : 2

Delete Node: 10

Node 10 deleted successfully

Enter your choice : 4

Inorder traversal of tree is :

20 30 40 50 60 70 80 90 100 110 120 150

Enter your choice : 2

Delete Node: 20

Node 20 deleted successfully

Enter your choice : 3

Search Node: 20

Node 20 not found

Enter your choice : 4

Inorder traversal of tree is :

30 40 50 60 70 80 90 100 110 120 150

Enter your choice : 5