**RB TREE :**

package rbtree;

import java.util.Scanner;

/\*\*

\*

\* @author Pradyot Patil

\*/

public class RBtree {

public static void main(String[] args)

{

// TODO code application logic here'

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number of keys:");

int keys=sc.nextInt();

int i;

System.out.println("Enter keys: ");

// Node root=new Node(sc.nextInt(),null,null,0,ni);

Tree t=new Tree();

for(i=0;i<keys;i++)

{

RB\_insert(t,new Node(sc.nextInt(),t.nil,t.nil,0,t.nil));

DFS(t.root,t);

}

while(true)

{

System.out.println("Press 1 to delete,0 to exit");

int ch=sc.nextInt();

if(ch==1)

{

System.out.println("Enter node to delete:");

int del=sc.nextInt();

Node z=Find\_node(t.root,t,del);

System.out.println("z's key"+z.key);

RB\_delete(t,z);

}

else

break;

}

DFS(t.root,t);

}

static void RB\_insert(Tree t,Node z)

{

Node y=t.nil;

Node x=t.root;

while(x!=t.nil)

{

y=x;

if(z.key<x.key)

x=x.left;

else

x=x.right;

}

z.parent=y;

if(y==t.nil)

t.root=z;

else if (z.key<y.key)

y.left=z;

else

y.right=z;

z.left=t.nil;

z.right=t.nil;

z.color=1;

RB\_insert\_fixup(t,z);

}

static void RB\_insert\_fixup(Tree t,Node z)

{

int flag=0;

while(z.parent.color==1)

{

flag=1;

System.out.println("Property violated:inserted node's parent is red.");

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

{

Node y=z.parent.parent.right;

if(y.color==1)

{

z.parent.color=0;

y.color=0;

z.parent.parent.color=1;

z=z.parent.parent;

}

else

{

if(z==z.parent)

{

z=z.parent;

//leftrotate

leftRotate(t, z);

}

z.parent.color=0;

z.parent.parent.color=1;

//rightrotate

rightRotate(t, z.parent.parent);

}

}

else

{

Node y=z.parent.parent.left;

if(y.color==1)

{

z.parent.color=0;

y.color=0;

z.parent.parent.color=1;

z=z.parent.parent;

}

else

{

if(z==z.parent.left)

{

z=z.parent;

//rightrotate

rightRotate(t, z);

}

z.parent.color=0;

z.parent.parent.color=1;

//leftrotate

leftRotate(t, z.parent.parent);

}

}

}

if(flag==0)

System.out.println("Node inserted without any violation.");

t.root.color=0;

}

static void leftRotate(Tree t,Node x)

{

System.out.println("Left Rotated:"+x.key);

Node y;

y=x.right;

x.right=y.left;

if(y.left!=t.nil)

y.left.parent=x;

y.parent=x.parent;

if(x.parent==t.nil)

t.root=y;

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

x.parent.left=y;

else

x.parent.right=y;

y.left=x;

x.parent=y;

}

static void rightRotate(Tree t,Node x)

{

System.out.println("Right Rotated:"+x.key);

Node y;

y=x.left;

x.left=y.right;

if(y.right!=t.nil)

y.right.parent=x;

y.parent=x.parent;

if(x.parent==t.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;

}

static public void DFS(Node r,Tree t)

{

if(r==t.nil)

return;

DFS(r.left,t);

System.out.println("Key:"+r.key+"\tColor:"+r.color+"\tParent:"+r.parent.key);

DFS(r.right,t);

}

static public Node Find\_node(Node r,Tree t,int num)

{

while(r!=t.nil)

{

if(num==r.key)

return r;

else if(num<r.key)

{

r=r.left;

}

else

r=r.right;

}

return null;

}

static public void RB\_delete(Tree t,Node z)

{

Node x;

Node y=z;

int y\_ori\_color=y.color;

if(z.left==t.nil)

{

x=z.right;

RBTransplant(t,z,z.right);

}

else if(z.right==t.nil)

{

x=z.left;

RBTransplant(t,z,z.left);

}

else

{

y=TreeMin(t,z.right);

y\_ori\_color=y.color;

x=y.right;

if(y.parent==z)

x.parent=z;

else

{

RBTransplant(t,y,y.right);

y.right=z.right;

y.right.parent=y;

}

RBTransplant(t,z,y);

y.left=z.left;

y.left.parent=y;

y.color=z.color;

}

if(y\_ori\_color==0)

RBDeleteFixup(t,x);

}

static void RBTransplant(Tree t, Node u, Node v)

{

if(u.parent==t.nil)

t.root=v;

else if(u==u.parent.left)

u.parent.left=v;

else

u.parent.right=v;

v.parent=u.parent;

}

static Node TreeMin(Tree t, Node r)

{

while(r.left!=t.nil)

r=r.left;

return r;

}

private static void RBDeleteFixup(Tree t, Node x)

{

Node w;

while(x!=t.root && x.color==0)

{

if(x==x.parent.left)

{

w=x.parent.right;

if(w.color==1)

{

w.color=0;

x.parent.color=1;

leftRotate(t,x.parent);

w=x.parent.right;

}

if(w.left.color==0 && w.right.color==0)

{

w.color=1;

x=x.parent;

}

else if(w.right.color==0)

{

w.left.color=0;

w.color=1;

rightRotate(t,w);

w=x.parent.right;

}

w.color=x.parent.color;

x.parent.color=0;

w.right.color=0;

leftRotate(t,x.parent);

x=t.root;

}

else

{

w=x.parent.left;

if(w.color==1)

{

w.color=0;

x.parent.color=1;

rightRotate(t,x.parent);

w=x.parent.left;

}

if(w.right.color==0 && w.left.color==0)

{

w.color=1;

x=x.parent;

}

else if(w.left.color==0)

{

w.right.color=0;

w.color=1;

leftRotate(t,w);

w=x.parent.left;

}

w.color=x.parent.color;

x.parent.color=0;

w.left.color=0;

rightRotate(t,x.parent);

x=t.root;

}

}

x.color=0;

}

}

class Tree

{

Node root;

Node nil;

public Tree() {

//nil=null;

nil=new Node(-1,null,null,0,null);

nil.color=0;

root=nil;

}

public Tree(Node root) {

this.root = root;

}

}

class Node

{

int key;

Node left,right;

int color;//1 for red,0 for Black

Node parent;

public Node() {

}

public Node(int key, Node left, Node right, int color, Node parent) {

this.key = key;

this.left = left;

this.right = right;

this.color = color;

this.parent = parent;

}

}

**OUTPUT :**

Enter the number of keys: 8

Enter keys: 6

Node inserted without any violation.

Key:6 Color:0 Parent:-1

-----------------------------------------------------------------------

Enter key: 4

Node inserted without any violation.

Key:4 Color:1 Parent:6

Key:6 Color:0 Parent:-1

-----------------------------------------------------------------------

Enter key: 7

Node inserted without any violation.

Key:4 Color:1 Parent:6

Key:6 Color:0 Parent:-1

Key:7 Color:1 Parent:6

-----------------------------------------------------------------------

Enter key: 2

Property violated:inserted node's parent is red.

Key:2 Color:1 Parent:4

Key:4 Color:0 Parent:6

Key:6 Color:0 Parent:-1

Key:7 Color:0 Parent:6

-----------------------------------------------------------------------

Enter key: 5

Node inserted without any violation.

Key:2 Color:1 Parent:4

Key:4 Color:0 Parent:6

Key:5 Color:1 Parent:4

Key:6 Color:0 Parent:-1

Key:7 Color:0 Parent:6

-----------------------------------------------------------------------

Enter key: 20

Node inserted without any violation.

Key:2 Color:1 Parent:4

Key:4 Color:0 Parent:6

Key:5 Color:1 Parent:4

Key:6 Color:0 Parent:-1

Key:7 Color:0 Parent:6

Key:20 Color:1 Parent:7

-----------------------------------------------------------------------

Enter key: 16

Property violated:inserted node's parent is red.

Right Rotated:20

Left Rotated:7

Key:2 Color:1 Parent:4

Key:4 Color:0 Parent:6

Key:5 Color:1 Parent:4

Key:6 Color:0 Parent:-1

Key:7 Color:1 Parent:16

Key:16 Color:0 Parent:6

Key:20 Color:1 Parent:16

-----------------------------------------------------------------------

Enter key: 17

Property violated:inserted node's parent is red.

Key:2 Color:1 Parent:4

Key:4 Color:0 Parent:6

Key:5 Color:1 Parent:4

Key:6 Color:0 Parent:-1

Key:7 Color:0 Parent:16

Key:16 Color:1 Parent:6

Key:17 Color:1 Parent:20

Key:20 Color:0 Parent:16

Press 1 to delete,0 to exit

1

Enter node to delete: 6

deleted key 6

Right Rotated:20

Left Rotated:16

-----------------------------------------------------------------------

Press 1 to delete,0 to exit

1

Enter node to delete: 4

deleted key 4

-----------------------------------------------------------------------

Press 1 to delete,0 to exit

1

Enter node to delete: 20

deleted key 20

Right Rotated:7

-----------------------------------------------------------------------

Press 1 to delete,0 to exit

0

Key:2 Color:1 Parent:5

Key:5 Color:0 Parent:-1

Key:7 Color:0 Parent:5

Key:16 Color:0 Parent:17

Key:17 Color:1 Parent:7