/\* Name Anay Dombe Roll No-16

Devashish Ubale Roll No-10

Topic: Implementation of Red black tree using multithreading (File handling is used and language used is JAVA)\*/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.Scanner;

import java.lang.Thread;

import java.util.\*;

import java.io.\*;

import java.io.FileInputStream;

class RedBlackNode

{

RedBlackNode left, right;

int element;

int color;

/\* Constructor \*/

public RedBlackNode(int theElement)

{

this( theElement, null, null );

}

/\* parameterised Constructor \*/

public RedBlackNode(int theElement, RedBlackNode lt, RedBlackNode rt)

{

left = lt;

right = rt;

element = theElement;

color = 1;

}

}

/\* Class RBTree \*/

class RBTree

{

public RedBlackNode current;

public RedBlackNode parent;

public RedBlackNode grand;

public RedBlackNode great;

public RedBlackNode header;

public static RedBlackNode nullNode;

/\* static initializer for nullNode \*/

static

{

nullNode = new RedBlackNode(0);

nullNode.left = nullNode;

nullNode.right = nullNode;

}

/\* Black - 1 RED - 0 \*/

static final int BLACK = 1;

static final int RED = 0;

/\* Constructor \*/

public RBTree(int negInf)

{

header = new RedBlackNode(negInf);

header.left = nullNode;

header.right = nullNode;

}

/\* Function to check if tree is empty \*/

public boolean isEmpty()

{

return header.right == nullNode;

}

/\* Make the tree logically empty \*/

public void makeEmpty()

{

header.right = nullNode;

}

/\* Function to insert item \*/

public void insert(int item )

{

// System.out.println(item);

current = parent = grand = header;

nullNode.element = item;

while (current.element != item)

{

great = grand;

grand = parent;

parent = current;

current = item < current.element ? current.left : current.right;

// Check if two red children and fix if so

if (current.left.color == RED && current.right.color == RED)

handleReorient( item );

}

// Insertion fails if already present

if (current != nullNode)

return;

current = new RedBlackNode(item, nullNode, nullNode);

// Attach to parent

if (item < parent.element)

parent.left = current;

else

parent.right = current;

handleReorient( item );

}

public void handleReorient(int item)

{

// Do the color flip

current.color = RED;

current.left.color = BLACK;

current.right.color = BLACK;

if (parent.color == RED)

{

// Have to rotate

grand.color = RED;

if (item < grand.element != item < parent.element)

parent = rotate( item, grand ); // Start dbl rotate

current = rotate(item, great );

current.color = BLACK;

}

// Make root black

header.right.color = BLACK;

}

public RedBlackNode rotate(int item, RedBlackNode parent)

{

if(item < parent.element)

return parent.left = item < parent.left.element ? rotateWithLeftChild(parent.left) : rotateWithRightChild(parent.left) ;

else

return parent.right = item < parent.right.element ? rotateWithLeftChild(parent.right) : rotateWithRightChild(parent.right);

}

/\* Rotate binary tree node with left child \*/

public RedBlackNode rotateWithLeftChild(RedBlackNode k2)

{

RedBlackNode k1 = k2.left;

k2.left = k1.right;

k1.right = k2;

return k1;

}

/\* Rotate binary tree node with right child \*/

public RedBlackNode rotateWithRightChild(RedBlackNode k1)

{

RedBlackNode k2 = k1.right;

k1.right = k2.left;

k2.left = k1;

return k2;

}

public boolean search(int val)

{

return search(header.right, val);

}

private boolean search(RedBlackNode r, int val)

{

boolean found = false;

while ((r != nullNode) && !found)

{

int rval = r.element;

if (val < rval)

r = r.left;

else if (val > rval)

r = r.right;

else

{

found = true;

break;

}

found = search(r, val);

}

return found;

}

/\* Function for inorder traversal \*/

public void inorder()

{

inorder(header.right);

}

public void inorder(RedBlackNode r)

{

if (r != nullNode)

{

inorder(r.left);

char c = 'B';

if (r.color == 0)

c = 'R';

System.out.print(r.element +""+c+" ");

inorder(r.right);

}

}

}

class threading extends Thread

{

int ele;

RedBlackTreeTest rtest=new RedBlackTreeTest();

void array(int element)

{

ele=element;

}

public void run()

{

rtest.temp(ele);

}

}

class RedBlackTreeTest

{

static RBTree rbt = new RBTree(Integer.MIN\_VALUE);

void temp(int el)

{

System.out.println("Search result for "+el+"is :"+ rbt.search(el));

}

public static boolean areAllFalse(boolean[] array)

{

for(boolean b : array)

if(!b)

return true;

return false;

}

public static void main(String args[])throws InterruptedException

{

int elements[]=new int[100];

int n1,i,n2,count;

Scanner scan = new Scanner(System.in);

/\* Creating object of RedBlack Tree \*/

System.out.println("Red Black Tree Test\n");

/\*

System.out.println("Enter the number of elements to be inserted");

n1=scan.nextInt();

for(i=0;i<n1;i++)

{

System.out.println("Enter integer element to insert");

rbt.insert( scan.nextInt() );

}

\*/

try

{

//in=new FileInputStream("/home/adesh/Desktop");

//StringBuffer strContent=new StringBuffer();

FileInputStream fstream=new FileInputStream("E://rbt.txt");

DataInputStream in=new DataInputStream(fstream);

BufferedReader br=new BufferedReader(new InputStreamReader(in));

String strLine;

while((strLine=br.readLine())!=null)

{

int num=Integer.parseInt(strLine);

rbt.insert(num);

//System.out.println(num);

}

in.close();

}catch(Exception e)

{

}

/\* while((count=fr.read())!=-1)

{

//rbt.insert(dis.readInt());

}

fr.close();

\*/

System.out.println("+++++++++++++++++INORDER Traversal of Tree++++++++++++++++++++++");

/\* Display tree \*/

rbt.inorder();

System.out.print("\n");

System.out.println("Enter the number of elements to be searched");

n2=scan.nextInt();

threading t[]=new threading[n2];

System.out.println("Enter the integer elements to search");

for(i=0;i<n2;i++)

elements[i]=scan.nextInt();

//System.out.println("+++++++++++++++++INORDER Traversal of Tree++++++++++++++++++++++");

/\* Display tree \*/

//rbt.inorder();

System.out.print("\n");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Without Threading\*\*\*\*\*\*\*\*\*\*\*\*\n");

long startTime = System.nanoTime();

for(i=0;i<n2;i++)

System.out.println("Search result for "+ elements[i] +" is :"+rbt.search( elements[i] ));

long endTime = System.nanoTime();

long time1 = endTime - startTime;

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Using Threading\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

startTime = System.nanoTime();

for(i=0;i<n2;i++)

{

t[i]=new threading();

t[i].array(elements[i]);

t[i].start();

}

//endTime = System.nanoTime();

//long time2 = endTime - startTime;

boolean arr [] = new boolean[n2] ;

int flag=1;

for(i=0;i<n2;i++)

{

arr[i]=t[i].isAlive();

}

while(flag==1)

{

if(areAllFalse(arr)==true)

flag=0;

}

endTime = System.nanoTime();

long time2 = endTime - startTime;

System.out.println("Time required without threading is :"+time1+" nano secs\n"+"Time required with threading is :"+time2+" nano secs");

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SCREENSHOT/OUTPUT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*





