/\*\* **Java Program to Implement SplayTree** \*\*/

import java.util.Scanner;

/\*\* Class Node \*\*/

class SplayNode

{

SplayNode left, right, parent;

int element;

/\*\* Constructor \*\*/

public SplayNode()

{

this(0, null, null, null);

}

/\*\* Constructor \*\*/

public SplayNode(int ele)

{

this(ele, null, null, null);

}

/\*\* Constructor \*\*/

public SplayNode(int ele, SplayNode left, SplayNode right, SplayNode parent)

{

this.left = left;

this.right = right;

this.parent = parent;

this.element = ele;

}

}

/\*\* Class SplayTree \*\*/

class SplayTree

{

private SplayNode root;

private int count = 0;

/\*\* Constructor \*\*/

public SplayTree()

{

root = null;

}

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

public boolean isEmpty()

{

return root == null;

}

/\*\* clear tree \*\*/

public void clear()

{

root = null;

}

/\*\* function to insert element \*/

public void insert(int ele)

{

SplayNode z = root;

SplayNode p = null;

while (z != null)

{

p = z;

if (ele < p.element)

z = z.right;

else

z = z.left;

}

z = new SplayNode();

z.element = ele;

z.parent = p;

if (p == null)

root = z;

else if (ele < p.element)

p.right = z;

else

p.left = z;

Splay(z);

count++;

}

/\*\* rotate \*\*/

public void makeLeftChildParent(SplayNode c, SplayNode p)

{

if ((c == null) || (p == null) || (p.left != c) || (c.parent != p))throw new RuntimeException("WRONG");

if (p.parent != null)

{

if (p == p.parent.left)

p.parent.left = c;

else

p.parent.right = c;

}

if (c.right != null)

c.right.parent = p;

c.parent = p.parent;

p.parent = c;

p.left = c.right;

c.right = p;

}

/\*\* rotate \*\*/

public void makeRightChildParent(SplayNode c, SplayNode p)

{

if ((c == null) || (p == null) || (p.right != c) || (c.parent != p))

throw new RuntimeException("WRONG");

if (p.parent != null)

{

if (p == p.parent.left)

p.parent.left = c;

else

p.parent.right = c;

}

if (c.left != null)

c.left.parent = p;

c.parent = p.parent;

p.parent = c;

p.right = c.left;

c.left = p;

}

/\*\* function splay \*\*/

private void Splay(SplayNode x)

{

while (x.parent != null)

{

SplayNode Parent = x.parent;

SplayNode GrandParent = Parent.parent;

if (GrandParent == null)

{

if (x == Parent.left)

makeLeftChildParent(x, Parent);

else

makeRightChildParent(x, Parent);

}

else

{

if (x == Parent.left)

{

if (Parent == GrandParent.left)

{

makeLeftChildParent(Parent, GrandParent);

makeLeftChildParent(x, Parent);

}

else

{

makeLeftChildParent(x, x.parent);

makeRightChildParent(x, x.parent);

}

}

else

{

if (Parent == GrandParent.left)

{

makeRightChildParent(x, x.parent);

makeLeftChildParent(x, x.parent);

}

else

{

makeRightChildParent(Parent, GrandParent);

makeRightChildParent(x, Parent);

}

}

}

}

root = x;

}

/\*\* function to remove element \*\*/

public void remove(int ele)

{

SplayNode node = findNode(ele);

remove(node);

}

/\*\* function to remove node \*\*/

private void remove(SplayNode node)

{

if (node == null)

return;

Splay(node);

if( (node.left != null) && (node.right !=null))

{

SplayNode min = node.left;

while(min.right!=null)

min = min.right;

min.right = node.right;

node.right.parent = min;

node.left.parent = null;

root = node.left;

}

else if (node.right != null)

{

node.right.parent = null;

root = node.right;

}

else if( node.left !=null)

{

node.left.parent = null;

root = node.left;

}

else

{

root = null;

}

node.parent = null;

node.left = null;

node.right = null;

node = null;

count--;

}

/\*\* Functions to count number of nodes \*\*/

public int countNodes()

{

return count;

}

/\*\* Functions to search for an element \*\*/

public boolean search(int val)

{

return findNode(val) != null;

}

private SplayNode findNode(int ele)

{

SplayNode z = root;

while (z != null)

{

if (ele < z.element)

z = z.right;

else if (ele > z.element)

z = z.left;

else

return z;

}

return null;

}

/\*\* Function for inorder traversal \*\*/

public void inorder()

{

inorder(root);

}

private void inorder(SplayNode r)

{

if (r != null)

{

inorder(r.left);

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

inorder(r.right);

}

}

/\*\* Function for preorder traversal \*\*/

public void preorder()

{

preorder(root);

}

private void preorder(SplayNode r)

{

if (r != null)

{

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

preorder(r.left);

preorder(r.right);

}

}

/\*\* Function for postorder traversal \*\*/

public void postorder()

{

postorder(root);

}

private void postorder(SplayNode r)

{

if (r != null)

{

postorder(r.left);

postorder(r.right);

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

}

}

}

/\*\* Class SplayTreeTest \*\*/

public class SplayTreeTest

{

public static void main(String[] args)

{

Scanner scan = new Scanner(System.in);

/\*\* Creating object of SplayTree \*\*/

SplayTree spt = new SplayTree();

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

char ch;

/\*\* Perform tree operations \*\*/

do

{

System.out.println("\nSplay Tree Operations\n");

System.out.println("1. insert ");

System.out.println("2. remove ");

System.out.println("3. search");

System.out.println("4. count nodes");

System.out.println("5. check empty");

System.out.println("6. clear tree");

int choice = scan.nextInt();

switch (choice)

{

case 1 :

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

spt.insert( scan.nextInt() );

break;

case 2 :

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

spt.remove( scan.nextInt() );

break;

case 3 :

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

System.out.println("Search result : "+ spt.search( scan.nextInt() ));

break;

case 4 :

System.out.println("Nodes = "+ spt.countNodes());

break;

case 5 :

System.out.println("Empty status = "+ spt.isEmpty());

break;

case 6 :

System.out.println("\nTree Cleared");

spt.clear();

break;

default:

System.out.println("Wrong Entry \n ");

break;

}

/\*\* Display tree \*\*/

System.out.print("\nPost order : ");

spt.postorder();

System.out.print("\nPre order : ");

spt.preorder();

System.out.print("\nIn order : ");

spt.inorder();

System.out.println("\nDo you want to continue (Type y or n) \n");

ch = scan.next().charAt(0);

}

while (ch == 'Y'|| ch == 'y');

}

}

**Output:**

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

1

Enter integer element to insert

14

Post order : 14

Pre order : 14

In order : 14

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

1

Enter integer element to insert

28

Post order : 14 28

Pre order : 28 14

In order : 28 14

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

1

Enter integer element to insert

19

Post order : 28 14 19

Pre order : 19 28 14

In order : 28 19 14

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

1

Enter integer element to insert

63

Post order : 14 19 28 63

Pre order : 63 28 19 14

In order : 63 28 19 14

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

1

Enter integer element to insert

5

Post order : 63 19 14 28 5

Pre order : 5 28 63 14 19

In order : 63 28 19 14 5

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

1

Enter integer element to insert

7

Post order : 63 19 28 14 5 7

Pre order : 7 14 28 63 19 5

In order : 63 28 19 14 7 5

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

3

Enter integer element to search

24

Search result : false

Post order : 63 19 28 14 5 7

Pre order : 7 14 28 63 19 5

In order : 63 28 19 14 7 5

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

2

Enter integer element to remove

28

Post order : 19 5 7 14 63

Pre order : 63 14 19 7 5

In order : 63 19 14 7 5

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

3

Enter integer element to search

28

Search result : false

Post order : 19 5 7 14 63

Pre order : 63 14 19 7 5

In order : 63 19 14 7 5

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

2

Enter integer element to remove

14

Post order : 5 7 19 63

Pre order : 63 19 7 5

In order : 63 19 7 5

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

2

Enter integer element to remove

7

Post order : 63 5 19

Pre order : 19 63 5

In order : 63 19 5

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

4

Nodes = 3

Post order : 63 5 19

Pre order : 19 63 5

In order : 63 19 5

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

5

Empty status = false

Post order : 63 5 19

Pre order : 19 63 5

In order : 63 19 5

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

6

Tree Cleared

Post order :

Pre order :

In order :

Do you want to continue (Type y or n)

y

Splay Tree Operations

1. insert

2. remove

3. search

4. count nodes

5. check empty

6. clear tree

5

Empty status = true

Post order :

Pre order :

In order :

Do you want to continue (Type y or n)

n