Exercise BinarySearchTree

BinarySearchTree time complexity

Operation	Time complexity
Find	O(n)
Insert	O(n)
NumberOfElements	O(1)
Depth	O(n)
NumberOfLeaves	O(n)
ToString	O(n)

Treap time complexity

Operation	Time complexity
Find	O(log n)
Insert	O(log n)
NumberOfElements	O(1)
Depth	O(n)
NumberOfLeaves	O(n)
ToString	O(n)

Tree class

```
1  package inda5;
2
3  import java.io.StringWriter;
4
5  public class Tree<T extends Comparable<T>> {
6
7      private Node<T> nodes;    // First element in list.
8      private int size;
9      /**
10      * A node element.
11      */
12     public static class Node<T> {
13
14         public int key;
15         public T value;
16         public Node<T> left;
17         public Node<T> right;
18
19
20         public Node(int key,T value) {
21             this.key = key;
22             this.value = value;
23             this.left = null;
24             this.right = null;
25         }
26     }
27
28     /**
29      * Creates a new instance of the Tree class.
30      * @param main The main node (root).
31      */
32     public Tree(Node<T> main)
```

```
33      {
34          this.nodes = main;
35      }
36
37      /**
38       * Finds the values of the given key. If the key does not exist, null
             will be returned.
39       * @param key The key.
40       * @return The object associated with the key or null if the key does
             not exist.
41       */
42      public Node<T> Find(int key)
43      {
44          Node<T> temp = nodes;
45
46          while(temp != null && temp.key != key)
47          {
48              if (key < temp.key)
49                  temp = temp.left;
50              else
51                  temp = temp.right;
52          }
53
54          return temp;
55      }
56
57      /**
58       * Inserts a new value into given key. If the key already exists, the
             value is updated.
59       * @param key The key.
60       * @param value The value.
61       */
62      public void Insert(int key, T value)
63      {
64          size++;
65          if(nodes == null)
66          {
67              nodes = new Node<T>(key, value);
68              return;
69          }
70
71          Node<T> temp = nodes;
72
73          while(temp != null && temp.key != key )
74          {
75              if (key < temp.key && temp.left != null)
76                  temp = temp.left;
77              else if (key > temp.key &&  temp.right != null)
78                  temp = temp.right;
79              else
80                  break;
81          }
82
83          if (key == temp.key) {
84              temp.value = value; size--;
85          }
```

```
86          else if (key < temp.key)
87              temp.left = new Node<T>(key, value);
88          else
89              temp.right = new Node<T>(key, value);
90      }
91
92      public int NumberOfLeaves()
93      {
94          return getNumberOfLeaves(nodes);
95      }
96
97      /**
98       * Helper method for NumberOfLeaves
99       */
100     private int getNumberOfLeaves(Node<T> n)
101     {
102         if (n == null) return 0;
103         if (n.left == null && n.right == null) return 1;
104
105         return getNumberOfLeaves(n.left) + getNumberOfLeaves(n.right);
106     }
107
108     public int Deapth()
109     {
110         return getDeapth(nodes);
111     }
112
113     /**
114      * Helper method for Deapth
115      */
116     private int getDeapth(Node<T> n)
117     {
118         if (n == null) return 0;
119         return max(getDeapth(n.left), getDeapth(n.right)) + 1;
120     }
121
122     /**
123      * Finds the max value
124      */
125     private int max (int a, int b)
126     {
127         if (a > b)
128             return a;
129         return b;
130     }
131
132     /**
133      * Returns a string reprsentation of the tree, ordered.
134      */
135     @Override
136     public String toString()
137     {
138         StringWriter out = new StringWriter();
139         out.write("[");
140         inorder(nodes, out);
141         String a = out.toString();
```

```
142         return a.substring(0, a.length()-1) + "]";
143     }
144 
145     private void inorder(Node<T> node, StringWriter sw)
146     {
147         if (node.left != null)
148             inorder(node.left, sw);
149 
150         sw.write(node.value.toString());
151         sw.write(",");
152 
153         if (node.right != null)
154             inorder(node.right, sw);
155     }
156 }
```

BinarySearchTreeTest class

```
1  package inda5;
2 
3  import static org.junit.Assert.*;
4  import inda5.Tree.Node;
5 
6  import org.junit.Test;
7 
8  public class BinarySearchTreeTest {
9 
10     @Test
11     public void FindTest() {
12         Tree<String> tree = new Tree<String>(new Node<String>(100,"test"));
13 
14         tree.Insert(1, "hi");
15         tree.Insert(2, "world");
16         tree.Insert(3, "hello");
17         tree.Insert(4, "there");
18 
19         Node<String> a = tree.Find(3);
20         System.out.println(a.value);
21 
22         assertEquals("hello", a.value);
23         System.out.println(a.value);
24         assertEquals("hello", a.value);
25         assertEquals("hello", a.value);
26 
27     }
28 
29     @Test
30     public void InsertTest()
31     {
32         //fail();
33 
34         Tree<String> tree = new Tree<String>(new Node<String>(100,"test"));
35 
36         tree.Insert(1, "hi");
37         tree.Insert(2, "world");
38         tree.Insert(3, "hello");
```

```
39          tree.Insert(4, "there");
40      }
41
42
43      @Test
44      public void DeapthTest()
45      {
46          Tree<String> tree = new Tree<String>(new Node<String>(50,"test"));
47
48          tree.Insert(1, "hi");
49          tree.Insert(2, "world");
50          tree.Insert(3, "hello");
51          tree.Insert(4, "there");
52          System.out.println(tree.NumberOfLeaves() + "DeatphTest");
53          assertTrue(tree.NumberOfLeaves() == 1);
54
55
56          assertTrue(tree.Deapth() == 5);
57
58          tree.Insert(55, "hi");
59          tree.Insert(57, "world");
60          tree.Insert(79, "hello");
61
62          assertTrue(tree.Deapth() == 5);
63          assertEquals(tree.Find(57).value, "world");
64
65          tree.Insert(58, "world");
66          tree.Insert(71, "hello");
67
68          assertTrue(tree.Deapth() == 6);
69
70
71          //fail();
72      }
73
74      @Test
75      public void NumberOfLeavesTest()
76      {
77          Tree<String> tree = new Tree<String>(new Node<String>(44,"test"));
78
79          tree.Insert(17, "");
80          tree.Insert(8, "");
81          tree.Insert(32, "");
82          tree.Insert(28, "");
83          tree.Insert(41, "");
84          tree.Insert(88, "");
85          tree.Insert(65, "");
86          tree.Insert(97, "");
87          assertTrue(tree.NumberOfLeaves() == 5);
88
89          tree.Insert(40, "");
90          tree.Insert(43, "");
91          assertTrue(tree.NumberOfLeaves() == 6);
92
93      }
94
```

```
95      @Test
96      public void ToStringTest()
97      {
98          Tree<String> tree = new Tree<String>(new Node<String>(44,"test"));
99
100         tree.Insert(17, "a");
101         tree.Insert(8, "b");
102         tree.Insert(32, "c");
103         tree.Insert(28, "d");
104         tree.Insert(41, "e");
105         tree.Insert(88, "f");
106         tree.Insert(65, "g");
107         tree.Insert(97, "h");
108
109         System.out.println(tree.toString());
110         assertEquals(tree.toString(), "[b,a,d,c,e,test,g,f,h]");
111     }
112
113 }
```