

Homework Turnin

Account: 6G_06 (rgalanos@fcps.edu)
Section: 6G
Course: TJHSST APCS 2016-17
Assignment: 07-01

Receipt ID: 83dc8480313cbbce193d9ad000340013

Turnin Successful!

The following file(s) were received:

TreeLab.java (9201 bytes)

```
1. //name:    date:
2. import java.util.*;           //for the queue interface
3. public class TreeLab
4. {
5.     public static TreeNode root = null;
6.     // public static String s = "XCOMPUTERSCIENCE";
7.     // public static String s = "XThomasJeffersonHighSchool";
8.     public static String s = "XAComputerScienceTreeHasItsRootAtTheTop";
9.     public static void main(String[] args)
10.    {
11.        root = buildTree( root, s );
12.        System.out.print( display(root, 0) );
13.
14.        System.out.print("\nPreorder: " + preorderTraverse(root));
15.        System.out.print("\nInorder: " + inorderTraverse(root));
16.        System.out.print("\nPostorder: " + postorderTraverse(root));
17.
18.        System.out.println("\n\nNodes = " + countNodes(root));
19.        System.out.println("Leaves = " + countLeaves(root));
20.        System.out.println("Grandparents = " + countGrands(root));
21.        System.out.println("Only childs = " + countOnlys(root));
22.
23.        System.out.println("\nHeight of tree = " + height(root));
24.        System.out.println("Width = " + width(root));
25.        System.out.println("Min = " + min(root));
26.        System.out.println("Max = " + max(root));
27.
28.        System.out.println("\nBy Level: ");
29.        System.out.println(displayLevelOrder(root));
30.    }
31.    public static TreeNode buildTree(TreeNode root, String s)
32.    {
33.        root = new TreeNode(" " + s.charAt(1), null, null);
34.        for(int pos = 2; pos < s.length(); pos++)
35.            insert(root, " " + s.charAt(pos), pos,
36.                (int)(1 + Math.log(pos) / Math.log(2)));
37.
38.        insert(root, "A", 17, 5);
39.        insert(root, "B", 18, 5);
40.        insert(root, "C", 37, 6); //B's right child
41.        return root;
42.    }
43.
44.    public static void insert(TreeNode t, String s, int pos, int level)
45.    {
46.        TreeNode p = t;
47.        for(int k = level - 2; k > 0; k--)
48.            if((pos & (1 << k)) == 0)
49.                p = p.getLeft();
50.        else
```

```

51.     p = p.getRight();
52.     if((pos & 1) == 0)
53.         p.setLeft(new TreeNode(s, null, null));
54.     else
55.         p.setRight(new TreeNode(s, null, null));
56. }
57.
58.
59. private static String display(TreeNode t, int level)
60. {
61.     String toRet = "";
62.     if(t == null)
63.         return "";
64.     toRet += display(t.getRight(), level + 1); //recurse right
65.     for(int k = 0; k < level; k++)
66.         toRet += "\t";
67.     toRet += t.getValue() + "\n";
68.     toRet += display(t.getLeft(), level + 1); //recurse left
69.     return toRet;
70. }
71.
72. public static String preorderTraverse(TreeNode t)
73. {
74.     String toReturn = "";
75.     if(t == null)
76.         return "";
77.     toReturn += t.getValue() + " "; //preorder visit
78.     toReturn += preorderTraverse(t.getLeft()); //recurse left
79.     toReturn += preorderTraverse(t.getRight()); //recurse right
80.     return toReturn;
81. }
82. public static String inorderTraverse(TreeNode t)
83. {
84.     String toReturn = "";
85.     if(t == null)
86.         return "";
87.     toReturn += inorderTraverse(t.getLeft());
88.     toReturn += t.getValue() + " ";
89.     toReturn += inorderTraverse(t.getRight());
90.     return toReturn;
91. }
92. public static String postorderTraverse(TreeNode t)
93. {
94.     String toReturn = "";
95.     if(t == null)
96.         return "";
97.     toReturn += postorderTraverse(t.getLeft());
98.     toReturn += postorderTraverse(t.getRight());
99.     toReturn += t.getValue() + " ";
100.    return toReturn;
101. }
102. public static int countNodes(TreeNode t)
103. {
104.     if(t == null)
105.         return 0;
106.     return 1+countNodes(t.getLeft())+countNodes(t.getRight());
107. }
108. public static int countLeaves(TreeNode t)
109. {
110.     if(t == null)
111.         return 0;
112.     else if(t.getLeft() == null && t.getRight()==null)
113.         return 1;
114.     return countLeaves(t.getLeft())+countLeaves(t.getRight());
115. }
116. public static int countGrands(TreeNode t)
117. {
118.     if(t == null)
119.         return 0;
120.     else if(t.getLeft()!=null)
121.     {
122.         if(t.getLeft().getLeft()!=null || t.getLeft().getRight()!=null)
123.             return 1 + countGrands(t.getLeft()) + countGrands(t.getRight());
124.     }
125.     else if(t.getRight()!=null)
126.     {
127.         if(t.getRight().getLeft()!=null || t.getRight().getRight()!=null)
128.             return 1 + countGrands(t.getLeft()) + countGrands(t.getRight());
129.     }
130.     return 0;
131. }

```

```

132. public static int countOnlys(TreeNode t)
133. {
134.     if(t==null)
135.         return 0;
136.     else if(t.getLeft()==null&&!.getRight()!=null)
137.         return 1+countOnlys(t.getRight());
138.     else if(t.getLeft()!=null&&!.getRight()==null)
139.         return 1+countOnlys(t.getLeft());
140.     return countOnlys(t.getLeft())+countOnlys(t.getRight());
141. }
142. public static int height(TreeNode t)
143. {
144.     if(t == null)
145.         return -1;
146.
147.     int l = height(t.getLeft());
148.     int r = height(t.getRight());
149.
150.     return Math.max(l,r) + 1;
151. }
152. /* "width" is the longest path from leaf to leaf */
153. public static int width(TreeNode t)
154. {
155.     return height(t.getLeft()) + height(t.getRight()) + 2;
156. }
157. @SuppressWarnings("unchecked")//this removes the warning about needing to cast
158. public static Object min(TreeNode t)
159. {
160.     if(t == null)
161.         return null;
162.
163.     Object l = min(t.getLeft());
164.     Object r = min(t.getRight());
165.     Object min;
166.
167.     if(l == null && r==null)
168.         min = t.getValue();
169.     else if(l == null)
170.         if(((Comparable)t.getValue()).compareTo((Comparable)r)<0)
171.             min = t.getValue();
172.         else
173.             min = r;
174.     else if(r==null)
175.         if(((Comparable)t.getValue()).compareTo(l)<0)
176.             min = t.getValue();
177.         else
178.             min = l;
179.     else
180.     {
181.         if(((Comparable)l).compareTo((Comparable)r)<0)
182.             if(((Comparable)t.getValue()).compareTo((Comparable)l)<0)
183.                 min = t.getValue();
184.             else
185.                 min = l;
186.         else
187.         {
188.             if(((Comparable)t.getValue()).compareTo((Comparable)r)<0)
189.                 min = t.getValue();
190.             else
191.                 min = r;
192.         }
193.     }
194.     return min;
195. }
196. @SuppressWarnings("unchecked")//this removes the warning about needing to cast
197. public static Object max(TreeNode t)
198. {
199.     if(t == null)
200.         return null;
201.
202.     Object l = max(t.getLeft());
203.     Object r = max(t.getRight());
204.     Object max;
205.
206.     if(l == null && r==null)
207.         max = t.getValue();
208.     else if(l == null)
209.         if(((Comparable)t.getValue()).compareTo((Comparable)r)>0)
210.             max = t.getValue();
211.         else
212.             max = r;

```

```

213.     else if(r==null)
214.         if(((Comparable)t.getValue()).compareTo(l)>0)
215.             max = t.getValue();
216.         else
217.             max = l;
218.     else
219.     {
220.         if(((Comparable)l).compareTo((Comparable)r)>0)
221.             if(((Comparable)t.getValue()).compareTo((Comparable)l)>0)
222.                 max = t.getValue();
223.             else
224.                 max = l;
225.         else
226.         {
227.             if(((Comparable)t.getValue()).compareTo((Comparable)r)>0)
228.                 max = t.getValue();
229.             else
230.                 max = r;
231.         }
232.     }
233.     return max;
234. }
235. /* this method is not recursive. Use a local queue
236.    to store the children of the current node.*/
237. public static String displayLevelOrder(TreeNode t)
238. {
239.     String str = "";
240.     Queue<TreeNode> q = new LinkedList<TreeNode>();
241.     q.add(t);
242.
243.     while(!q.isEmpty())
244.     {
245.         TreeNode temp = q.remove();
246.         str+=temp.getValue().toString();
247.
248.         if(temp.getLeft()!=null)
249.             q.add(temp.getLeft());
250.         if(temp.getRight()!=null)
251.             q.add(temp.getRight());
252.     }
253.     return str;
254. }
255. }
256. /*****
257.
258. ----jGRASP exec: java Lab01
259.
260.      E
261.     E
262.    C
263.   M
264.  N
265. T
266. E
267. C
268. I
269. U
270. C
271. O
272. S
273.   C
274.  B
275. P
276.  A
277. R
278.
279. Preorder: C O P R A S B C U C I M T E N E C E
280. Inorder:  R A P B C S O C U I C E T N M C E E
281. Postorder: A R C B S P C I U O E N T C E E M C
282.
283. Nodes = 18
284. Leaves = 8
285. Grandparents = 5
286. Only childs = 3
287.
288. Height of tree = 5
289. Width = 8
290. Min = A
291. Max = U
292.
293. By Level:

```

```
294.  COMPUTERSCIENCEABC
295.  *****/
296.      /* TreeNode class for the AP Exams */
297.
298.  class TreeNode
299.  {
300.      private Object value;
301.      private TreeNode left, right;
302.
303.      public TreeNode(Object initValue)
304.      {
305.          value = initValue;
306.          left = null;
307.          right = null;
308.      }
309.
310.      public TreeNode(Object initValue, TreeNode initLeft, TreeNode initRight)
311.      {
312.          value = initValue;
313.          left = initLeft;
314.          right = initRight;
315.      }
316.
317.      public Object getValue()
318.      {
319.          return value;
320.      }
321.
322.      public TreeNode getLeft()
323.      {
324.          return left;
325.      }
326.
327.      public TreeNode getRight()
328.      {
329.          return right;
330.      }
331.
332.      public void setValue(Object theNewValue)
333.      {
334.          value = theNewValue;
335.      }
336.
337.      public void setLeft(TreeNode theNewLeft)
338.      {
339.          left = theNewLeft;
340.      }
341.
342.      public void setRight(TreeNode theNewRight)
343.      {
344.          right = theNewRight;
345.      }
346.  }
347.
```