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)

```
//name:    date:
import java.util.*;         //for the queue interface
public class TreeLab
{
   public static TreeNode root = null;
  // public static String s = "XCOMPUTERSCIENCE";
  // public static String s = "XThomasJeffersonHighSchool";
   public static String s = "XAComputerScienceTreeHasItsRootAtTheTop";
   public static void main(String[] args)
   {
      root = buildTree( root, s );
      System.out.print( display(root, 0) );

      System.out.print("\nPreorder: " + preorderTraverse(root));
      System.out.print("\nInorder: " + inorderTraverse(root));
      System.out.print("\nPostorder: " + postorderTraverse(root));

      System.out.println("\n\nNodes = " + countNodes(root));
      System.out.println("Leaves = " + countLeaves(root));
      System.out.println("Grandparents = " + countGrands(root));
      System.out.println("Only childs = " + countOnlys(root));

      System.out.println("\nHeight of tree = " + height(root));
      System.out.println("Width = " + width(root));
      System.out.println("Min = " + min(root));
      System.out.println("Max = " + max(root));

      System.out.println("\nBy Level: ");
      System.out.println(displayLevelOrder(root));
   }
   public static TreeNode buildTree(TreeNode root, String s)
   {
      root = new TreeNode("" + s.charAt(1), null, null);
      for(int pos = 2; pos < s.length(); pos++)
         insert(root, "" + s.charAt(pos), pos,
            (int)(1 + Math.log(pos) / Math.log(2)));

      insert(root, "A", 17, 5);
      insert(root, "B", 18, 5);
      insert(root, "C", 37, 6); //B's right child
      return root;
   }

   public static void insert(TreeNode t, String s, int pos, int level)
   {
      TreeNode p = t;
      for(int k = level - 2; k > 0; k--)
         if((pos & (1 << k)) == 0)
            p = p.getLeft();
         else
```

```
            p = p.getRight();
      if((pos & 1) == 0)
         p.setLeft(new TreeNode(s, null, null));
      else
         p.setRight(new TreeNode(s, null, null));
   }


   private static String display(TreeNode t, int level)
   {
      String toRet = "";
      if(t == null)
         return "";
      toRet += display(t.getRight(), level + 1); //recurse right
      for(int k = 0; k < level; k++)
         toRet += "\t";
      toRet += t.getValue() + "\n";
      toRet += display(t.getLeft(), level + 1); //recurse left
      return toRet;
   }

   public static String preorderTraverse(TreeNode t)
   {
      String toReturn = "";
      if(t == null)
         return "";
      toReturn += t.getValue() + " ";  //preorder visit
      toReturn += preorderTraverse(t.getLeft());         //recurse left
      toReturn += preorderTraverse(t.getRight());        //recurse right
      return toReturn;
   }
   public static String inorderTraverse(TreeNode t)
   {
      String toReturn = "";
      if(t == null)
         return "";
      toReturn += inorderTraverse(t.getLeft());
      toReturn += t.getValue() + " ";
      toReturn += inorderTraverse(t.getRight());
      return toReturn;
   }
   public static String postorderTraverse(TreeNode t)
   {
      String toReturn = "";
      if(t == null)
         return "";
      toReturn += postorderTraverse(t.getLeft());
      toReturn += postorderTraverse(t.getRight());
      toReturn += t.getValue() + " ";
      return toReturn;
   }
   public static int countNodes(TreeNode t)
   {
      if(t == null)
         return 0;
      return 1+countNodes(t.getLeft())+countNodes(t.getRight());
   }
   public static int countLeaves(TreeNode t)
   {
      if(t == null)
         return 0;
      else if(t.getLeft() == null && t.getRight()==null)
         return 1;
      return countLeaves(t.getLeft())+countLeaves(t.getRight());
   }
   public static int countGrands(TreeNode t)
   {
      if(t == null)
         return 0;
      else if(t.getLeft()!=null)
      {
         if(t.getLeft().getLeft()!=null || t.getLeft().getRight()!=null)
            return 1 + countGrands(t.getLeft()) + countGrands(t.getRight());
      }
      else if(t.getRight()!=null)
      {
         if(t.getRight().getLeft()!=null || t.getRight().getRight()!=null)
            return 1 + countGrands(t.getLeft()) + countGrands(t.getRight());
      }
      return 0;
   }
```

```
public static int countOnlys(TreeNode t)
{
   if(t==null)
      return 0;
   else if(t.getLeft()==null&&t.getRight()!=null)
      return 1+countOnlys(t.getRight());
   else if(t.getLeft()!=null&&t.getRight()==null)
      return 1+countOnlys(t.getLeft());
   return countOnlys(t.getLeft())+countOnlys(t.getRight());
}
public static int height(TreeNode t)
{
   if(t == null)
      return -1;

   int l = height(t.getLeft());
   int r = height(t.getRight());

   return Math.max(l,r) + 1;
}
   /* "width" is the longest path from leaf to leaf */
public static int width(TreeNode t)
{
   return height(t.getLeft()) + height(t.getRight()) + 2;
}
@SuppressWarnings("unchecked")//this removes the warning about needing to cast
public static Object min(TreeNode t)
{
   if(t == null)
      return null;

   Object l = min(t.getLeft());
   Object r = min(t.getRight());
   Object min;

   if(l == null && r==null)
      min = t.getValue();
   else if(l == null)
      if(((Comparable)t.getValue()).compareTo((Comparable)r)<0)
         min = t.getValue();
      else
         min = r;
   else if(r==null)
      if(((Comparable)t.getValue()).compareTo(l)<0)
         min = t.getValue();
      else
         min = l;
   else
   {
      if(((Comparable)l).compareTo((Comparable)r)<0)
         if(((Comparable)t.getValue()).compareTo((Comparable)l)<0)
            min = t.getValue();
         else
            min = l;
      else
      {
         if(((Comparable)t.getValue()).compareTo((Comparable)r)<0)
            min = t.getValue();
         else
            min = r;
      }
   }
   return min;
}
@SuppressWarnings("unchecked")//this removes the warning about needing to cast
public static Object max(TreeNode t)
{
   if(t == null)
      return null;

   Object l = max(t.getLeft());
   Object r = max(t.getRight());
   Object max;

   if(l == null && r==null)
      max = t.getValue();
   else if(l == null)
      if(((Comparable)t.getValue()).compareTo((Comparable)r)>0)
         max = t.getValue();
      else
         max = r;
      else if(r==null)
         if(((Comparable)t.getValue()).compareTo(l)>0)
            max = t.getValue();
         else
            max = l;
      else
      {
         if(((Comparable)l).compareTo((Comparable)r)>0)
            if(((Comparable)t.getValue()).compareTo((Comparable)l)>0)
               max = t.getValue();
            else
               max = l;
         else
         {
            if(((Comparable)t.getValue()).compareTo((Comparable)r)>0)
               max = t.getValue();
            else
               max = r;
         }
      }
      return max;
   }
      /* this method is not recursive.  Use a local queue
      to store the children of the current node.*/
   public static String displayLevelOrder(TreeNode t)
   {
      String str = "";
      Queue<TreeNode> q = new LinkedList<TreeNode>();
      q.add(t);

      while(!q.isEmpty())
      {
         TreeNode temp = q.remove();
         str+=temp.getValue().toString();

         if(temp.getLeft()!=null)
            q.add(temp.getLeft());
         if(temp.getRight()!=null)
            q.add(temp.getRight());
      }
      return str;
   }
}
/*******************************************************

   ----jGRASP exec: java Lab01

         E
      E
         C
   M
         N
      T
         E
C
         I
      U
         C
   O
         S
               C
            B
      P
            A
         R

Preorder: C O P R A S B C U C I M T E N E C E
Inorder: R A P B C S O C U I C E T N M C E E
Postorder: A R C B S P C I U O E N T C E E M C

Nodes = 18
Leaves = 8
Grandparents = 5
Only childs = 3

Height of tree = 5
Width = 8
Min = A
Max = U

By Level:
 COMPUTERSCIENCEABC
*******************************************************/
     /* TreeNode class for the AP Exams */

class TreeNode
{
   private Object value;
   private TreeNode left, right;

   public TreeNode(Object initValue)
   {
      value = initValue;
      left = null;
      right = null;
   }

   public TreeNode(Object initValue, TreeNode initLeft, TreeNode initRight)
   {
      value = initValue;
      left = initLeft;
      right = initRight;
   }

   public Object getValue()
   {
      return value;
   }

   public TreeNode getLeft()
   {
      return left;
   }

   public TreeNode getRight()
   {
      return right;
   }

   public void setValue(Object theNewValue)
   {
      value = theNewValue;
   }

   public void setLeft(TreeNode theNewLeft)
   {
      left = theNewLeft;
   }

   public void setRight(TreeNode theNewRight)
   {
      right = theNewRight;
   }
}

```