```java
/**
 * @author Bishoy A Abdelmalik
 * @class COMP 282 meeting at 2:00 PM
 * @Assigment Program #2
 * @DateTurnedIn  Sept 30, 2019
 * @description
 * this file includes both the StringAVLNode that have the code for creating the node
 * and the StringAVL tree that has all the code for the tree and its methods
 */
import java.util.ArrayList;
import java.util.Deque;
import java.util.LinkedList;

class StringAVLTree {

    // should really be private but I need access
    // for my test program to work
    StringAVLNode root;
    // just one constructor
    public StringAVLTree() {
        this.root=null;
    }
    // Rotate the node to the right
    private static StringAVLNode rotateRight(StringAVLNode t) {
        StringAVLNode newRoot=t.getLeft();// get the pointer to the new root
        t.setLeft(t.getLeft().getRight());//set left pointer of the old root to the right of the new root
        newRoot.setRight(t);//set the right of the new root to the old root
        t=newRoot;//new root equal to t so we return it
        return t;
    }
    // Rotate the node to the left
    private static StringAVLNode rotateLeft(StringAVLNode t) {
        StringAVLNode newRoot=t.getRight();// get the pointer to the new root
        t.setRight(t.getRight().getLeft());//set right pointer of the old root to the left of the new root
        newRoot.setLeft(t);//set the left of the new root to the old root
        t=newRoot;//new root equal to t so we return it
        return t;
    }
    // For these next four, be sure not to use any global variables
    // and no extra counting parameters in the recursive methods, e.g.,
```

```java
    // the recursive height method should just have one parameter, the
    // StringAVLNode
    // Return the height of the tree - not to be used anywhere in insert or delete
    public int height() { return height(this.root);}
    private static int height(StringAVLNode root) {
        int returnValue=0;//assumes empty root
        if(root!=null){//if root isn't null
            int depth;
            if(root.getBalance()==0){
                depth=height(root.getLeft());//get height of left subtree if they are both the same
            }else if (root.getBalance()>0){
                depth=height(root.getRight());//get height of right subtree
            }else {
                depth=height(root.getLeft());//get height of left subtree

            }
            returnValue = 1 + depth;

        }
        return returnValue;
    }



    // nadir return the distance to the closest leaf
    public int nadir() {
        int returnValue=nadir(this.root)+1;
        if(this.root==null){
            //since we add one to make the distance counting start from 1 at the first tree level
            // reset the value to 0 if the tree is actually empty
            returnValue=0;
        }
        return returnValue;
    }
    private static int nadir(StringAVLNode root) {
        int returnValue=0;//assumes empty root
        if(root!=null &&root.getLeft()==null&& root.getRight()==null){
            //got to a leaf or root has no children
            returnValue=0;
        }else if(root!=null ){//if root isn't null
```

```
81              int depthL=nadir(root.getLeft());//get height of left subtree
82              int depthR=nadir(root.getRight());//get height of right subtree
83              if(depthL<depthR) {//decide which side is shorter
84                  returnValue = 1 + depthL;
85              }else {
86                  returnValue=1+depthR;
87              }
88          }
89          return returnValue;
90      }
91
92      // delete methods not used was not required but was in skeleton code
93      public void delete(String d) {}
94      private StringAVLNode delete(StringAVLNode t, String d) { return t;}
95
96      // Return the number of leaves in the tree
97      public int leafCt() { return leafCt(this.root); }
98      private static int leafCt(StringAVLNode root) {
99          int returnValue;
100         if(root==null){//empty tree return 0
101             returnValue=0;
102         }else if(root.getLeft()==null&&root.getRight()==null){//reached end of tree || one node tree
103             returnValue=1;
104         }else {//add left and right sub tree leafct
105             returnValue=leafCt(root.getLeft())+leafCt(root.getRight());
106         }
107         return returnValue;
108     }
109
110     // Return the number of perfectly balanced AVL nodes
111     public int balanced() {return balanced(this.root);}
112     private static int balanced(StringAVLNode root) {
113         int returnValue=0;//assume its null
114         if(root != null) {//if not null
115             if (root.getBalance() == 0) {// if balance =0
116                 returnValue = 1;
117             }
118             returnValue += balanced(root.getLeft())+balanced(root.getRight());
119         }
120         return returnValue;
```

```java
    }

    // Return the inorder successor, i.e., the next larger value in the tree
    // or null if there is none or str is not in the tree
    public String successor(String str) {
        StringAVLNode returnValue=null;//the node that we will return its value
        StringAVLNode parent=null;//parent of node will be used if we need to look at parents

        //parents stack for all the parents of the node we will look for
        // will be used if we need to look more than one parent up
        Deque<StringAVLNode> parents=new LinkedList<>();
        StringAVLNode root=this.root;//root of the tree and will be used as the node we want its successor
        StringAVLNode current=this.root;//will be used in the loop
        boolean found=false;//if its found set this to true
        //find the node
        while(current!=null){
            if(str.compareToIgnoreCase(current.getItem())==0) {//found
                root=current;
                found=true;
                current=null;
            }else if(str.compareToIgnoreCase(current.getItem())<0){//left
                parent=current;
                parents.push(parent);
                current=current.getLeft();
            }else {//right
                parent=current;
                parents.push(parent);
                current=current.getRight();
            }
        }
        if(!found || root==null) {//if not found in the tree or root equal null
            returnValue=null;
        }else if(root.getRight()!=null){
            // if node we want its successor have a right subtree then the successor is there
            StringAVLNode tempNode=root.getRight();//get the right node of the node we want its successor
            while(tempNode.getLeft()!=null){//go all the way to the left
                tempNode=tempNode.getLeft();
            }
            returnValue=tempNode;
        }else{//if it doesnot have a right subtree
```

```
161             if(parent==null){
162                 //there were no successor on the right of the required node
163                 // and no parent meaning no successor
164                 returnValue=null;
165             }else if (parent.getLeft()!=null&&str.compareToIgnoreCase(parent.getLeft().getItem())==0){
166                 //the parent is the successor
167                 returnValue=parent;
168             }else if(parent.getRight()!=null&&str.compareToIgnoreCase(parent.getRight().getItem())==0){
169                 //required element is right of parent
170                 //here we use the parents stack from earlier once
171                 // we find a parent that is bigger we exit and that is the successor
172                 if(!parents.isEmpty()){
173                     parents.pop();//remove parent of current node (ie the node that is equal to str)
174                 }
175                 boolean exit=false;
176                 while (!exit&&!parents.isEmpty()){
177                     parent=parents.pop();
178                     if(str.compareTo(parent.getItem())<0){//if parent is bigger than input string
179                         returnValue=parent;
180                         exit=true;
181                     }else{
182                         returnValue=null;
183                     }
184                 }
185 
186             }
187 
188         }
189         String returnString=(returnValue!=null)?returnValue.getItem():null;
190         return returnString;
191     }
192     //insert a node and do nothing if node exists
193     public void insert(String str) {
194         this.root=insert(str,this.root);
195     }
196     private static StringAVLNode insert(String str, StringAVLNode t) {
197         if(t==null){
198             t=new StringAVLNode(str);
199         }else if(str.compareToIgnoreCase(t.getItem())==0) {
200             //Silence is golden..
```

```
201             //input is in the tree do nothing
202         }
203         else if(str.compareToIgnoreCase(t.getItem())<0){//insert left
204             int OldBalance;
205             if(t.getLeft()!=null){
206                 OldBalance=t.getLeft().getBalance();
207             }else {
208                 OldBalance=282;
209             }
210             t.setLeft(insert(str,t.getLeft()));
211             int newBalance=t.getLeft().getBalance();
212             if((OldBalance==0&& newBalance!=0)|| OldBalance==282){
213                 t.setBalance(t.getBalance()-1);
214                 if(t.getBalance()==-2){//either ll or lr
215                     if(t.getLeft().getBalance()<0){//ll
216                         t=rotateRight(t);//single rotation
217                         t.setBalance(0);
218                         t.getRight().setBalance(0);
219                     }else {//lr
220                         t.setLeft(rotateLeft(t.getLeft()));
221                         t=rotateRight(t);
222                         //fix balance factor
223                         if (t.getBalance() > 0) {
224                             t.getLeft().setBalance(-1);
225                             t.getRight().setBalance(0);
226                         } else {
227                             if (t.getBalance() == 0) {
228                                 t.getRight().setBalance(0);
229                                 t.getLeft().setBalance(0);
230                             } else {
231 
232                                 t.getLeft().setBalance(1);
233                                 t.getLeft().setBalance(0);
234                                 t.getRight().setBalance(1);
235                             }
236                         }
237                         t.setBalance(0);
238                     }
239                 }
240             }
```

```
241         }else {//insert right
242             int OldBalance;
243             if(t.getRight()!=null){
244                 OldBalance=t.getRight().getBalance();
245             }else {
246                 OldBalance=282;
247             }
248             t.setRight(insert(str,t.getRight()));
249             int newBalance=t.getRight().getBalance();
250             if((OldBalance==0&& newBalance!=0)|| OldBalance==282){
251                 t.setBalance(t.getBalance()+1);
252                 if(t.getBalance()==2){//either rr or rl
253                     if(t.getRight().getBalance()>0){//rr
254                         t=rotateLeft(t);//single rotation
255                         t.setBalance(0);
256                         t.getLeft().setBalance(0);
257                     }else {//rl
258                         t.setRight(rotateRight(t.getRight()));
259                         t = rotateLeft(t);
260                         if (t.getBalance() > 0) {
261                             t.getLeft().setBalance(-1);
262                             t.getRight().setBalance(0);
263                         } else {
264                             if (t.getBalance() == 0) {
265                                 t.getLeft().setBalance(0);
266                                 t.getRight().setBalance(0);
267                             } else {
268                                 t.getLeft().setBalance(0);
269                                 t.getRight().setBalance(1);
270                             }
271                         }
272                         t.setBalance(0);
273                     }
274                 }
275             }
276         }
277         return t;
278     }
279 
280     // who are you? Put your name here!
```

```java
    public static String myName() {
        return "Bishoy Abdelamlik";

    }
} // end of StringAVLTree class
class StringAVLNode {
    private String item;
    private int balance;
    private StringAVLNode left, right;

    // just one constructor, please
    public StringAVLNode(String str) {
        this.left=null;
        this.right=null;
        this.balance=0;
        this.item=str;
    }
    public int getBalance() {return this.balance;}
    public void setBalance(int bal) {
        this.balance=bal;
    }
    public String getItem() {return this.item;}
    // no setItem
    public StringAVLNode getLeft() {return this.left;}
    public void setLeft(StringAVLNode pt) {
        this.left=pt;
    }
    public StringAVLNode getRight() {return this.right;}
    public void setRight(StringAVLNode pt) {
        this.right=pt;
    }
}
```