```java
package Assign_2;

/** This class is an implementation of the main class
 for this Virus Tree program.
 * Micah Rose-Mighty
 * 6498935
 * 2020-10-08
 * Created using IntelliJ
 */

import java.text.MessageFormat;

public class Main {

    public static void main(String[] args) {

        System.out.println("Tree Created: ");
        VirusTree vt = new VirusTree("
tree_of_virus_input.txt");
        System.out.println();

        System.out.println("Depth of Tree: ");
        System.out.print(vt.depth());
        System.out.println();
        System.out.println();

        System.out.println("Breadth-First Traversal
: ");
        vt.bfs();
        System.out.println();

        System.out.println("Pre-order Traversal: ");
        vt.preorder();
        System.out.println();

        System.out.println("Post-order Traversal: ");
        vt.postorder();
        System.out.println();

        String v1 = "HCoV-OC43";
        String v2 = "Hcov-229E";
        System.out.println(MessageFormat.format("
Distance between {0} and {1}: ", v1, v2));
        vt.distance(v1, v2);
```

```
41          System.out.println();
42
43          String v3 = "SARS-CoV";
44          String v4 = "Zika virus";
45          System.out.println(MessageFormat.format("
    Distance between {0} and {1}: ", v3, v4));
46          vt.distance(v3, v4);
47          System.out.println();
48
49      }
50  }
51
```

```java
package Assign_2;

/** This class is an implementation of the VirusTree Structure along with all the required methods.
 * Micah Rose-Mighty
 * 6498935
 * 2020-10-08
 * Created using IntelliJ
 */

import java.io.File;
import java.io.IOException;
import java.text.MessageFormat;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Scanner;

public class VirusTree {

    private VirusTreeNode root; // tree root node

    public VirusTree() { root = null; } // empty tree constructor

    public VirusTree(String filename){ // Constructor that creates tree structure from given input file
        this();
        try(Scanner scanner = new Scanner(new File(filename))) {
            while(scanner.hasNextLine()){
                String[] parts = scanner.nextLine().trim().split(",");

                for(int i = parts.length-1; i>=1; i--){
                    if(!insert(parts[0], parts[i])){
                        throw new IllegalArgumentException("Can not find parent node: " + parts[0]);
                    }
                }
                System.out.println(MessageFormat.format("{0}: {1}", parts[0], String.join(" -> ", Arrays.copyOfRange(parts, 1, parts.length))));
```

```
35              }
36          }
37          catch (IOException e) {
38              throw new RuntimeException(e);
39          }
40      }
41
42      public int depth() { // Method for getting the
    depth of the given tree using inner recursive method
    for calculation.
43          if(root == null){
44              return 0;
45          }
46          return depthStep(root);
47      }
48
49      public void bfs(){ // Method for outputting the
    given tree nodes in breadth-first traversal order.
50          if (root == null) {
51              return;
52          }
53
54          LinkedList<VirusTreeNode> queue = new
    LinkedList<>();
55          queue.addLast(root);
56
57          while(!queue.isEmpty()){
58              VirusTreeNode current = queue.removeFirst
    ();
59              System.out.println(current.info);
60
61              VirusTreeNode currChild = current.
    firstChild;
62              while(currChild != null){
63                  queue.addLast(currChild);
64                  currChild = currChild.nextSibling;
65              }
66          }
67      }
68
69      public void preorder(){ // Method for outputting
    the given tree in Pre-order Traversal Order by
    calling traverse method and setting it to true.
70          traverse(true);
```

```
71      }
72
73      public void postorder(){ // Method for
    outputting the given tree in Post-order Traversal
    Order by calling traverse method and setting it to
    false.
74          traverse(false);
75      }
76
77      public void distance(String info1, String info2
    ){ //Method for outputting the distance between two
    nodes in the given tree.
78          if(root == null) {
79              throw new IllegalArgumentException("Can'
    t find node: " + info1);
80          }
81
82          LinkedList<String> path1 = getPath(root,
    info1, new LinkedList<>());
83          if(path1 == null){
84              throw new IllegalArgumentException("Can'
    t find node: " + info1);
85          }
86
87          LinkedList<String> path2 = getPath(root,
    info2, new LinkedList<>());
88          if(path2 == null){
89              throw new IllegalArgumentException("Can'
    t find node: " + info2);
90          }
91
92          for (int i = path1.size() - 1; i>=0; i--) {
    // For loop to find last common ancestor starting
    from last node in first path and if found in second
    path then that is common ancestor.
93              String pathInfo = path1.get(i);
94              int index = path2.indexOf(pathInfo);
95              if(index >= 0) {
96                  int distance = Math.max(path1.size
    () - 1 - i, path2.size() - 1 - index);
97                  System.out.println(MessageFormat.
    format("The distance between {0} and {1} is {2}.
    They have common ancestor {3}.", info1, info2,
    distance, pathInfo));
```

```
 98                 return;
 99             }
100         }
101         throw new IllegalArgumentException(
    MessageFormat.format("There does not exist a common
    ancestor node for {0} and {1}", info1, info2));
102     }
103 
104     private LinkedList<String> getPath(VirusTreeNode
     currentNode, String info, LinkedList<String>
    currentList) { // Private recursive method for
    searching a path to a given node.
105         String currentInfo = currentNode.info;
106         currentList.add(currentInfo);
107         if(currentInfo.equals(info)) {
108             return currentList;
109         }
110 
111         VirusTreeNode currChild = currentNode.
    firstChild;
112         while (currChild != null){
113             LinkedList<String> result = getPath(
    currChild, info, currentList);
114             if(result != null) {
115                 return result;
116             }
117             currChild = currChild.nextSibling;
118         }
119         currentList.removeLast();
120         return null;
121     }
122 
123     private void traverse(boolean preorder) { //
    Inner traverse method the outputs Pre-order
    Traversal if true and Post-order Traversal if false.
124         if (root == null) {
125             return;
126         }
127         traverseStep(root, preorder);
128     }
129 
130     private void traverseStep(VirusTreeNode
    currentNode, boolean preorder) { // Inner recursive
    traverse order method that is based on the given
```

```
130 node.
131
132         if (preorder) {
133             System.out.println(currentNode.info);
134         }
135
136         VirusTreeNode currChild = currentNode.
    firstChild;
137         while(currChild != null) {
138             traverseStep(currChild,preorder);
139             currChild = currChild.nextSibling;
140         }
141
142         if (!preorder) {
143             System.out.println(currentNode.info);
144         }
145     }
146
147     private int depthStep(VirusTreeNode currentNode
    ) { // Inner recursive method for calculating depth.
148         int max = 0;
149
150         VirusTreeNode currChild = currentNode.
    firstChild;
151         while (currChild != null) {
152             max = Math.max(max, 1 + depthStep(
    currChild));
153             currChild = currChild.nextSibling;
154         }
155         return max;
156     }
157
158     private boolean insert(String parentInfo, String
     childInfo) { // Private method for adding parent-
    child pair into the given tree. Returns true if pair
     is added successfully and false otherwise.
159
160         if (root == null) {
161             root = new VirusTreeNode(parentInfo);
162             root.firstChild =  new VirusTreeNode(
    childInfo);
163             root.nextSibling = null;
164             root.firstChild.firstChild = null;
165             root.firstChild.nextSibling = null;
```

```
166
167             return true;
168         }
169         return insertStep(root, parentInfo,
    childInfo);
170     }
171
172     private boolean insertStep(VirusTreeNode
    currentNode, String parentInfo, String childInfo) {
    // Inner recursive method for inserting a parent-
    child pair into the given tree. Also returns true if
     pair was successfully added and false otherwise.
173
174         if (currentNode.info.equals(parentInfo)) {
175             VirusTreeNode newNode = new
    VirusTreeNode(childInfo);
176             newNode.firstChild = null;
177             newNode.nextSibling = currentNode.
    firstChild;
178             currentNode.firstChild = newNode;
179
180             return true;
181         }
182         VirusTreeNode currChild = currentNode.
    firstChild;
183         while (currChild != null) {
184             if (insertStep(currChild, parentInfo,
    childInfo)) {
185
186                 return true;
187             }
188             currChild = currChild.nextSibling;
189         }
190         return false;
191     }
192 }
193
```

```java
package Assign_2;

/** This class is an implementation of the VirusTree Node.
 * Micah Rose-Mighty
 * 6498935
 * 2020-10-08
 * Created using IntelliJ
 */


public class VirusTreeNode {

    String info;

    VirusTreeNode firstChild;
    VirusTreeNode nextSibling;

    VirusTreeNode(String info){ this.info = info; } //Node data constructor
}

```

```
 1 Virus,DNA Virus,RNA Virus
 2 RNA Virus,Flaviviridae,Filoviridae,Coronaviridae
 3 Flaviviridae,Flavivirus,Hepacivirus
 4 Filoviridae,Cuevavirus,Dianlovirus,Ebolavirus,
   Marburgvirus
 5 Coronaviridae,Alphacoronavirus,Betacoronavirus,
   Deltacoronavirus,Gammacoronavirus,Alphaletovirus
 6 Flavivirus,Zika virus,Yellow fever virus,West Nile
   virus
 7 Hepacivirus,GBV-B,hepatitis C virus
 8 Ebolavirus,Ebola virus,Tai Forest virus,Bombali virus
 9 Marburgvirus,Marburg virus
10 Alphacoronavirus,Hcov-229E,Hcov-NL63
11 Betacoronavirus,HCoV-OC43,HCoV-HKU1
12 Betacoronavirus,SARS-CoV
13 Betacoronavirus,MERS-CoV
14 DNA Virus,Adenoviridae,Poxviridae
15 Adenoviridae,Mastadenovirus
16 Poxviridae,Orthopoxvirus
17 Mastadenovirus,Canine hepatitis virus
18 Orthopoxvirus,Cowpox virus,Smallpox virus
```