```java
import java.io.*;
import java.util.*;

class BTNode{

    private int nodeid;
    private int data;
    private int levelNum;
    private BTNode leftChildPtr;
    private BTNode rightChildPtr;

    public BTNode(){}

    public  void setNodeId(int id){
        nodeid = id;
    }

    public  int getNodeId(){
        return nodeid;
    }

    public  void setData(int d){
        data = d;
    }

    public  int getData(){
        return data;
    }

    public  void setLevelNum(int level){
        levelNum = level;
    }

    public  int getLevelNum(){
        return levelNum;
    }

    public  void setLeftChildPtr(BTNode ptr){
        leftChildPtr = ptr;
    }

    public  void setRightChildPtr(BTNode ptr){
        rightChildPtr = ptr;
    }

    public  BTNode getLeftChildPtr(){
        return leftChildPtr;
    }

    public  BTNode getRightChildPtr(){
        return rightChildPtr;
    }

    public  int getLeftChildID(){
        if (leftChildPtr == null)
            return -1;

        return leftChildPtr.getNodeId();
    }

    public  int getRightChildID(){
        if (rightChildPtr == null)
            return -1;

```

```
65          return rightChildPtr.getNodeId();
66      }
67  }
68
69
70
71  class Node{
72
73      private int data;
74      private Node nextNodePtr;
75      private Node prevNodePtr;
76
77      public Node(){}
78
79      public void setData(int d){
80          data = d;
81      }
82
83      public int getData(){
84          return data;
85      }
86
87      public void setNextNodePtr(Node nodePtr){
88          nextNodePtr = nodePtr;
89      }
90
91      public Node getNextNodePtr(){
92          return nextNodePtr;
93      }
94
95      public void setPrevNodePtr(Node nodePtr){
96          prevNodePtr = nodePtr;
97      }
98
99      public Node getPrevNodePtr(){
100         return prevNodePtr;
101     }
102
103 }
104
105 class Queue{
106
107     private Node headPtr;
108     private Node tailPtr;
109
110     public  Queue(){
111         headPtr = new Node();
112         tailPtr = new Node();
113         headPtr.setNextNodePtr(null);
114         tailPtr.setPrevNodePtr(null);
115     }
116
117     public Node getHeadPtr(){
118         return headPtr;
119     }
120
121     public  Node getTailPtr(){
122         return tailPtr;
123     }
124
125     public  boolean isEmpty(){
126
127         if (headPtr.getNextNodePtr() == null)
128             return true;
```

```
129
130             return false;
131         }
132
133
134         public  void enqueue(int data){
135
136             Node newNodePtr = new Node();
137             newNodePtr.setData(data);
138             newNodePtr.setNextNodePtr(null);
139
140             Node lastNodePtr = tailPtr.getPrevNodePtr();
141
142             if (lastNodePtr == null){
143
144                 headPtr.setNextNodePtr(newNodePtr);
145                 newNodePtr.setPrevNodePtr(null);
146
147             }
148             else{
149
150                 lastNodePtr.setNextNodePtr(newNodePtr);
151                 newNodePtr.setPrevNodePtr(lastNodePtr);
152
153             }
154
155             tailPtr.setPrevNodePtr(newNodePtr);
156
157         }
158
159
160         public  int dequeue(){
161
162             Node firstNodePtr = headPtr.getNextNodePtr();
163             Node nextNodePtr = null;
164
165             int poppedData = -100000; //empty queue
166
167             if (firstNodePtr != null){
168                 nextNodePtr = firstNodePtr.getNextNodePtr();
169                 poppedData = firstNodePtr.getData();
170             }
171             else
172                 return poppedData;
173
174             if (nextNodePtr != null){
175                 nextNodePtr.setPrevNodePtr(null);
176                 headPtr.setNextNodePtr(nextNodePtr);
177             }
178             else{
179                 headPtr.setNextNodePtr(null);
180                 tailPtr.setPrevNodePtr(null);
181             }
182
183             return poppedData;
184
185         }
186
187
188         public  int peek(){
189
190             Node firstNodePtr = headPtr.getNextNodePtr();
191
192             if (firstNodePtr != null)
```

```
193             return firstNodePtr.getData();
194         else
195             return -100000; //empty queue
196 
197     }
198 
199 
200 }
201 
202 
203 class BinaryTree{
204 
205     private int numNodes;
206     private BTNode arrayOfBTNodes[];
207 
208     public BinaryTree(int n){
209         numNodes = n;
210         arrayOfBTNodes = new BTNode[numNodes];
211 
212         for (int id = 0; id < numNodes; id++){
213             arrayOfBTNodes[id] = new BTNode();
214             arrayOfBTNodes[id].setNodeId(id);
215             arrayOfBTNodes[id].setLevelNum(-1);
216             arrayOfBTNodes[id].setLeftChildPtr(null);
217             arrayOfBTNodes[id].setRightChildPtr(null);
218         }
219     }
220 
221     public  void setLeftLink(int upstreamNodeID, int downstreamNodeID){
222         arrayOfBTNodes[upstreamNodeID].setLeftChildPtr(arrayOfBTNodes[downstreamNodeID]);
223     }
224 
225     public  void setRightLink(int upstreamNodeID, int downstreamNodeID){
226 
        arrayOfBTNodes[upstreamNodeID].setRightChildPtr(arrayOfBTNodes[downstreamNodeID])
        ;
227     }
228 
229 
230     public void printLeafNodes(){
231 
232         for (int id = 0; id < numNodes; id++){
233 
234             if (arrayOfBTNodes[id].getLeftChildPtr() == null &&
            arrayOfBTNodes[id].getRightChildPtr() == null)
235                 System.out.print(id + " ");
236             }
237 
238         System.out.println();
239     }
240 
241 
242     public  boolean isLeafNode(int nodeid){
243 
244         if (arrayOfBTNodes[nodeid].getLeftChildPtr() == null &&
        arrayOfBTNodes[nodeid].getRightChildPtr() == null)
245             return true;
246 
247         return false;
248     }
249 
250 
251     public  int getNodeHeight(int nodeid){
252 
```

```
253         if (nodeid == -1)
254             return -1;
255 
256         if (isLeafNode(nodeid) )
257             return 0;
258 
259         int leftChildID = arrayOfBTNodes[nodeid].getLeftChildID(); // -1 if not exist
260         int rightChildID = arrayOfBTNodes[nodeid].getRightChildID(); // -1 if not exist
261 
262         return Math.max(getNodeHeight(leftChildID), getNodeHeight(rightChildID)) + 1;
263 
264     }
265 
266 
267     public  int getTreeHeight(){
268         return getNodeHeight(0);
269     }
270 
271 
272     public  void assignLevelNumbers(){
273 
274         Queue queue = new Queue();
275         queue.enqueue(0);
276         arrayOfBTNodes[0].setLevelNum(0);
277 
278         while (!queue.isEmpty()){
279 
280             int firstNodeInQueue = queue.dequeue();
281 
282             int leftChildID = arrayOfBTNodes[firstNodeInQueue].getLeftChildID();
283             if (leftChildID != -1){
284                 queue.enqueue(leftChildID);
285 
                    arrayOfBTNodes[leftChildID].setLevelNum(arrayOfBTNodes[firstNodeInQueue].
                    getLevelNum()+1);
286             }
287 
288             int rightChildID = arrayOfBTNodes[firstNodeInQueue].getRightChildID();
289             if (rightChildID != -1){
290                 queue.enqueue(rightChildID);
291 
                    arrayOfBTNodes[rightChildID].setLevelNum(arrayOfBTNodes[firstNodeInQueue]
                    .getLevelNum()+1);
292             }
293 
294         }
295 
296 
297     }
298 
299     public  int getDepth(int nodeid){
300         return arrayOfBTNodes[nodeid].getLevelNum();
301     }
302 
303 }
304 
305 
306 class BinaryTreeDepth{
307 
308     public static void main(String[] args){
309 
310     try{
311 
312     Scanner input = new Scanner(System.in);
```

```
313
314         String filename;
315         System.out.print("Enter a file name: ");
316         filename = input.next();
317
318         int numNodes;
319         System.out.print("Enter number of nodes: ");
320         numNodes = input.nextInt();
321
322         BinaryTree binaryTree = new BinaryTree(numNodes);
323
324         FileReader fr = new FileReader(filename);
325         BufferedReader br = new BufferedReader(fr);
326
327         String line = null;
328
329         while ( (line = br.readLine()) != null){
330
331             StringTokenizer stk = new StringTokenizer(line, ",: ");
332
333             int upstreamNodeID = Integer.parseInt(stk.nextToken());
334
335             int childIndex = 0;
336
337             while (stk.hasMoreTokens()){
338
339                 int downstreamNodeID = Integer.parseInt(stk.nextToken());
340
341                 if (childIndex == 0 && downstreamNodeID != -1)
342                     binaryTree.setLeftLink(upstreamNodeID, downstreamNodeID);
343
344                 if (childIndex == 1 && downstreamNodeID != -1)
345                     binaryTree.setRightLink(upstreamNodeID, downstreamNodeID);
346
347                 childIndex++;
348
349             }
350
351         }
352
353
354         binaryTree.assignLevelNumbers();
355
356         for (int id = 0; id < numNodes; id++)
357             System.out.println("Depth of Node " + id + " : " + binaryTree.getDepth(id) );
358
359
360         }
361         catch(Exception e){e.printStackTrace();}
362
363         }
364     }
```

```
Enter a file name: binaryTreeFile_1.txt
Enter number of nodes: 10
Depth of Node 0 : 0
Depth of Node 1 : 1
Depth of Node 2 : 1
Depth of Node 3 : 2
Depth of Node 4 : 2
Depth of Node 5 : 2
Depth of Node 6 : 3
Depth of Node 7 : 3
Depth of Node 8 : 3
Depth of Node 9 : 4
```