  
  
**Assignment Cover Sheet**

|  |  |
| --- | --- |
| **Subject Code: CSCI203** |  |
| **Subject Name: Algorithms and Data Structure** |  |
| **Submission Type: Online** |  |
| **Assignment Title: CSCI203 Assignment 2 Design** |  |
| **Student Name: Abdul Rahman Al Sbeinaty, Karan Maitra** |  |
| **Student Number: 6979270, 6346777** |  |
| **Student Phone/Mobile No.: 055-9419429, 054-5156770** |  |
| **Student E-mail:** [**arfas380@uowmail.edu.au**](mailto:arfas380@uowmail.edu.au)**, karanmaitra@gmail.com** |  |
| **Lecturer Name: Halim** |  |
| **Due Date: 21/02/2022** |  |
| **Date Submitted: 21/02/2022** |  |

|  |  |
| --- | --- |
| **PLAGIARISM:** The penalty for deliberate plagiarism is FAILURE in the subject. Plagiarism is cheating by using the written ideas or submitted work of someone else. UOWD has a strong policy against plagiarism.  The University of Wollongong in Dubai also endorses a policy of non-discriminatory language practice and presentation.  **PLEASE NOTE:**STUDENTS MUST RETAIN A COPY OF ANY WORK SUBMITTED | **DECLARATION:** I/We certify that this is entirely my/our own work, except where I/we have given fully-documented references to the work of others, and that the material contained in this document has not previously been submitted for assessment in any formal course of study. I/we understand the definition and consequences of plagiarism.  **Signature of Student:** |

|  |  |  |
| --- | --- | --- |
| |  | | --- | | **Optional Marks:** | | **Comments:** | |

https://my.uowdubai.ac.ae/images/scissors.gif

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Lecturer Assignment Receipt**(To be filled in by student and retained by Lecturer upon return of assignment) | | | **Subject:** | **Assignment Title:** | | **Student Name:** | **Student Number:** | | **Due Date:** | **Date Submitted:** | | **Signature of Student:** | | |

https://my.uowdubai.ac.ae/images/scissors.gif

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Student Assignment Receipt** (To be filled in and retained by Student upon submission of assignment) | | | **Subject:** | **Assignment Title:** | | **Student Name:** | **Student Number:** | | **Due Date:** | **Date Submitted:** | | **Signature of Lecturer** | | |

Contents

**No table of contents entries found.**

# Brief Explanation

* The approach we used to solve this problem was firstly we ask the user at what time does he want to see the infection because then while scanning we would get rid of unnecessary connections that exceed that time which works as an upper limit. Then we scan the infected computer’s file and sort the nodes within that file and we connect all the pre-infected nodes to a virtual node that connects them all together that way we can create an imaginary tree and after that we scan the dataset file and add the edges or so-called connections to the tree. Then we used a DFS method of traversing through the tree to find the infected computers and the sequence that got them infected.

# Pseudocode

Class Node

{

Attributes:

Int label;

Int infectionTime;

String infectionSequence;

//setters and getters for the attributes

}

Class Edge

{

Attributes:

Node n1,n2;

Int connectionTime;

//setters and getters for the attributes

}

Class Graph

{

Attributes:

List<Edge>[] map; //this will be initialized with an empty list of edges within every index

Node[] nodes; //this will be filled with default nodes from 0-100 (100 being the virtual node) with infectionTime set to -1 as a default value

//setters and getters

Functions:

Node getNodeByLabel(int label){return nodes[label];}

Void addEdge(int label1,int label2,int connectionTime)

{

Node n1 <- nodes[label1];

Node n2 <- nodes[label2];

Edge edge <- new Edge(n1,n2,connectionTime);

Map[label1].add(edge);

}

List<Node> DFSForInfected(int root)

{

List<Edge> stack;

Stack.addAll(map[root]);

While(stack.size() != 0)

{

Edge curr <- stack.remove(0);

Node n1 <- nodes[curr.getN1().getLabel()];

Node n2 <- nodes[curr.getN2().getLabel()];

If(n1.getInfectionTime() >= 0 && n2.getInfectionTime() < 0 || n1.getLabel() == 100)

{

Infect(n1,n2,curr.getConnectionTime());

Stack.addAll(0,map[curr.getN2().getLabel())];

}

}

List<Node> infectedNodes;

For(Node n in infectedNodes)

If(n.getInfectionTime() >= 0) infectedNodes.add(n);

Return infectedNodes;

}

Void infect(Node n1,Node n2,int connectionTime)

{

If(n1.getInfectionTime() >= 0 && n2.getInfectionTime() < 0)

{

If(connectionTime >= n1.getInfectionTime())

{

N2.setInfectionTime(connectionTime);

N2.setInfectionSequence(n1.getInfectionSequence() + “ ” + n1.getLabel());

}

}

}

}

Class Main

{

Main function()

{

Scanner scan; //scans from user

FileScanner infectedScanner; //scans from infected computers file

FileScanner dataSetScanner; //scans from dataset file

infectedScanner.nextLine(); //clears the first line

dataSetScanner.nextLine(); //clears the first line

print(“Enter time to find infected computers:”);

int time <- scan.nextInt();

long start = currentTimeInMillis;

Graph graph;

List<Node> infected;

While(infectedScanner.hasNextInt())

{

Int label <- infectedScanner.nextInt();

Int infectionTime <- infectedScanner.nextInt();

If(infectionTime <= time)

{

Node temp <- graph.getNodeByLabel(label);

Temp.setInfectionTime(infectionTime);

Infected.add(temp);

}

}

Sort(infected); //ascending

For(Node n : infected) graph.addEdge(100,n.getLabel(),n.getInfectionTime());

While(dataSetScanner.hasNextInt())

{

Int l1 <- dataSetScanner.nextInt();

Int l2 <- dataSetScanner.nextInt();

Int connectionTime <- dataSetScanner.nextInt();

If(connectionTime <= time)

{

Node n2 <- graph.getNodeByLabel(l2);

If(n2.getInfectionTime() >= 0) graph.addEdge(l2,l1,connectionTime);

Else graph.addEdge(l1,l2,connectionTime);

}

}

Print(“Infected computers at time “ + time + “ are:”);

List<Node> infectedNodes <- graph.DFSForInfected(100);

FileWriter fw; //print to a file

For(Node n in infectedNodes)

{

If(n.getLabel() != 100)

Fw.write(“Computer#” + n.getLabel() + “ at time “ + n.getInfectionTime()

+ “:[“ + n.getInfectionSequence() + “ ]\n”);

}

Fw.close();

Long execTime = currentTimeInMillis – start;

Print(“Executed in “ + (execTime/1000.) + “ secs”);

}

}

# Time Complexity

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Code | S/E | Frequency | Total Steps | Notation |
| Node getNodeByLabel(int label) |  |  |  |  |
| { |  |  |  |  |
| Return nodes[label]; | 1 | 1 | 1 | Θ(1) |
| } |  |  |  |  |
| Total: |  |  | 1 | Θ(1) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Code | S/E | Frequency | Total Steps | Notation |
| Void addEdge(int label1,int label2, int connectionTime) |  |  |  |  |
| { |  |  |  |  |
| Node n1 <- nodes[label1]; | 1 | 1 | 1 | Θ(1) |
| Node n2 <- nodes[label2]; | 1 | 1 | 1 | Θ(1) |
| Edge edge <- new Edge(n1,n2,connectionTime); | 1 | 1 | 1 | Θ(1) |
| Map[label1].add(edge); | 1 | 1 | 1 | Θ(1) |
| } |  |  |  |  |
| Total: |  |  | 4 | Θ(1) |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Code | S/E | Frequency | Total Steps | Notation |
| List<Node> DFSForInfected(int root) |  |  |  |  |
| { |  |  |  |  |
| List<Edge> stack; | 1 | 1 | 1 | Θ(1) |
| Stack.addAll(map[root]); | 1 | 1 | 1 | Θ(1) |
| While(stack.size() != 0) |  |  |  |  |
| { |  |  |  |  |
| Edge curr <- stack.remove(0); |  |  |  |  |
| Node n1 <- nodes[curr.getN1().getLabel()]; |  |  |  |  |
| Node n2 <- nodes[curr.getN2().getLabel()]; |  |  |  |  |
| If(n1.getInfectionTime() >= 0 && n2.getInfectionTime() < 0 || n1.getLabel() == 100) |  |  |  |  |
| { |  |  |  |  |
| Infect(n1,n2,curr.getConnectionTime()); |  |  |  |  |
| Stack.addAll(0,map[curr.getN2().getLabel()]); |  |  |  |  |
| } |  |  |  |  |
| } |  |  |  |  |
| List<Node> infectedNodes; |  |  |  |  |
| For(Node n in infectedNodes) |  |  |  |  |
| If(n.getInfectionTime() >= 0) |  |  |  |  |
| infectedNodes.add(n); |  |  |  |  |
| Return infectedNodes; |  |  |  |  |
| } |  |  |  |  |
| Total: |  |  | 4 | Θ(1) |