Computer Networks (CSD304) – Socket Programming
Graded Assignment

Due Date: April 28, 2022 (11:59 pm)

Grading: This assignment has a **weightage of 10%** in your overall 100 points.

Guidelines

- ✓ This assignment aims to make the students familiar with socket programming in computer networks.
- ✓ **This assignment is to be completed individually.**
- ✓ **Programming Language to be used: C or Java -> programs written in any other programming language will not be evaluated and a score of 0 will be given.**
- ✓ Use either UDP or TCP sockets for this assignment.
- ✓ Code should be easy to understand (make proper use of comments, don't overuse them).
- ✓ Assignment submitted after due date and time will not be evaluated and a score of zero will be awarded for this assignment.
- ✓ Material copied from Internet or elsewhere will attract penalty - Plagiarism will not be tolerated.
    - Plagiarism below 60% - No penalty or Minor penalty.
    - Plagiarism between 60% and 75% - 50% marks deduction
    - Plagiarism greater than 75% - 100% marks deduction.

Submission

Each student must upload the following files on Blackboard:

- Paste your code and screenshots of input and output screens (paste them in this file) - Name the document as Socket_CN2022_FirstName_LastName.pdf.
- **Upload zip file of .c (server.c and client.c) OR .java (server.java and client.java)** file on BB.

Question

Write a program that involves a client and a server. The client sends server 4 values, for example *X, n, B, C* where, X is the adjacency matrix of a directed graph with *m* nodes (let's say 5 nodes: A B C D E), and *n* is the length of the path from node B to node C.

The server responds back with two responses:

i. Positive Y response (or Negative N response) if there exists (or doesn't exist) a path of length n from B to C.
ii. The image of the directed graph with nodes proving the validity of the response.

For simplicity, assume that the above graph can have minimum 3 nodes and maximum 10 nodes

For example: Let's take a 3-node directed graph:

Case 1: Client sends the following to the server:

Input:

0	*1*	*0*
1	*0*	*1*
0	*0*	*0*

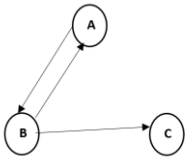
, 2, A, C

Where, there is an adjacency matrix, 2 is the length of the path from node A to node C – that server has to check whether it exists or not.

Server should return the following:

Output 1: Yes, there exists a path of length 2 from node A to node C.

Output 2: Graph:



Case 2: Client sends the following to the server:

Input:

0	*1*	*0*
1	*0*	*1*
0	*0*	*0*

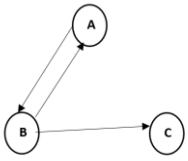
, 2, C, A

Where, there is an adjacency matrix, 2 is the length of the path from node C to node A.

Server should return the following:

Output 1: No, there is no path of length 2 from node C to node A.

Output 2: Graph:

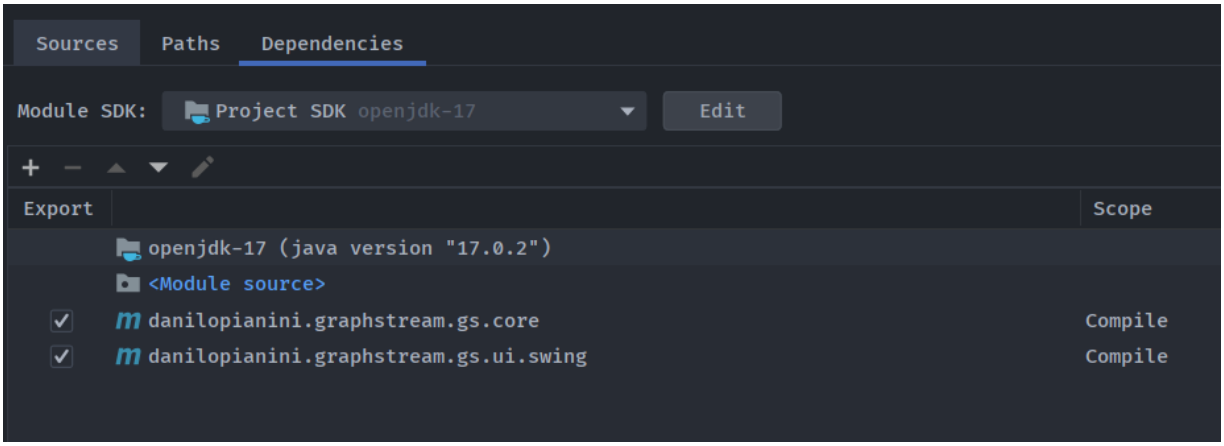


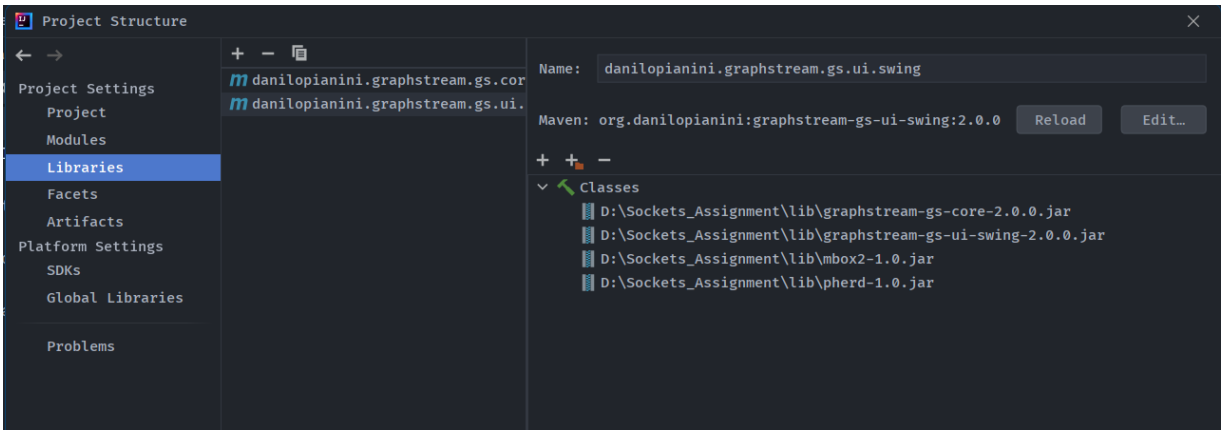
Computer Networks (CSD304) – Socket Programming
Graded Assignment

Submission Template

To run the program:

- Install the libraries **gs-core** and **gs-ui-swing** (gs: GraphStream)
- Used Maven to create the project. So, install these libraries through maven or maybe through official website(http://graphstream-project.org/download/) and add .jar files.
- Load the dependencies in the project and import the required packages to run the project successfully.





- Change the path to read the created image (w.r.t your file directory)

```
//READING THE STORED IMAGE
BufferedImage image = ImageIO.read(new File( pathname: "D:\\Sockets_Assignment\\graph.jpg"));
```

Change the pathname according to your directory in which this project exists or where the created graph image gets saved.

Computer Networks (CSD304) – Socket Programming
Graded Assignment

❖ Screenshots of Input and Output Screens

Sample Input 1:

Number of nodes: 6

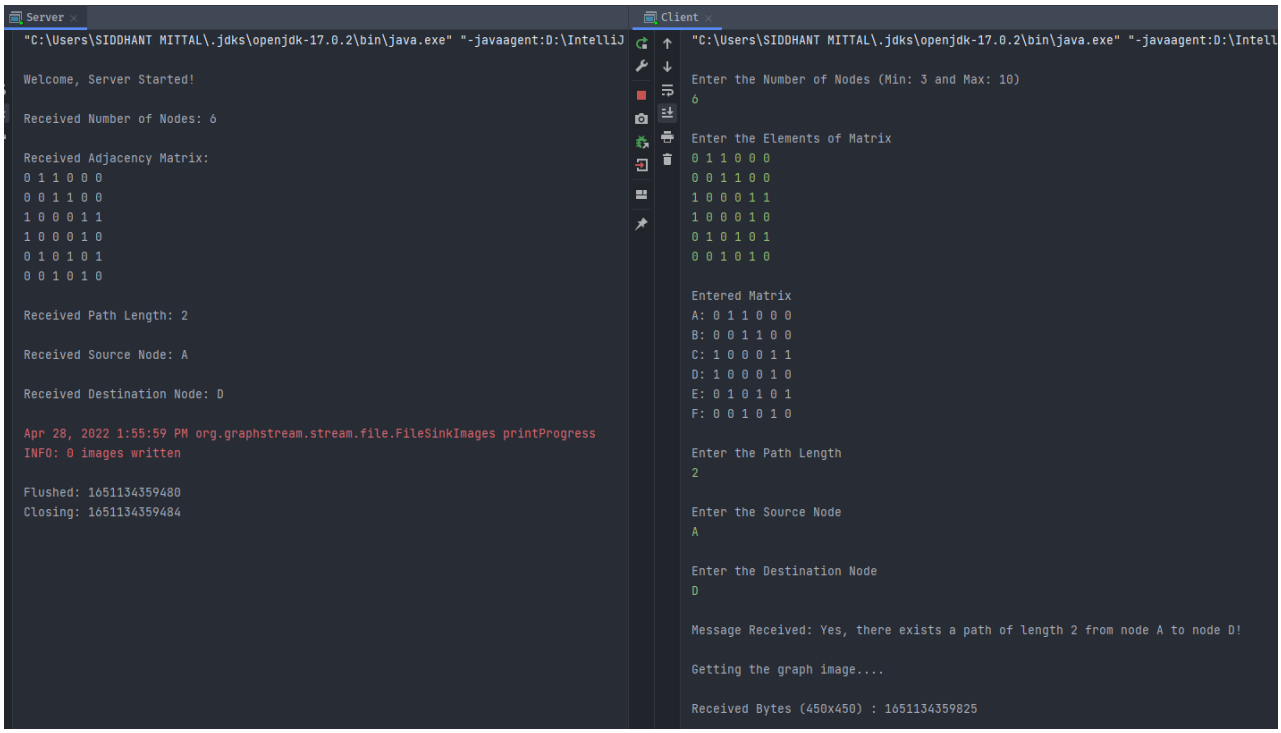
Matrix: 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0

Path Length: 2

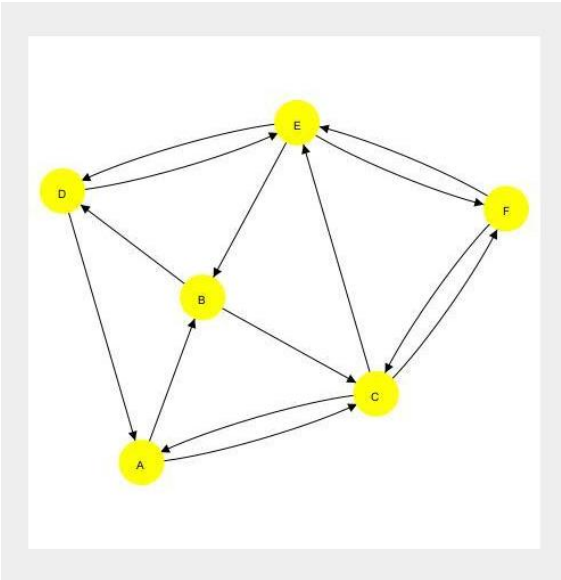
Source Node: A

Destination Node: D

SERVER + CLIENT WINDOWS:

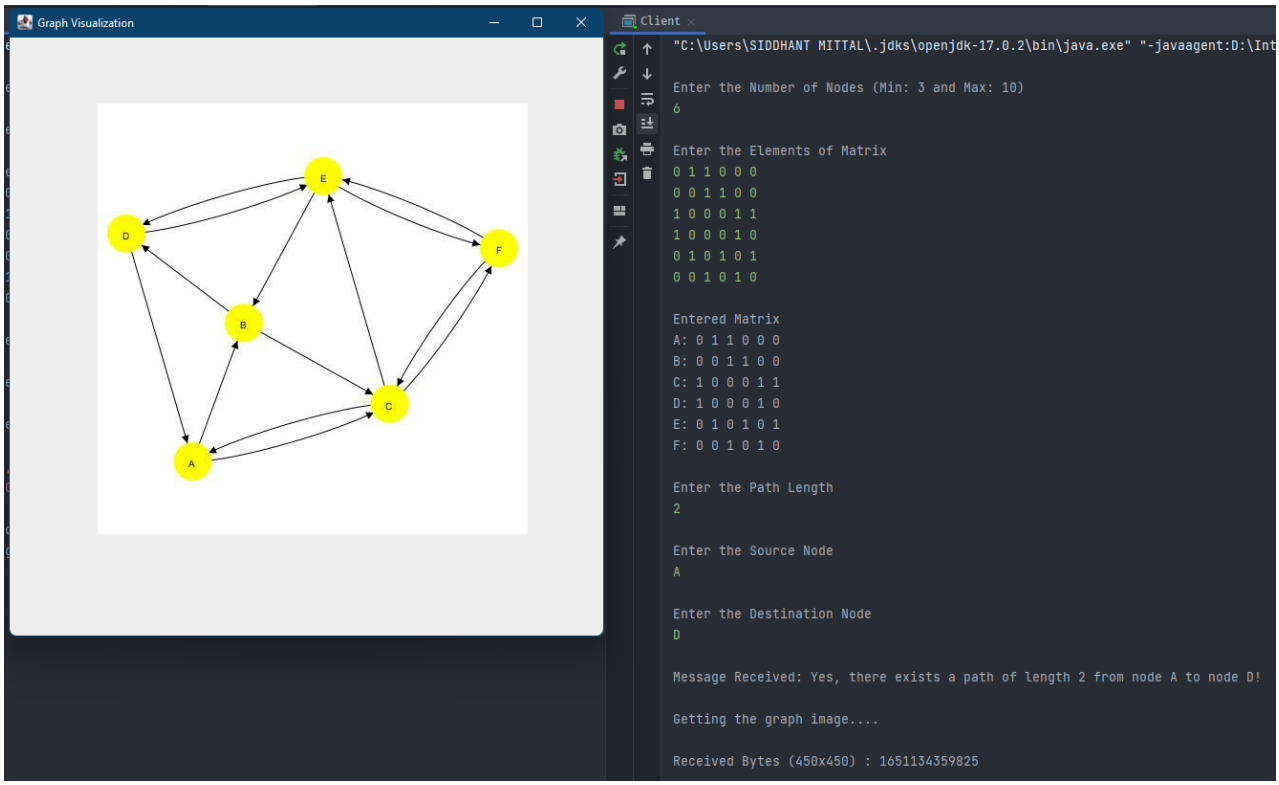


VISUALIZING GRAPH:



Computer Networks (CSD304) – Socket Programming
Graded Assignment

GRAPH + CLIENT WINDOW



Sample Input 2:

Number of nodes: 6

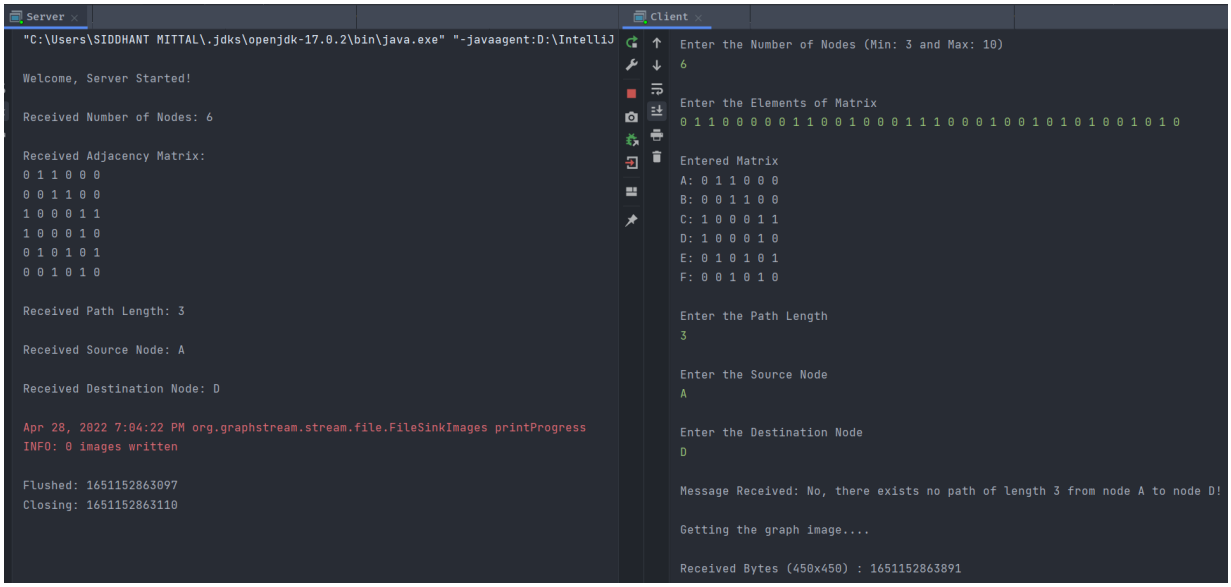
Matrix: 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0

Path Length: 2

Source Node: A

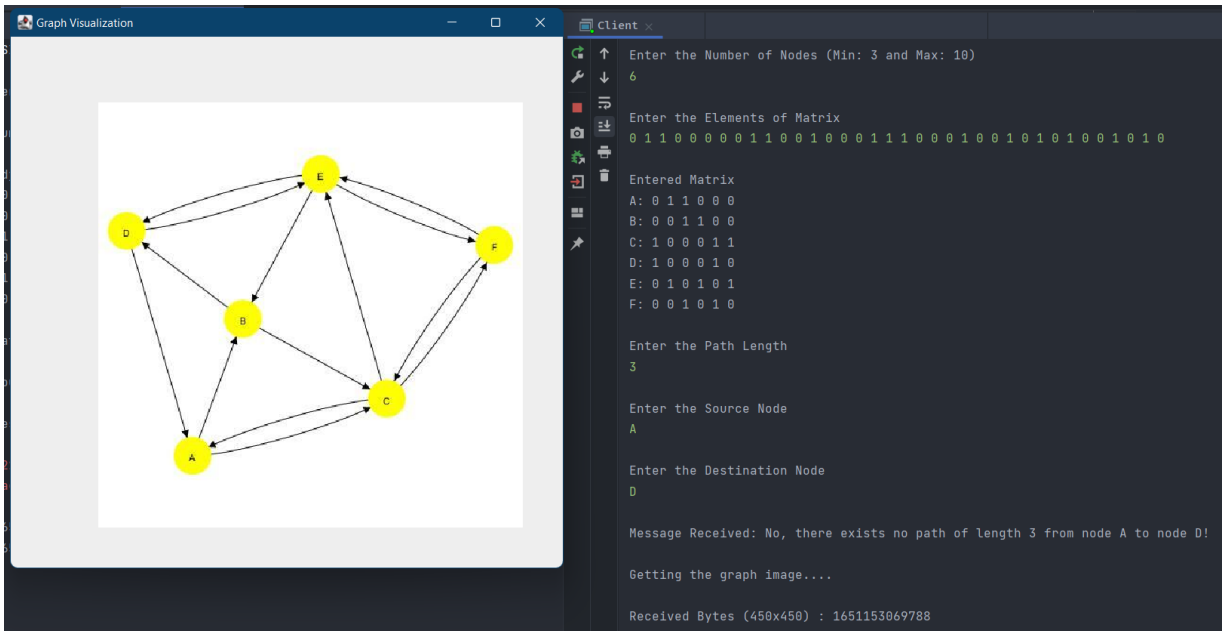
Destination Node: D

SERVER + CLIENT WINDOWS:

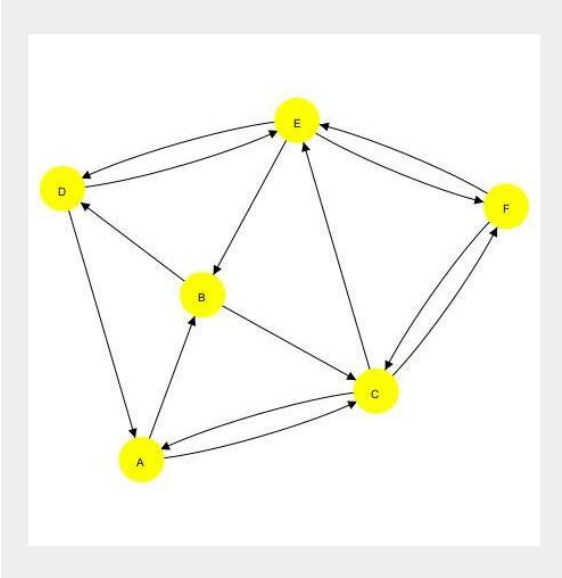


Computer Networks (CSD304) – Socket Programming
Graded Assignment

GRAPH + CLIENT WINDOW



VISUALIZING GRAPH:



❖ Server side code –

```
❖ /*
   * CN-2022 GRADED LAB
   * SIDDHANT MITTAL
   * 1910110388
   */

  /*SERVER SIDE CODE*/

  /* IMPORTS */
  import java.io.*;
  import java.net.*;
  import java.util.*;
  import java.util.List;
  import javax.imageio.ImageIO;
  import java.awt.image.BufferedImage;
  import java.nio.ByteBuffer;
  import org.graphstream.graph.Node;
  import org.graphstream.graph.implementations.MultiGraph;
  import org.graphstream.stream.file.FileSinkImages;
  import org.graphstream.stream.file.FileSinkImages.OutputType;
  import org.graphstream.stream.file.FileSinkImages.LayoutPolicy;

  public class Server {
      /* GLOBAL VARIABLES */
      static int nodes;
      static int[][] gAdjMatrix = new int[10][10];
      static DataOutputStream output;

      /* FUNCTION TO CHECK IF DESIRED PATH LENGTH IS PRESENT IN THE
  GIVEN LENGTHS */
      public static boolean lengthCheck(ArrayList<Integer>[] list, int
  source, int dest, int vertices, int reqLength) {

          //ARRAY OF VISITED NODES
          boolean[] hasVisited = new boolean[vertices];
          ArrayList<Integer> path = new ArrayList<>();

          //ADD THE SOURCE NODE AND SUBTRACT 1
          path.add(source);
          ArrayList<Integer> pathLen = new ArrayList<>();

          //CALL DFS FUNCTION RECURSIVELY
          DFS(list, source, dest, pathLen, hasVisited, path);

          //RETURN IF PATH LENGTH EXIST OR NOT
          return pathLen.contains(reqLength);
      }

      /* FUNCTION THAT RECURSIVELY CHECKS PATH */
      private static void DFS(ArrayList<Integer>[] list, Integer
  source, Integer dest, List<Integer> lengths, boolean[] hasVisited,
  List<Integer> pathList) {

          if (source.equals(dest)) {

              //ADD PATH LENGTH TO LIST
              lengths.add(pathList.size()-1);
```

```
            //IF NODE FOUND RETURN
            return;
        }

        //MARK CURR NODE AS VISITED
        hasVisited[source] = true;

        //VISIT ALL ADJ NODES FOR ALL VERTICES
        for (Integer i : list[source]) {
            if (!hasVisited[i]) {
                pathList.add(i);
                DFS(list, i, dest, lengths, hasVisited, pathList);
                pathList.remove(i);
            }
        }

        //MARK THE CURR NODE
        hasVisited[dest] = false;
    }

    /* FUNCTION TO TRANSFORM MATRIX TO LIST */
    public static ArrayList<Integer>[] matrix_to_list(int[][]
matrix){

        /* STORING NUMBER OF VERTICES */
        int v = matrix[0].length;

        ArrayList<Integer>[] list = new ArrayList[v];

        /* CREATING A NEW LIST FOR EACH VERTEX */
        for(int i=0;i<v;i++){
            list[i]=(new ArrayList<Integer>());
        }

        /* STORING THE VERTICES IN LIST */
        for (int i = 0; i < matrix[0].length; i++) {
            for (int j = 0; j < matrix.length; j++) {
                if (matrix[i][j] >= 1) {
                    list[i].add(j);
                }
            }
        }

        return list;
    }

    /* MAIN FUNCTION */
    public static void main(String[] args)throws Exception {

        //SETTING SYSTEM PROP FOR INTEGRATING GRAPHSTREAM WITH SWING
        System.setProperty("org.graphstream.ui", "swing");

        try{
            //CREATING SERVER SOCKET BINDED TO A PORT
            ServerSocket serverSocket = new ServerSocket(5678);
            System.out.println();
            System.out.println("Welcome, Server Started!");

            while(true){
                Socket socket = serverSocket.accept();
```

```
                //DATA INPUT STREAM TO TAKE INPUT FROM CLIENT
                DataInputStream input = new
DataInputStream(socket.getInputStream());

                //OUTPUT STREAM TO STORE THE OUTPUT AND SEND TO
CLIENT
                output = new
DataOutputStream(socket.getOutputStream());

                //STORING THE INPUT MATRIX
                nodes = input.readInt();
                for (int i = 0; i < nodes; i++)
                    for (int j = 0; j < nodes; j++)
                        gAdjMatrix[i][j] = input.readInt();

                //READING DATA FROM CLIENT
                int length = input.readInt();
                int source = input.readInt();
                int dest = input.readInt();

                System.out.println();

                System.out.println("Received Number of Nodes: " +
nodes + "\n");

                System.out.println("Received Adjacency Matrix:");
                for (int i = 0; i < nodes; i++) {
                    for (int j = 0; j < nodes; j++) {
                        System.out.print(gAdjMatrix[i][j] + " ");
                    }
                    System.out.println();
                }

                System.out.println();

                System.out.println("Received Path Length: " + length
+ "\n");

                System.out.println("Received Source Node: " +
(char)((int)source + (int)'A') + "\n");

                System.out.println("Received Destination Node: " +
(char)((int)dest + (int)'A') + "\n");


                //CONVERTING MATRIX TO LIST
                ArrayList<Integer>[] adjList;
                adjList = matrix_to_list(gAdjMatrix);

                //INITIALIZING A GRAPH (GRAPHSTREAM)
                MultiGraph graph = new MultiGraph("USE");

                //SETTING UP ATTRIBUTES FOR GRAPH
                graph.setAttribute("ui.quality");
                graph.setAttribute("ui.antialias");

                //CREATING GRAPH NODES
                for(int i=0;i<nodes;i++){
                    graph.addNode(String.valueOf(i+1));
                }
```

```
                    //DESIGNING THE NODES
                    for(int i=0;i<nodes;i++){
                        Node e1=graph.getNode(String.valueOf(i+1));
                        e1.setAttribute("ui.style", "shape:circle;fill-
color: yellow;size: 40px;");
                        e1.setAttribute("ui.label",
String.valueOf((char)(i+ (int)'A')));
                    }

                    //CONSTRUCTING THE EDGES WRT THE MATRIX RECEIVED
                    for(int i=0;i<nodes;i++){
                        for(int j=0;j<nodes;j++) {
                            if(gAdjMatrix[i][j]==1) {
                                String init = String.valueOf(i + 1);
                                String end = String.valueOf(j + 1);
                                String id = init+end;
                                graph.addEdge(id, init, end, true);
                            }
                        }
                    }

                    //TAKING SCREENSHOT OF THE CREATED GRAPH
                    FileSinkImages img = FileSinkImages.createDefault();

                    //SETTING UP ALL ATTRIBUTES OF IMAGE
                    img.setOutputType(OutputType.JPG);
                    img.setResolution(400,400);

img.setLayoutPolicy(LayoutPolicy.COMPUTED_FULLY_AT_NEW_IMAGE);

                    //STORING THE IMAGE
                    img.writeAll(graph,"graph.jpg");

                    //READING THE STORED IMAGE
                    BufferedImage image = ImageIO.read(new
File("D:\\Sockets_Assignment\\graph.jpg"));

                    //CREATING A BYTE ARRAY FOR OUTPUT STREAM
                    ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();

                    //WRITING THE IMAGE
                    ImageIO.write(image, "jpg", byteArrayOutputStream);

                    //CHECK IF THE PATH EXISTS
                    boolean flag = lengthCheck(adjList, source, dest,
nodes, length);

                    //SENDING THE RESPONSE TO CLIENT AFTER CHECKING
                    char res;
                    if(flag)
                        res = 'Y';
                    else
                        res = 'N';

                    //SENDING THE RESPONSE
                    output.writeChar(res);

                    //BYTE ARRAY TO STORE THE IMAGE AS BYTES
                    byte[] size =
ByteBuffer.allocate(4).putInt(byteArrayOutputStream.size()).array();
```

```
                output.write(size);

                //SENDING THE IMAGE AS BYTES
                output.write(byteArrayOutputStream.toByteArray());
                output.flush();

                System.out.println();

                System.out.println("Flushed: " +
System.currentTimeMillis());
                //Thread.sleep(120000);
                System.out.println("Closing: " +
System.currentTimeMillis());
            }
        } catch(IOException e){
            System.out.println("ERROR: " + e);
        }
    }
}
```

❖ Client **Side Code –**

```
❖ /*
   * CN-2022 GRADED LAB
   * SIDDHANT MITTAL
   * 1910110388
   */

  /*CLIENT SIDE CODE*/

  /* IMPORTS */
  import java.awt.*;
  import java.awt.image.BufferedImage;
  import java.io.*;
  import java.net.Socket;
  import java.nio.ByteBuffer;
  import java.util.Scanner;
  import javax.imageio.ImageIO;
  import javax.swing.*;

  public class Client extends JFrame{

      //GLOBAL IMAGE VARIABLE
      static Image global_img;

      public void paint(Graphics g) {
          super.paint(g);
          Image img = global_img;
          //PAINTING IMAGE FROM RECEIVED BYTES
          g.drawImage(img, 100, 100, this);
      }

      /* MAIN FUNCTION */
      public static void main(String[] args) {

          Scanner input = new Scanner(System.in);

          System.out.println();
```

```
        /* TAKING NUMBER OF NODES FROM USER */
        System.out.println("Enter the Number of Nodes (Min: 3 and
Max: 10)");
        int nodes = input.nextInt();

        System.out.println();

        // DECLARING THE MATRIX
        int[][] adjMat = new int[nodes][nodes];
        int val;

        // TAKING THE MATRIX AS INPUT FROM USER
        System.out.println("Enter the Elements of Matrix");
        for (int i = 0; i < nodes; i++)
            for (int j = 0; j < nodes; j++) {
                val = input.nextInt();
                if(val >= 1)
                    val = 1;
                else
                    val = 0;
                adjMat[i][j] = val;
            }

        System.out.println();

        // DISPLAYING THE ENTERED MATRIX WRT THE NODE
        System.out.println("Entered Matrix");
        StringBuilder s = new StringBuilder();
        for (int i = 0; i < nodes; i++) {
            s.append((char) (i + (int) 'A')).append(": ");
            for (int j : adjMat[i]) {
                s.append(j).append(" ");
            }
            s.append("\n");
        }
        System.out.print(s);

        System.out.println();

        //TAKING PATH LENGTH AS INPUT
        System.out.println("Enter the path length");
        int length = input.nextInt();

        System.out.println();

        //TAKING SOURCE NODE AS INPUT AND CONVERTING IT TO INDEX
VALUE
        System.out.println("Enter the start node");
        int source =
(int)Character.toUpperCase(input.next().charAt(0)) - (int)'A';

        System.out.println();

        //TAKING DESTINATION NODE AS INPUT AND CONVERTING IT TO INDEX
VALUE
        System.out.println("Enter the end node");
        int dest = (int)Character.toUpperCase(input.next().charAt(0))
- (int)'A';

        System.out.println();
```

```
        // ESTABLISHING TCP CONNECTION TO COMMUNICATE WITH SERVER
        try {
            //INITIALIZING A CLIENT SIDE CONNECTION
            Socket clientSocket = new Socket("localhost", 5678);

            //INPUT STREAM OBJECT TO TAKE INPUT
            DataInputStream dataInput = new
DataInputStream(clientSocket.getInputStream());

            //OUTPUT STREAM OBJECT TO GET OUTPUT
            DataOutputStream dataOutput = new
DataOutputStream(clientSocket.getOutputStream());

            //SENDING DATA TO SERVER

            //SENDING THE NUMBER OF NODES NAD ADJACENCY MATRIX
            dataOutput.writeInt(nodes);
            dataOutput.flush();

            for (int i = 0; i < nodes; i++)
                for (int j = 0; j < nodes; j++)
                    dataOutput.writeInt(adjMat[i][j]);
            dataOutput.flush();

            //PATH LENGTH SENT
            dataOutput.writeInt(length);
            dataOutput.flush();

            //SENDING SOURCE AND DESTINATION NODES
            dataOutput.writeInt(source);
            dataOutput.flush();
            dataOutput.writeInt(dest);
            dataOutput.flush();

            //READ RESPONSE FROM SEVER
            char res = dataInput.readChar();

            //CONVERTING INT TO CHARS
            char sNode = (char)((int)source + (int)'A');
            char dNode = (char)((int)dest + (int)'A');

            String message = "";

            //CHECK RESPONSE AND CREATE DISPLAY MESSAGE
            if(res == 'Y'){
                message = "Yes, there exists a path of length " +
length + " from node " + sNode + " to node " + dNode + "!";
            }else if(res == 'N'){
                message = "No, there exists no path of length " +
length + " from node " + sNode+ " to node " + dNode + "!";
            }

            System.out.println("Message Received: " + message);

            System.out.println();

            //CREATING A BYTE ARRAY
            byte[] sizeAr = new byte[4];

            //READING THE BYTES SENT FROM SERVER
            dataInput.read(sizeAr);
```

```
            //FIND THE SIZE OF BYTES ARRAY
            int size = ByteBuffer.wrap(sizeAr).asIntBuffer().get();

            //CREATING A BYTE ARRAY FOR IMAGE
            byte[] imageArray = new byte[size];
            dataInput.read(imageArray);

            //CREATING THE IMAGE FROM RECEIVED BYTES
            BufferedImage image = ImageIO.read(new
ByteArrayInputStream(imageArray));

            //STORING THE IMAGE IN GLOBAL VARIABLE TO SEND IT TO
CONSTRUCTOR (PAINT) FOR PAINTING THE IMAGE
            global_img = image;

            //DISPLAYING THE IMAGE RECEIVED USING JFrame
            JFrame frame = new Client();
            frame.setTitle("Graph Visualization");
            frame.setSize(600, 600);
            frame.setVisible(true);

            System.out.println("Getting the graph image....");
            System.out.println();
            System.out.println("Received Bytes (" + image.getHeight()
+ "x" + image.getWidth() + ") : " + System.currentTimeMillis());

            //CLOSING THE CONNECTION
            dataOutput.close();
            clientSocket.close();

        } catch (IOException ex){
            System.out.println("ERROR: " + ex);
        }
    }
}
```
