Computer Networks (CSD304) – Socket Programming Assignment

Due Date: November 25, 2020 (10 pm)

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: Java**
- ✓ 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.
- ✓ Materials copied from the Internet or otherwise will attract penalty.

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

Submission

Each student must upload the following files on Blackboard:

a) Client.java file - The java file must contain your name and roll no (as comments).

b) Server.java file - The java file must contain your name and roll no (as comments).

c) Paste your code and screenshots of input and output screens (paste them in this file) - Name the document as Socket_CN2020_FirstName_LastName.pdf. [**You are required to strictly follow the naming convention**.]

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 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:

(a) positive Y response (or negative N response) if there exists (or doesn't exist) a path of length n from B to C.

(b) the image of the directed graph with nodes A B C D E proving the validity of the response.

For simplicity, assume a 5-node graph with nodes named A, B, C, D, E.

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*

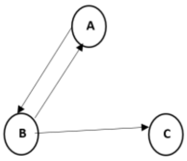
, 2, A, C

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*

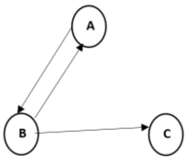
, 2, C, A

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

Server should return the following:

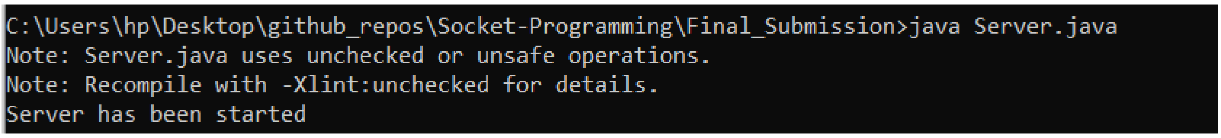
Output 1: No, there is no path of length 2 from node C to node A.

Output 2: Graph:



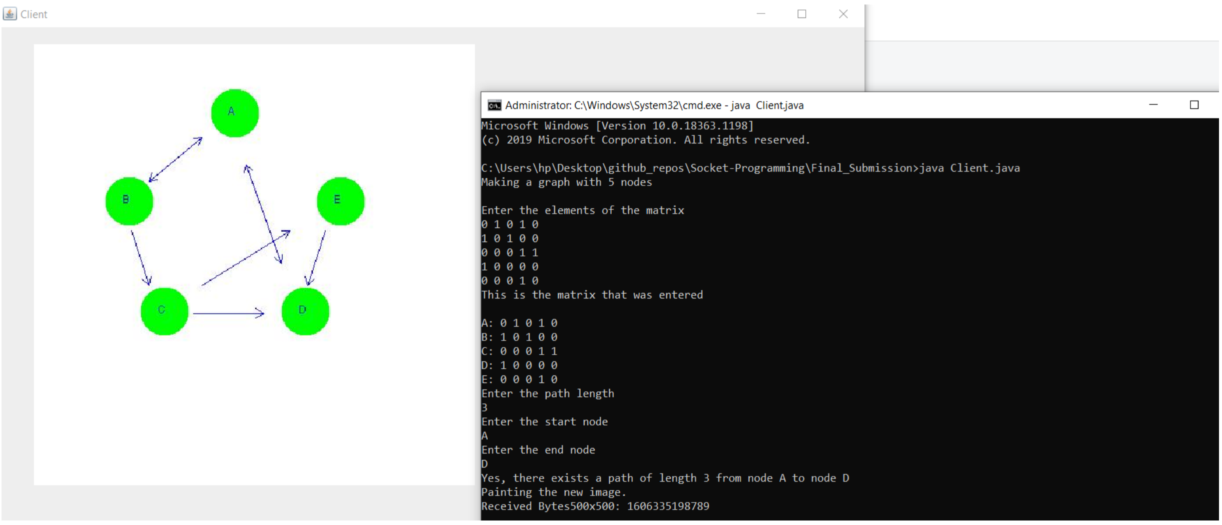
<u>Submission Template</u>

\\Screenshots of Input and Output Screens



```
C:\Users\hp\Desktop\github_repos\Socket-Programming\Final_Submission>java Server.java
Note: Server.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Server has been started
```

Computer Networks (CSD304) – Socket Programming Assignment


Client
A
B
E
C
D
Administrator: C:\Windows\System32\cmd.exe - java Client.java

```
Microsoft Windows [Version 10.0.18363.1198]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\hp\Desktop\github_repos\Socket-Programming\Final_Submission>java Client.java
Making a graph with 5 nodes

Enter the elements of the matrix
0 1 0 1 0
1 0 1 0 0
0 0 0 1 1
1 0 0 0 0
0 0 0 1 0
This is the matrix that was entered

A: 0 1 0 1 0
B: 1 0 1 0 0
C: 0 0 0 1 1
D: 1 0 0 0 0
E: 0 0 0 1 0
Enter the path length
3
Enter the start node
A
Enter the end node
D
Yes, there exists a path of length 3 from node A to node D
Painting the new image.
Received Bytes500x500: 1606335198789
```

\\ Link to the Github repository

Here, all my work can be tracked right from day 1 and how it kept on changing and updating over the period of time. This is to maintain authenticity of my work. Link: https://github.com/prakharrathi25/Socket-Programming

The repository is currently private and will be made public after the deadline to avoid anyone else from accessing it. Additionally, I can provide access to the repository by adding as a collaborator, in case the access is needed earlier.

\\ Walkthrough Video

I have also created a walkthrough video to show my work in action and how to run the program effectively. This can be used for testing the code. Link: https://drive.google.com/file/d/16W3K3VP6XFmY8qrgebQGMOHyWbqdAof-/view?usp=sharing

\\Server side code – put the code here

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package com.mycompany.serverside;

/* Import java Packages */
import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.nio.ByteBuffer;
import java.util.*;
import java.util.List;
import javax.swing.*;
import java.awt.*;
import java.awt.image.BufferedImage;

/**
 * Name: Prakhar Rathi
 * Roll Number: 1810110169
 */

public class Server extends JFrame{

    // Global adjmatrix
    static int[][] globalAdjMatrix = new int[5][5];

    // Declare global image
    static Image global_img;

    // Global socket variable
//    static ServerSocket serverSocket;
//    static Socket socket;
//    static DataInputStream input;
    static DataOutputStream output;

    /* function to convert adjacency matrix to adjacency list */
    public static ArrayList<Integer>[] mat_to_list(int[][] m){

        // Collect number of vertices
        int vertices = m[0].length;

        // Declare an Array
        ArrayList<Integer>[] adjList = new ArrayList[vertices];

        // Create a new list for each vertex to store the vertices
        for (int i = 0; i < vertices; i++) {
            adjList[i] = (new ArrayList<Integer>());
        }

        // Store the vertices in the adjacency list
        for (int i = 0; i < m[0].length; i++) {
            for (int j = 0; j < m.length; j++) {
                if (m[i][j] >= 1) {
                    adjList[i].add(j);
                }
            }
        }

        return adjList;

    }

    // Set the loaction of the points in the graph
    Point A = new Point(220, 80);
    Point B = new Point(100, 180);
    Point C = new Point(140, 305);
    Point E = new Point(340, 180);
    Point D = new Point(300, 305);

    int drawnEdges1[] = new int[25];
    int drawnEdges2[] = new int[25];
    // Overriding the paint function and including exception handling in
the same
    public void paint(Graphics g) {
        global_img = createGraphImage();
    }

    public static BufferedImage toBufferedImage(Image img)
    {
        if (img instanceof BufferedImage)
        {
            return (BufferedImage) img;
        }

        // Create a buffered image with transparency
        BufferedImage bimage = new BufferedImage(img.getWidth(null),
img.getHeight(null), BufferedImage.TYPE_INT_ARGB);

        // Draw the image on to the buffered image
        Graphics2D bGr = bimage.createGraphics();
        bGr.drawImage(img, 0, 0, null);
        bGr.dispose();

        // Return the buffered image
        return bimage;
    }

    // Function to draw the tip of the edge in the graph
    private void drawEdgeTip(Graphics g, int x1, int y1, int x2, int y2) {
        int x,y;
        double rads = 0.5236;
        double hyp_multiplier = 10;
        int diff_y = y2 - y1;
        int diff_x = x2 - x1;
        double t = Math.atan2(diff_y, diff_x);
        double r = rads + t;
        for (int j = 0; j < 2; j++) {
            x = (int)(x2 - hyp_multiplier * Math.cos(r));
            y = (int)(y2 - hyp_multiplier * Math.sin(r));
            g.drawLine(x2, y2, x, y);
            r = t - rads;
        }
    }

    public void drawEdge(Graphics g, int x1, int y1, int x2, int y2) {
        int midx = (x1 + x2)/2;
        int midy = (y1 + y2)/2;
        x1 = (midx + x1)/2;
        x2 = (midx + x2)/2;
        y1 = (midy + y1)/2;
        y2 = (midy + y2)/2;

        // Add the connecting line
        g.drawLine(x1, y1, x2, y2);

        // Add the tip of the edge to the connecting graph
        drawEdgeTip(g, x1, y1,x2, y2);

    }

    // Function to visualise the graph nodes and the edges
    private Image createGraphImage() {

        // Create a buffered image object
        BufferedImage image = new BufferedImage(500, 500,
BufferedImage.TYPE_INT_RGB);

        // Instantiate graphics object
        Graphics g = image.getGraphics();

        /* Design the ovals which are the nodes */
        g.fillRect(0, 0, image.getWidth(), image.getHeight());
        g.setColor(Color.green); // node colors
        g.fillOval(200, 50, 55, 55);
        g.fillOval(80, 150, 55, 55);
        g.fillOval(320, 150, 55, 55);
        g.fillOval(120, 275, 55, 55);
        g.fillOval(280, 275, 55, 55);

        g.setColor(Color.blue); // text and edge color
        /* Add the node names */
        g.drawString("A", A.x, A.y);
        g.drawString("B", B.x, B.y);
        g.drawString("C", C.x, C.y);
        g.drawString("D", D.x, D.y);
        g.drawString("E", E.x, E.y);

        for (int i = 0; i < 25; i++) {
            drawnEdges1[i] = -1;
            drawnEdges2[i] = -1;
        }
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
                if (globalAdjMatrix[i][j] > 0) {
                    int n = 0;
                    int m = 0;
                    if (drawnEdges1[0] == -1)
                        drawnEdges1[0] = i;
                    else {
                        while (drawnEdges1[n] != -1)
                            n++;
                        drawnEdges1[n] = i;
                    }
                    if (drawnEdges2[0] == -1)
                        drawnEdges2[0] = j;
                    else {
                        while (drawnEdges2[m] != -1)
                            m++;
                        drawnEdges2[m] = j;
                    }
                }
            }
        }

        for(int i=0; i<25;i++){
            if(drawnEdges1[i]>-1 && drawnEdges2[i]>-1){
                int a1=0,b1=0,a2=0,b2=0;
                switch(drawnEdges1[i]){
                    case 0:
                        a1 = A.x;
                        b1 = A.y;
                        break;
                    case 1:
                        a1 = B.x;
                        b1 = B.y;
                        break;
                    case 2:
                        a1 = C.x;
                        b1 = C.y;
                        break;
                    case 3:
                        a1 = D.x;
                        b1 = D.y;
                        break;
                    case 4:
                        a1 = E.x;
                        b1 = E.y;
                        break;
                }
                switch(drawnEdges2[i]){
                    case 0:
                        a2 = A.x;
                        b2 = A.y;
                        break;
                    case 1:
                        a2 = B.x;
                        b2 = B.y;
                        break;
                    case 2:
                        a2 = C.x;
                        b2 = C.y;
                        break;
                    case 3:
                        a2 = D.x;
                        b2 = D.y;
                        break;
                    case 4:
                        a2 = E.x;
                        b2 = E.y;
                        break;
                }
                drawEdge(g, a1, b1, a2, b2);
//                  System.out.println("Drew an edge");
            }
//              System.out.println("Outside Loop");
        }

        // Return the image
        return image;
    }

    /* Function to check if the required path length in the given path
lengths */
    public static boolean checkPathLength(ArrayList<Integer>[] list, int
source, int dest, int vertices, int reqLength) {
        // Create an array of visited nodes
        boolean[] hasVisited = new boolean[vertices];
        ArrayList<Integer> pathList = new ArrayList<>();

        // add the source node to the path (subtract 1 from path length)
        pathList.add(source);
        ArrayList<Integer> pathLength = new ArrayList<>();
        // Call recursive DFS function
        pathLengthDFS(list, source, dest, pathLength, hasVisited, 
pathList);

        // Return whether the path length exists or not
        return pathLength.contains(reqLength);
    }

    /* Function to recursively check the path and then add to the list of 
path lengths */
    private static void pathLengthDFS(ArrayList<Integer>[] adjList, Integer 
s, Integer d, List<Integer> lengths, boolean[] hasVisited, List<Integer> 
funcPathList) {

        if (s.equals(d)) {

            // Add the path length to the path lengths list (subtract 1 to 
remove source node)
            lengths.add(funcPathList.size()-1);

            // If we have found a matching node then we can directly return 
after adding the length to tha path lenght
            return;
        }

        // Mark the current node as visited
        hasVisited[s] = true;

        // Recur to all the adjacent nodes for all the vertices
        for (Integer i : adjList[s]) {
            if (!hasVisited[i]) {
                // store current node in the path to begin traversal
                funcPathList.add(i);
                pathLengthDFS(adjList, i, d, lengths, hasVisited, 
funcPathList);

                // remove current node from the path
                funcPathList.remove(i);
            }
        }

        // Mark the current node
        hasVisited[d] = false;
    }


    public static void main(String args[])throws Exception{
        try{

            // create a server socket and bind it to the port number
            ServerSocket serverSocket = new ServerSocket(9000);
            System.out.println("Server has been started");

            while(true){

                // Create a new socket to establish a virtual pipe
                // with the client side (LISTEN)
                Socket socket = serverSocket.accept();

                // Create a datainput stream object to communicate with the 
client (Connect)
```

```
                DataInputStream input = new 
DataInputStream(socket.getInputStream());

                // Create a dataoutput stream object to communicate with 
the client (Connect)
                output = new DataOutputStream(socket.getOutputStream());

                // Read the data from the client (Receieve)
                int pathLength = input.readInt();
                int start = input.readInt();
                int end = input.readInt();

                // Collect the nodes and the matrix through the data
                int nodes = input.readInt();
                for (int i = 0; i < nodes; i++)
                    for (int j = 0; j < nodes; j++)
                        globalAdjMatrix[i][j] = input.readInt();

                // Convert adjacency matrix to adjacency list
                ArrayList<Integer>[] adjList = new ArrayList[nodes];
                adjList = mat_to_list(globalAdjMatrix);

                // Check whether path length is present in the array
                boolean pathExists = checkPathLength(adjList, start, end, 
nodes, pathLength);

                // Send the Y or N to the client
                char response;
                if(pathExists)
                    response = 'Y';
                else
                    response = 'N';

                // Send the response
                output.writeChar(response);

                /* Graph Visualisation */
                JFrame frame = new Server();
                frame.setSize(600, 600);
                frame.setVisible(true);
                frame.setTitle("Graph Visualisation from Server");
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                while(global_img == null){
                    System.out.println("Stay Idle for Transmission");
                }

                //flushing the img here
                ByteArrayOutputStream arrayOutputStream = new 
ByteArrayOutputStream();
                BufferedImage temp = toBufferedImage(global_img);
                ImageIO.write(temp, "jpg", arrayOutputStream);

                byte[] size = 
ByteBuffer.allocate(4).putInt(arrayOutputStream.size()).array();
                output.write(size);
                output.write(arrayOutputStream.toByteArray());
                output.flush();
                System.out.println("Flushed Bytes: " + 
System.currentTimeMillis());
                global_img = null;
                System.out.println("Closing: " +
System.currentTimeMillis());

            }
        } catch(IOException e){}
    }
}
```

\\Client Side Code – put the code here

```
package com.mycompany.clientside;

import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
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.*;

/**
 * Name: Prakhar Rathi
 * Roll Number: 1810110169
 */

public class Client extends JFrame {

    // Global Image
    static Image global_img;

    public void paint(Graphics g) {

        System.out.println("Painting the new image.");

        super.paint(g);
        Image img = global_img;

        // Draw the image from the bytes
        g.drawImage(img, 50, 50, this);

    }
    /* Main Function */
    public static void main(String[] args) {

        // Collect inputs from the user
        Scanner input = new Scanner(System.in);

        /* Read the matrix */
        int nodes = 5;
        System.out.println("Making a graph with 5 nodes\n");

        // Declare the matrix
        int adjMatrix[][] = new int[nodes][nodes];
        int entry;
        // Read the matrix values
        System.out.println("Enter the elements of the matrix");
        for (int i = 0; i < nodes; i++)
            for (int j = 0; j < nodes; j++) {
                entry = input.nextInt();
                if(entry >= 1)
                    entry = 1;
                else
                    entry = 0;
                adjMatrix[i][j] = entry;
            }

        // Display the entered matrix
        System.out.println("This is the matrix that was entered\n");
        StringBuilder s = new StringBuilder();
        for (int i = 0; i < nodes; i++) {
            s.append((char)(i+ (int)'A') + ": ");
            for (int j : adjMatrix[i]) {
                s.append((j) + " ");
            }
            s.append("\n");
        }
        System.out.print(s.toString());

        // Input path length
        System.out.println("Enter the path length");
        int pathLength = input.nextInt();

        // Input starting and ending nodes (convert alphabets to index
values)
        System.out.println("Enter the start node");
        int start = (int)Character.toUpperCase(input.next().charAt(0)) -
(int)'A';

        System.out.println("Enter the end node");
        int end = (int)Character.toUpperCase(input.next().charAt(0)) -
(int)'A';

        // TCP Connection and communication with the server
        try {
            // Make a new client side connection
            Socket clientSocket = new Socket("localhost", 9000);

            // Make a new inputstream object
            DataInputStream dataInput = new
DataInputStream(clientSocket.getInputStream());

            // Create an output stream
            DataOutputStream dataOutput = new
DataOutputStream(clientSocket.getOutputStream());

            // Send data to the server

            // Send Path length
            dataOutput.writeInt(pathLength);
            dataOutput.flush();

            // Send start and end
            dataOutput.writeInt(start);
            dataOutput.flush();
            dataOutput.writeInt(end);
            dataOutput.flush();

            // Send the number of nodes and the matrix
            dataOutput.writeInt(nodes);
            dataOutput.flush();
            for (int i = 0; i < nodes; i++)
                for (int j = 0; j < nodes; j++)
                    dataOutput.writeInt(adjMatrix[i][j]);
                    dataOutput.flush();

            // Read the response input from the server
            char response = dataInput.readChar();

            // Convert start and end node to alphabets
            char startNode = (char)((int)start + (int)'A');
            char endNode = (char)((int)end + (int)'A');
            String statement = "";

            // Check response from the server
            if(response == 'Y'){
                statement = "Yes, there exists a path of length " +
pathLength + " from node " + startNode + " to node " + endNode;
            }else if(response == 'N'){
                statement = "No, there exists no path of length " +
pathLength + " from node " + startNode+ " to node " + endNode;
            }

            // Print the statement
            System.out.println(statement);

            /* Load the image */

            // getting img from server
            byte[] sizeAr = new byte[4];
            dataInput.read(sizeAr);

            int size = ByteBuffer.wrap(sizeAr).asIntBuffer().get();
            byte[] imageArray = new byte[size];
            dataInput.read(imageArray);
            BufferedImage image = ImageIO.read(new
ByteArrayInputStream(imageArray));
            global_img = image;

            JFrame frame = new Client();
            frame.setTitle("Client");
            frame.setSize(1000, 1000);
            frame.setVisible(true);

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

            // Call the constructor to load image
            dataOutput.close();
            clientSocket.close(); // close the connection

        } catch (IOException ex){}

    }
}
```

Computer Networks (CSD304) – Socket Programming Assignment