COMPUTER NETWORKS PRACTICAL

1. Write a program to implement a HTTP client-server communication using TCP sockets. The client sends a string to the server. The server reverses the string and sends it back.

ALGORITHM:

1. Server listens on a TCP port and accepts incoming client connections.
2. Client connects to the server and sends a string message.
3. Server receives the message, reverses the string.
4. Server sends the reversed string back to the client.
5. Both client and server close their respective sockets and streams.

CODE:

SERVER:

```
import java.io.*;
import java.net.*;

public class ReverseServer {
  public static void main(String[] args) {
    try (ServerSocket serverSocket = new ServerSocket(9090)) {
      System.out.println("Server is running and waiting for a client...");

      Socket socket = serverSocket.accept();
      System.out.println("Client connected.");

      BufferedReader in = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));
      PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

      String received = in.readLine();
      String reversed = new StringBuilder(received).reverse().toString();

      out.println(reversed);

      in.close();
      out.close();
      socket.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}
```

CLIENT:

```
import java.io.*;
import java.net.*;

public class ReverseClient {
    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", 9090)) {
            BufferedReader userInput = new BufferedReader(
                new InputStreamReader(System.in));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));

            System.out.print("Enter a string to reverse: ");
            String input = userInput.readLine();

            out.println(input);
            String reversed = in.readLine();

            System.out.println("Reversed string from server: " + reversed);

            in.close();
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2. Develop a UDP-based chat system where two users can send and receive messages simultaneously.

ALGORITHM:

1. Server creates a DatagramSocket and waits to receive messages.
2. Client reads input and sends messages using DatagramSocket to the server's IP and port.
3. Both server and client have two threads: one for sending, one for receiving.
4. Messages are exchanged using DatagramPacket.
5. Both programs loop continuously to allow real-time chat.

CODE:

SERVER:

```
import java.net.*;
import java.io.*;
```

```
public class UDPServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket(5000); // Server port
        InetAddress clientAddress = InetAddress.getByName("localhost");
        int clientPort = 6000; // Expected client port

        // Thread to send messages to client
        new Thread(() -> {
            try {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                while (true) {
                    String msg = br.readLine();
                    byte[] buffer = msg.getBytes();
                    DatagramPacket packet = new DatagramPacket(buffer, buffer.length, clientAddress, clientPort);
                    socket.send(packet);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }).start();

        // Thread to receive messages from client
        new Thread(() -> {
            try {
                byte[] buffer = new byte[1024];
                while (true) {
                    DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
                    socket.receive(packet);
                    String msg = new String(packet.getData(), 0, packet.getLength());
                    System.out.println("Client: " + msg);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }).start();
    }
}
```

CLIENT:

```
import java.net.*;
import java.io.*;

public class UDPClient {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket(6000); // Client port
        InetAddress serverAddress = InetAddress.getByName("localhost");
        int serverPort = 5000; // Server port

        // Thread to send messages to server
        new Thread(() -> {
            try {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                while (true) {
                    String msg = br.readLine();
                    byte[] buffer = msg.getBytes();
                    DatagramPacket packet = new DatagramPacket(buffer, buffer.length,
serverAddress, serverPort);
                    socket.send(packet);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }).start();

        // Thread to receive messages from server
        new Thread(() -> {
            try {
                byte[] buffer = new byte[1024];
                while (true) {
                    DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
                    socket.receive(packet);
                    String msg = new String(packet.getData(), 0, packet.getLength());
                    System.out.println("Server: " + msg);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }).start();
    }
}
```

3. Implement a file transfer protocol using TCP where the client requests a file and the server sends it if available.

ALGORITHM:

Server

1. Create a ServerSocket and wait for client connections.
2. Accept the client socket and read the requested filename.
3. Check if the file exists; if yes, send file size and content.
4. If not found, send a "file not found" message.
5. Close connections after transfer is complete.

Client

1. Connect to server using a Socket.

2. Send the filename to the server.
3. If file is found, receive size and content, save locally.
4. If not found, display error message.
5. Close the connection.

CODE:

SERVER:

```
import java.io.*; import java.net.*;

public class TCPFileServer {
    public static void main(String[] args) throws Exception {
        ServerSocket ss = new ServerSocket(5000);
        Socket s = ss.accept();
        DataInputStream dis = new DataInputStream(s.getInputStream());
        DataOutputStream dos = new DataOutputStream(s.getOutputStream());
        String fileName = dis.readUTF();
        File file = new File(fileName);
        if (file.exists()) {
            dos.writeUTF("FOUND");
            dos.writeLong(file.length());
            FileInputStream fis = new FileInputStream(file);
            byte[] buffer = new byte[4096]; int read;
            while ((read = fis.read(buffer)) > 0) dos.write(buffer, 0, read);
            fis.close();
        } else dos.writeUTF("NOT_FOUND");
        s.close(); ss.close();
    }
}
```

CLIENT:

```
import java.io.*; import java.net.*;

public class TCPFileClient {
    public static void main(String[] args) throws Exception {
        Socket s = new Socket("localhost", 5000);
        DataOutputStream dos = new DataOutputStream(s.getOutputStream());
        DataInputStream dis = new DataInputStream(s.getInputStream());
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter file name to request: ");
        String fileName = br.readLine();
        dos.writeUTF(fileName);
        String response = dis.readUTF();
        if (response.equals("FOUND")) {
            long size = dis.readLong();
            FileOutputStream fos = new FileOutputStream("received_" + fileName);
            byte[] buffer = new byte[4096]; int read;
            while (size > 0 && (read = dis.read(buffer, 0, (int)Math.min(buffer.length, size))) > 0)
{
                fos.write(buffer, 0, read); size -= read;
            }
            fos.close(); System.out.println("File received.");
        } else System.out.println("File not found on server.");
        s.close();
    }
}
```

4. Simulate DNS resolution using UDP sockets.

ALGORITHM:

DNS Server

1. Create a DatagramSocket and wait for incoming requests.
2. On receiving a hostname, look it up in a predefined mapping.
3. If found, send back the corresponding IP address.
4. If not found, send an error message.
5. Repeat indefinitely for multiple requests.

DNS Client

1. Create a DatagramSocket and send a hostname to the server.
2. Wait for a response packet from the server.
3. Display the received IP address or error message.
4. Allow user to input hostnames.
5. Repeat or exit on user command.

CODE:

SERVER:

```
import java.net.*;
import java.util.*;

public class UDP_DNSServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket(5000);
        byte[] buffer = new byte[1024];
        Map<String, String> dnsTable = Map.of(
            "google.com", "142.250.190.14",
            "yahoo.com", "98.137.11.163",
            "openai.com", "104.18.12.123"
        );

        while (true) {
            DatagramPacket request = new DatagramPacket(buffer, buffer.length);
            socket.receive(request);
            String hostname = new String(request.getData(), 0, request.getLength());
            String ip = dnsTable.getOrDefault(hostname, "Host not found");
            byte[] responseData = ip.getBytes();
            DatagramPacket response = new DatagramPacket(responseData, responseData.length,
                    request.getAddress(), request.getPort());
```

```
          socket.send(response);
        }
    }
}
```

CLIENT:

```
import java.net.*;
import java.io.*;

public class UDP_DNSClient {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket();
        InetAddress serverAddr = InetAddress.getByName("localhost");
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        while (true) {
            System.out.print("Enter hostname (or 'exit'): ");
            String hostname = br.readLine();
            if (hostname.equalsIgnoreCase("exit")) break;
            byte[] sendData = hostname.getBytes();
            DatagramPacket request = new DatagramPacket(sendData, sendData.length, serverAddr, 5000);
            socket.send(request);

            byte[] buffer = new byte[1024];
            DatagramPacket response = new DatagramPacket(buffer, buffer.length);
            socket.receive(response);
            String ip = new String(response.getData(), 0, response.getLength());
            System.out.println("IP Address: " + ip);
        }

        socket.close();
    }
}
```

5. Simulate Distance Vector Routing algorithm and display routing tables

Algorithm:

1. Initialize the distance table for each router with direct link costs; unknown routes are set to infinity.
2. Each router sends its distance vector to its immediate neighbors.
3. On receiving a vector, each router updates its table using Bellman-Ford: new_cost = cost_to_neighbor + neighbor’s_cost_to_destination.
4. Repeat exchanges until no updates occur (i.e., tables converge).
5. Display the final routing tables for all routers.

CODE:

```
import java.util.*;
public class DistanceVectorRouting {
    static final int INF = 999, N = 4;
    public static void main(String[] args) {
        int[][] g = {{0, 1, 3, INF}, {1, 0, 1, 4}, {3, 1, 0, 2}, {INF, 4, 2, 0}};
        int[][][] t = new int[N][N][2];

        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++) {
                t[i][j][0] = g[i][j];
                t[i][j][1] = (g[i][j] == INF || i == j) ? -1 : j;
            }

        boolean updated;
        do {
            updated = false;
            for (int i = 0; i < N; i++)
                for (int j = 0; j < N; j++)
                    for (int k = 0; k < N; k++)
                        if (g[i][k] != INF && t[k][j][0] != INF && g[i][k] + t[k][j][0] < t[i][j][0]) {
                            t[i][j][0] = g[i][k] + t[k][j][0];
                            t[i][j][1] = k;
                            updated = true;
                        }
        } while (updated);

        for (int i = 0; i < N; i++) {
            System.out.println("Router " + i + ": Dest Cost NextHop");
            for (int j = 0; j < N; j++)
                System.out.printf("%d\t%s\t%s\n", j, t[i][j][0] == INF ? "INF" : t[i][j][0] + "", t[i][j][1] == -1 ? "-" : t[i][j][1] + "");
            System.out.println();
        }
    }
}
```

OUTPUT:

```
Router 0: Dest Cost NextHop
0    0   -
1    1   1
2    2   1
3    4   1

Router 1: Dest Cost NextHop
0    1   0
1    0   -
2    1   2
3    3   2

Router 2: Dest Cost NextHop
0    2   1
```

1	**1**	**1**
2	**0**	**-**
3	**2**	**3**

6. Simulate Link State Routing algorithm using Dijkstra’s method.

ALGORITHM:

1. Initialize a distance array dist[] for each router (node), set distances to infinity, except for the source node (set to 0).
2. Create a visited[] array to track if a node has been processed.
3. For each router, find the node with the smallest tentative distance (not visited yet), and update its neighboring nodes' distances.
4. Repeat step 3 until all nodes have been visited.
5. Print the final routing table showing the shortest path from the source node to all other nodes.

CODE:

```
import java.util.*;

public class LinkStateRouting {
    static final int INF = 999, N = 4;

    public static void main(String[] args) {
        int[][] graph = {{0, 1, 3, INF}, {1, 0, 1, 4}, {3, 1, 0, 2}, {INF, 4, 2, 0}};
        int[] dist = new int[N];
        Arrays.fill(dist, INF); dist[0] = 0; // Source node (router 0)
        boolean[] visited = new boolean[N];

        // Dijkstra's algorithm
        for (int i = 0; i < N; i++) {
            int u = -1, minDist = INF;
            for (int j = 0; j < N; j++) if (!visited[j] && dist[j] < minDist) { u = j; minDist = dist[j]; }
            visited[u] = true;
            for (int v = 0; v < N; v++) if (graph[u][v] != INF && !visited[v] && dist[u] + graph[u][v] < dist[v]) dist[v] = dist[u] + graph[u][v];
        }

        // Print routing table
        System.out.println("Routing Table from Source (Router 0):");
        for (int i = 0; i < N; i++) System.out.println("Dest " + i + ": " + (dist[i] == INF ? "INF" : dist[i]));
    }

}
```

OUTPUT:

```
Routing Table from Source (Router 0):
```

Dest 0: 0

Dest 1: 1

Dest 2: 2

Dest3:4

7. Simulate CRC Error Detection and validate the error checking capability.

ALGORITHM:

1. Append zeros (length = generator - 1) to the data.
2. Perform binary division (XOR) of the data by the generator polynomial.
3. The remainder after division is the CRC checksum.
4. Append the CRC to the original data to create the transmitted frame.
5. At receiver side, divide the received frame by the generator; if remainder is zero, no error; else, error detected.

CODE:

```
public class CRC {
   static String xor(String a, String b) {
       String res = "";
       for (int i = 1; i < b.length(); i++) res += (a.charAt(i) == b.charAt(i)) ? '0' : '1';
       return res;
   }

   static String divide(String data, String gen) {
       int pick = gen.length(); String tmp = data.substring(0, pick);
       while (pick < data.length())
           tmp = xor(tmp.charAt(0) == '1' ? gen : "0".repeat(pick), tmp) + data.charAt(pick++);
       return xor(tmp.charAt(0) == '1' ? gen : "0".repeat(pick), tmp);
   }

   public static void main(String[] args) {
       String data = "100100", gen = "1101", pad = "0".repeat(gen.length() - 1);
       String crc = divide(data + pad, gen), frame = data + crc;
       System.out.println("Frame: " + frame);
       System.out.println("Check: " + (divide(frame, gen).equals("0".repeat(gen.length() - 1)) ?
"No Error" : "Error"));
   }
}
```

OUTPUT:

```
Frame: 100100111
Check: No Error
```

8. Write a program to implement ARP using TCP.

ALGORITHM:

1. Server stores a mapping of IP addresses to MAC addresses (ARP table).
2. Client sends an IP address to the server (ARP request).
3. Server looks up the IP in its ARP table.
4. If found, server returns the corresponding MAC address (ARP reply).
5. If not found, server responds with "MAC not found".

CODE:

SERVER:

```
import java.io.*; import java.net.*; import java.util.*;
public class ARPServer {
  public static void main(String[] args) throws IOException {
    Map<String, String> arpTable = Map.of("192.168.1.1", "AA:BB:CC:DD:EE:01",
"192.168.1.2", "AA:BB:CC:DD:EE:02");
    ServerSocket ss = new ServerSocket(8888);
    Socket s = ss.accept();
    BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));
    PrintWriter out = new PrintWriter(s.getOutputStream(), true);
    String ip = in.readLine();
    out.println(arpTable.getOrDefault(ip, "MAC not found"));
    s.close(); ss.close();
  }}
```

CLIENT:

```
import java.io.*; import java.net.*;

public class ARPClient {

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

    Socket s = new Socket("localhost", 8888);

    PrintWriter out = new PrintWriter(s.getOutputStream(), true);

    BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));

    out.println("192.168.1.1"); // Change to test other IPs

    System.out.println("MAC Address: " + in.readLine());
```

```
                s.close();

            }

        }
```

9.Write a TCP socket program for a simple HTTP echo client that downloads a web page from a given host.

ALGORITHM:

1. Open a TCP socket to the target web server on port 80 (HTTP).
2. Send an HTTP GET request for the root page (e.g., GET / HTTP/1.1).
3. Include required headers like Host and Connection.
4. Read the response from the server.
5. Display the HTML content received.

CODE:

```
public class HTTPEchoClient {

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

        Socket socket = new Socket("www.google.com", 80); // Connecting to Google

        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));



        // Send HTTP GET request

        out.println("GET / HTTP/1.1");

        out.println("Host: www.google.com");

        out.println("Connection: close");

        out.println(); // End of headers

        // Read and print response

        String line;

        while ((line = in.readLine()) != null) System.out.println(line);

        socket.close();
    }}
```

OUTPUT:

```
HTTP/1.1 301 Moved Permanently

Location: http://www.google.com/

Content-Type: text/html; charset=UTF-8

Content-Length: 219

Date: Sun, 12 May 2025 10:00:00 GMT

...

<!doctype html><html><head><title>301 Moved</title></head>

<body><h1>301 Moved</h1>

The document has moved <a href="http://www.google.com/">here</a>.

</body></html>
```

10.Simulate RARP using UDP.

ALGORITHM:

1. Client sends its MAC address to the server via a UDP packet.
2. Server receives the MAC and looks it up in a predefined RARP table.
3. If a match is found, the server sends back the corresponding IP address.
4. Client receives the response and displays the IP.
5. If no match, the server responds with "IP not found".

CODE:

SERVER:

```
import java.net.*;
import java.util.*;
public class RARPServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket(9999);
        byte[] buf = new byte[1024];
        Map<String, String> table = Map.of("AA:BB:CC:DD:EE:01", "192.168.1.10");
        while (true) {
            DatagramPacket req = new DatagramPacket(buf, buf.length);
            socket.receive(req);
            String mac = new String(req.getData(), 0, req.getLength());
            String ip = table.getOrDefault(mac, "IP not found");
            byte[] reply = ip.getBytes();
            DatagramPacket res = new DatagramPacket(reply, reply.length, req.getAddress(), req.getPort());
            socket.send(res);
        }
    }
}
```

CLIENT:

```
import java.net.*;

public class RARPClient {

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

        DatagramSocket socket = new DatagramSocket();

        String mac = "AA:BB:CC:DD:EE:01";

        byte[] reqData = mac.getBytes();

        InetAddress serverAddr = InetAddress.getByName("localhost");

        DatagramPacket req = new DatagramPacket(reqData, reqData.length, serverAddr, 9999);

        socket.send(req);

        byte[] resData = new byte[1024];

        DatagramPacket res = new DatagramPacket(resData, resData.length);

        socket.receive(res);

        System.out.println("Received IP: " + new String(res.getData(), 0, res.getLength()));

        socket.close();

    }

}
```

11. Simulate Congestion Control Algorithm (e.g., TCP Tahoe or AIMD).

ALGORITHM:

1. **Start with congestion window (cwnd)** of 1 and threshold (ssthresh) set to a high value.
2. **Use exponential growth** (slow start): cwnd doubles until it hits ssthresh.
3. On **packet loss (detected via timeout)**, set ssthresh = cwnd / 2, reset cwnd = 1.
4. After threshold is reached, use **linear increase** (congestion avoidance).
5. Repeat for each transmission round to simulate congestion control behavior.

CODE:

```
public class TCPTahoeSimulation {
    public static void main(String[] args) {
        int cwnd = 1, ssthresh = 16;
        int rounds = 10;
        for (int i = 1; i <= rounds; i++) {
            System.out.println("Round " + i + ": cwnd = " + cwnd);
            if (i == 5) { // Simulate packet loss at round 5
                System.out.println("Packet loss detected! Timeout occurs.");
                ssthresh = cwnd / 2;
                cwnd = 1;
            } else if (cwnd < ssthresh) {
                cwnd *= 2; // Slow start
            } else {
                cwnd += 1; // Congestion avoidance
            }
        }
    }
}
```

OUTPUT:

```
Round 1: cwnd = 1
Round 2: cwnd = 2
Round 3: cwnd = 4
Round 4: cwnd = 8
Round 5: cwnd = 16
Packet loss detected! Timeout occurs.
Round 6: cwnd = 1
Round 7: cwnd = 2
Round 8: cwnd = 4
Round 9: cwnd = 8
Round 10: cwnd = 9
```

12.Write a program Echo Client and Server using TCP.

ALGORITHM:

1. Server starts and listens on a TCP port for client connections.
2. On connection, it reads data from the client.
3. The server sends the same data (echo) back to the client.
4. Client connects, sends a message, and reads the echoed response.
5. Both sides close the connection after communication.

CODE:

SERVER:

```
import java.net.*; import java.io.*;

public class EchoServer {
    public static void main(String[] args) throws IOException {
        ServerSocket server = new ServerSocket(6000);
        Socket client = server.accept();
        BufferedReader in = new BufferedReader(new
InputStreamReader(client.getInputStream()));
        PrintWriter out = new PrintWriter(client.getOutputStream(), true);
        String msg = in.readLine();
        out.println("Echo: " + msg);
        client.close(); server.close();
    }
}
```

CLIENT:

```
import java.net.*; import java.io.*;

public class EchoClient {
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost", 6000);
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        out.println("Hello Server!");
        System.out.println("Server says: " + in.readLine());
        socket.close();
    }
}
```

13. Simulation of error correction code.

ALGORITHM:

1. Encode 4-bit data into 7 bits by adding 3 parity bits.
2. Transmit the 7-bit code (simulate error by flipping one bit).
3. Receiver calculates syndrome from received bits.
4. Syndrome indicates error position (if any).
5. Correct the error and extract the original 4-bit data.

CODE:

```
import java.io.*;
class CRCnor {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter Generator: ");
        String gen = br.readLine();
        System.out.print("Enter Data: ");
        String data = br.readLine();
        String code = data + "0".repeat(gen.length() - 1); // Padding with zeros
        code = data + div(code, gen);
        System.out.println("Transmitted Code Word: " + code);
        System.out.print("Enter Received Code Word: ");
        String rec = br.readLine();
        System.out.println(Integer.parseInt(div(rec, gen)) == 0 ? "No errors" : "Contains
errors");
    }

    static String div(String num1, String num2) {
        String remainder = num1.substring(0, num2.length());
        while (remainder.length() < num1.length()) {
            remainder = remainder.substring(1).replaceAll("[01]", "");
            remainder += num1.charAt(remainder.length());
        }
        return remainder.substring(1);
    }
}
```

OUTPUT:

```
Enter Generator: 1101
Enter Data: 101100
Transmitted Code Word: 101100110
Enter Received Code Word: 101100110
No errors
```

14.Simulate Link State Routing algorithm using open shortest path algorithm.

ALGORITHM:

1. Each router sends a link state advertisement (LSA) containing its known neighbors and link costs.
2. Each router receives LSAs from all other routers to build the network topology.
3. Run Dijkstra's algorithm for each router to compute the shortest paths to all other routers.
4. Update routing tables based on the computed shortest paths.
5. Repeat the process to handle changes in the network, such as link failures or new connections.

CODE:

```
import java.util.*;
public class DijkstraLSR {
    public static void main(String[] args) {
        int[][] g = {
            {0, 1, 4, 0},
            {1, 0, 2, 6},
            {4, 2, 0, 3},
            {0, 6, 3, 0}
        };

        for (int src = 0; src < 4; src++) {
            int[] dist = new int[4]; boolean[] vis = new boolean[4];
            Arrays.fill(dist, 999); dist[src] = 0;

            for (int i = 0; i < 4; i++) {
                int u = -1, min = 999;
                for (int j = 0; j < 4; j++)
                    if (!vis[j] && dist[j] < min) { min = dist[j]; u = j; }
                vis[u] = true;

                for (int v = 0; v < 4; v++)
                    if (g[u][v] > 0 && dist[u] + g[u][v] < dist[v])
                        dist[v] = dist[u] + g[u][v];
            }
            System.out.println("Router " + src + ": " + Arrays.toString(dist));
        }
    }
}
```

OUTPUT:

```
Router 0: [0, 1, 3, 6]
Router 1: [1, 0, 2, 5]
Router 2: [3, 2, 0, 3]
Router 3: [6, 5, 3, 0]
```

15.Write a program to implement a echo client-server communication using TCP sockets.

ALGORITHM:

SERVER:

1. Create a server socket on a port.
2. Wait and accept client connection.
3. Read message from the client.
4. Echo the same message back.
5. Close the connection.

CLIENT:

1. Connect to server via socket.
2. Send a message.
3. Read echo from server.
4. Display the response.
5. Close the connection.

CODE:

SERVER:

```
import java.io.*; import java.net.*;

public class EchoServer {

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

      ServerSocket ss = new ServerSocket(5000);

      Socket s = ss.accept();

      BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));

      PrintWriter out = new PrintWriter(s.getOutputStream(), true);

      out.println("Echo: " + in.readLine());

      s.close(); ss.close();

   }

}
```

CLIENT:

```
import java.io.*; import java.net.*;

public class EchoClient {

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

      Socket s = new Socket("localhost", 5000);

      PrintWriter out = new PrintWriter(s.getOutputStream(), true);

      BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));

      out.println("Hello, Server!");

      System.out.println("From Server: " + in.readLine());

      s.close();

   }

}
```