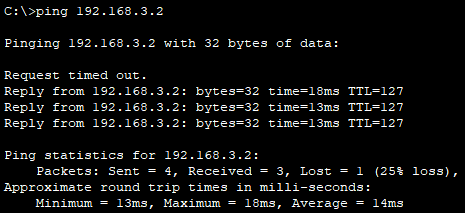
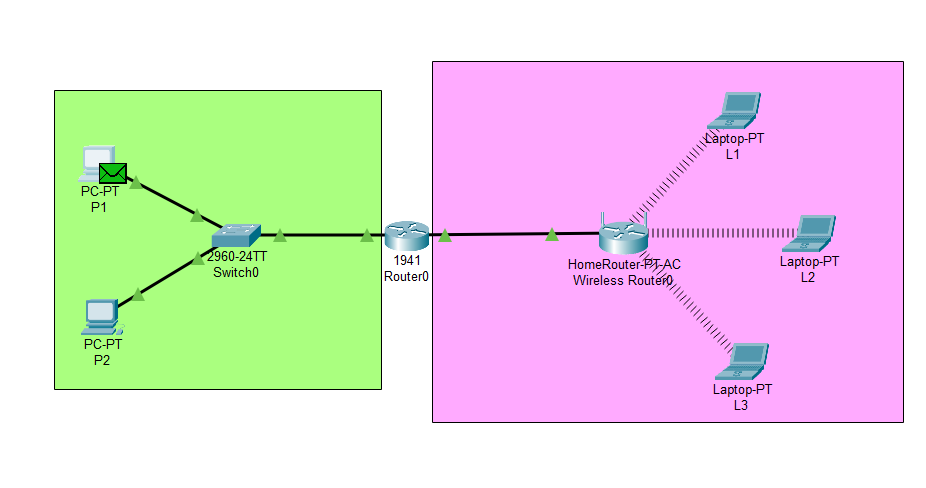
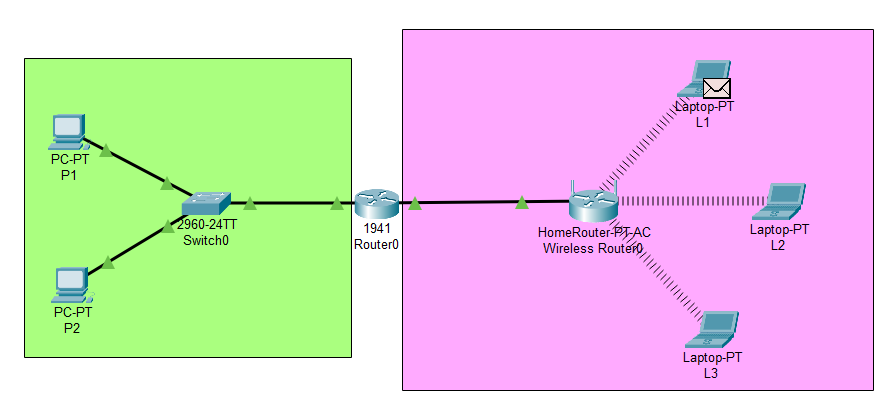
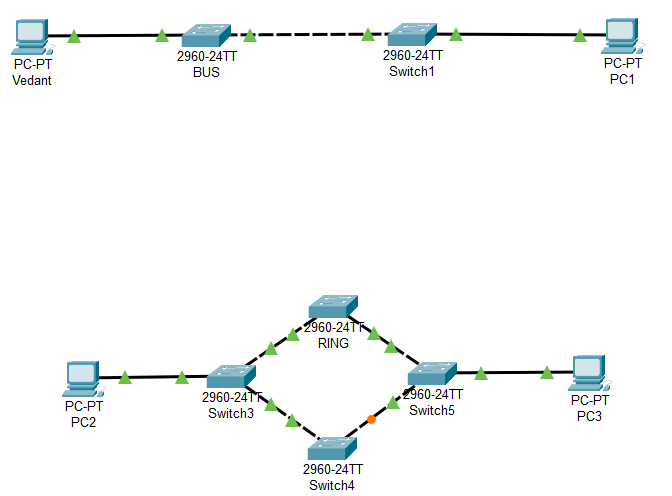
Assignment 1

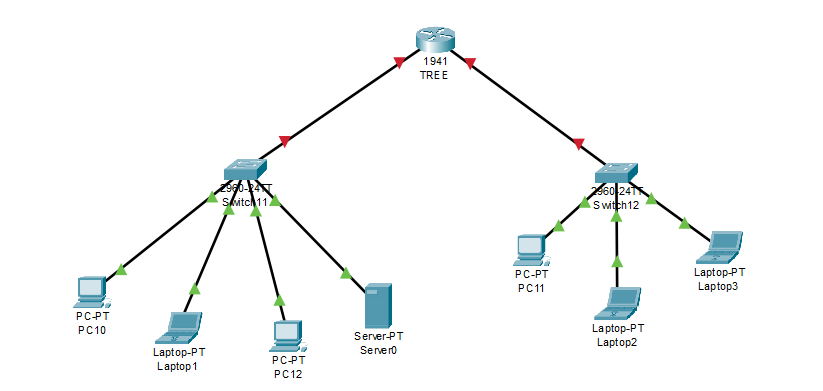


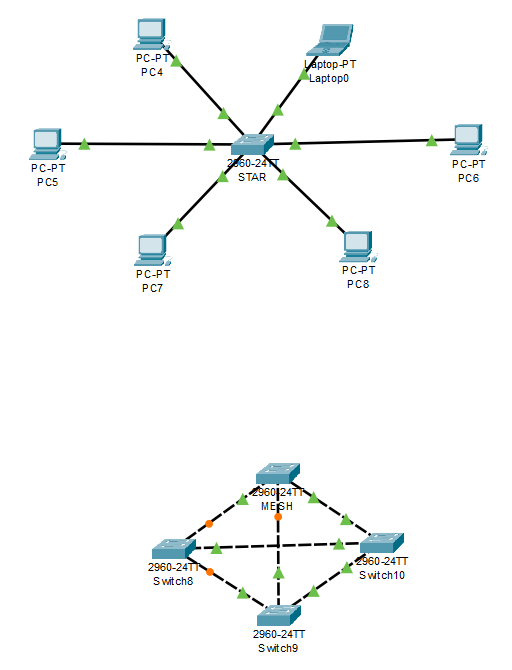




Assignment 2







Assignment 3

import java.util.Scanner;

public class HammingCode {

public static int[] encodeHammingCode(int[] data) {

int m = data.length;

int r = 0;

while (Math.pow(2, r) < (m + r + 1)) {

r++;

}

int[] encoded = new int[m + r];

int j = 0;

for (int i = 1; i <= encoded.length; i++) {

if (Math.pow(2, j) == i) {

j++;

} else {

encoded[i - 1] = data[m - i + j];

}

}

for (int i = 0; i < r; i++) {

int parityIndex = (int) Math.pow(2, i) - 1;

int parityValue = 0;

for (int k = parityIndex; k < encoded.length; k += (2 \* (parityIndex + 1))) {

for (int l = k; l < k + parityIndex + 1 && l < encoded.length; l++) {

parityValue ^= encoded[l];

}

}

encoded[parityIndex] = parityValue;

}

return encoded;

}

public static int[] detectAndCorrectHammingCode(int[] encoded) {

int r = 0;

int n = encoded.length;

// Find the number of parity bits

while (Math.pow(2, r) < n + 1) {

r++;

}

int errorPosition = 0;

for (int i = 0; i < r; i++) {

int parityIndex = (int) Math.pow(2, i) - 1;

int parityValue = 0;

for (int k = parityIndex; k < n; k += (2 \* (parityIndex + 1))) {

for (int l = k; l < k + parityIndex + 1 && l < n; l++) {

parityValue ^= encoded[l];

}

}

if (parityValue != 0) {

errorPosition += parityIndex + 1;

}

}

if (errorPosition != 0) {

System.out.println("Error detected at position: " + errorPosition);

encoded[errorPosition - 1] ^= 1; // Correct the error by flipping the bit

} else {

System.out.println("No errors detected.");

}

return encoded;

}

public static int[] extractData(int[] encoded) {

int r = 0;

int n = encoded.length;

while (Math.pow(2, r) < n + 1) {

r++;

}

int[] data = new int[n - r];

int j = 0;

for (int i = 1; i <= n; i++) {

if (Math.pow(2, j) == i) {

j++;

} else {

data[i - j - 1] = encoded[i - 1];

}

}

return data;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of data bits:");

int dataBits = scanner.nextInt();

int[] data = new int[dataBits];

System.out.println("Enter the data bits one by one:");

for (int i = 0; i < dataBits; i++) {

data[i] = scanner.nextInt();

}

int[] encoded = encodeHammingCode(data);

System.out.println("Encoded Hamming Code:");

for (int bit : encoded) {

System.out.print(bit + " ");

}

System.out.println();

System.out.println("Introduce error by flipping a bit? (yes/no):");

String error = scanner.next();

if (error.equalsIgnoreCase("yes")) {

System.out.println("Enter the position to flip (1-based index):");

int position = scanner.nextInt();

encoded[position - 1] ^= 1;

}

int[] correctedCode = detectAndCorrectHammingCode(encoded);

System.out.println("Corrected Hamming Code:");

for (int bit : correctedCode) {

System.out.print(bit + " ");

}

System.out.println();

int[] extractedData = extractData(correctedCode);

System.out.println("Extracted original data:");

for (int bit : extractedData) {

System.out.print(bit + " ");

}

System.out.println();

scanner.close();

}

}

Output:



Assignment 4

package Go\_back\_N;

import java.util.Random;

import java.util.Scanner;

public class SlidingWindowProtocol {

static Random *random* = new Random();

public static void goBackN(int totalFrames, int windowSize) {

int sentFrames = 0;

while (sentFrames < totalFrames) {

int windowEnd = Math.*min*(sentFrames + windowSize, totalFrames);

System.***out***.println("\nSending frames " + sentFrames + " to " + (windowEnd - 1));

for (int i = sentFrames; i < windowEnd; i++) {

boolean success = *random*.nextBoolean();

if (success) {

System.***out***.println("Frame " + i + " successfully received.");

} else {

System.***out***.println("Frame " + i + " lost or damaged. Resending window from frame " + i);

sentFrames = i;

break;

}

if (i == windowEnd - 1) {

sentFrames = windowEnd;

}

}

}

System.***out***.println("All frames successfully transmitted using Go-Back-N.\n");

}

public static void selectiveRepeat(int totalFrames, int windowSize) {

boolean[] received = new boolean[totalFrames];

int sentFrames = 0;

while (sentFrames < totalFrames) {

int windowEnd = Math.*min*(sentFrames + windowSize, totalFrames);

System.***out***.println("\nSending frames " + sentFrames + " to " + (windowEnd - 1));

for (int i = sentFrames; i < windowEnd; i++) {

if (!received[i]) {

boolean success = *random*.nextBoolean();

if (success) {

System.***out***.println("Frame " + i + " successfully received.");

received[i] = true;

} else {

System.***out***.println("Frame " + i + " lost or damaged. Will retransmit.");

}

}

}

while (sentFrames < totalFrames && received[sentFrames]) {

sentFrames++;

}

}

System.***out***.println("All frames successfully transmitted using Selective Repeat.\n");

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("Sliding Window Protocol Simulation in Peer-to-Peer Mode");

System.***out***.println("Enter total number of frames to send:");

int totalFrames = scanner.nextInt();

System.***out***.println("Enter the window size:");

int windowSize = scanner.nextInt();

System.***out***.println("\nChoose a mode:\n1. Go-Back-N\n2. Selective Repeat");

int mode = scanner.nextInt();

switch (mode) {

case 1:

*goBackN*(totalFrames, windowSize);

break;

case 2:

*selectiveRepeat*(totalFrames, windowSize);

break;

default:

System.***out***.println("Invalid mode selected.");

}

scanner.close();

}

}

Sliding Window Protocol Simulation in Peer-to-Peer Mode

Enter total number of frames to send:

8

Enter the window size:

4

Choose a mode:

1. Go-Back-N

2. Selective Repeat

1

Sending frames 0 to 3

Frame 0 successfully received.

Frame 1 successfully received.

Frame 2 successfully received.

Frame 3 successfully received.

Sending frames 4 to 7

Frame 4 successfully received.

Frame 5 lost or damaged. Resending window from frame 5

Sending frames 5 to 7

Frame 5 successfully received.

Frame 6 successfully received.

Frame 7 successfully received.

All frames successfully transmitted using Go-Back-N.

Sliding Window Protocol Simulation in Peer-to-Peer Mode

Enter total number of frames to send:

8

Enter the window size:

4

Choose a mode:

1. Go-Back-N

2. Selective Repeat

2

Sending frames 0 to 3

Frame 0 successfully received.

Frame 1 successfully received.

Frame 2 lost or damaged. Will retransmit.

Frame 3 successfully received.

Sending frames 2 to 5

Frame 2 successfully received.

Frame 4 successfully received.

Frame 5 lost or damaged. Will retransmit.

Sending frames 5 to 7

Frame 5 successfully received.

Frame 6 successfully received.

Frame 7 successfully received.

All frames successfully transmitted using Selective Repeat.

Assignment 5

package subnetting;

import java.util.Scanner;

public class Subnetting {

public static int[] convertToIntArray(String ip) {

String[] parts = ip.split("\\.");

int[] ipAddress = new int[4];

for (int i = 0; i < 4; i++) {

ipAddress[i] = Integer.*parseInt*(parts[i]);

}

return ipAddress;

}

public static void printIPAddress(int[] ip) {

System.***out***.println(ip[0] + "." + ip[1] + "." + ip[2] + "." + ip[3]);

}

public static int calculateSubnetBits(int subnets) {

int bits = 0;

while (Math.*pow*(2, bits) < subnets) {

bits++;

}

return bits;

}

public static int[] calculateSubnetMask(int subnetBits) {

int[] subnetMask = {255, 255, 255, 0};

int mask = 0;

for (int i = 0; i < subnetBits; i++) {

mask += (1 << (7 - i));

}

subnetMask[3] = mask;

return subnetMask;

}

public static int calculateHostsPerSubnet(int subnetBits) {

return (int) Math.*pow*(2, (8 - subnetBits)) - 2; // Subtracting 2 for network and broadcast addresses

}

public static void printSubnets(int[] ipAddress, int[] subnetMask, int subnets, int hostsPerSubnet) {

System.***out***.println("Subnet Addresses:");

int increment = 256 / subnets;

for (int i = 0; i < subnets; i++) {

int[] subnetAddress = ipAddress.clone();

subnetAddress[3] = i \* increment;

*printIPAddress*(subnetAddress);

}

System.***out***.println("\nEach subnet can have " + hostsPerSubnet + " usable hosts.");

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("Enter a Class C IP address (e.g., 192.168.1.0):");

String ipInput = scanner.nextLine();

int[] ipAddress = *convertToIntArray*(ipInput);

System.***out***.println("Enter the number of subnets you want:");

int subnets = scanner.nextInt();

int subnetBits = *calculateSubnetBits*(subnets);

int[] subnetMask = *calculateSubnetMask*(subnetBits);

System.***out***.println("Subnet Mask:");

*printIPAddress*(subnetMask);

int hostsPerSubnet = *calculateHostsPerSubnet*(subnetBits);

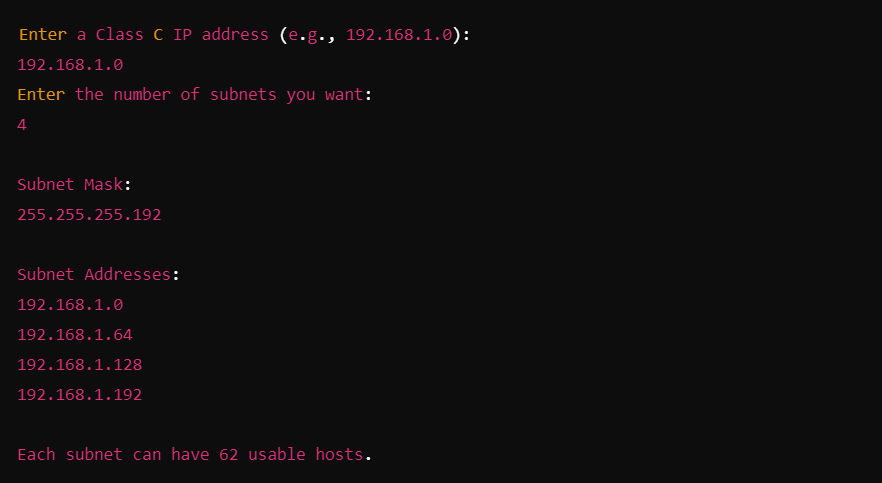
*printSubnets*(ipAddress, subnetMask, subnets, hostsPerSubnet);

scanner.close();

}

}

Output:



Assignment 6

import java.util.Arrays;

import java.util.Scanner;

public class RoutingProtocols {

// Dijkstra's Algorithm for Link State Routing Protocol

public static void dijkstra(int graph[][], int src) {

int V = graph.length; // Number of vertices

int[] dist = new int[V]; // The output array. dist[i] will hold the shortest distance from src to i

boolean[] sptSet = new boolean[V]; // sptSet[i] will be true if vertex i is included in shortest path tree

// Initialize all distances as INFINITE and sptSet[] as false

Arrays.fill(dist, Integer.MAX\_VALUE);

Arrays.fill(sptSet, false);

// Distance of source vertex from itself is always 0

dist[src] = 0;

// Find shortest path for all vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum distance vertex from the set of vertices not yet processed.

int u = minDistance(dist, sptSet);

// Mark the picked vertex as processed

sptSet[u] = true;

// Update dist[] for adjacent vertices of the picked vertex

for (int v = 0; v < V; v++) {

// Update dist[v] if it is not in sptSet, there is an edge from u to v, and total weight

// of path from src to v through u is smaller than the current value of dist[v]

if (!sptSet[v] && graph[u][v] != 0 && dist[u] != Integer.MAX\_VALUE && dist[u] + graph[u][v] < dist[v]) {

dist[v] = dist[u] + graph[u][v];

}

}

}

// Print the calculated shortest distances

printSolution(dist, src);

}

// A utility method to find the vertex with minimum distance value, from

// the set of vertices not yet included in shortest path tree

private static int minDistance(int dist[], boolean sptSet[]) {

int min = Integer.MAX\_VALUE, minIndex = -1;

for (int v = 0; v < dist.length; v++) {

if (!sptSet[v] && dist[v] <= min) {

min = dist[v];

minIndex = v;

}

}

return minIndex;

}

// Method to print the shortest path solution

private static void printSolution(int dist[], int src) {

System.out.println("Vertex \t\t Distance from Source (Node " + src + ")");

for (int i = 0; i < dist.length; i++) {

System.out.println(i + " \t\t " + dist[i]);

}

}

// Bellman-Ford Algorithm for Distance Vector Routing Protocol

public static void bellmanFord(int graph[][], int src) {

int V = graph.length;

int[] dist = new int[V];

// Initialize distances from src to all other vertices as INFINITE

Arrays.fill(dist, Integer.MAX\_VALUE);

dist[src] = 0;

// Relax all edges |V| - 1 times

for (int i = 0; i < V - 1; i++) {

for (int u = 0; u < V; u++) {

for (int v = 0; v < V; v++) {

if (graph[u][v] != 0 && dist[u] != Integer.MAX\_VALUE && dist[u] + graph[u][v] < dist[v]) {

dist[v] = dist[u] + graph[u][v];

}

}

}

}

// Check for negative-weight cycles

for (int u = 0; u < V; u++) {

for (int v = 0; v < V; v++) {

if (graph[u][v] != 0 && dist[u] != Integer.MAX\_VALUE && dist[u] + graph[u][v] < dist[v]) {

System.out.println("Graph contains a negative weight cycle.");

return;

}

}

}

// Print the calculated shortest distances

printSolution(dist, src);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of nodes in the network:");

int V = scanner.nextInt();

int[][] graph = new int[V][V];

System.out.println("Enter the adjacency matrix of the network (0 if no direct link):");

for (int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {

graph[i][j] = scanner.nextInt();

}

}

System.out.println("Choose the routing protocol:\n1. Link State (Dijkstra's Algorithm)\n2. Distance Vector (Bellman-Ford Algorithm)");

int choice = scanner.nextInt();

System.out.println("Enter the source node:");

int src = scanner.nextInt();

switch (choice) {

case 1:

dijkstra(graph, src);

break;

case 2:

bellmanFord(graph, src);

break;

default:

System.out.println("Invalid choice!");

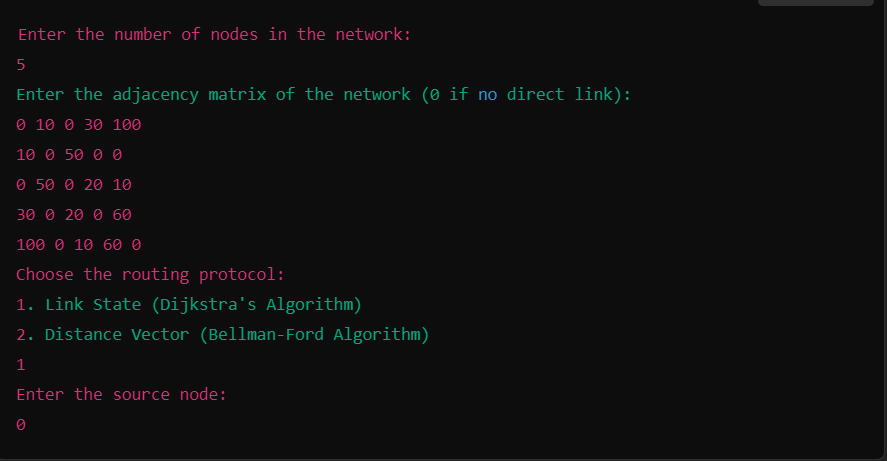
}

scanner.close();

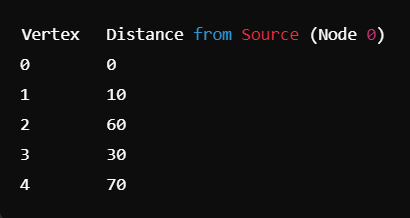
}

}

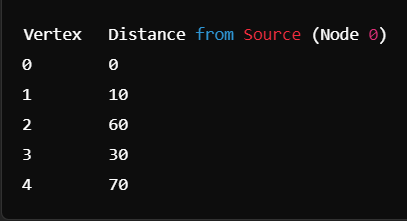
Input:



Output dijkstra:

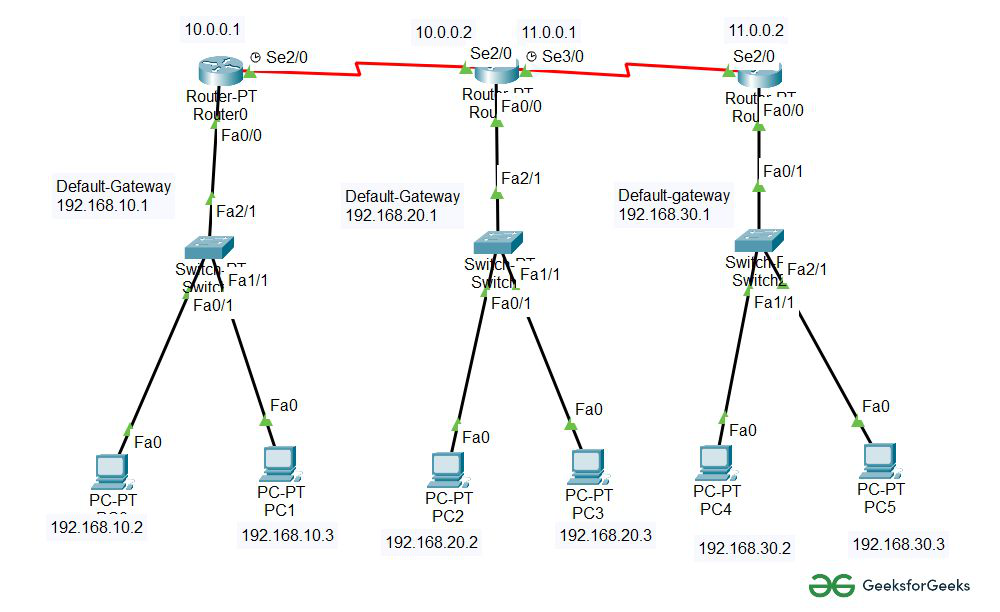


Output BellmanFord:



Assignment 7

| **S.NO** | **Device** | **IPv4 Address** | **Subnet mask** | **Default Gateway** |
| --- | --- | --- | --- | --- |
| **1.** | PC0 | 192.168.10.2 | 255.255.255.0 | 192.168.10.1 |
| **2.** | PC1 | 192.168.10.3 | 255.255.255.0 | 192.168.10.1 |
| **3.** | PC2 | 192.168.20.2 | 255.255.255.0 | 192.168.20.1 |
| **4.** | PC3 | 192.168.20.3 | 255.255.255.0 | 192.168.20.1 |
| **5.** | PC4 | 192.168.30.2 | 255.255.255.0 | 192.168.30.1 |
| **6.** | PC5 | 192.168.30.3 | 255.255.255.0 | 192.168.30.1 |



Assignment 8

package tcp;

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class TCPClient {

public static void main(String[] args) {

try {

Socket socket = new Socket("localhost", 8080);

System.***out***.println("Connected to server.");

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

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

Scanner scanner = new Scanner(System.***in***);

System.***out***.println("Choose an option: \n1. Say Hello \n2. File Transfer \n3. Calculator");

int choice = scanner.nextInt();

scanner.nextLine();

switch (choice) {

case 1:

// a. Say Hello to Each Other

out.println("HELLO");

String helloResponse = in.readLine();

System.***out***.println("Server says: " + helloResponse);

break;

case 2:

// b. File Transfer

out.println("FILE");

*receiveFile*(socket, "received\_example.txt");

break;

case 3:

// c. Calculator

System.***out***.println("Enter a calculation (e.g., 12 + 5): ");

String expression = scanner.nextLine();

out.println("CALCULATOR");

out.println(expression);

String result = in.readLine();

System.***out***.println("Calculation result: " + result);

break;

default:

System.***out***.println("Invalid choice.");

break;

}

socket.close();

} catch (Exception e) {

e.printStackTrace();

}

}

private static void receiveFile(Socket socket, String saveAs) {

try {

byte[] buffer = new byte[4096];

InputStream is = socket.getInputStream();

FileOutputStream fos = new FileOutputStream(saveAs);

int bytesRead;

while ((bytesRead = is.read(buffer)) > 0) {

fos.write(buffer, 0, bytesRead);

}

fos.close();

System.***out***.println("File received and saved as " + saveAs);

} catch (IOException e) {

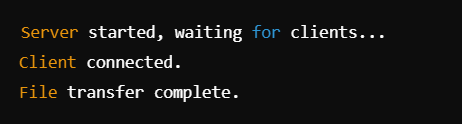
e.printStackTrace();

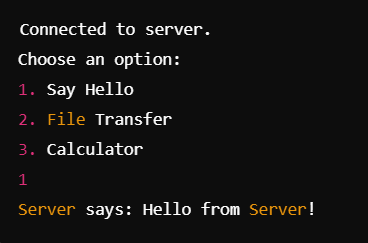
}

}

}

Output:





Assignment 9

package udp;

import java.io.\*;

import java.net.\*;

public class UDPServer {

private static final int PORT = 9876;

private static final int BUFFER\_SIZE = 1024;

public static void main(String[] args) {

DatagramSocket socket = null;

try {

socket = new DatagramSocket(PORT);

System.out.println("UDP Server is running on port " + PORT);

byte[] receiveBuffer = new byte[BUFFER\_SIZE];

DatagramPacket packet = new DatagramPacket(receiveBuffer, receiveBuffer.length);

socket.receive(packet);

String fileName = new String(packet.getData(), 0, packet.getLength());

System.out.println("Receiving file: " + fileName);

FileOutputStream fos = new FileOutputStream("received\_" + fileName);

boolean receiving = true;

while (receiving) {

receiveBuffer = new byte[BUFFER\_SIZE];

packet = new DatagramPacket(receiveBuffer, receiveBuffer.length);

socket.receive(packet);

if (packet.getLength() == 0) {

receiving = false;

System.out.println("File transfer complete.");

} else {

fos.write(packet.getData(), 0, packet.getLength());

}

}

fos.close();

} catch (IOException e) {

e.printStackTrace();

} finally {

if (socket != null && !socket.isClosed()) {

socket.close();

}

}

}

}

package udp;

import java.io.\*;

import java.net.\*;

public class UDPClient {

private static final String SERVER\_ADDRESS = "localhost"; // Change this to the server's IP address if on a different machine

private static final int SERVER\_PORT = 9876;

private static final int BUFFER\_SIZE = 1024;

public static void main(String[] args) {

DatagramSocket socket = null;

try {

socket = new DatagramSocket();

BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the file path to send:");

String filePath = userInput.readLine();

File file = new File(filePath);

if (!file.exists()) {

System.out.println("File not found.");

return;

}

String fileName = file.getName();

byte[] fileNameBytes = fileName.getBytes();

DatagramPacket fileNamePacket = new DatagramPacket(fileNameBytes, fileNameBytes.length, InetAddress.getByName(SERVER\_ADDRESS), SERVER\_PORT);

socket.send(fileNamePacket);

System.out.println("Sending file: " + fileName);

FileInputStream fis = new FileInputStream(file);

byte[] sendBuffer = new byte[BUFFER\_SIZE];

int bytesRead;

while ((bytesRead = fis.read(sendBuffer)) != -1) {

DatagramPacket packet = new DatagramPacket(sendBuffer, bytesRead, InetAddress.getByName(SERVER\_ADDRESS), SERVER\_PORT);

socket.send(packet);

}

DatagramPacket endPacket = new DatagramPacket(new byte[0], 0, InetAddress.getByName(SERVER\_ADDRESS), SERVER\_PORT);

socket.send(endPacket);

System.out.println("File transfer complete.");

fis.close();

} catch (IOException e) {

e.printStackTrace();

} finally {

if (socket != null && !socket.isClosed()) {

socket.close();

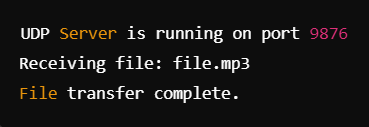
}

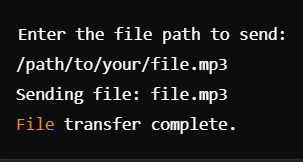
}

}

}

Output:





Assignment 10

package dns;

import java.net.\*;

import java.util.Scanner;

public class DNSLookup {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nDNS Lookup Options:");

System.out.println("1. Find IP address from URL");

System.out.println("2. Find URL from IP address");

System.out.println("3. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

scanner.nextLine();

switch (choice) {

case 1:

// Option 1: Find IP address from URL

System.out.print("Enter URL (e.g., www.google.com): ");

String url = scanner.nextLine();

findIPAddress(url);

break;

case 2:

// Option 2: Find URL from IP address

System.out.print("Enter IP address (e.g., 142.250.190.78): ");

String ipAddress = scanner.nextLine();

findHostname(ipAddress);

break;

case 3:

// Option 3: Exit

System.out.println("Exiting DNS Lookup.");

scanner.close();

return;

default:

System.out.println("Invalid choice, please try again.");

}

}

}

private static void findIPAddress(String url) {

try {

InetAddress inetAddress = InetAddress.getByName(url);

System.out.println("IP Address for " + url + ": " + inetAddress.getHostAddress());

} catch (UnknownHostException e) {

System.out.println("Unknown Host: " + e.getMessage());

}

}

private static void findHostname(String ipAddress) {

try {

InetAddress inetAddress = InetAddress.getByName(ipAddress);

String hostName = inetAddress.getCanonicalHostName();

System.out.println("Hostname for " + ipAddress + ": " + hostName);

} catch (UnknownHostException e) {

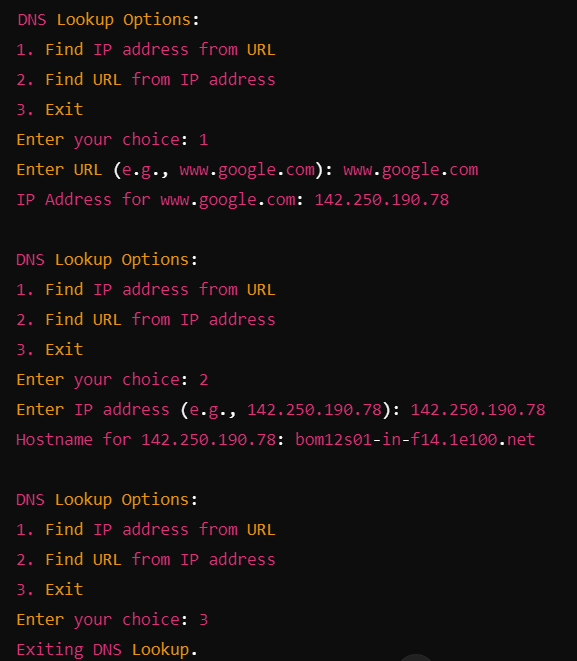
System.out.println("Invalid IP Address: " + e.getMessage());

}

}

}

Output:



Assignment 11

public class RemoteSoftwareInstaller {

public static void main(String[] args) {

String host = "192.168.1.100";

String user = "admin";

String password = "password123";

String command = "sudo apt-get install -y curl";

try {

// Set up JSch session

JSch jsch = new JSch();

Session session = jsch.getSession(user, host, 22);

session.setPassword(password);

// Skip host key checking

session.setConfig("StrictHostKeyChecking", "no");

session.connect();

// Open an execution channel

ChannelExec channel = (ChannelExec) session.openChannel("exec");

channel.setCommand(command);

// Capture command output

InputStream in = channel.getInputStream();

channel.connect();

// Read command output

byte[] tmp = new byte[1024];

while (in.read(tmp) != -1) {

System.out.print(new String(tmp));

}

// Close channel and session

channel.disconnect();

session.disconnect();

System.out.println("Software installed successfully!");

} catch (Exception e) {

e.printStackTrace();

}

}

}

Output:

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following NEW packages will be installed:

curl

0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.

Need to get 234 kB of archives.

After this operation, 788 kB of additional disk space will be used.

Get:1 http://archive.ubuntu.com/ubuntu focal/main amd64 curl 7.68.0-1ubuntu2.6 [234 kB]

Fetched 234 kB in 1s (234 kB/s)

Selecting previously unselected package curl.

(Reading database ... 34567 files and directories currently installed.)

Preparing to unpack .../curl\_7.68.0-1ubuntu2.6\_amd64.deb ...

Unpacking curl (7.68.0-1ubuntu2.6) ...

Setting up curl (7.68.0-1ubuntu2.6) ...

Processing triggers for man-db (2.9.1-1) ...

Software installed successfully!