**DEPARTMENT OF COMPUTER SCIENCE**

ITCS 6166 – COMPUTER COMMUNICATION AND NETWORKING

**Programming Project -2**

Implementing Distance Vector Routing Protocol

**SUBMITTED TO**

**Dewan T. Ahmed, Ph.D.**

**SUBMITTED BY:**

**BHARAT SAI TEJA YANTRAPATI**

**(801167000)**

**MOUNIKA RAYANA**

**(801168148)**

**MUKESH KUMAR PULLABHATLA**

**(801165738)**

1. Introduction**:**

The protocol implemented in this project is Distance Vector Routing.

* 1. DVR Protocol:

In Distance-Vector Routing, the first thing each node creates is its own least-cost tree with the rudimentary information it has about its immediate neighbours.

The incomplete trees are exchanged between immediate neighbours to make the trees more and more complete and to represent the whole routing network.

We can say that in distance vector routing, a router continuously tells all its neighbours what it knows about the whole routing network.

Let 𝑑𝑥(𝑦) : estimate of least cost from x to y

Then, x maintains distance vector Dx = [Dx(y):y belongs to N]

then 𝑑𝑥(𝑦) =

Link cost changes:

* Node detects local link cost change
* Updates routing info, recalculates distance vector
* If DV changes, notify neighbors

Node x:

Knows cost to each neighbour v:c(x,v)

Maintains its neighbours’ distance vectors. For each neighbour v, x maintains

Dv = [Dv(y):y belongs to N]

1. Implementation:

The distance vector routing protocol in our case is implemented using JAVA. There are two java files:

1) Master.java – The purpose of the master is to take the input as a directory path as an input. It then gets access to the number of routers present in the path and assigns port numbers to each of those corresponding routers.

2) Slave.java – The purpose of the Slave class is to find the shortest distance to other routers. The other functions are:

It also detects link cost changes and cost recalculation. The wait time we had allowed for both receiving the distance vector from neighbours and after re-computing is 6 and 9 seconds.

Key Points:

The wait time we had allowed for both receiving the distance vector from neighbours and after re-computing is 7 and 8 seconds.

• We made use of the process builder to invoke multiple processes for the different routers.

• There are two threads used respectively for reading and writing the distance vectors.

• Error Handling mechanism for entering the invalid port number has been included.

1. Procedure to run the project:

Go to the command prompt and set the folder to “src” folder and compile the files by running Master.java and Slave.java using the command “javac \*.java”.

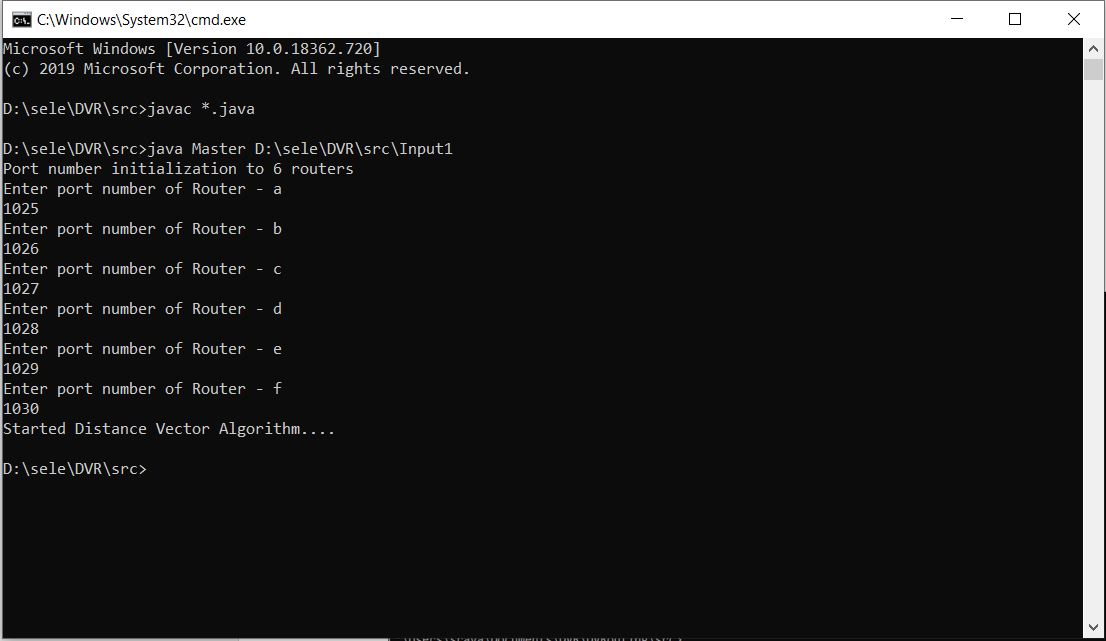
Then run by giving the following command “java Master <Input1 file path>”

If we are checking for link weight/cost and computing shortest path using Distance vector Protocol Algorithm, then the command should be given by compiling using the command “java Master <Input2 file path>”

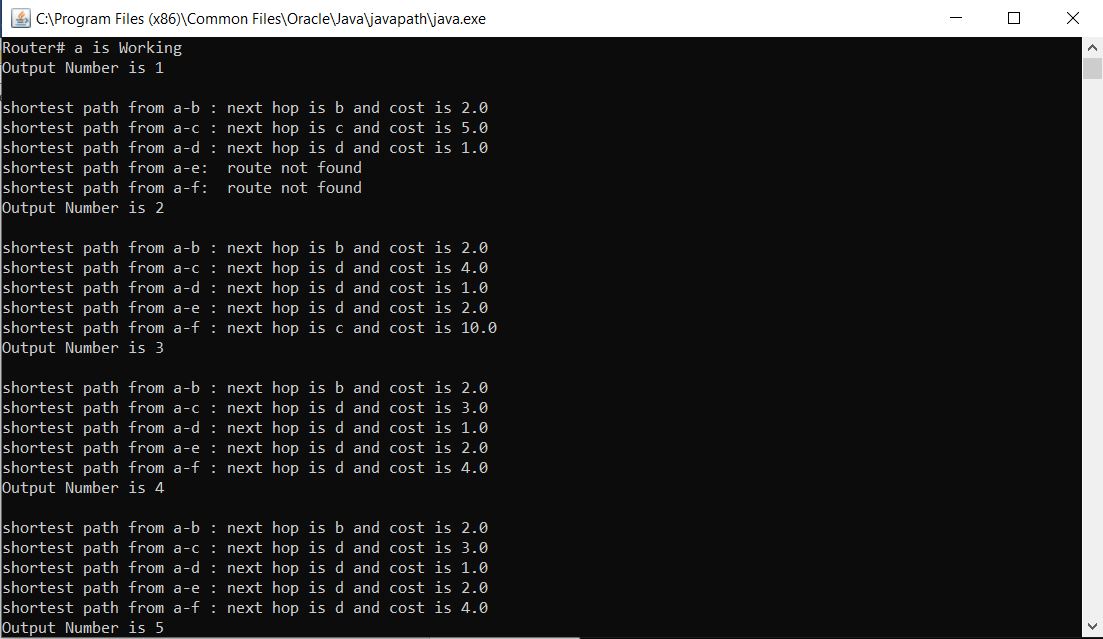
Output:

Case1:

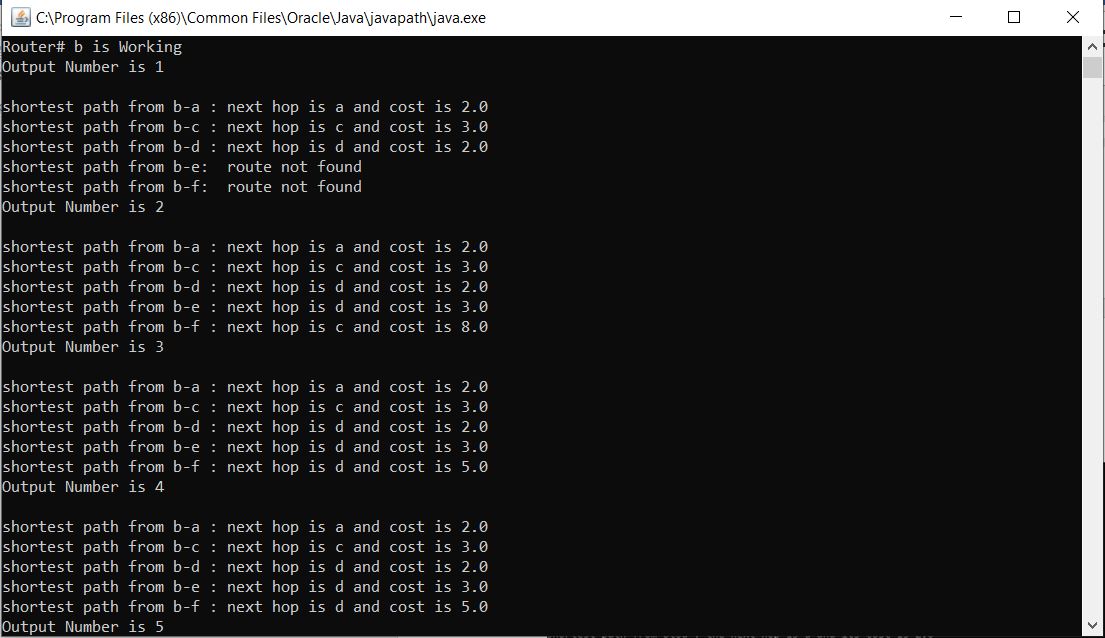
Access the .dat files and then assign the port numbers to each of the six routers.



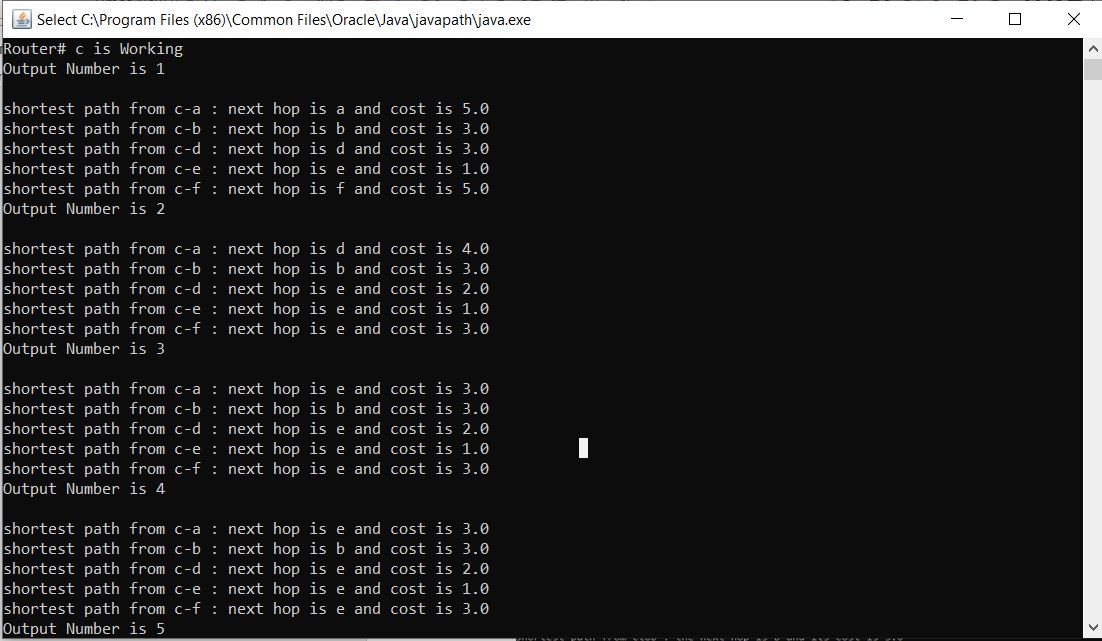
Router a:



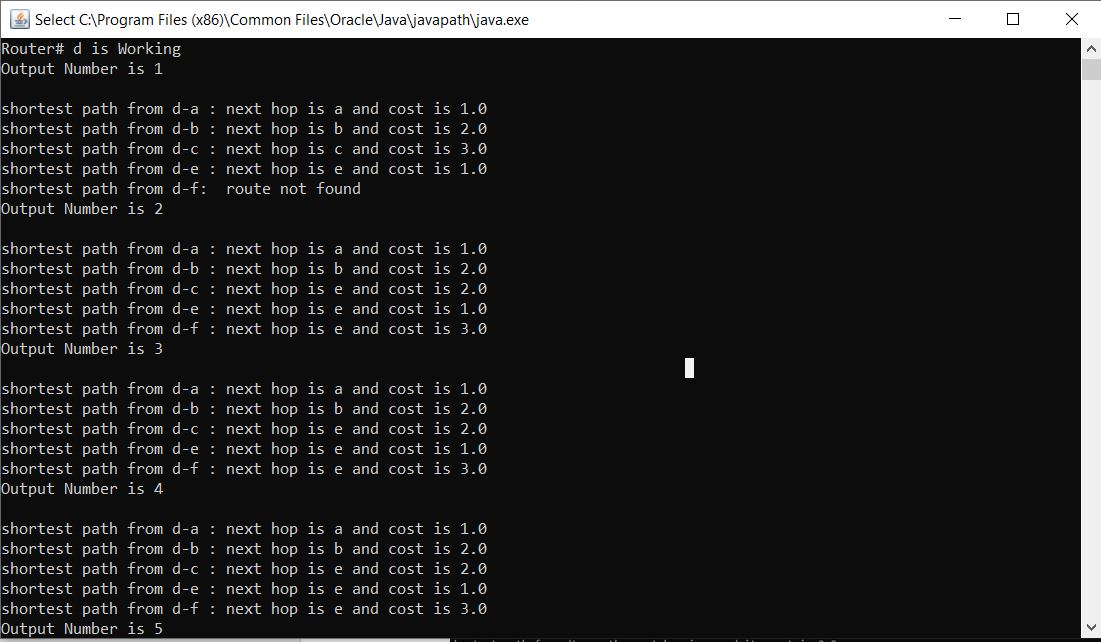
Router b:



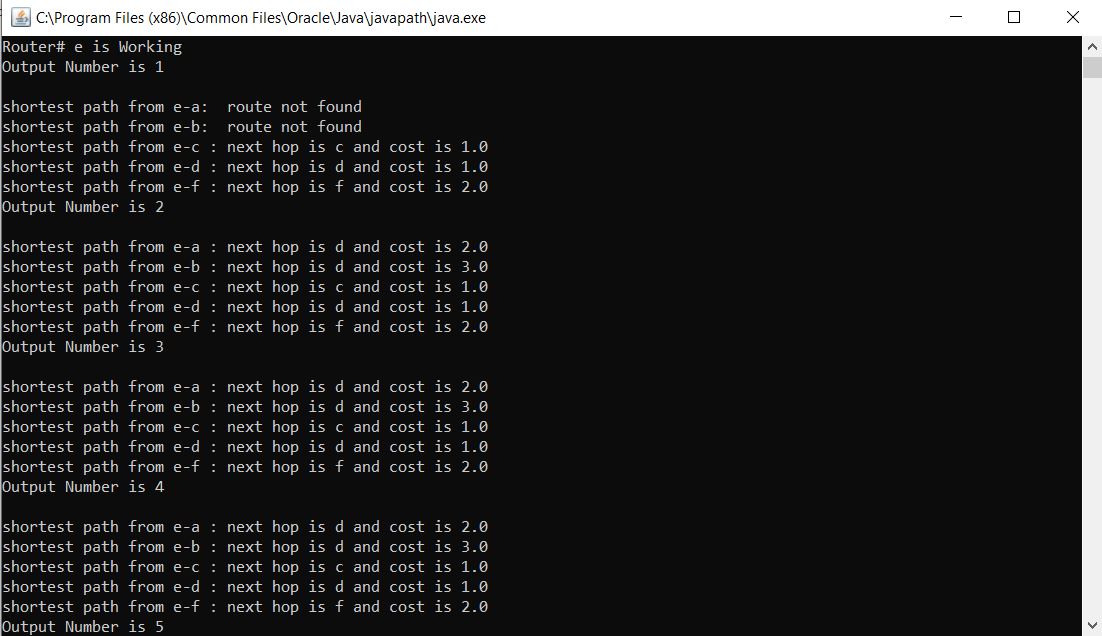
Router c:



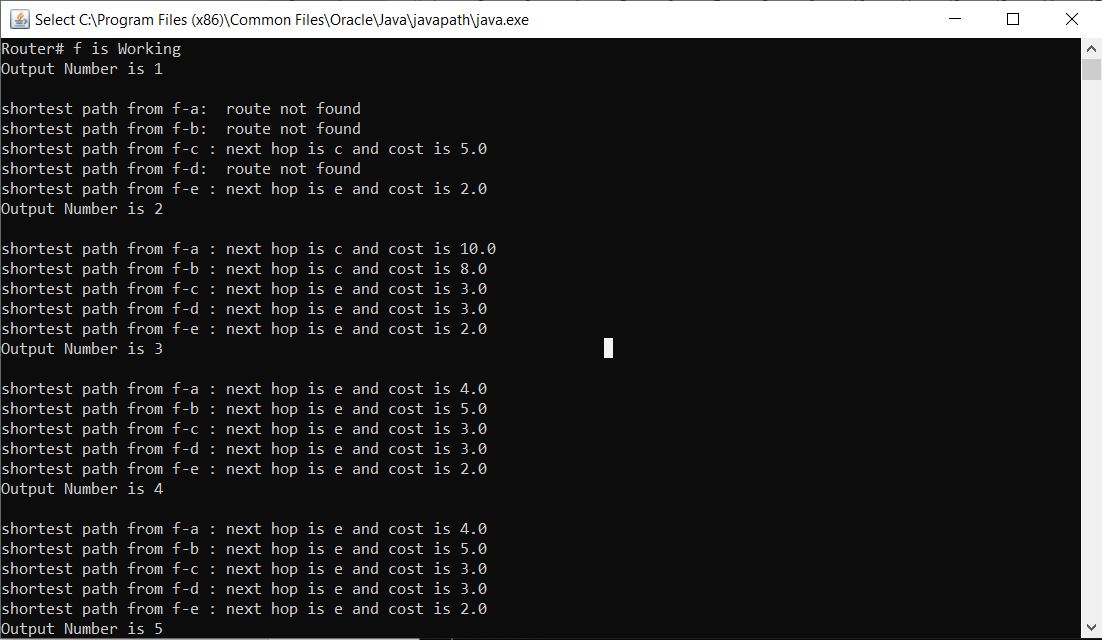
Router d:



Router e:



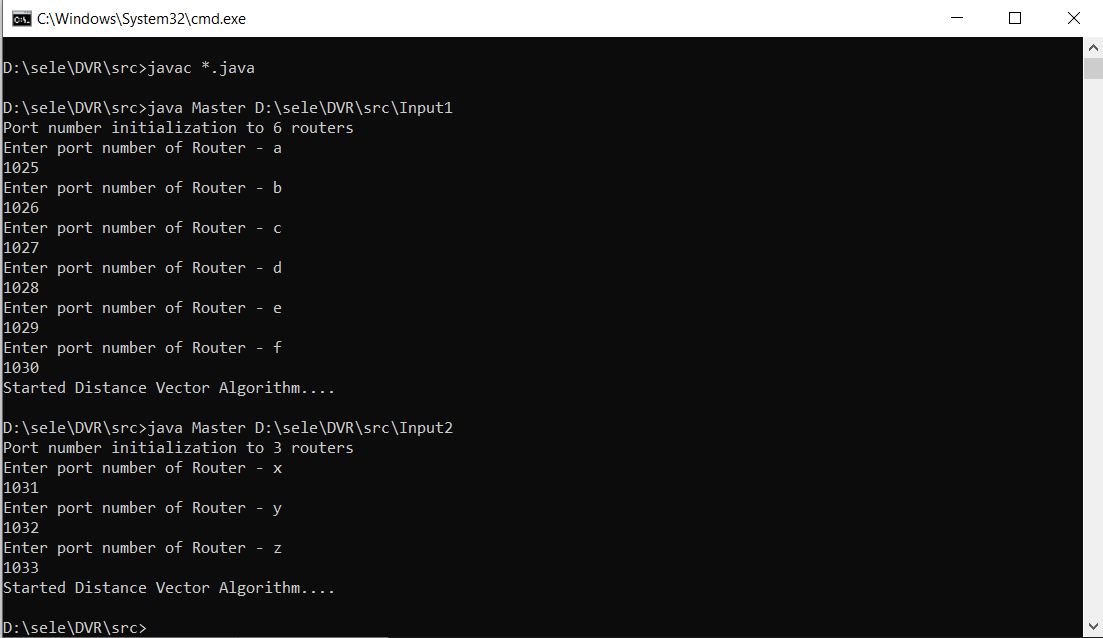
Router f:



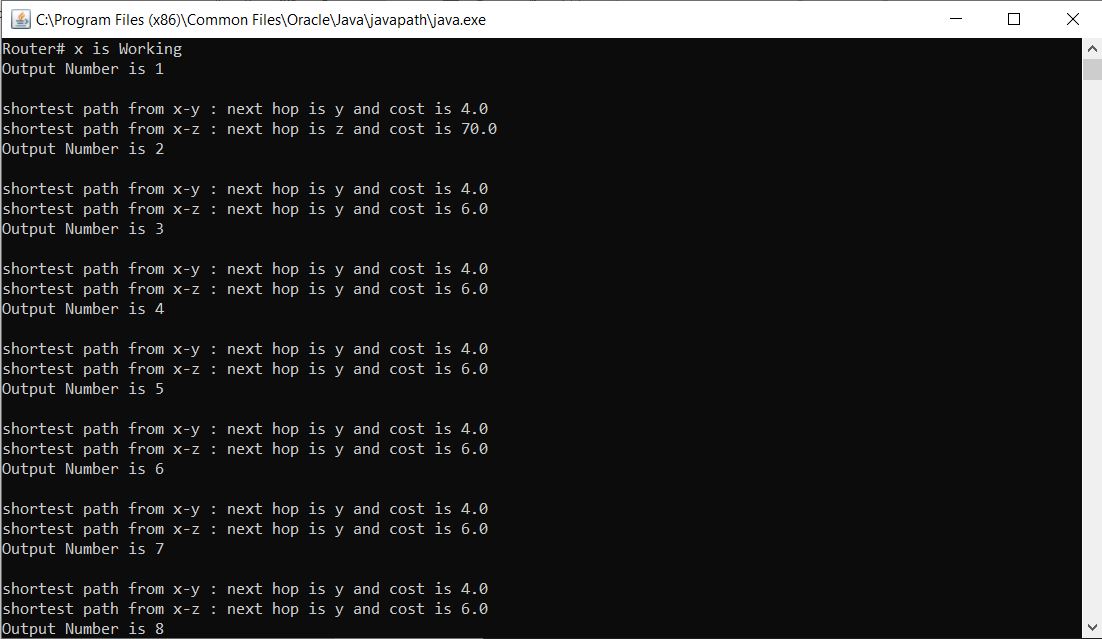
Case 2:

For Input 2 we take routers x,y,z by changing the distance between routers

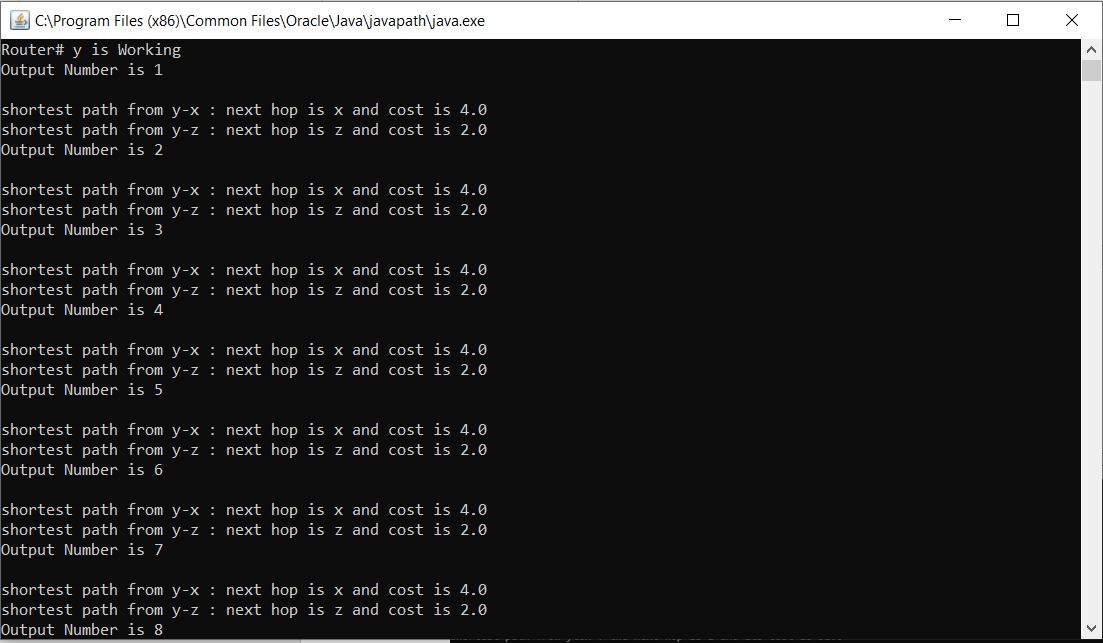
By using Distance vector algorithm, the distance between x and z changes from 70(initial hop is x) to 6(hop changes to y)



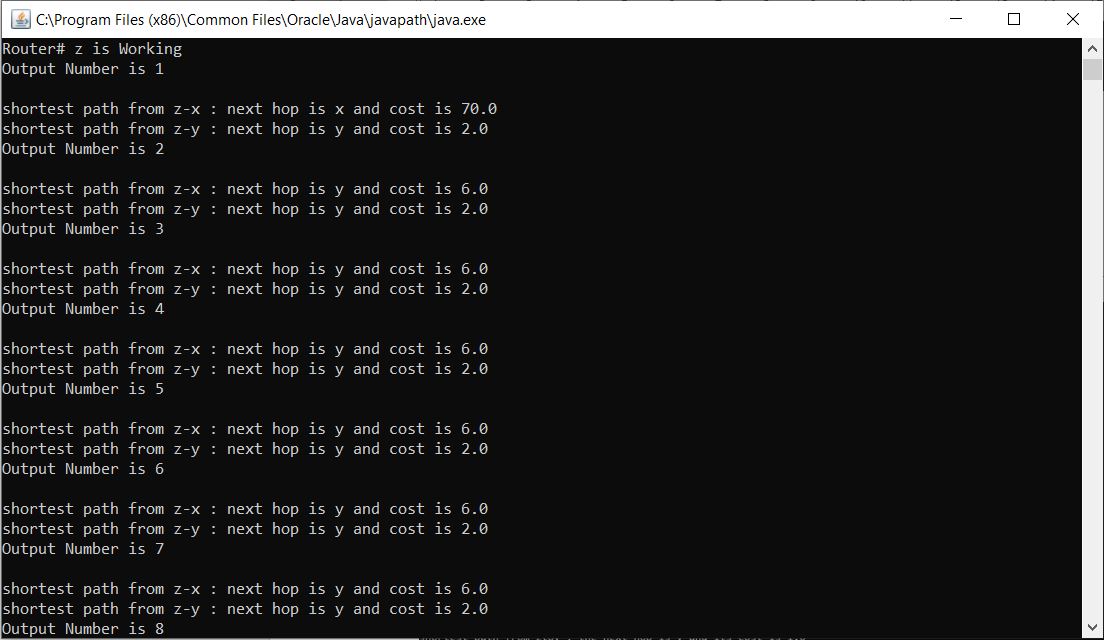
Router x:



Router y:



Router z:



1. **Java Code:**

**Master.java:**

**# Program is used to read data from input files, take the port numbers and call the main program #that calculates the minimum distances**

**import java.util.HashSet;**

**import java.util.Scanner;**

**import java.io.File;**

**import java.io.IOException;**

**/\*\***

**\* @author Mukesh, Bharat, Mounika**

**\***

**\*/**

**public class Master {**

**/\*\***

**\* @param args**

**\*/**

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

**// creating Hashset variable to give port for evey node**

**HashSet<Integer> portSet = new HashSet<>();**

**String path = args[0]; // reading the file from main method arguments**

**File dirPath = new File(path); // accesing file directory path**

**// validation to check the input arguments of file path**

**if(argumentsValidation(args, dirPath)) {**

**return;**

**}**

**// Accessing data from the file**

**File data[] = dirPath.listFiles();**

**int dataSize = data.length;**

**int[] ports = new int[dataSize];**

**String dvNodes = "";**

**System.out.println("Port number initialization to " + dataSize + " routers");**

**//validation of input numbers from scanner and accessing the distance vector nodes**

**dvNodes = portInputs(portSet, data, dataSize, ports, dvNodes);**

**// code for sending to slave.**

**for (int i = 0; i < dataSize; i++) {**

**ProcessBuilder pb = new ProcessBuilder("cmd.exe", "/c", "start java Slave " + (i + 1) + " \"" + data[i].getParent().replace("\\", "/") + "\" " + dataSize + dvNodes);**

**try {**

**pb.start();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**System.out.println("Started Distance Vector Algorithm....");**

**}**

**/\*\***

**\* @param portSet**

**\* @param data**

**\* @param dataSize**

**\* @param ports**

**\* @param dvNodes**

**\* @return**

**\*/**

**private static String portInputs(HashSet<Integer> portSet, File[] data, int dataSize, int[] ports, String dvNodes) {**

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

**for (int i = 0; i < dataSize; i++) {**

**String rNum = data[i].getName();**

**rNum = rNum.substring(0, rNum.indexOf(".dat"));**

**System.out.println("Enter port number of Router - " + rNum);**

**boolean check = true;**

**while (check) {**

**try {**

**int portNum = Integer.parseInt(scanner.nextLine());**

**if (portNum <= 1024 || portNum >= 65536)**

**throw new NumberFormatException();**

**if (portSet.contains(portNum))**

**throw new Exception();**

**check = false;**

**ports[i] = portNum;**

**portSet.add(portNum);**

**} catch (NumberFormatException e) {**

**System.out.println("Enter valid Port number between 1024 and 65536.");**

**check = true;**

**} catch (Exception e) {**

**System.out.println("Port already in use");**

**check = true;**

**}**

**}**

**dvNodes=dvNodes+ " " + rNum + ":" + ports[i];**

**}**

**scanner.close();**

**return dvNodes;**

**}**

**/\*\***

**\* @param args**

**\* @param dirPath**

**\*/**

**private static boolean argumentsValidation(String[] args, File dirPath) {**

**boolean flag=false;**

**if (args.length == 0) {**

**System.out.println("Enter file location");**

**flag= true;**

**} else if (args.length > 1) {**

**System.out.println("Input Format is Incorrect:More than one parameter is entered");**

**flag= true;**

**}**

**if (!dirPath.isDirectory()) {**

**System.out.println(" Entered Path is not correct");**

**flag= true;**

**}**

**return flag;**

**}**

**}**

**Slave.Java:**

**#Checks for the changes in distance between routers and computes the shortest #distance continuously**

**import java.util.Arrays;**

**import java.io.File;**

**import java.io.FileReader;**

**import java.io.IOException;**

**import java.net.DatagramPacket;**

**import java.net.DatagramSocket;**

**import java.io.BufferedReader;**

**import java.net.InetAddress;**

**import java.net.SocketException;**

**/\*\***

**\* @author Mukesh, Bharat, Mounika**

**\***

**\*/**

**public class Slave {**

**// initialization of required variables for application of DVR**

**public static DatagramSocket routerSocket;**

**public static File routerFile;**

**public static double[] routerMyVector;**

**public static double[][] routerVectors;**

**public static String[] routerNodes,listNextHop,routersNeighbouring;**

**public static int[] routerPorts;**

**public static int routerId,routersCount = 1;**

**/\*\***

**\* @param args**

**\*/**

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

**// Accessing the arguments from the main method.**

**int totalLength = Integer.parseInt(args[2]); // total length value from the arguments.**

**int currentRId = Integer.parseInt(args[0]); // current router Id from the arguments**

**String path = args[1]; // path from the arguments**

**setParams(totalLength, args, currentRId, path);**

**ActionThread writeThread = new ActionThread("w");**

**// starting an action thread to write action;**

**writeThread.start();**

**ActionThread readThread = new ActionThread("r");**

**// starting an action thread to read action;**

**readThread.start();**

**// looping to run the process until the user termination.**

**while (true)**

**;**

**}**

**/\*\***

**\* Distance vector Algorithm logic to find the shortest path**

**\* no return statements or method arguments**

**\*/**

**public static void distanceVectorRoutingAlgorithm() {**

**// iteration over accessing the neighboring nodes.**

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

**// accessing the index nodes costs.**

**int index = findIndex(routersNeighbouring[i]);**

**// iteration to find the next hop for shortest path**

**for (int j = 0; j < routerMyVector.length; j++) {**

**if (j == routerId - 1) {**

**continue;**

**} else if (i == 0) {**

**routerMyVector[j] = routerVectors[routerId - 1][index] + routerVectors[index][j];**

**listNextHop[j] = routersNeighbouring[i];**

**} else {**

**if (routerMyVector[j] > routerVectors[routerId - 1][index] + routerVectors[index][j]) {**

**listNextHop[j] = routersNeighbouring[i];**

**routerMyVector[j] = routerVectors[routerId - 1][index] + routerVectors[index][j];**

**}**

**}**

**}**

**}**

**}**

**/\*\***

**\* @param vector**

**\* @param portNum**

**\* method to update cost of shortest path.**

**\*/**

**public synchronized static void updateDistanceVectors(String[] vector, int portNum) {**

**int index = 0;**

**int portLength=routerPorts.length;**

**// iteration over port length**

**while(index < portLength)**

**{**

**if (routerPorts[index] == portNum)**

**{**

**break;**

**}**

**index++;**

**}**

**if (index == portLength)**

**{**

**return;**

**}**

**for (int i = 0; i < vector.length; i++)**

**{**

**routerVectors[index][i] = Double.parseDouble(vector[i]);**

**}**

**}**

**/\*\***

**\* @param length**

**\* @param args**

**\* @param RId**

**\* @param parentPath**

**\*/**

**public static void setParams(int length, String[] args, int RId, String parentPath) {**

**// initializing and assigning the values of global static variables(routerId,routerVectors,routerPorts,routerNodes).**

**routerId = RId;**

**routerVectors = new double[length][length];**

**routerPorts = new int[length];**

**routerNodes = new String[length];**

**// accessing routerNodes and routerPorts through iterating over length**

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

**Arrays.fill(routerVectors[i], Double.MAX\_VALUE);**

**routerVectors[i][i] = 0.0;**

**String[] temp = args[i + 3].split(":");**

**routerNodes[i] = temp[0];**

**routerPorts[i] = Integer.parseInt(temp[1]);**

**}**

**routerMyVector = new double[length];**

**listNextHop = new String[length];**

**Arrays.fill(routerMyVector, Double.MAX\_VALUE);**

**routerMyVector[routerId - 1] = 0.0;**

**routerFile = new File(parentPath + "/" + routerNodes[routerId - 1] + ".dat");**

**// accessing socket**

**try {**

**routerSocket = new DatagramSocket(routerPorts[routerId - 1]);**

**} catch (SocketException e) {**

**e.printStackTrace();**

**}**

**System.out.println("Router# " + routerNodes[routerId - 1] + " is Working");**

**}**

**/\*\***

**\* method to read routers.**

**\* no return statements or method arguments.**

**\*/**

**public static void read() {**

**try {**

**Arrays.fill(routerVectors[routerId - 1], Double.MAX\_VALUE);**

**routerVectors[routerId - 1][routerId - 1] = 0.0;**

**BufferedReader bReader = new BufferedReader(new FileReader(routerFile));**

**int length = Integer.parseInt(bReader.readLine());// accessing length from the router file**

**routersNeighbouring = new String[length];**

**// iteration over length**

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

**String[] temp = bReader.readLine().split(" ");**

**int index = findIndex(temp[0]);**

**routersNeighbouring[i] = temp[0];**

**if (routersCount == 1) {**

**listNextHop[index] = temp[0];**

**routerMyVector[index] = Double.parseDouble(temp[1]);**

**} else {**

**}**

**routerVectors[routerId - 1][index] = Double.parseDouble(temp[1]);**

**}**

**bReader.close();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**/\*\***

**\* no return statements or method arguments.**

**\* reading the data from socket.**

**\*/**

**public static void readPacketData() {**

**// looping to run the process until the user termination.**

**boolean check = true;**

**while (check) {**

**try {**

**String action = "u";**

**byte[] data = new byte[1024];**

**int size = data.length;**

**DatagramPacket packet = new DatagramPacket(data, size);**

**routerSocket.receive(packet);**

**int length = packet.getLength();**

**String vector = new String(packet.getData(), 0, length);**

**ActionThread updateThread = new ActionThread(action, vector, packet.getPort());**

**updateThread.start();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**/\*\***

**\* no return statements or method arguments.**

**\* method to display the shortest path cost and next hop output.**

**\*/**

**public static void output() {**

**System.out.println("Output Number is " + routersCount++);**

**System.out.println();**

**String src = routerNodes[routerId - 1];**

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

**if (i != (routerId - 1)) {**

**String dest = routerNodes[i];**

**if (routerMyVector[i] == Double.MAX\_VALUE) {**

**System.out.println("shortest path from " + src + "-" + dest + ": " + " route not found");**

**} else {**

**System.out.println("shortest path from " + src + "-" + dest + " : next hop is " + listNextHop[i]**

**+ " and cost is " + routerMyVector[i]);**

**}**

**}**

**}**

**}**

**/\*\***

**\* @param temp**

**\* @return**

**\* method to find the index value**

**\*/**

**public static int findIndex(String temp) {**

**int pos = -1;**

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

**if (routerNodes[i].equals(temp)) {**

**pos = i;**

**break;**

**}**

**}**

**return pos;**

**}**

**/\*\***

**\* no return statements or method arguments.**

**\* method to broadcast the updated forwarding table to neighbouring routers**

**\*/**

**public static void broadCastToNeighbours() {**

**try {**

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

**String data = "";**

**for (int j = 0; j < routerMyVector.length; j++) {**

**if (routersNeighbouring[i].equals(listNextHop[j])) {**

**data = data + Double.MAX\_VALUE + ":";**

**} else {**

**data += routerMyVector[j] + ":";**

**}**

**}**

**DatagramPacket packet = new DatagramPacket(data.getBytes(), data.getBytes().length);**

**packet.setAddress(InetAddress.getByName("localhost"));**

**packet.setPort(routerPorts[findIndex(routersNeighbouring[i])]);**

**routerSocket.send(packet);**

**}**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**/\*\***

**\* @author Bharat, Mukesh, Mounika.**

**\* Class for thread creation to continuously process read write and update operations.**

**\***

**\*/**

**class ActionThread extends Thread {**

**private String action;**

**private String vector;**

**private int portNum;**

**/\*\***

**\* @param action**

**\* @param vector**

**\* @param port**

**\*/**

**public ActionThread(String action, String vector, int port) {**

**this.action = action;**

**this.vector = vector;**

**this.portNum = port;**

**}**

**/\*\***

**\* @param action**

**\*/**

**public ActionThread(String action) {**

**this.action = action;**

**}**

**/\*\***

**\*no return statements or method arguments.**

**\*/**

**public void run() {**

**if (action.equalsIgnoreCase("r")) {**

**Slave.readPacketData();**

**} else if (action.equalsIgnoreCase("w")) {**

**while (true) {**

**try {**

**Slave.read();**

**Slave.output();**

**Slave.broadCastToNeighbours();**

**// sleep time for thread to maintain a buffer time for 7000 millisec**

**Thread.sleep(7000);**

**Slave.distanceVectorRoutingAlgorithm();**

**// sleep time for thread to maintain a buffer time for 8000 millisec**

**Thread.sleep(8000);**

**} catch (Exception e) {**

**}**

**}**

**} else if (action.equalsIgnoreCase("u")) {**

**Slave.updateDistanceVectors(vector.split(":"), portNum);**

**}**

**}**

**}**