

**A PROJECT REPORT ON** **IMPLEMENTATION OF DISTANCE VECTOR ROUTING PROTOCOL.**

**Computer Communications and Networks**  
**ITCS 6166/8166, Spring 2020**

**Project Members: Professor: Dr. Dewan T Ahmed**

**Prabhav A. Jani (801166353) TA: Divya Patel**

**Salma Roohi Khayum (801168027) Sahithi Priya Gutta**

**Shriya Sharma (801168387)**

**ABSTRACT**

Distance Vector Routing or DVR for short is a routing algorithm used to find the shortest/best route for data packets to traverse based on distance. There are a lot of factors involved in DVR such as cost, latency, and availability of routers but in this project, we will be focusing on cost. The term *distance vector* refers to the fact that the protocol manipulates *vectors,* arrays of distances to other nodes in the network. The distance vector algorithm was the original ARPANET routing algorithm and was implemented more widely in LANs with the Routing Information Protocol (RIP).

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| 1. | Algorithm | 4 |
| 2. | Program Introduction | 5 |
| 3. | Program design and Implementation | 6-7 |
| 4. | How to run the program | 7-9 |
| 5. | Output | 10 |
| 6. | References | 11 |

**ALGORITHM:**

Distance Vector Algorithm (vertices, edges, vertex V, source S)

for each vertex V in vertices:

next hop[v] = v // initialize predecessor or next hop value to the same port

cost[source] = 0 // The cost from the source to itself is assigned to zero

Repetitive Edge relaxation:

for i = 1 to n-1: // n stands for the total number of vertices. Iterate through all the vertices

for each edge (u, v) with weight w in edges:

if cost[u] + w < cost[v]:

cost[v] = cost[u] + w

next hop [v] = u

return cost [], next hop []

Running time of the algorithm is O(V+E). Here V is the total number of vertices and E is the total number of edges in the graph.

**Program Introduction:**

We have used Python language in our program. The version used is Python 3.8

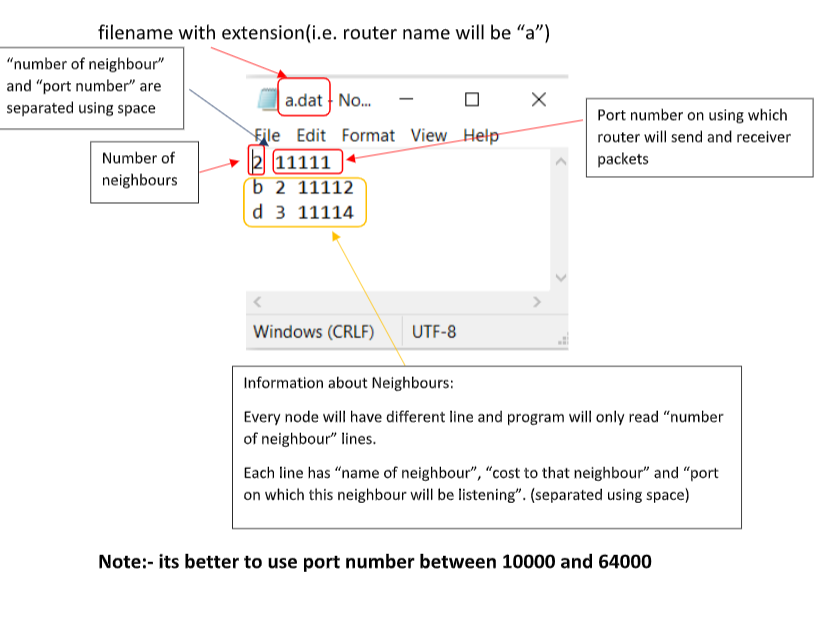
We have used global variables in our program like

* “name”  it is name of current node, it is used in place of IP address.
* “my\_port” this stores the port number used by the current node.
* “my\_neighbour” used to sore the node names of the neighbors and the port number on which they listen (neighbour nodes=>['a->11111\n', 'c->11113\n', 'e->11115']).
* “costs\_found\_in\_file” this is used to have costs to the neighbour, it will be used to check the file for cost change.
* “My\_routing\_table” my\_routing\_table is nested dictonary node will point to another dictionary containing cost and hop {'a':{'cost':2,'hop':'b'}}

After initializing global variables, the program will ask for the name of the input file with “.dat” extension.

From the name of the file, the program will set the name variable to the name provided by the file name excluding the “.dat” extension. After this, the program will read the file and the first line gives the information about the number of neighbors it is connected to and the port number it would use for UDP data transfer (sending and receiving routing tables). The following lines in the file will have the name of the neighbor node, cost to the neighbor node, and port number used by the neighbor node for UDP data transfer (sending and receiving routing tables). Each line will represent a neighboring node specific information.

For example:



**Program Design and Implementation:**

The program implements a specific procedure every 15 seconds. This procedure consists of the following operations:

1. Reading the file for any update (cost change)
2. Sending the current routing table to the neighbors
3. Listen for the packets from neighbors
4. Print the current shortest path to other nodes

For the first operation, the function **readFromFileForUpdate** is used.

Function readFromFileForUpdate is used to track any update in the cost to the neighbor in the input file. If there is an update found, this function will be used to set the cost of the neighbor node again. It will also change the cost to the nodes accessed through this neighbor (nodes for which the hop is this neighbor). If there is no update, the program will continue to iterate through the next neighboring nodes and check for any updates in the cost.

For the second operation, the function **sendRoutingTableToNeighbours** is used.

A new message string is created for every node starting with the current node name and the rows in the routing table separated using comma. For every neighbor, while iterating through the routing table, it will check for “hop”. If its hop is same as the name of the neighbor, that row from the routing table will not be included in the message string. This technique is also known as **Split horizon**. This is implemented to avoid recursive update problem.

Example of message string: b,b'0'b,a'2.0'a,c'5.0'c

For the third operation, the function URTUM  is used.

Firstly, program will listen for the UDP packets. After the packet is received, message from this packet is passed to this function. URTUM function is used to Update Routing Table Using Message. Message string is then split/divided using a comma and then stored in a list variable. The first element in the variable represents the node which sent the routing information. Furthermore, each element is then identified as a row of routing table sent by the neighbor. This consists of the node, cost and hop each separated by **‘** symbol.

Current routing table and the newly obtained routing tables are compared. Here, there can be 3 cases.

1. If there is any node in the newly obtained routing table, which is not present in the current routing table. This new node is inserted in the current routing table with cost equal to the “cost to the neighbor who sent this table” + “cost from neighboring node to the new node” (Cost provided in the new routing table).
2. If the hop in our current routing table for any node is same as the name of the neighbor who sent the new routing table, the routing cost is again set to “Cost to the neighbor who sent this table” + “Cost provided in the newly arrived routing table to that node”.
3. If hop in our current routing table for any node is **not** same as the name of the neighbor who sent the new routing table, the cost to that node is stored in a temporary variable “**original\_cost**”. This original cost is compared with **“Cost to the neighbor who sent this table” + “Cost provided in the newly arrived routing table to that node”.** Minimum cost out of these two is stored in the new variable as a **new\_cost**. **Original\_cost** and the **new\_cost** variables are compared. If both are the same, then there is no change in the current routing table. Otherwise, the cost to that node is set to a new\_cost variable and hop is changed to the name of the neighbor who sent this new routing table.

Finally, to print the current shortest path to other nodes we use **printShortestPath** function

This function uses the routing table to print the shortest path from the current node to the other nodes in the specified format.

For example:

 Output Number >2

Shortest path b-b: the next hop is b and the cost is 0

Shortest path b-a: the next hop is a and the cost is 2.0

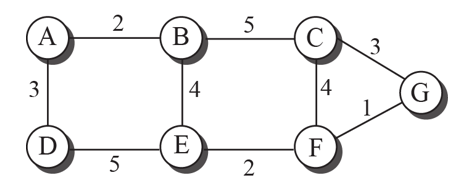
Shortest path b-c: the next hop is c and the cost is 5.0

Shortest path b-e: the next hop is e and the cost is 4.0

Here, Output Number is a global variable incremented every time this function is called.

**How to run the program:**

If you want to create below structure:



1. Create a new folder and paste the provided program and create the .dat files in it. Please refer the steps below in creating a “.dat” file for each node.
2. Double click on the .py program file, a command prompt will open and ask for the input file.

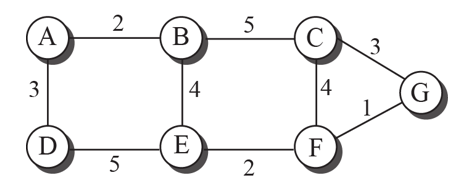
**Make sure you have python installed before running this program**

1. Choose a port no. for every node in image

(suggestion: as it is alphabets choose port range starting from 11111 and add +1 for next alphabet)

11113

11112



11117

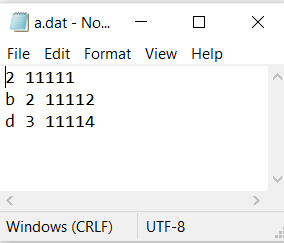
11116

11115

11114

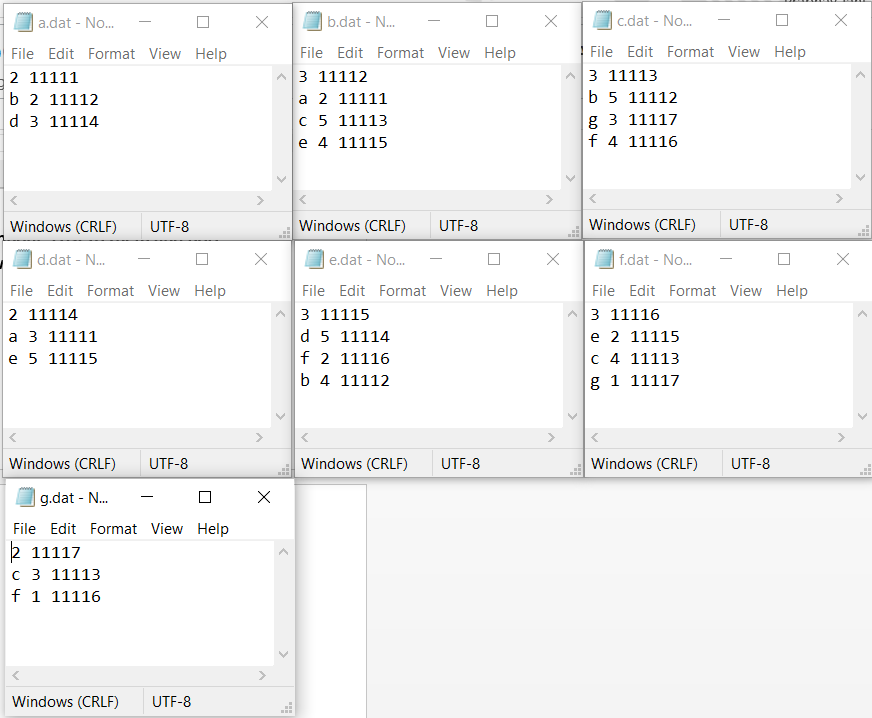
11111

1. Select a node and create file in same folder with its name and with content as follows:

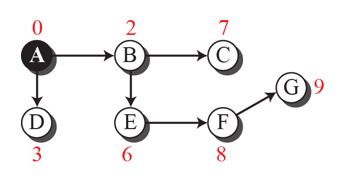


**Note :- Provide appropriate names of neighbor, cost to them and port number on which they are listening, as it will send and receive packets from neighbors.**

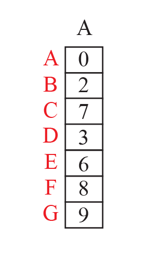
1. Do the same for all other nodes



1. Open terminal for each node (or just run the program as it has infinite loop it will show the output) and provide name of the file
2. For given figure shortest path from “a” to every node will be

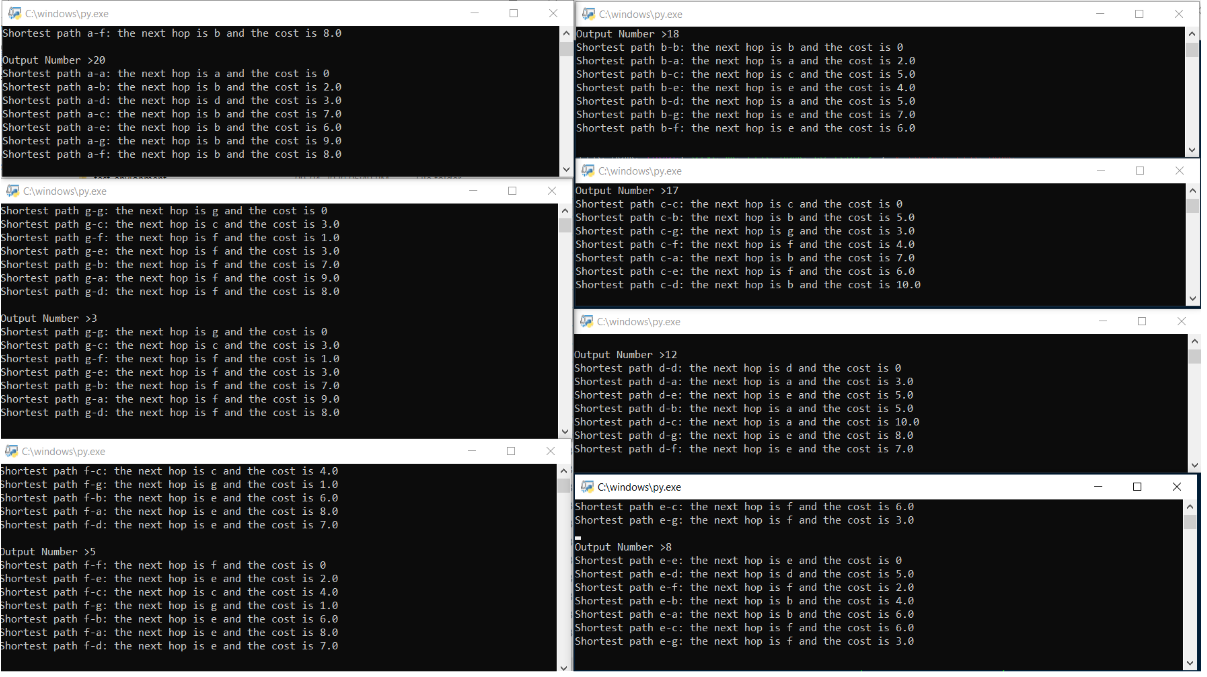


1. Routing table will be:



The sequence could be out of order as the information may arrive out of order

**Output:**



References:

<https://stackoverflow.com/questions/44916411/pythons-udp-crashes-when-sending-to-ip-port-that-isnt-listening>

<https://www.w3schools.com/python/default.asp>

<https://pythontic.com/modules/socket/udp-client-server-example>

<https://en.wikipedia.org/wiki/Distance-vector_routing_protocol>