CSE 4344 LAB 2 – Distance Vector Routing Spring 2023

Objectives:

1. Simulate Distance Vector Routing
2. Continued work with IDEs and GUIs

Due: 04/16, 11:59:59 pm Difficulty: Medium

These will be individual projects. For Lab 2 the same development rules apply as for lab 1. You can use either Java or Python as the programming language.

You are to implement a simulation of Distance Vector routing. We will work with routers as nodes and links as edges, ignoring the fact that routers actually work with network addresses, not router numbers. You will read a plain text input file which will describe the network. Each line in the file will describe a link. The line will have 3 numbers, the numbers of the two nodes followed by the cost of the link. There will be a maximum of 6 nodes and a maximum of 4 links connecting to any one node. For example, using numbers for the nodes instead of letters, a network might look like this:

```
1 2 7
2 3 1
1 5 1
4 5 2
2 5 8
4 3 2
```

Input Files

You program should be able to read an input file, in a txt format as described above, eg.,

```
1 2 7
2 3 1
1 5 1
4 5 2
2 5 8
4 3 2
```

You should be able to change the input file to be read by the program.

Initializing the node

Each node should maintain a DV table, as discussed in class. After reading the input file, each node should be bootstrapped and DV table initialized base on the given txt file. For example, in the above example, node 1 will have an initial DV table as below. (I omitted the next hop for obtaining least cost)

1	2	3	4	5
0	7	-	-	1

Sending DV to Neighbors

Each node shall send its DV to its neighbors and receive DVs from neighbors too.

Passing information among the nodes can take different forms. The nodes can connect with each other using sockets when they need to. In this case, you can implement each node as separate process or thread.

Alternatively, methods for passing information among nodes can be use a shared buffer, or even a file. The node process can read the file or buffer to obtain new DV from neighbors.

Updating Own DV

Upon receiving DVs from neighbors, a node should update its own DV if updates are indeed required. If any updates occur, new generated DV should be sent to neighbors.

Stable State

If no updates are happening at any node in the network, we reach a stable state, and the program can pause.

GUI Requirements

You should have a GUI implementation that allows the user to see the Distance Vector table and the routing table in each node.

The GUI can take various forms. However, there are some basic controls and functions that are mandatory to be implemented.

You will need to show the DVs for all the nodes in the network. It is desirable to use normal windows to do this (either in separate windows for each node individually or in one window with all nodes displayed therein). However, if you have difficulties in manipulating the GUI, a simple print out using the command line will suffice too. Please neatly format the print outs so that they are easy to read. You should observe the DVs and understand the process of the algorithm.

You should be able to run the simulation in a controlled way so that the user can observe the changes in the tables as the nodes run each step in the algorithm. You can assume that all the nodes exchange tables at the same time and then make one iteration with the new information. This can be a button in the GUI window or hitting a key in the command prompt. Include DVs for each step in your writeup.

The system should be able to detect when the algorithm reaches a stable state. i.e., the nodes are not getting any new information. It should indicate this on the GUI. The GUI should also display the number of cycles that the system takes to reach a stable state. Observe this information when the stable state is reached and include it in your writeup.

You should also include an option (button or check box, or prompt) that allows the simulation to run without intervention and display a total time until the algorithm reaches a stable state. Include this total run time in your writeup.

Adjusting Link Cost

You should be able to adjust the cost of any link in the network. You should simulate a line failure by setting the cost of a link to infinity. You should then run this network **from the state it was in.** Run it once in the original single step mode so you can see the tables change. Observe the algorithm's progress and record your observation in your writeup. Then run it again without the intervention and record the running time in your writeup. Infinity is 16.

You should then simulate a line repair by setting the cost back to what it was originally. You should then make the same runs and observations as in the previous step and include them in your writeup.

Write-up:

Your write-up should follow the same rules as with lab 1. In addition, the writeup should include the ***observations discussed above***. In each case, include a comment – at least one sentence, on what you conclude from the observation.

Submission:

Your submission should follow the same rules as with lab 1.

Grading:

Points element:

05	Select the input file
10	Initial DV tables set up in GUI
10	System detects a stable state
20	Runs algorithm correctly in single step mode – record DVs
10	Runs algorithm without stopping – displays time
10	Change a link cost and run from the previous state
30	Writeup – include all elements discussed above
05	Comments in code

Your comments should follow the same rules as with lab 1.
Deductions for failing to follow directions will follow the same rules as with lab 1.

Important Note:

The same rules about collusion and plagiarism apply as with lab 1.