CS/ECE 438: Communication Networks **Spring 2022**

Machine Problem 3

Handed Out: March 27*,* 2022 *Due: April* 10*, 2022, 23:59 CST*

TA: Federico Cifuentes-Urtubey

Abstract

This machine problem tests your understanding of the distance vector and the link state routing algorithms.

1 Introduction

In this MP, you will implement the link state and distance vector routing protocols. You will write two separate programs: one that implements the link state protocol, and one that implements the distance vector protocol. Both programs will read the same file formats to get the network's topology and what messages to send.

2 Router Programs

Your program should contain a collection of imaginary routing nodes that carry out their routing protocol (link state or distance vector, for the corresponding program). These imaginary nodes are just data-structures in your program. There is no socket programming involved.

Once their tables have converged, for each node (in ascending order of node ID), write out the node's forwarding table (see "Output format" section for details). Then, have some of your nodes send some data to some other nodes, with the data forwarded according to the nodes' forwarding tables: the sources, destinations, and message contents are specified in the message file; see below for format.

Then, one at a time, apply each line in the topology changes file (see below) in order, and repeat the previous instructions after each change. The nodes in your {distance vector, link state} program should use the {DV, LS} algorithm to arrive at a correct forwarding table for the network they're in.

3 Tie breaking

We would like everyone to have consistent output even on complex topologies, so we ask you to follow specific tie-breaking rules.

1. Distance Vector Routing: when two equally good paths are available, your node should choose the one whose next-hop node ID is lower.

2. Link State: When choosing which node to move to the finished set next, if there is a tie, choose the lowest node ID.

3. If a current-best-known path and newly found path are equal in cost, choose the path whose last node before the destination has the smaller ID.
   Example: source is 1, and the current-best-known path to 9 is $1 \rightarrow 4 \rightarrow 12 \rightarrow 9$. We are currently adding node 10 to the finished set. $1 \rightarrow 2 \rightarrow 66 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 9$ costs the same as path $1 \rightarrow 4 \rightarrow 12 \rightarrow 9$. We will switch to the new path, since 10<12.

4 Input Formats

The program reads three files: the *topology* file, the *topology changes* file, and the *message* file.

Your programs will be run using the following commands:

```
./linkstate topofile messagefile changesfile
```

```
./distvec topofile messagefile changesfile
```

All files have their items delimited by newlines. The *topology* file represents the initial topology. The *topology changes* file represents a sequence of changes to the topology, to be applied one by one. The *message* file describes which nodes should send data to whom once the routing tables converge. (The tables should converge before the topology changes start, as well as after each change).

All messages in the *message* file are sent every time the tables converge. So, if there are two message lines in the messagefile, and two changes in the changesfile, a total of (2(initial) + 2(after first change in changesfile has been applied) + 2(second line in changesfile has been applied)) = 6 messages are to be sent.

A line in the *topology* file represents a link between two nodes and is structued this way:

```
<ID of a node> <ID of another node> <cost of the link between them>
```

The *topology changes* file has the exact same format. The first line in the changes file is the first change to apply, second is the second, etc. Cost -999 (and only -999) indicates that the previously existing link between the two nodes is broken. (Real link costs are always positive; never zero or negative.)

A line in the *message* file looks like

```
<source node ID> <dest node ID> <message text>
```

The files we test your code with will follow this description exactly, with nothing extraneous or improperly formatted.

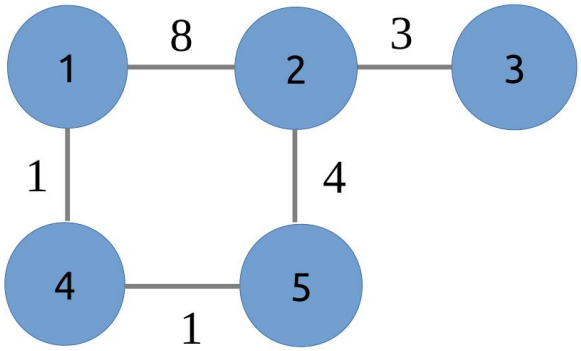

Figure 1: Sample Topology

Example topology file:

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

This would correspond to a topology of Figure 1.

Example message file:

```
2 1 here is a message from 2 to 1

3 5 this one gets sent from 3 to 5!
```

The message file would cause "here is a message from 2 to 1" to be sent from 2->1, and "this one gets sent from 3 to 5!" from 3->5. Note that node **IDs could go to double digits**.

Example changes file:

```
2 4 1
2 4 -999
```

This would add a cost 1 link between 2 and 4, and then remove it afterwards.

5 Output Format

Write all output described in this section to a file called "`output.txt`".

The forwarding table format should be:

<destination> <nexthop> <pathcost>

where nexthop is the neighbor we hand destination's packets to, and pathcost is the total cost of this path to destination. The table should be sorted by destination.

Example for node 2 from the example topology:

```
1 5 6
2 2 0
3 3 3
4 5 5
5 5 4
```

As you can see, the node's entry for itself should list the nexthop as itself, and the cost as 0. If a destination is not reachable, do not print its entry. That's one single space in between each number, with each row on its own line. Remember, you're printing all nodes' tables at once. So, the example for node 2 would have been preceded by a similarly formatted table for node 1, and followed by the tables of 3, 4, and 5.

When a message is to be sent, print the source, destination, path cost, path taken (including the source, but NOT the destination node), and message contents in the following format:
''`from <x> to <y> cost <path_cost> hops <hop1> <hop2> <...> message <message>`''
e.g. : ''`from 2 to 1 cost 6 hops 2 5 4 message here is a message from 2 to 1`''
Print messages in the order they were specified in the messages file. If the destination is not reachable, please say ''`from <x> to <y> cost infinite hops unreachable message <message>`''

Please do not print anything else; any diagnostic messages or the like should be commented out before submission. However, if you want to organize the output a little, it's okay to print as many blank lines as you want in between lines of output.

Both messagefile and changesfile can be empty. In this case, the program should just print the forwarding table.

The output file will have the general layout as follows:

```
<topology entries for node 1>
<topology entries for node 2>
<topology entries for node 3>
<topology entries for node ...>
<message output line 1>
<message output line 2>
<message output line ...>
```

——— At this point, 1^{st} change is applied

```
<topology entries for node 1>
<topology entries for node 2>
<topology entries for node 3>
<topology entries for node ...>
<message output line 1>
<message output line 2>
<message output line ...>
```

——— At this point, 2^{nd} change is applied

```
<topology entries for node 1>
<topology entries for node 2>
<topology entries for node 3>
<topology entries for node ...>
<message output line 1>
```

```
<message output line 2>
<message output line ...>
```
——— And so on...

6 Notes

All the notes for the previous MPs still apply. We are not repeating those here for brevity.

New information:

1. Your project must include a Makefile whose default target makes executables called `distvec` and `linkstate`. **DO NOT move the Makefile from its original location; otherwise, the Autograder will not be able to compile your code.**
2. Command line format:
   ```
   ./distvec topofile messagefile changesfile
   ./linkstate topofile messagefile changesfile
   ```

Submission instructions are same as for previous MPs. Tests generally take **10-15 minutes**, and there may be a queue of students. You can see where you are in the queue at `http://mobius03.cs.illinois.edu:4380/queue/queue_mp3.html`.

Please refer to MP0, MP1, and MP2 instructions for other notes.