CSCE 416 (Fall 2019); Lab Assignment 5
Routing simulation

Points possible: 100
Due date: Dec. 5, 2019
Group Member Allowed?: Yes, one partner.

In this assignment, you will implement two routing protocols following the *Dijkstra (Link-state)* and *Bellman-Ford (Distance vector)* routing algorithms. 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.

Router Programs

Your program should contain a collection of imaginary routers that carry out their routing protocol (link-state or distance vector, for the corresponding program). The routers in your {link-state, distance vector} program should use the {Dijkstra, Bellman-Ford} algorithm to calculate the correct forwarding table. Once their forwarding tables have converged, for each router (in ascending order of router ID), write out the router's forwarding table (see "Output format" section for details). Then, have some of your routers send some data to some other routers, with the data forwarded according to the routers' 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 paragraph's instructions after each change.

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

— Distance Vector: When two equally good paths are available, your router should choose the one whose next-hop router ID is lower.
— Link-state:
  — When choosing which router to move to the finished set, if there is a tie, choose the lowest router ID.
  — If a current-best-known path and newly found path are equal in cost, choose the path whose last router ID before the destination has a 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 router 10 to the finished set. $1 \rightarrow 2 \rightarrow 66 \rightarrow 34 \rightarrow 5 \rightarrow 10 \rightarrow 9$ costs the same as path $1 \rightarrow 4 \rightarrow 12 \rightarrow 9$. We will switch to the new path since ID 10 $<$ ID 12.

Input Formats

Your JAVA filenames should be lsrouter.java (for Dijkstra/Link-state) and dvrouter.java (for Bellman-Ford/Distance vector). Each program reads three files: the topology file; the topology changes file; and the message file.

The command line format to run each program should be:

```
$ java lsrouter topofile changesfile messagefile
$ java dvrouter topofile changesfile messagefile.
```

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 routers should send data to whom once the forwarding tables converge. (The tables should converge before the topology changes start, as well as after each change.)

A line in the topology file represents a link between two routers, and looks like:
<ID of a router> <ID of another router> <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 is broken. (Real link costs are always positive; never zero or negative.)

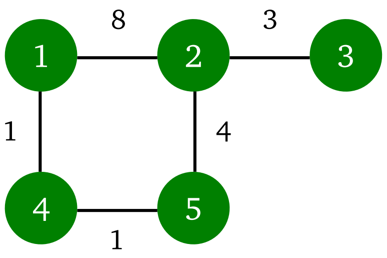

Figure 1: Network topology. Routers with unique ID and link costs.

A line in the message file looks like:
<source router ID> <destination router ID> <message text>
The files we test your code with will follow this description exactly, with nothing extraneous or improperly formatted.

Example topology file (see Figure 1):
```
1 2 8
2 3 3
2 5 4
4 1 1
4 5 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 router IDs could go to double digits.

Example changes file:
```
2 4 1
1 2 3
2 4 -999
```

This would add a link between 2 and 4 with cost 1, change the cost to 3 of the link between 1 and 2, and then remove the link between 2 and 4.

Output Format

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

The forwarding table of each router format should be:
destination nexthop pathcost

Where nexthop is the neighbor the router will send the destination's packets to, and pathcost is the total cost of this path to destination. The table should be sorted by destination.
Example for router 2 from the example topology (Figure 1):

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

As you can see, the router's entry for itself should list the nexthop as itself, and the cost as 0. That's one single space in between each number, with each row on its own line.

Remember, you should print all routers' tables at once. So, the example for router 2 would have been preceded by a similarly formatted table for router 1, and followed by the tables of router 3, 4, and 5.

When a message is to be sent, print the source, destination, path taken (including the source and destination router), and message contents in the following format:
"from <x> to <y>: hops <hop1> <hop2> <...>; message: <message>"
For example: "from 2 to 1: hops 2 5 4 1; 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 print "from <x> to <y>: hops unreachable; message: <message>"

Your program (Link-state and Distance vector) should execute $N + 1$ times, where N is the number of changes in the changes file. For each execution, calculate, print the forwarding tables and print the messages. Here is the order in which your program should print output in the "output.txt" file:

1. For each router in the topology: print the forwarding table according to the format shown above
2. For each line in the message file: print the message string according to the format shown above
3. For each line in the changes file: apply the change one at a time to the current topology and go to Step 1

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 ok to print as many blank lines as you want in between lines of output.

Submission

⋆ First, create a zip file containing the lsrouter.java (for Dijkstra/Link-state), dvrouter.java (for Bellman-Ford/Distance vector), and Makefile;
⋆ Second, rename the zip file as YOURLASTNAME_la5.zip (YOURLASTNAME in all caps). If you are working in a group of two, then you should include an additional README file inside the zip file. The README file should briefly, but clearly, discuss the responsibilities of both students in the group. Also, rename the zip file as STUDENT1LASTNAME_STUDENT2LASTNAME_la5.zip;
⋆ Third, upload it on the Blackboard.

Grade Breakdown

(1) 10%: Your code is modular and well-documented. You can implement different functionalities in different classes, but your main classes should be lsrouter and dvrouter. Include all the .java files inside Makefile. Grader will use make to compile the files. Zero point if the files do not compile.

(2) 70%: If the routers can display the correct routing table for both Link-state and Distance vector and under different topology changes.
(3) 20%: If any router can send message to a destination with appropriate message format for both Link-state and Distance vector and under different topology changes.

Notes

Same as in lab assignment 4.