CSCD58 Computer Networks - Winter 2016

Programming Assignment 3

Distance Vector Routing

Due: April 3rd, 2016, 5:00PM

In this assignment, you will implement *DVRouter* to perform the routing table update feature of distance vector routing.

Specifications

DVRouter executes to maintain *RoutingTable*, the routing table of the router it is running on. *DVRouter* maintains *RoutingTable* based on distance vector updates it receives from its neighbors. *DVRouter* reads distance vector updates from *InUpdates.txt* – the file to emulate distance vector messages from one-hop neighbors. We name a distance vector update message from a specific neighbor of *DVRouter* as *DVUpdate*. Whenever *DVRouter* finishes processing a *DVUpdate*, it reports the entire *RoutingTable* to *OutUpdates.txt* regardless of whether there has been an update on *RoutingTable* or not. *InUpdates.txt* and *OutUpdates.txt* are respectively the only input and output of *DVRouter* throughout its entire processing.

By assumption of this assignment, the *DVUpdate*-s (the entries in *InUpdates.txt*) *DVRouter* receives are the only means *DVRouter* finds out about the other routers, including its next-hop neighbors, in the network. (Recall that, in distance vector routing, a router receives updates only from its neighbors.)

InUpdates.txt contains the *DVUpdates* in their order of arrival to *DVRouter*. Upon each *DVUpdate*, *DVRouter* makes the necessary changes, if any, to its *RoutingTable*. Refer to the material we covered in class and to your textbook on distance vector routing and table updates.

***RoutingTable*:** Each entry of *RoutingTable* has the following fields:

- *destination*: the destination router
- *nextHop*: the best next hop to transmit the packets towards *destination*
- *totalCost*: the known total cost of path from *DVRouter* itself to *destination* through *nextHop*.

A suggested implementation for *RoutingTable* is a static *HashMap* member field in *DVRouter* of which the keys are *destination*-s, values are objects of a *RoutingTableType*. *RoutingTableType* can be a static inner class or a package-access class that contains the above member fields.

Assume the basic routing table maintenance specified here as the overall scope of this assignment. Any solution or concern for counting to infinity is out of context and is excluded. Likewise, do not consider any implementation of keep-alive or any other maintenance components that would be in a full distance vector routing protocol.

You can add components in addition to those mentioned here as long as they comply with these specifications. However, do not add other public classes. All your code should be contained in file *DVRouter.java*.

File Formats

InUpdates.txt and *OutUpdates.txt* are respectively the only input and output of *VRouter* throughout its entire processing. *DVRouter* reads *InUpdates.txt* from and writes *OutUpdates.txt* to the default directory.

Assume that the ID of the router *DVRouter* is running on is "DV". By this, every other router know *this* router as "DV" and refer to it so in their update messages. Assume that "DV" is an entry in every *DVUpdate DVRouter* receives.

InUpdates.txt and *OutUpdates.txt* have the following format (See Figure 1 for examples of *InUpdates.txt* and *OutUpdates.txt* files):

***InUpdates.txt*:** Each entry in *InUpdates.txt* is a *DVUpdate*, i.e., a distance vector update from a specific neighbor of *DVRouter*. Each consecutive pair of *DVUpdate*-s in *InUpdates.txt* is separated by a blank line. A *DVUpdate* entry from a specific router, say from *RouterR* to *DVRouter* is of the following form:

First line: *sequenceNumber*

is a decimal number to identify the message for operational purposes. You will be using these *sequenceNumber*-s while showing the current state of *RoutingTable* upon every *DVUpdate* (see *sequenceNumber* in *OutUpdates.txt*.)

Second line: ID of the router ("*RouterR*" in this case) sending this update.

Subsequent lines: *destination; totalCost*

where *destination* is the destination address and *totalCost* is the minimum of the path costs (as known to *RouterR*) from *RouterR* itself to *destination*. Each pair of *destination* and *totalCost* (each pair on a separate line) represents an entry in *DVUpdate RouterR* is sending at the time.

Assume the following on a *DVUpdate* from a router *RouterR*:

- *RouterR* never appears on a *DVUpdate* from *RouterR* itself. Recall that the *totalCost* on such an entry would always be 0.
- A missing router, say *RouterP*(≠*RouterR*), in a *DVUpdate* from *RouterR* means that *RouterR* is currently seeing the distance between itself and *RouterP* as infinity.

***OutUpdates.txt*:** Each entry is the overall content of *RoutingTable* upon a specific *DVUpdate* from a neighbor of *DVRouter*. The consecutive entries are separated by a blank line each. Each entry in *OutDates.txt* is of the following form:

First Line: *sequenceNumber*
is the *sequenceNumber* of the *DVUpdate* (see *InUpdates.txt*) that *DVRouter* has just processed and is now writing the results of that process. Recall: *DVRouter* should write the entire content of its *RoutingTable* to *OutUpdates.txt* upon every *DVUpdate* even though there is no change in *RoutingTable* after the processing of that *DVUpdate*.

Subsequent lines: *destination*, *nextHop*, *totalCost*
as given in *RoutingTable* structure above.

1 A DV; 7 B; 20 E; 15 2 B A; 40 C; 20 F; 20 DV; 10 E; 5 3 A B; 20 DV; 5 C; 5 D; 40 E; 15	1 A; A; 7 B; A; 27 E; A; 22 2 A; A; 7 B; B; 10 C; B; 30 F; B; 30 E; B; 15 3 A; A; 5 B; B; 10 C; A; 10 D; A; 45 F; B; 30 E; B; 15
(a)	**(b)**

Figure 1. *InUpdates.txt* and *OutUpdates.txt* file examples. a) Contents of a sample *InUpdates.txt* file, b) contents of the *OutUpdates.txt* file corresponding to the file in part (a).

Submission Instructions

- Submit on BlackBoard only the file *DVRouter.java*.
- The submission deadline is April 3rd, 2016, **5:00PM sharp**.
- Write the full name and student ID of both team members as a comment at the top of *DVRouter.java* you submit.
- BlackBoard is setup to allow up to 3 submissions. Among multiple submissions—if any, the one with the latest timestamp within deadline will be marked.
- All code must be written in Java 1.7.
- Make sure your code compiles correctly. Classes with compile time errors won't be marked.
- Ensure the quality of your code. In addition to the specifications given here, your code should be readable and well written by Java standards.