CSE 232: Assignment 4
Topic: Link State Routing
Due date: December 9, 2022

In this assignment, we provide the implementation of the Distance Vector Routing (DVR) algorithm called the Routing Information Protocol (RIP). You have to modify the given implementation to Link State Routing (LSR) based protocol.

During the implementation process, you are expected to identify the changes required to the given code (for example, data structures, routing algorithm, functions that send/receive routing table updates, etc.). These changes should be implemented to generate a functionally correct LSR implementation.

You can download the code of the DVR protocol from here.

(A) Steps to run the given code

The code folder has five files. The sample input and output files will help you test your code.

You can compile your code as follows to create the 'rip' executable.

$ *g++ -std=c++11 main.cpp routing_algo.cpp -o rip*

The code is written to take input from standard input (terminal), so you can redirect an input text file to standard input as follows, in order to run the code.

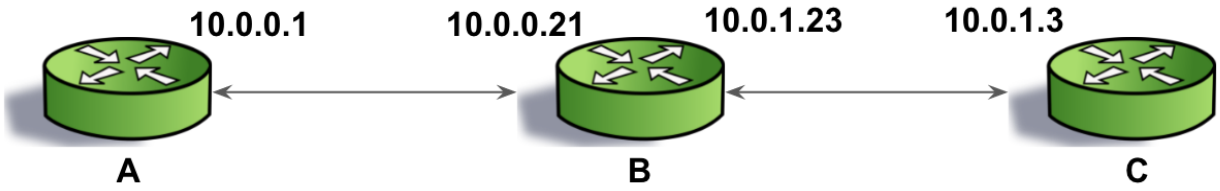
$ ./rip < sampleinput.txt

(B) Input (i.e., topology) description

Below is a sample input file provided to you (sampleinput.txt).

```
3
A B C
A 10.0.0.1 10.0.0.21 B
B 10.0.0.21 10.0.0.1 A
B 10.0.1.23 10.0.1.3 C
C 10.0.1.3 10.0.1.23 B
EOE
```

The overall topology represented by the above input is as follows.



The first line in the input file defines the number of nodes in the network for this input instance, followed by the labels for each node. Then, the next few lines (until EOE is found) will consist of the assignment of an IP address to each interface of the node along with the IP address of the

node to which it is connected-to. Here, in the above example at line #3, interface of A with an IP address 10.0.0.1 is connected to an interface of B with IP address 10.0.0.21.

(C) DVR code base description

1. The code in *main.cpp* parses the input and creates a data structure of the list of the nodes in the network, along with information about the interfaces at each node. It also initializes the routing table at each node with entries of zero cost only to itself. All the relevant data structures are defined in node.h.
2. Once the input has been parsed, a call is made to the function 'routingAlgo', which is implemented in the file *routing_algo.cpp*. The current routing algorithm computes the routing table entries at all nodes after processing the neighbor update messages (serially). Once the routing tables converge, the function "*printRT*" is used to print routing tables of each node.
3. During the execution of the routingAlgo function, a node may wish to send its routing table entries to other nodes. This is accomplished by calling the *sendMsg()* function in the node's object. Calling sendMsg at a node will result in a corresponding call to the *recvMsg* function at all its neighbors.
4. The recvMsg function runs the DVR algorithm, and updates the routing table. You will find the data structures and functions implemented in *node.h* useful; look over them carefully. In particular, the function *isMyInterface(string eth)* returns true/false depending on whether the argument passed is the own interface of a node or not, and will be helpful during your route computation algorithm.

In summary, the function routingAlgo is invoked by main, and runs the distance vector routing protocol at all nodes in the network. This function causes the nodes to send messages of their routing tables to their neighbors, who will process these routing tables in *recvMsg*. The data structures corresponding to the messages exchanged are all defined in *node.h*.

(D) Output description

If you run the given version of the code (for the given sampleinput.txt), it will print the routing table entries at each node after convergence, as shown below.

```
Printing the routing tables after the convergence
A:
10.0.0.1 | 10.0.0.1 | 10.0.0.1 | 0
10.0.0.21 | 10.0.0.21 | 10.0.0.1 | 1
10.0.1.23 | 10.0.0.21 | 10.0.0.1 | 1
10.0.1.3 | 10.0.0.21 | 10.0.0.1 | 2
B:
10.0.0.21 | 10.0.0.21 | 10.0.0.21 | 0
10.0.1.23 | 10.0.1.23 | 10.0.1.23 | 0
10.0.0.1 | 10.0.0.1 | 10.0.0.21 | 1
10.0.1.3 | 10.0.1.3 | 10.0.1.23 | 1
C:
10.0.1.3 | 10.0.1.3 | 10.0.1.3 | 0
10.0.0.21 | 10.0.1.23 | 10.0.1.3 | 1
10.0.1.23 | 10.0.1.23 | 10.0.1.3 | 1
10.0.0.1 | 10.0.1.23 | 10.0.1.3 | 2
```

The output consists of the label of a node, followed by the routing table entries at that node, in the format {destination ip | next hop | my ethernet interface | hop count}. This output is printed out by the call to *printTable()* for each node in the *'printRT'* function of *routing_algo.cpp*. Note that the function printTable sorts the routing table entries so that the output is deterministic for a given input, enabling ease in grading at our end.

(E) Problem Statement

Q.1. Complete the code to implement LSR [5+5]

a. You must complete the 'routingAlgo' function so that it correctly computes the routing table entries at all nodes. You must figure out when the routing tables have converged after which the routing tables can be printed out using the *"printRT"* function (already called at the end of 'routingAlgo').

b. The *sendMsg* function is already implemented in node.h but that is as per DVR, you should modify this as per LSR. You must complete the *recvMsg* function in order to process routing table updates at a given node. The *recvMsg* function should run Dijkstra's algorithm (for each node), compute and update the routing table entries.

Q.2. Modify the code/input implemented for Q.1. to generate the converged routing tables for the network topology shown in the figure below. [5]