CSE 232: Assignment 4
Topic: Distance Vector Routing
Due date: November 23, 2021

In this assignment, you will write code to implement a distance vector routing protocol called Routing Information Protocol (RIP). You can download the code of the assignment from here.

The code folder has five files. You will only need to edit the file routing_algo.cpp unless specified. You must not make changes to node.h or main.cpp unless specified in the question. 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*

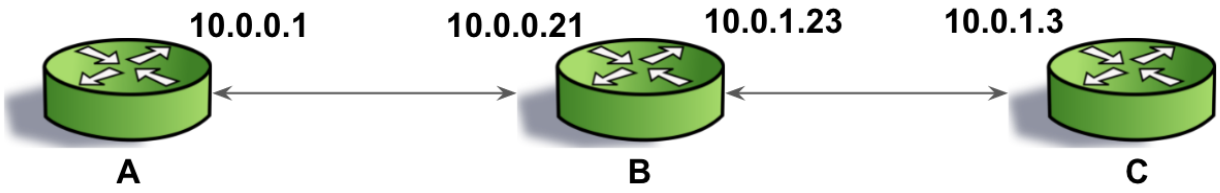
The code is written to take input from standard input (terminal), so you can redirect an input text file to standard input as follows, in order to run the code.

$ ./rip < sampleinput.txt

Below is a sample input file provided to you.

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



Following is the description of the code base provided.

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. ***You must complete the 'routingAlgo' function so that it correctly computes the routing table entries at all nodes. The neighbor update messages should be processed serially, i.e., process the messages as per the order, A, B, and C.*** Once the routing tables converge, the function "printRT" is used to print routing tables of each node.
3. The function 'routingAlgo' must run multiple iterations of the Bellman-Ford update algorithm at every node in the list of nodes, in order to compute and update routing table entries. 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').
4. 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. The sendMsg function is already implemented in node.h; ***you must complete the recvMsg function in order to process a routing table update from a neighbor at a given node***.
5. The recvMsg function will run the Bellman-Ford update algorithm of the distance vector routing protocol, and update its routing table as required. 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, you will need to fill in the two functions: routingAlgo and recvMsg in the file routing_algo.cpp. The function routingAlgo is invoked by main, and should run the distance vector routing protocol at all nodes in the network. This function should cause 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.

If you run the given version of the code, it will print only the default routing table entries at each node that point to itself. For example, your output for the given sampleinput.txt may initially look like this.

```
A:
10.0.0.1 | 10.0.0.1 | 10.0.0.1 | 0
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
C:
10.0.1.3 | 10.0.1.3 | 10.0.1.3 | 0
```

If you implement the routing protocol correctly, the correct output from your program when run against the above sample input should be as follows.

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 autograding at our end.

Problem Statement

Q.1. Complete the code [5+5]

Q.2. Modify the code implemented for Q.1. as follows. [5]

A. After the routing tables converge, change the routing table entry of node B and C such that link B->C and C->B reaches the maximum hop count limit, i.e.16.
    1. You should implement this change within the 'routingAlgo' function itself. You will have to add a new function to class 'node' in order to update the routing table entry. Name it 'updateTblEntry()'. Propagate this change (using DVR protocol).
    2. As mentioned earlier, ensure that the neighbor update messages are processed serially within the 'routingAlgo' function.
B. Modify the code such that each node prints (to the standard output) the updated routing table entries, at the end of each iteration. You should not modify the print function implemented for the solution to Q.1.
C. Can you spot any problem?

Q.3. Implement poisoned-reverse solution. Use the code implemented in Q.2. and modify the link costs from B->C and C->B as mentioned in Q.2. You will have to change node.h file, sendMsg, and recvMsg functions. You should understand the 'RoutingEntry' class, and modify it to implement this solution. [5]