Link State Routing

Due May 4 by 11:59pm **Points** 100 **Submitting** a file upload **Available** until May 4 at 11:59pm

This assignment was locked May 4 at 11:59pm.

The goal of this assignment is to create a virtual network of routers that send routing data to one another. This assignment models the way that actual dynamic network routing is performed.

Routers should be represented by an appropriately designed class or data type. Each router should maintain a data structure that stores references to other "directly" connected routers, which can be referenced by id, along with the cost of the link. These routers will exchange information necessary to build a routing table. Each router will be advertising access to a particular named network, which is a string that should be stored in the router class. You will need to store other information in the router class to enable the remainder of the algorithm to work, such as an undirected graph representing the network of routers as that router currently understands it.

You will be implementing a "link state routing" algorithm. Link state routing algorithms work on real networks by sending a "link state packet" around the network, with routers forwarding the packet as appropriate. We will be simulating this packet with instances of another class, that will contain the following information:

- ID of the router that originates the LSP.
- A sequence number indicating how many LSPs have been sent by the originating router; i.e., each newly originated LSP has a higher sequence number than all previous LSPs. The first sequence number should be 1; ignore the possibility of wraparound.
- Time to live, decremented each time the LSP is forwarded; with an initial value of 10.
- Either of the following at your option:
  - A list that indicates each reachable network (indicated by the network name stored in the router's string).
  - A list that indicates each directly connected router, the network behind each one, and the cost to get to that router.

The router class should contain a function/method named something similar to receivePacket that should take as its only arguments - (1) an instance of the link state packet class, (2) ID of the router that forwarded the LSP. A router that receives an LSP should first decrement the LSP's TTL. Next, the receiving router should discard the LSP and not use its information if either (1) the TTL has reached zero, or (2) the receiving router has already seen an LSP from the same originating router with a sequence number higher than or equal to the sequence number in the received LSP. If the LSP is not discarded, its information should be compared to the receiving router’s connectivity graph and the LSP should be sent out (in the form of function calls) to all directly connected routers except the one on which it was received.

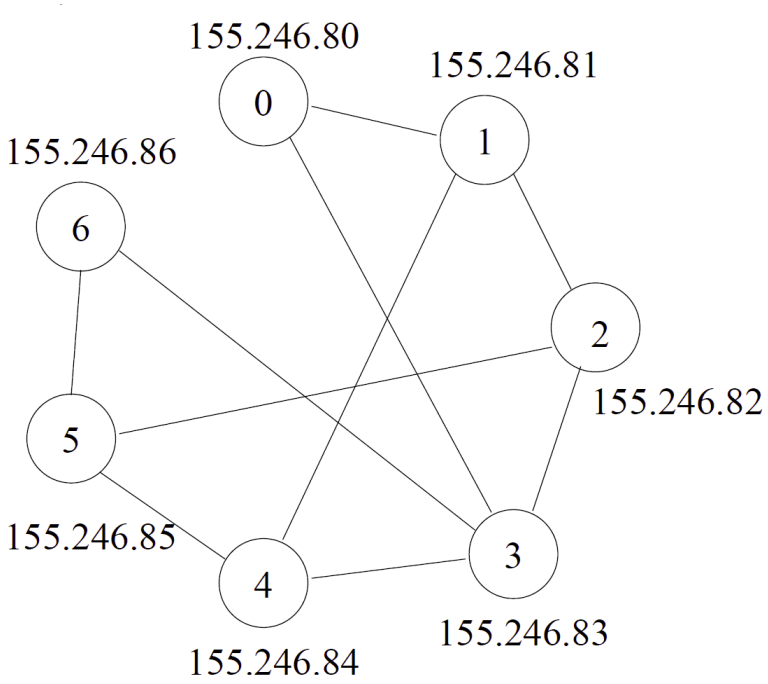
The router class should also contain a function named something similar to originatePacket that takes no arguments. This function will perform two functions. It should cause the router to generate an LSP packet based on the current state of the network as it understands it, and send it to all directly connected routers. Before it sends the packet, however, it should also increment a "tick" counter and consider if there are any directly connected routers from which it has not received a packet in 2 ticks. If that occurs, the router should alter its graph to reflect that the cost of the link to the other router is now infinity (or some arbitrarily huge number that you choose to represent infinity, if your language does not have a special infinity value.).

Each router should maintain a "routing table" consisting of <network, outgoing link, cost> triples. If connectivity has changed (as indicated by a received link state packet), the receiving router should run Dijkstra’s all-pairs shortest-path algorithm on its updated connectivity graph to re-compute its routing table of *<network, outgoing link, cost>* triples. *Outgoing link* means the directly connected router to which this router would send data in order to route the data using the shortest path. It does not mean the destination router. (Consider the advantages and disadvantages of implementing the graph using an adjacency matrix or using an adjacency list.)

The initial network will be set up by reading in a file, infile.dat, that has the following format:

```
[router-id] [network-name]
 [directly-linked-router-id] [link-cost]
 [directly-linked-router-id] [link-cost]
 [directly-linked-router-id] [link-cost]
 etc...
[router-id] [network-name]
 [directly-linked-router-id] [link-cost]
 [directly-linked-router-id] [link-cost]
 [directly-linked-router-id] [link-cost]
 etc...
etc...
```

Each [bracketed] text above represents the name of a bit of data, and will not appear with brackets around it in the file. "etc..." represents that the pattern above it will be repeated. link-cost is optional and should be assumed to be 1 if missing
An example network, and picture representing that network, follows:




```
0 155.246.80
 1
 3
1 155.246.81
 0
 2
 4
2 155.246.82
 1
 3
 5
3 155.246.83
 0
 2
 4
 6
4 155.246.84
 1
 3
 5
5 155.246.85
 2
 4
 6
6 155.246.86
 3
 5
```

Note, there is no guarantee that the router id numbers will be sequential as they were in the above example. There could be, for example, a two-router setup with id *123* and *84719*.

Write a main function that proceeds as follows:

- At the beginning of the program, create routers and initialize their direct links by reading in infile.dat.
- Prompt the user if they would like to continue (enter "C"), quit (enter "Q"), print the routing table of a router ("P" followed by the router's id number), shut down a router (enter "S" followed by the id number), or start up a router (enter "T" followed by the id).
- If the user chooses to continue, you should call the originatePacket() function on every router in whatever order you choose. Then prompt again.
- If the user shuts down a router, change the router object so that it does not send out any LSP or do anything in response to originatePacket or receivePacket function calls.
- If the user starts up a router, change the router object so it once again behaves normally. (Routers initially are started, not stopped.)
- If the user chooses to print the routing table, display the table in the following format, and then prompt again:

```
network, outgoing link
network, outgoing link
```

```
network, outgoing link
network, outgoing link
etc...
```

- If the user chooses to quit, exit the program.

In any of the above file formats, all non-newline whitespace may be represented by one or more tab or space characters.

This final project cannot be extended and no late submissions are permitted for any reason at any time without first filing a petition for incomplete in the course. *This rule will be followed strictly*, and submissions will be blocked by Canvas at the due date. If you want to avoid potential technology-related issues such as internet connectivity or your clock and Canvas's clock not being in sync, please plan on submitting the assignment early.

Note: You must not submit your "node_modules" folder if you are working on NodeJs/JavaScript. (Just submit your JavaScript source code and package.json file)