CptS 223 PA #4

In this assignment, you will build a system that simulates a simple network. In this network, it is possible to send a text message between any two nodes (vertices in graph speak). In order to simplify this task, we will make the network and all nodes omniscient, meaning that we always have perfect knowledge of the network's state. In real networking, this is often not the case. Typically, a given node knows only about a very small subsection of any given network. This can make routing between nodes less exact, but taking a more realistic approach would be too ambitious for CptS 223.

Creating the Network

Before you can transmit messages between nodes on your network, you must first build the network from a text file. As was the case with PA #3, your program should accept the name of the network file from the command line (PA4.exe network1.txt) **or**, when a parameter is not provided (PA4.exe), from the user directly.

The network text is broken up into two segments. The first segment defines the nodes in the network (one per line). The second segment contains the edges present in the graph in the following format <source> <dest> <weight>. Consider the following example file:

```
1
2
3
1 2 5
1 3 10
2 3 4
```

This file defines three nodes (1, 2, 3) and three links (edges) between the nodes:

- Node 1 is connected to Node 2. The weight of this link is 5.
- Node 1 is also connected to Node 3. The weight of this link is 10.
- Node 2 is only connected to Node 3. The weight of this link is 4.

In the context of networking, we can conceptualize the weight of a given link to represent the latency of communication between any given two nodes. For example, sending a message to a computer in Europe would obviously take longer than sending a message to a computer in Seattle.

Routing a Message

For this assignment, we will be sending strings between network nodes. As with real networking, messages are often broken down into smaller chunks. These chunks are sometimes called packets. In our example, a packet will consist of a single character within the original message. For example, a message of "Hello" would be broken down into 5 packets to be transmitted. Note that because packets are not guaranteed to arrive in order, you must also keep track of the original ordering of characters. In the real world, packets also contain information to mark the beginning and end of a message. For the purposes of this assignment, we will ignore that requirement. For more information on how I wrote my message class, see the section on UML diagrams.

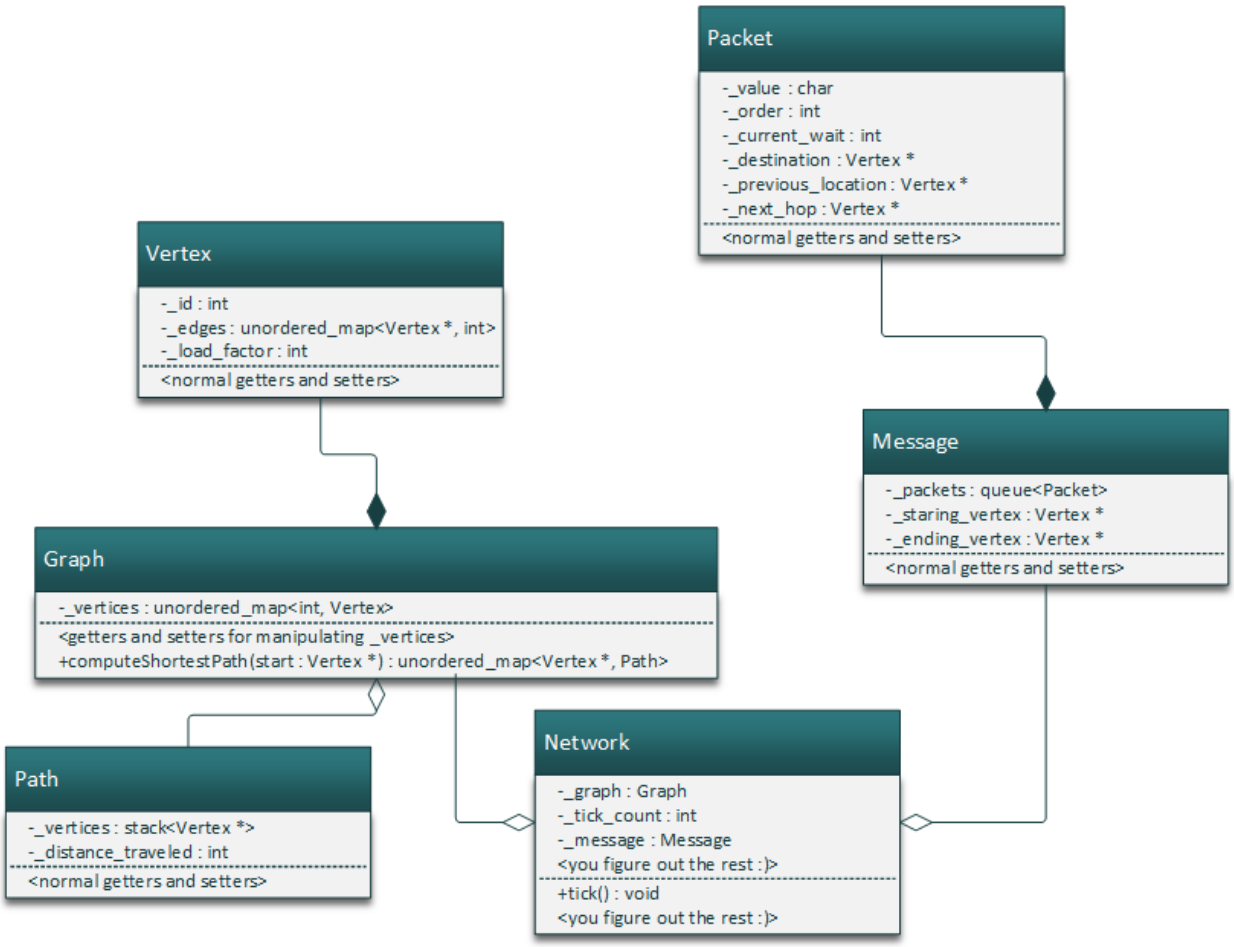
After obtaining a valid source node, destination node, and message from the user, begin transmission of the message. The steps for this loop are as follows:

1. If the message still has packets that need to be transmitted, queue the next packet for transmission at the starting location:
    a. Compute the shortest route between the starting location and the packet's final destination using Dijkstra's Algorithm (See note on Dijkstra's Algorithm).
    b. Determine the next intermediary transmission node. On a Path = v1, v2, v3, ... , vn where v1 = starting location, the next intermediary node would be v2.
    c. The arrival time of the packet at v2 is determined by the natural, unmodified weight of the link between v1 and v2 and the load factor of v2. Thus wait = weight * load(v2). Queue the packet's arrival at the appropriate time in the future. Note that a packet whose transit time is 1 should arrive at the destination node during the following loop. In other words, it should be delivered during step #2.
    d. Because v1 and v2 are communicating, increase the load factor of each node by 1.
2. For each packet presently in the network:
    a. Decrement the packet's expected arrival time
    b. If the packet's arrival time is less than or equal to 0, the packet is considered to have arrived at its next destination.
        i. Decrease the load factor of both the source and destination nodes by 1
        ii. If the packet has not reached its final destination, schedule another transmission using the method described in #1a - 1d. Be sure to alter the nodes that are transmitting the packet. For example, if the packet just arrived at v2 from v1, the next transmission would occur between v2 and v3.
        iii. If the packet has reached its final destination, put the packet in a list of completed packets (this will be used to display the final transmission results)

A note on Dijkstra's Algorithm

We are using a node's "load factor" to indicate the speed at which a given node operates. As a node becomes more saturated with network traffic, it likely will take the node longer to respond to any given message. To model this behavior, calculate an edge's weight to be the product of the edge's unadjusted weight and the node's load factor (weight = vertex.getWeight() * vertext.getLoadFactor()).

UML Diagrams



Above are the classes that I used to solve the problem. This configuration is only a suggestion; you are free to make up your own classes and relationships. In my program, I use the Vertex class to represent a given node in the network. Graph contains a collection of Vertices and also implement's Dijkstra's algorithm (method computeShortestPath). Lastly, I use the Network class to handle the running of the simulation. As such, the Network class actually tracks a given packet's progress through Graph. Note that there are probably much more efficient and/or elegant ways to implement the network simulation.

Due Date

This assignment is due by midnight on Monday, December 7, 2015.

Grading

You are allowed to complete this assignment in teams. A team consists of at most two students (including you). Students on teams will receive the same grade. Your grade will be determined as follows:

- [60 – 15 each test case] Your program's outputs match the expected output generated for a given test case.
- [20] Your program uses Dijkstra's algorithm to compute the shortest path
- [10] Does the code conform to good coding practices of Objected Oriented Programming principles or is code haphazardly slapped together in main()?
- [10] Is the code documented well and generally easy to read (with helpful comments and pointers/tips for the TA)?