Assignment 6 Requirements Document

Description

For this program, you will create a program that implements Dijkstra's algorithm.

Basics

You will be given the files test6.cpp, test6.h, graph.cpp, and graph.h. These will contain all of the methods that can be called by the test engine. You are allowed to implement more methods and functions if you wish, and you may create any other files that are required to implement Dijkstra's algorithm.

Program Requirements

1. The following methods must be implemented for class Graph:
    a. Graph() - Constructor for Graph. When finished, the graph shall be an empty directed graph.
    b. ~Graph() - Destructor for Graph.
    c. LoadGraph(*fileName*) - Loads the graph found in the file *fileName*. If the graph is invalid, it shall not be loaded. Edges where the source and destination vertices are the same shall be ignored. LoadGraph will only be called once for a given graph object. The distance of all vertices shall be initialized to INFINITY.
    d. RunDijkstra(*startVertex*) - Runs Dijkstra's algorithm, using the vertex named *startVertex* as the starting vertex. After the algorithm has been run, the distances shall be stored in the vertices until the next call of RunDijkstra. If a vertex is not reachable, it shall have a distance of INFINITY.
    e. GetDistance(*vertex*) - Returns the distance for the vertex with name *vertex*. If there is no such vertex, returns -1.
    f. GetNumberOfUpdates(*vertex*) - Returns the number of times that the distance of the vertex with the name *vertex* was reduced the last time that RunDijkstra was called. If there is no such vertex, returns -1. Note that the initialization of distances does not count, so a non-starting vertex with a distance of INFINITY will have 0 updates.

2. The function GetINFINITY returns the value of INFINITY. It is guaranteed to be a value that never matches a distance of any vertex in testing, so you may compare if a distance is equal to GetINFINITY() to see if it has a distance that has been set or not. This function is defined in test6.cpp (and may be changed during testing), so do not define it in your code outside of test6.cpp. You are free to change GetINFINITY in your copy of test6.cpp for unit testing.

Graph Requirements

A valid graph shall meet the following requirements:

1. A valid graph must have at least one valid edge.
2. Valid vertices have names consisting of a single character. That character may be an uppercase letter, a lowercase letter, or a digit.
3. Valid edges have a source vertex and a destination vertex which are both valid. In addition, the vertices are not the same.
4. Weights of valid edges shall be nonnegative.

Graph Format

The format of the file for LoadGraph shall be as follows:

[Vertices]
vertex1
vertex2
…
vertex*m*

[Edges]
vertex1src vertex1dest weight1
vertex2src vertex2dest weight2
…
vertex*n*src vertex*n*dest weight*n*

While the file may not exist, if the file exists and is not blank, then it is in the correct format Although the values themselves are not guaranteed to be valid, the weights are guaranteed to be integer values (so the library function atoi will work). However, there may be blank lines throughout the file (all of which should be ignored).

Submission Information

There are two submissions on Gradescope for this assignment. The first is the practice submission. I recommend that you submit to that one first because it will test to make sure that your program compiles and that every function can be called correctly. The second is the final submission. This compiles the program and tests it. Your grade is based on the results from the final submission.

Deliverables

For each submission, you will need to submit all files except test6.cpp. While you may include test6.cpp in your submission, the test engine will use an altered copy of that file to test your code. All files required to implement your solution must be turned in by Sunday, April 23, at 11:59 P.M.

Other Notes/Reminders

Only use the standard C/C++ libraries; do not use others. Do not share your code with others or post your code publicly, as this constitutes an Academic Integrity violation. Do not print anything, and do not read from or write to files that aren't in the specifications, as doing these things may cause your program to fail cases.