Lab 10: Shortest path problem

1. Introduction

In the lecture last week we simulated Dijkstra's algorithm on a sheet of paper. The goal of this lab is to translate the procedure into valid C++ code. More specifically, you will implement Dijkstra's algorithm as a part of your Graph class, so each instance of your Graph class holds information of shortest paths (once they are calculated) along with its structural information.

The code skeleton is under: **/comp/15/files/l10**

1. Log in to the CS homework server.
2. Move to your **comp15** directory that you created in Lab 1.
3. Create a directory named **lab10** under the comp15 directory.
4. Move to the lab directory.
5. Copy the code skeleton to the current working directory.
6. Leverage the features of Git to manage your progress toward writing the program.

The code skeleton provides you with seven source files: **test.cpp**, **Graph.hpp**, **Graph.cpp**, **Vertex.hpp**, **Vertex.cpp**, **Pair.hpp**, **MinHeap.hpp** as well as two object (relocatable) files: **Pair.o**, **MinHeap.o**

- Note that the copy constructor and assignment operator of the Graph class and of the Vertex class are commented out, so that this lab can be completed within our lab time. However, you are encouraged to work on them later on.
- **test.cpp** provides you with three tests with assert() that your implementation is supposed to pass.
- The Pair class has already been implemented and pre-compiled for you. Please use the Pair class to represent outgoing edges.
  - The "first" of an instance of the Pair class is a pointer to the destination vertex and the "second" is the weight of *this* edge.
  - In the destructor of the Pair class, the vertex pointed by the "first" will not be deleted.
- The MinHeap class has already been implemented and pre-compiled for you. You may find it useful when you need a min priority queue in your implementation.
  - The default constructor, copy constructor, assignment operator and insert() method has not been implemented. (They will throw an exception.)
  - The user-defined constructor takes a vector of pointers to vertices, and then constructs itself as a min heap of the given pointers to vertices, in which the distance of each vertex is used as the key to maintain its min-heap property.

- The rebuild() method can be used to maintain the min-heap property of *this* min-heap for cases where the distance information of any vertex in *this* min-heap is updated.

- **Important note**: Because of the design choices made in the MinHeap class (e.g. the insert() method is disabled), let's assume two things in this lab:
  - Every single vertex in a graph is reachable from the start vertex.
  - The ID of the start vertex is 0.

2. Requirements

1. Implement the Vertex class.
   a. You can use the vector, which is one of C++ STL, to hold its outgoing edges.
   b. The default constructor: You will define what should happen when the default constructor is used.
   c. The addOutgoingEdge() and getOutgoingEdgeAt(): You will define what should happen when the given information is not appropriate.
2. Implement the Graph class.
   a. You can use the vector, which is one of C++ STL, to hold all vertices and edges.
   b. The addEdge(), setShortestPathsFrom(), getShortestPath() methods: If the given information is not appropriate, throw a `std::runtime_error` with the message (case-sensitive): **Invalid**
   c. The getShortestPath() method: If the given "fromID" is different from the "startID" that was specified when the setShortestPathsFrom() was called, throw a `std::runtime_error` with the message (case-sensitive): **Invalid**
   d. You may use `INT_MAX` as an alternative to the infinity when initialising the distance of each vertex. (You will need to include the `climits` header.)
   e. The format of getShortestPath(): Please find the expected output in **test.cpp**.
3. Pass the tests with assert() written in **test.cpp** with no memory leaks and no memory errors.

3. README

Create the README file that includes the following categories with appropriate section headers.

1. **Name**: Your name.
2. **Date**: The last updated date.
3. **Summary**: A brief summary of the lab.
4. **Files**: A list of files that are necessary to build and test the program.
5. **Instructions**: A sequence of commands to compile and test the program. Note that you are expected to report procedures without using the make command.
6. **References**: A list of citations to information used to complete the lab.

4. Submission

Submit your files listed below using Gradescope.
Files: **test.cpp Graph.hpp Graph.cpp Vertex.hpp Vertex.cpp README**