ECE250: Lab Project 5

Due Date: Friday, April 17th 2020, 11:00 pm

1. Project Description

The goal of this project is to write a C++ implementation to find shortest path in a graph using **Dijktsra's algorithm**. Using your implementation, a user can find the shortest path between two given cities using an undirected graph.

We ask you to write a C++ class **undirectedGraph**, representing cities (as nodes) and roads (as edges) connecting these cities. This is a weighted graph in which each edge represents the corresponding distance (as a double data type) between two cities. You have to write your own implementation of the undirected graph. It is not allowed to use the C++ Standard Library, *except vectors*, to implement your data structures.

2. Program Design

Write a short description of your design. You will submit this document along with your C++ solution files for marking. This document must include your design decisions. Please refer to the course website for "Programming Guidelines" and the expected content for your design document.

3. Project Requirements

Write a test program (named **undirectedGraphtest.cpp**) that reads commands from standard input and writes the output to standard output. The program will respond to the commands described in this section.

Command	**Parameters**	**Description**	**Output**
i	*name*	Inserts a node (city) to a graph.	**success:** if the insertion command was successful **failure:** if the insertion command was unable to complete since the city was already in the graph
setd	*name1;name2;d*	Assigns a distance (*d*) to the edge (road) connecting two cities (*name1* and *name2).* **NOTE:** If this distance between *name1* and *name2* has already been set, updates the distance to *d*.	**success:** if the command was able to assign the distance to the connection or to update an existing distance between two cities successfully **failure:** if the command was unable to complete because one or both cities do not exist, or *d* is invalid (<=0), or both nodes are identical (*name1=name2)*
s	*name*	Searches for a city with the specified *name.*	**found** *name* **not found**

Command	**Parameters**	**Description**	**Output**
degree	*name*	Prints the degree of the city (*name).*	**degree of** *name value* **failure:** if the city (*name)* is not found
graph_nodes		Returns the number of nodes (cities) in the graph.	**number of nodes** *value*
graph_edges		Returns the number of edges (roads) in the graph.	**number of edges** *value*
d	*name1;name2*	Prints the distance between two cities (*name1* and *name2)* along the edge directly connecting them. **Note:** Two cities (*name1* and *name2*) must be adjacent to have a valid direct distance.	**direct distance** *name1* **to** *name2 value* **failure:** if one or both nodes (cities) are not found, or two nodes are not directly connected, or both nodes are identical (*name1=name2)*
shortest_d	*name1;name2*	Finds the shortest distance between two cities (*name1* and *name2).*	**shortest distance** *name1* **to** *name2 value* **failure:** if one or both cities are not found, or cities are not reachable from each other, or both nodes are identical (*name1=name2)*
print_path	*name1;name2*	Prints the path between two cities (*name1* and *name2*) such that the sum of the distances of its constituent edges (roads) is minimized (shortest path).	*name1 … name2* **failure:** if one or both cities are not found, or two cities are not reachable from each other, or both nodes are identical (*name1=name2)*
clear		Deletes all the nodes and edges from the graph.	**success**

Provide an analysis for the **time complexity** (average case, best case, and worst case) of your implementation of Dijkstra's algorithm.

- **Test Files**

The course website contains example input files for the corresponding output files. The files are named *test01.in*, *test02.in* and so on with the output files named *test01.out, test02.out* and so on.

4. How to Submit Your Program

Once you have completed your solution and tested it comprehensively in your computer or on the lab computers, you have to transfer your files to the *eceUbuntu server* and test there since we perform the automated testing using this environment. Once you finish testing in the *eceUbuntu server*, you will create a compressed file (tar.gz) that should contain:

- A typed document (maximum three pages) describing your design. A document beyond 3 pages will not be marked. Submit this document in PDF format. The name of this file should be:

  xxxxxxxx _design_ p***n***.pdf in which ***xxxxxxxx*** is your UW user id (*e.g*., jsmith) and ***n*** is the project number that is 5 (five) for this submission.

- A test program (**undirectedGraphtest.cpp)** that reads the commands and writes the output.
- Required header files and classes (ending in *.h .hpp .cpp*).
- A make file (named Makefile), with instructions on how to compile your solution and create an executable file named **undirectedGraphdriver**

The name of your compressed file should be ***xxxxxxxx***_p***n***.tar.gz, where ***xxxxxxxx*** is your UW user id (e.g., jsmith) and ***n*** is the project number that is 5 (five) for this submission.