ECE250: Lab Project 3

Due Date: Friday, November 27, 2020, 11:00 pm

1. Project Description

The goal of this project is to write a C++ implementation for the Minimum Spanning Tree (MST) of a weighted undirected graph, using Kruskal's algorithm. We consider the *n* nodes in the graph to be numbered from *0* to *n – 1*. This means a graph with 4 nodes has nodes named *0*, *1*, *2* and *3*. Each edge has a weight (a positive number of double type) associated with it. You can represent the graph as an adjacency matrix or an adjacency list.

In order to build the minimum spanning tree T, the Kruskal's algorithm adds one edge to the T (initialized with an empty graph) in each step. To make sure that this procedure does not form loops in the MST, we need to keep track of the connected components of T. In your implementation, **you must write a *Disjoint sets* class** to assist you with this task. *Disjoint sets* is a well-known data structure for grouping *n* elements (nodes) into a collection of disjoint sets (connected components). You can read more information on disjoint sets from Chapter 21 of CLRS book and the Lecture#27. We recommend that you write the *disjoint sets* class using linked lists.

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 follow the *Design Document Template* given on LEARN, under "Contents | Course Materials| Tutorials | Tutorial #1 | Design Document Template" when writing your document.

3. Project Requirements

Write a test program (named **kruskaltest.cpp**) which will read commands from standard input and write the output to standard output. The program will respond to the commands described in this section.

For this project, you are required to handle some error conditions as exceptions (indicated in the table bellow as "**invalid argument**"). That is, the tree function will "throw" an exception called "illegal_argument" and the driver will "catch" the exception.

Command	**Parameters**	**Description**	**Output**
n	*m*	*m* is the number of nodes. ie. if m = 5, the graph has nodes 0,1,2,3 and 4. Note this this will always be the first command inputted, and will never be called subsequently.	**success** **invalid argument** if m < 0.

Command	**Parameters**	**Description**	**Output**
i	*u;v;w*	Inserts an edge between nodes u and v with weight w (a double type).	**success** if the insertion was successful. If there is already an edge connecting u and v, this command will update the weight for the edge. **invalid argument** if u or v are outside the valid range, or w <= 0.
d	*u;v*	Deletes the edge between nodes u and v.	**success** if the deletion was successful. **failure** if there is no edge connecting u and v. **invalid argument** if u or v are outside the valid range.
degree	*u*	Returns the degree of vertex u.	**degree of u is** d_u if node u is a valid node. **invalid argument** if u is outside the valid range.
edge_count		Returns the total number of edges in the graph.	**edge count is** n_edges For an empty graph, the edge count is 0.
clear		Removes all the edges from the graph.	**success**
mst		Calculates the minimum spanning tree and returns its total weight.	**mst** weight **not connected** if the graph is not connected.
exit		Last command of all input files	This command does not print any output.

The **time complexity** of Kruskal's algorithm depends on the implementation of the disjoint-set data structure. Implement **mst** with the best possible running time based on your disjoint-set data structure. You can refer to Sections 21 and 23.2 in CLRS and the Lecture#27 for details on the analysis for the running time that you will need to add to your design document.

- **Test Files**

When you create input files, be sure to follow the example files. Note that each line ends with a UNIX end of line character (std::endl), and that there are no blank spaces before this character. Each input file ends with the line “exit” – followed by std::endl.

Text files created on DOS/Windows machines have different line endings than files created on Unix/Linux.

- **Checking for memory leaks**

When testing your program you need to ensure that all memory allocated has been freed adequately. You can use the **valgrind** utility available in the ECE Unix server (eceUbuntu) to ensure that your program is handling this correctly.

In this project, you can use the following command in eceUbuntu, to identify memory leaks in your executable named `kruskaldriver`:

```
valgrind --leak-check=yes ./kruskaldriver < t01/test.in
```

Memory leaks are usually associated with deficiencies in the destructor function of a class.

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 3 (three) for this submission.

- A test program (**kruskaltest.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 **kruskaldriver**

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 3 (three) for this submission.