## Program Description

The goal of this assignment was to implement a Graph ADT. The first part of the assignment focused on on inserting edges, getting the weight between two vertices, and building the Graph. The second half of the assignment focused on building a minimum spanning tree from a regular tree.

## Purpose of the Assignment

The purpose of this assignment was to learn how to implement a graph data structure and how to find the minimum spanning tree of a graph. In class we broadly talked about how to the data structure and algorithm work. The purpose of this assignment was to implement what we learned in class into C++.

## Data Structures Description

The graph itself is a data structure, but the implementation in this lab requires several other data structures. Vectors were used as locators for vertices and edges and provided direct access to certain elements as opposed to iterating through an entire list to find data. A disjoint set was used to implement Kruskal's Minimum Spanning  Tree algorithm. DListNodes were also used in the insertEdge function.

Graphs are very useful for visualizing data and the minimum spanning tree is useful for finding the shortest path to connect all data points or nodes.

## Function and Algorithm Description

The buildGraph() function works by creating n slots in a vector to store vertices, where n is the number of vertices. O(n)

The insertEdge() function works by creating a new entry in the EdgeList between the two specified vertices. The adjacency list for each vertex is then updated to account for this new entry. Finally the number of total edges counter is updated. O(n)

The getWeight() function works by iterating through the entire EdgeList until the specific edge is found, it will then return the weight of that edge. O(n)

The sortEdge() function works by making a copy of the EdgeList called copy_EdgeList. The program the iterates through this list and finds the smallest weight. That edge is then pushed into another vector called sorted_list. The edge is then removed from copy_EdgeList. This process repeats until copy_EdgeList is empty. At this point sorted_list will have all the edges in order of smallest weight to largest weight. The EdgeList is then updated to be the same as sorted_list. $O(n^2)$

The MSTAlgo() function first sorts the EdgeList from smallest to largest weights using the sortEdge() function. Then each vertex is assigned its own Disjoint set. The program then iterates through the sorted EdgeList. If two vertices aren't in the same disjoint set, their sets are unioned and the MST EdgeList is updated to account for an edge between these two vertices. If two vertices are already in the same set, then the algorithm skips over that specific edge. After the EdgeList has been exhausted. The program iterates through the MST edge list and computes the total weight of all edges. This value is returned once the program has finished executing. O(n) (assuming sortEdge is constant)

### Program Organization and Description of Classes

The main class in this assignment was the Graph data structure. Functions were declared in Graph.h with function implementations placed in Graph.cpp. Inside Graph.h are some smaller classes that improve program organization, they are the Edge class and the Vertex class. The Edge class stores two integers representing vertex values, and one double value that stores the weight of the edge. The vertex class stores the value of the vertex and a pointer to an edge object. The Disjoint set implementation is saved in disjointset.h. The Graph class also has an exception class called GraphException.

### Compile Instructions

Navigate to source directory. Run “make”. Then run the command “./main test1.mat”
The program will then run.

### Input / Output Formatting

The program only accepts input in the form of a command line argument to an input file. The input file must be formatted precisely in order for the program to work. The first integer in the file specifies the number of vertices, the second integer specifies the number of edges. Then starting on the next line, the adjacency lists begin. Each adjacency list is terminated with a “-1”.

### Logical Exceptions

The program will throw an error if the number of vertices or edges is less than zero for a graph.
If the current number of edges in a graph (some how) falls below 1, an error will be thrown.
When finding a disjoint set, if the key value is less than zero the user will be presented with an error status.
If trying to union two disjoint sets, the user will be presented with an error status.
When making a set, if the key value is greater than the nodeLocator size, an error will be presented.
If the disjoint set is initialized with a number less than 1, an error status will be presented.

### C++ Object Oriented or Generic Programming Features

Some smaller parts of this assignment rely on templating, such as the disjoint set.

### **Tests** (using the given input file)

```
dhcp-10-202-147-183:A6_suppl_part2 kyle$ ./main test1.mat
The Adjacency Matrix of the Graph is:
    0    9    3    5
    9    0    0    2
    3    0    0    0
    5    2    0    0
The total value of the Minimum Spanning Tree is: 10
The Minimum Spanning Tree is:
Node  Node  Weight
  1     3      2
  0     2      3
  0     3      5
dhcp-10-202-147-183:A6_suppl_part2 kyle$
```