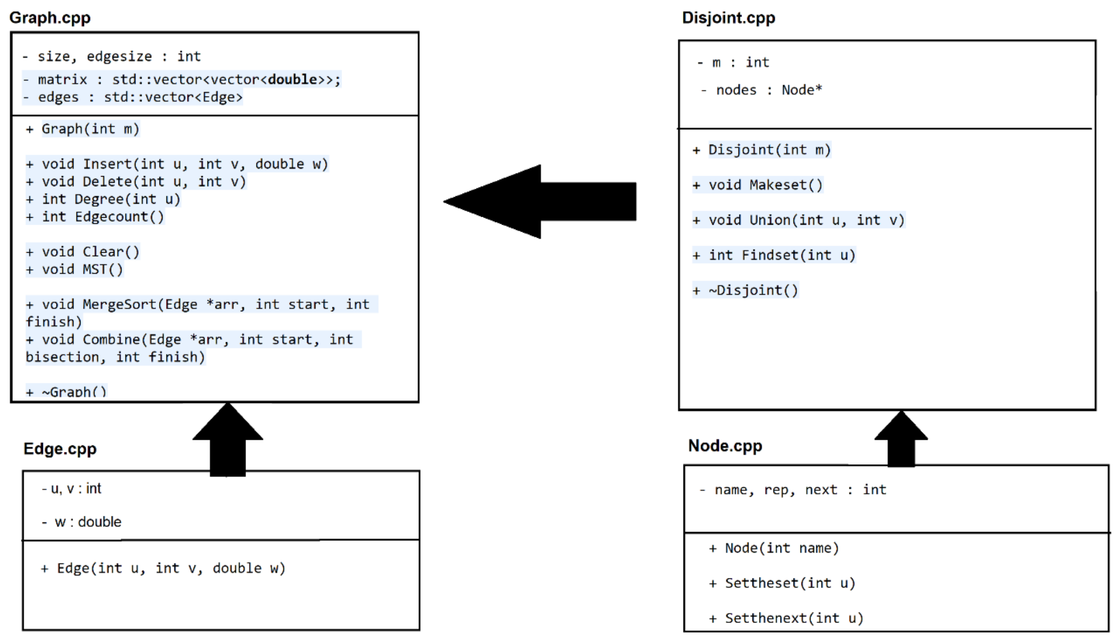
Matthew Panizza, 20822324

Overview of Classes

For this project, I used five main classes: “Graph.cpp”, “Disjoint.cpp”, “Node.cpp” “Edge.cpp”, and “InvalidArgument.cpp”. The Graph class contains project functions such as Insert() and MST(). The Disjoint contains its three operation and does not use STL container classes. Node and Edge are just used as objects that store properties. InvalidArgument is empty and used for try/catch. Explicit detail of functions in Graph and Disjoint and seen below under “Function Design and Performance”.

UML Diagram



Function Design and Performance

Graph.cpp

The Graph class is one of the required classes for this project and takes parameters: int m. This is the number of nodes in the graph. In the constructor, a class variable “int size” is set to m. Then, an class adjacency matrix is initialized with infinity values (vary large numbers). This class includes functions for all commands (i.e. delete, degree, mst). Graph did not need a destructor, as all allocated memory is freed/deleted.

void MST() – O(elog$_2$v) time complexity:

This is the function in the Graph class that performs Kruskal’s algorithm. I outline the steps and time complexity calculation below.

First of all, serval variables are defined, including an array of unsorted edges. This is O(1) constant time. Next, the array of unsorted edges is sorted. To ensure $e\log_2 v$, merge sort ($e\log_2 e$) is used.

After this set up, Makeset() from Disjoint is called. This is O(1) constant time (explained below). Then, a loop iterates through each edge in sorted order, then calls Union(int u, int v) and updates the weight if the two nodes of the current edge are not in the same set. This loop is ($e\log_2 v$) time. Next, a loop of size v checks every vertex to ensure they are all in the same set (v time).

Lastly, either "not connected" or the weight is printed in O(1) constant time.

Time Analysis: $O(1) + O(e\log_2 e) + O(1) + Oe\log_2 v + O(1) = O(e\log_2 ev + 3) = O(e\log_2 v)$

Disjoint.cpp

The Disjoint class is one of the required classes for this project and takes parameters: int m. This is the number of nodes in the graph when MST() is called. A pointer called "nodes" of type Nodes is defined as a class variable. In the constructor, a class variable "int size" is set to m. This class includes functions for all Disjoint operations (Makeset, Union, Findset). A destructor is included in this class to free the array of nodes from memory.

void Makeset() – O(v) time complexity

First, this function will allocate the appropriate amount of memory for the class variable nodes in O(1) time. Then, the function in Disjoint simply fills the array with nodes in O(v) time.

void Union(int u, int v) – O($\log_2 v$) time complexity

For any node, the worst possible amount of times it can be put into a new set is $\log_2 v$ times. This is because for any amount of nodes v, only $\log_2 v$ -1 sub graphs can be created before the graph is fully connected. Also, some variables are defined in O(1) time, which gives $O(1) + O(\log_2 v)$ which is $O(\log_2 v)$ time.

int Findset(int u) – O(1) time complexity

This function simply returns the current representative for node u in the nodes array. This operation takes place in O(1) time.

Matthew Panizza, 20822324

Test Cases

- To begin testing, I simply tested the given test cases on the eceUbuntu server and had successfully results where I also, making sure there were no memory leaks.

- Custom case #1: I made sure that duplicating works. Specifically, inserting an edge with the same nodes, multiple times. Also, I ensured that the clear command could be called twice in a row.

- Custom case #2: I made sure that the n command can be called multiple times with arrows both valid and invalid. For example, I tested calling n 1 then n 2 to make sure it updates. Then, I tested n 5, n -2, n 6 to ensure updating still works after there was an error.

- Custom Case #3: I made sure that double edges are accounted for properly. For example, if you enter i 2;2;5, the I ensured that node was added to the matrix and increased the nodes' degress by 2 instead of 1. I also tried that same command twice in a row to ensure the weight updates and does add onto.

- Custom Case #4: Accounting for an insert or delete command where u is larger than v in I u;v;w. I tested this case to allow for this scenario to be acceptable input.

- Then, I tested an extensive list of random test cases.

- Lastly, I made sure all line endings worked on the UNIX server and re-tested for and corrected all memory leaks in my program.