ECE 250 Project 4
Weighted Graph

1. Overview of Classes

I've implemented two classes for the weighted graph function.

Class graph

- The graph class consists of multiple functions that can be used to manipulate the weighted graph or calculate the MST and cost. Such functions include, adding or removing edges to and from the graph.

Member Variables:

1. adjList (vector<vector<tupel<int, int>>>) – stores the adjacency list for each vertex.
2. totalCost (int) – the cost (sum of the weights) via the MST.

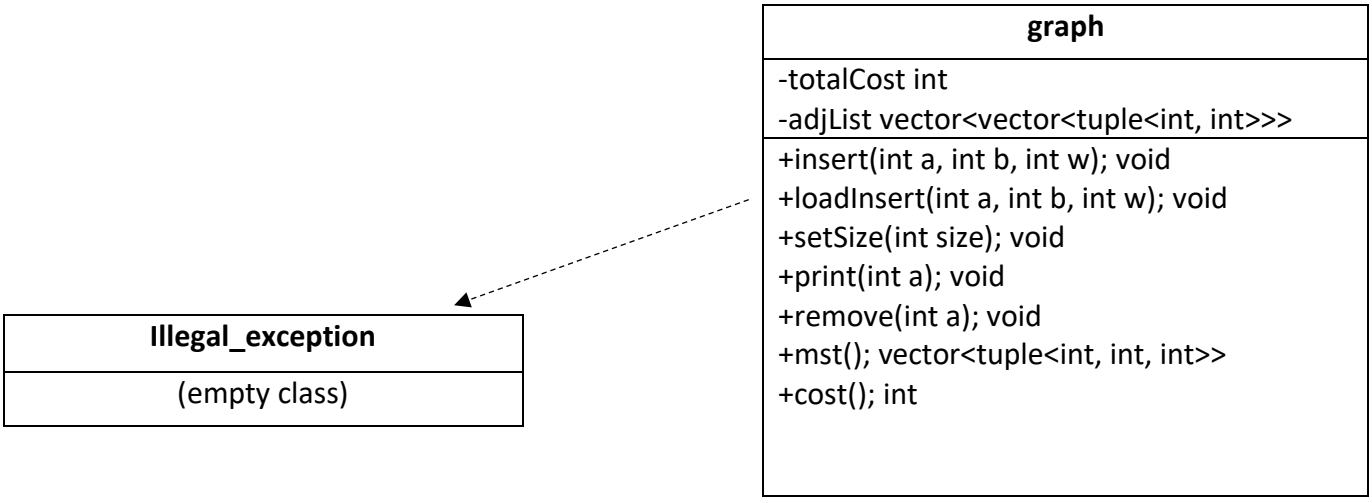
Member Functions:

1. insert – Inserts a new edge into the graph with a weight.
2. loadInsert – Add all the edges and weights from the dataset into the graph.
3. setSize – Set the size of adjList
4. print – Prints a list of all the adjacent vertices for a certain vertex.
5. remove – Removes a vertex from the graph along with all its edges. That vertex is then also removed from any adjacency lists that it was adjacent to.
6. mst – Finds the MST of the graph. Returns a vector<tuple<int, int, int>> the first two int's being the vertices and the last int being the weight.
7. cost – Returns the cost of the graph via the MST.

Class illegal_exception

- An empty class that is thrown when an error occurs with invalid input.

2. UML Class Design



3. Constructors/Destructors and Design Decisions

graph Class:

There is no constructor for this class as the graph gets created when edges are being added during runtime.

No operators were overloaded in my code, as there was no need. The only date type I dealt with were int's.

4. Runtime Implementation

For the insert function, the worse possible runtime is O(max(a, b)). This is the case since if the size of adjList is less than either a or b, adjList must be resized to either a or b, depending on which value is larger. However, on average, if adjList does not need to be resized, the runtime is O(1) for the push_back function, when adding to the adjacency list.

For the print function, it iterates through the adjacency list of vertex 'a' to print the vertices adjacent to vertex 'a', Thereform the run time depends on the size of the adjacency list for vertex 'a', giving a runtime of O(degree(a)).

The delete function contains 2 for loops for removing the vertex from the other vertices adjacency lists. The outer for loop iterates through all the vertices in the graph, having a run time of O(V), V being the number of vertices. For each vertex, I iterate through its adjacency list, to check if it contains vertex 'a'. In the worse-case scenario, all edges contain vertex 'a', thus giving the worse possible runtime of O(V * E).

For the mst function I used the Kruskal algorithm, with a runtime of O(E log (V)), E being the number of edges and V being the number of vertices in the graph. In the mst function, a vector<tuple<int, int, int>> variable called 'edges' is created with run time O(E) and populated with each edge and weight in the graph. After the edges are populated, the disjoint set is initialized and used to ensure that no edges with the same point are added to the mst. The initialization of the disjoint set is O(V). To compute the mst, I iterate through 'edges' and check each edge with the disjoint set to determine if the edge endpoints are already connected to the mst. If they are not connected, the edge is added to the mst, giving a runtime of O(E log (V)).

The runtime of the cost function is also O(E log (V)). This is because the cost is calculated at the same time the mst is being calculated. When an edge and weight is added to the mst, the weight of that edge is added to the total cost.