addVertex

The addVertex function takes in a string label and it first check if the label is empty and if it is it will throw an exception. Then we will set up the new vertex's distance, shortest path, and a Boolean function to check if we find it or not for later use. Then we push the new vertex in the double end queue.

Time complexity: O (1)

removeVertex

When we want to remove a vertex from the queue, we use a variable to mark the beginning of the queue and we traverse the entire queue until the end and once we find the vertex we erase it from the queue. When we remove the vertex, we also need to remove the edges and we use the same method of traversing the whole string and once we find the edges we remove it.

Time complexity: O (V + E), V is number of vertices and E is number of edges.

addEdge

The addedge function adds new edge between two given vertexes. It will setup the edges and appoint the weight to the edge. Then we push it in the edge's double ended queue.

Time complexity: O (1)

removeEdge

The removeEdge function finds use a temporary variable and it goes through the entire queue and once it find the edge with the given two labels it will remove it from the graph, leaving the vertexes alone.

Time complexity: O (E)

shortestPath

The shortestPath function uses four helper functions to help the find the shortest path between two vertexes in the graph.

1. It first uses a path start function to do the setup of the traversal with the vertex class. Once we find the desired vertex node, we set the minimum distance to 0 and we clear the path in case there are left over values, and we push "a" which is the value into the end. Then adds a new path to the container in the graphs class in Graph.hpp.

2. Then it uses the construct path function to makes the list in the priority queue in ascending order of nodes by their distances (using edge/weight). While the list of the paths is not empty, we set the second element from the top of the queue to be minimum element and we pop the first element and then we find minimum vertex index and its minimum distance as well as setting the boolean value to true.

3. Then it uses the lay path function finds the shortest distance in the priority queue from vector A to vector B. The function puts the nodes in the vector with minimum weight and get shortest distances from vector A to B. When our start variable is the end variable, we get the minimum

distance and update our shortest distance variable. We also need to put the least weight in the string and update the path variable with the minimum path.

4. Finally it uses the path reset function to restore all of the vertexes of the current graph to its default value.

In the end of the function, we return the shortest distance variable as an unsigned long number.

Time complexity: O (|E| + |V| log |V|)