Assignment 4
Name: Vhera Kaey Vijayaraj
Student ID: 28903013

Task 1

For this task, in order to find the quickest path from a source to a target, Djikstra's algorithm was used. A min heap is first contructed with the source node inside. Next, while the target vertex has not been visited, the vertex with the minimum distance will be at the root. This vertex is then popped from the heap. This has a time complexity of O(log V). If the node that has been popped has not been visited, the distance will update to the distance of that vertex. The vertex's edges will then be transversed.

Again, if the vertex in the edge list has already been visited, nothing happens. If it has been discovered already (means its already in the heap), if theres a current shorter distance, the distance of that vertex in the heap will be updated. It will be updated using a list of positions in the heap class. This is so that we can access the positions of the vertices in the heap with a complexity of O(1). The decrease_distance function in the minheap class has a complexity of O(log V) as it is updating the distance and then rising the nodes which has the least distance.

If the vertex has not been discovered (meaning it is not in the heap yet), the disttance will be the distance of the node before it added with the current distance. This is done repeatedly until the target node is visited. This means that we have got the shortest distance from the source to the target. In order to get the path of all the vertices which have the shortest distance, a predecessor value is stored in the vertex class. This value tells us the node before in which the node is connected to. The predecessor value is updated everytime if the path is changed depending on the edge.

To get the vertices, we start from the target node and keep taking the predecessor of all the nodes and appending it to a list until the current vertex is the source. If target vertex has not been visited (meaning there is no path from the source to the target), the function will return an empty list with a distance of negative one. Otherwise, the function will return a list with the path and the shortest distance.

The time complexity for this function is O(V log V) + O(E log V) which is equal to O(E log V) since we can assume that E > V. E is the total number of edges and V is the totoal number of vertices. The time complexity of O(V log V) comes from inserting the vertices into the min heap. O(E log V) is when we are relaxing all the edges as we are only visiting each edge once.

The space complexity for this function is O(E+V). E is the total number of edges and V is the total number of edges. This is due to the fact that we are storing every vertice in the graph. We are also storing every edge in a list in the vertex class as each vertice is connected to other vertices with edges.

Task 2

For task 2, the function augmentGraph changes the variable is_redlightcam and is_toll in the vertex and edge class if that vertex or edge is a camera or toll. Since we can't pass through edges which have a toll and go to vertices which are a camera, statements were added to Djikstra's algorithm from task 1. After popping the root from the min heap, it is checked to see if it is a red light camera. If it is (means we should never go to that node), it goes back to the beginning of the loop. There is no need to go through its edges since we can never visit that vertex in the first place.

Next, when the edges are being looped through, the edge is checked to see if it is a toll or not. If it is, the edge will not be considered. This is repeated until the target vertex is visited. The path will then be the shortest distance without going through any tolls or red light cameras.

The time complexity of this function is O(E log V). E is the total number of edges and V is the total number of vertices. The inserting of the items into the min heap is O(V log V) and when we relax each edge and since we are only visiting it once, the complexity of that is O(E log V). The total time complexity os O(E log V) + O(V Log V). Since we can assume that E > V, the time complexity will be O(E log V).

The space complexity for this function is O(E+V). E is the total number of edges and V is the total number of edges. This is due to the fact that we are storing every vertice in the graph. We are also storing every edge in a list in the vertex class as each vertice is connected to other vertices with edges.

Task 3

For this task, we are supposed to find the shortest detour path passing through atleast one of the service nodes. A new graph class is created so that I can use it as a class variable in the Graph class. When the buildGraph function is called, the reverse graph is also built. Same goes for the addService function.

Firstly, a reversed graph is built (a graph following the original graph, but all its edges are reversed). Djikstra is then run two times. First, it is run with the original graph. Then, it is run with the reversed graph. When it reaches a vertex which is a service node, the distance going forwards or backwards, depending on whether we are going from source to service node or service node to target is stored in the Vertex class as a variable called distance_forward and distance_backward. Next, once we have all the values stored in the service nodes, we access those service nodes by using the vertices list. We check if there exists a path for a node if both the distance_forward and distance_backward are valid distances (this would mean there is a path from the source to the service node and service node to the target).

The results are then appended to a list. The value with the minimum distance is retrieved from the list. Then, the nodes are used to find the path and the nodes used for going from the source to service node and from the service node to the source. The paths are then combined and returned together with the distance. If there is no valid path, the function returns [[],-1].

The time complexity for this function is O(2E log V) which = O(E log V), since 2 is a constant. E is the total number of edges and V is the total number of vertices. This is because, we are running Djikstra twice. Djikstra has a complexity of O(E log V). The inserting of the items into the min heap is O(V log V) and when we relax each edge and since we are only visiting it once, the complexity of that is O(E log V). The total time complexity os O(E log V) + O(V Log V). Since we can assume that E > V, the time complexity will be O(E log V) for Djikstra.

The space complexity for this function is O(E+V). E is the total number of edges and V is the total number of edges. This is due to the fact that we are storing every vertice in the graph. We are also storing every edge in a list in the vertex class as each vertice is connected to other vertices with edges.