Report

1. **Function Outline:** The first task is basically using the dijkstra algorithm to solve this problem. When constructing the discovered list, min heap was used so the item with the smallest distance can be gotten in O(1) time. Updating, inserting and removing are also convenient since min heap was used in the algorithm. After the target node was put into the finalized list, the algorithm would stop as there is no point to keep going since the target is finalized. Then another function would be called to recover the path of this finalized list. A variable was kept in the algorithm so it can state whether it is possible to reach the target from the source node.

Worst-case complexity: The worst time complexity is O(ElogV) since E is much smaller than ElogV so the recovering function is within this complexity. O(logV) is for inserting and removing in the min heap and E for generating these procedures E times since there are E edges. The worst case space complexity for this task is O(E+V) since the input graph would be O(E+V) and the finalized list and discovered list would take O(V) at most so it's still O(E+V) space.

2. **Function Outline:** For the second task, the basic idea is to modify the graph a little bit at first and then add conditions to the dijkstra algorithm to make it work. So after reading from the file and got all the needed properties, insert a condition to check if this vertex or this edge can be used or not. If not, just don't insert it into the list or using it as the information to update. Then just using the result same as the previous task to find the shortest path.

Worst-case complexity: The worst time complexity for this task is o(ElogV) which is the same as above. This is because this task also used the dijkstra algorithm and added a simple condition to check whether a vertex or an edge is usable or not. So the time complexity would stay the same and so does the space complexity. It would still be O(E+V) and is the same as the input graph

3. **Function Outline:** For the last task, the general idea is to add the service point into the graph first and then construct a reverse graph for calculation. So after the dijkstra of the normal graph, a finalized list that contains all the shortest path from the source to every node has been generated. This is when the dijkstra algorithm would be run for a second time on the reverse graph. Since the graph is reversed, a list contains the shortest paths from every node to the target is now available. So a new list that combines this 2 information can now be made and looping through this list while checking if it passes any service or not can give us the desired shortest path within only going through the dijkstra twice.

Worst-case complexity: The worst time complexity for this task is O(ElogV) since dijkstra only run 2 times and loops that takes O(V) time would also be run several times. But since O(E) is much smaller than O(ElogV), the overall time complexity would still be O(ElogV). The worst space complexity is still O(E+V) and the same as the size of the input graph. The processing would generate several new lists that take O(V) space at most so the worst space complexity would still be O(E+V).