Name: Kerry Zheng

Student ID: 28794346

FIT2004: Prac 12, Red-light Camera Detector

I create a list the size of the number of vertices and initialise all to be false and read in the vertices file. This creates a list that we can determine if a vertex is a red light camera or not in O(1) time. Setting True to the element that corresponds to the vertex number in each line of the file. Then I read in every edge in the edges list and keeping note if the edge is a toll road and create an adjacency list out of it. This takes O(V+E) time and space complexity.

To find the shortest path to every vertex from the source under the condition that the path cannot contain toll roads and a vertex is a red light unless it is the destination I implemented Dijkstra's algorithm using an array-based min-heap that I update to keep tracking of the discovered vertices. Firstly, I create an extra list the size of the number of vertices, called vertices list, so that I keep track of where each vertex is located in the heap, because it is array based, so I can update nodes in O(1) time. So I first add the source vertex into the min heap and then while the min heap is not empty, I get the vertex, v, with the smallest distance, which is located at the top of the heap, and remove it from the heap by swapping it with the last node, deleting the min, and sift down the node we just swapped into the root. I check if the vertex is a red-light camera or not. If it is a red-light camera, I instantly move it to finalised and add it to a list of finalised cameras. Otherwise, for each outgoing edge of the vertex, u, I first check if it's a toll road. If it is, I don't not consider that edge. Otherwise, if vertex u has not been discovered or finalised, I add it to the min heap and up heap it, and keeping track of the vertex v, so we can recover the path. Else if the current distance to vertex u is greater than the distance from vertex v plus the edge weight connecting v and u, then I update the value of edge distance to be that, and also the previous vertex to be equal to v, and upheap that node because updating the node may have violated the heap property. Because we can find the node in the min-heap in O(1) time the upheap has O(logV) time complexity. Then I move the vertex v to finalised. Whether or not vertex v is finalised or not, I record where each vertex is located in the finalised list in a list called position in finalised list of size V so that I can access the previous vertex when recovering the shortest path in O(1) time. In finding the shortest path to every reachable vertex under the given condition, it takes O(VlogV + ElogV) time

complexity and O(V) space complexity, because the while loop runs V times and every time in the while loop we get the minimum. Then to repair the heap property we swap the last node to the root and sink. This takes O(logV) time. The entire for loop runs E times, and each time the for loop is run we have to upheap the node we just added or updated. This takes O(logV) time, so the for loop has O(ElogV) time complexity, thus time complexity is O(VlogV + ElogV). It has O(V) space because our heap, the finalised list, the finalised camera list and the position in finalised list all have at most size V.

Every time two vertices are swapped in the min-heap, the indexes that correspond to the vertex number in the vertices list are also swapped, except when we are deleting a node. Then the node we swap to the root is set to 1 and the node we are deleting is set to –1 to indicate it is finalised.

To recover the shortest path to k closest cameras, for each camera up until k or the size of the finalised camera list, because we record the previous vertex to that node that is the shortest path, and we know exactly, that is O(1) time, where each vertex is located in the finalised list if its finalised given by the finalised position list, we add the previous vertex and then the previous vertex of the previous vertex's location in the finalised list is given by the finalised position list. This has time and space complexity O(P) where P is the total number of edges printed in the output.

Therefore, my program has O(VlogV + ElogV) time complexity and O(V + E) space complexity to determine the k closest cameras, and O(VlogV + ElogV + P) time complexity and O(V + E + P) space complexity.