About the data structure:

For this project, the following information needs to be stored in memory in a space-saving, and quick-accessing way.

1. Nodes and connections: the possible nodes will need to be stored, as well as the possible direct connection of every node. To achieve quick access, adjacency matrix is preferred. However, it would require too much memory: for the second input file, with 87575 vertices, matrix would require 87575^2 nodes. If every node is an int, that would require almost 16 GB of memory. Therefore, adjacency list is better in this situation.
2. Properties of every vertex in the list:
    a. Every vertex's position
    b. Distance from source toward that node
    c. The previous visited node
3. Optimization: Based on the size of the input map data, I prefer using an array of linked lists to represent the adjacency list. This would save a lot of time to access and find the nodes.

About the algorithm:

1. Initializing by writing infinity to all nodes' distance property; previous visited node is not defined Find the starting node, mark it with distance 0, and previous as -10, indicating it is the first node Create a set of unvisited nodes using a priority queue, according to the distances.
2. Find the starting node in the array, find all the nodes that can be directly reached from the first node, update these nodes' distances with their distances towards the first node. These nodes are the second set of nodes. Update the previous visited node of every node in the second set. Adjust the priority queue if necessary
3. Go and visit every node in the second set. Find the node that can be directly reached from the second set of nodes. Only update the distances of the third set of nodes if the new distance is lower than the current distance.
4. Check for halting condition:
    a. The destination has been visited
    b. The smallest distance among the nodes in the unvisited set is infinity
5. Applying 2nd and 3rd steps again and again, until the last node has been reached or cannot be reached.

Therefore, for this project, the nodes will be a defined data structure. Also, they will be used several times. First, create these data structure according to the input file, and put it in an array. Then, according to neighbors of every node, create the array of linked lists. After that, read the query file, starting the algorithm, and create a priority queue of unvisited nodes.

A min heap, or a priority queue will be used to keep the unvisited nodes.