```
/***********************************************************************
 * Project Report Template
 * Project 3 (Map Routing), ECE368
 ***********************************************************************/
```

Name: Aqlan Najwan Mohd Zakil Hussin
Login: amohnzak

```
/***********************************************************************
 *  Explain your overall approach to the problem and a short
 *  general summary of your solution and code.
 ***********************************************************************/
```

I created a structure called Graph to store all the information regarding the graph. It contains information such as number of vertices, number of edges, source (starting point), destination (end point), adjacency list, points (which stores the x and y coordinates of vertices), predecessor (which is an array that stores the nodes the path took) and distance (which is the shortest path distance). For the algorithm, I implement priority queue and it works as follows:

```
1) Initialize distances of all vertices as infinite.

2) Create an empty priority_queue pq.  Every item
   of pq is a pair (weight, vertex). Weight (or
   distance) is used used as first item of pair
   as first item is by default used to compare
   two pairs

3) Insert source vertex into pq and make its
   distance as 0.

4) While either pq doesn't become empty
    a) Extract minimum distance vertex from pq.
       Let the extracted vertex be u.
    b) Loop through all adjacent of u and do
       following for every vertex v.

           // If there is a shorter path to v
           // through u.
           If dist[v] > dist[u] + weight (u, v)

               (i) Update distance of v, i.e., do
                     dist[v] = dist[u] + weight (u, v)
               (ii) Insert v into the pq (Even if v is
                    already there)

5) Print distance array dist[] to print all shortest
   paths.
```

Here is the list of special library I used:

1) Math.h: I used sqrt function to calculate the distance between vertices.

2) Assert.h: I used assert function to make sure the index of the vertices is greater than or equal to 0 and less than number of vertices.

```
/**********************************************************************
 *  Known bugs / limitations of your program / assumptions made.
 **********************************************************************/
I assumed that the index of the vertices is greater than or equal to 0 and
less than the total number of vertices provided by the first line of the
first input file. I also assumed that the edges connect only existed vertices
that are listed in the first input file before the edges are listed.


/**********************************************************************
 *  List whatever help (if any) that you received.
 **********************************************************************/
I looked up the internet on how to implement Djikstra’s algorithm using
priority queue. I used the idea found on geeksforgeeks website.


/**********************************************************************
 *  Describe any serious problems you encountered.
 **********************************************************************/
At first, I had problem when trying to print the nodes on the shortest path
because I only save the predecessor. So, I can read the path backwards, from
destination to the source. What I do to solve this problem is, I added
another loop to iterate from destination to source until I found the node
that are not printed yet.



/**********************************************************************
 *  List any other comments/feedback here (e.g., whether you
 *  enjoyed doing the exercise, it was too easy/tough, etc.).
 **********************************************************************/
The exercise is challenging and required a lot of research to get ideas on
how the algorithm works and how to implement it.
```