Parsing input files
I would first read the first line and store the number of vertices and edges, so I know how many lines to read for each of them. I would store the co-ordinates as well as the connections in auxilliary data structures so I can parse them easier. Then I would read the first node in the graph and begin to populate my graph data structure (described below). I would look for the connections it has, and find the Euclidian distance to each of its adjacent nodes and store all this information in my graph data structure. Then I would repeat this procedure for all the nodes in the graph until they have all been covered.

Data structures
To store the graph, I would an array of size 'n', where 'n' is the number of the vertices in the graph. Each slot in the array is a head node to a linked list representing all the edges to other adjacent nodes in the graphs. Each node in this linked list will have 2 data fields, the name of the node and its edge weight to the head node of the list. I prefer using this implementation over the 2-dimensional array implementation because it seems more memory efficient (since my 'n' is around 87000).

Dijkstra's Algorithm
This algorithm is used to find the shortest path between two vertices in a graph. For example, lets say a graph has 6 vertices named A-G. The shortest path between A and G is needed. First a root vertex is identified (can be either A or G). This is now the 'current' node. Any shorter paths to adjacent vertices are updated using the edge weights between the current node and all its adjacent nodes. Then, weights of all the adjacent nodes (excluding nodes already visited nodes) are compared and the lowest one is picked. This will be the new current node. Mark the old current node as 'visited'. Repeat the process with the new current node until all nodes have been visited. Each node's weight now represents the shortest path between it and the root node.