Asheem Chhetri

ECE 368
Project 3
Map Routing: Milestone 1

Implement Dijkstra's shortest path algorithm for weighted undirected graphs.

Inputs:
File 1: The first input file will represent a map, which is an undirected graph whose vertices are points on a plane and are connected by edges whose weights are Euclidean distances. (Vertices correspond to cities and the edges to roads connected to them)
File 2: The second input file contains a list of search queries with the first line containing the number of such queries and each of the following lines containing one query in the form of a source vertex and destination vertex pair
To parse the input search file:
It consists the number of the queries (on the first line) and followed by the queries it has starting point and a destination. Thus, we can get the corresponding number of starting points and destination points by reading the file.

Goal:
Given a map file and a query file as inputs, we need to compute the shortest path from each source listed in the query file to the corresponding destination using Dijkstra's algorithm.

Dijkstra's algorithm:
This algorithm finds the length of an *optimal* path between two vertices in a graph. (*Optimal* can mean *shortest* or *cheapest* or *fastest* or optimal in some other sense: it depends on how you choose to label the edges of the graph.) The algorithm will work as following (based on current understanding):

1. Choose the source vertex
2. Define a set S of vertices, and initialize it to empty set. As the algorithm progresses, the set S will store those vertices to which a shortest path has been found.
3. Label the source vertex with 0, and insert it into S.
4. Consider each vertex not in S connected by an edge from the newly inserted vertex. Label the vertex not in S with the label of the newly inserted vertex + the length of the edge.
    - But if the vertex not in S was already labelled, its new label will be min(label of newly inserted vertex + length of edge, old label)
5. Pick a vertex not in S with the smallest label, and add it to S.
6. Repeat from step 4, until the destination vertex is in S or there are no labelled vertices not in S.

If the destination is labelled, its label is the distance from source to destination. If it is not labelled, then there is no path from the source to the destination.

Data Structures to be used:

Since we know the format of the input graph file, it is consisted of number of vertices and edges (The first line), x and y coordinates for all the vertices and all the edges (pairs of vertices). I will be using struct for each node. In each node structure, it contains the x and y coordinates integer variables, and then followed by the array of integer which represent the distance to other vertices that are conjoint with the node itself.
We can read the header line which is the number of vertices and edges. Firstly, read the corresponding lines of coordinates for all the vertices and finally get all the index of conjoint vertices into the array.

```
struct node{
        int x; //x coordinate
        int y; //y coordinate
        int distance[]; //distance between the current node and other nodes
        int number; //number of connected nodes
}
```

Note: This is the initial plan of implementation, which may or may not change as I start working on the project.