Driving Route Finder

You will be implementing a driving route finder (like Google Maps) for the Edmonton area. The application will involve a combination of the Arduino (as client) and a desktop computer (as server). The user will be able to scroll around with a joystick on a zoomable map of Edmonton and select a start and end point for their route. The Arduino will communicate these points to the desktop, which has all of the street information for Edmonton. The desktop will find the shortest path (by distance along the path) and return the waypoints of this path to be displayed as lines overlaid on the original map. The user can then repeatedly query new points to receive new routes.

We will provide the Arduino code for scrollable/zoomable maps, which will just display the latitude and longitude of a selected point. In class we will develop python code for directed graphs, and discuss algorithms for route finding efficiently. We will also provide you with a text file containing information on the Edmonton road graph. You are responsible for implementing the following:

1. Routine for computing least cost paths efficiently.
2. Routines for building the graph by reading the provided text file. A cost function for use with Dijkstra's
3. A python route finding server that provides paths through the specified protocol.
4. Arduino client program that queries server for user requested routes (as specified above).
5. Displaying the route on the Arduino overlaid on the map.

The assignment must be submitted in two parts. **The first part is due Friday, March 1. The second part is due Friday, March 8**. Both must be submitted by 11:59pm.

If you're looking for some optional extra functionality for your program after this basic functionality is completed, consider connecting the route finder with our restaurant finder to find paths to nearby restaurants.

Part I: Server

For part I, you will implement the server side of the required functionality (items 1-3 in the list above).

Dijkstra's Algorithm

You will need to implement Dijkstra's algorithm for finding least cost paths matching the following interface.

```
path = least_cost_path(G, start, dest, cost)

least_cost_path returns a least cost path in the digraph G from vertex
```

```
start to vertex dest, where costs are defined by the cost function.
cost should be a function that takes a single edge argument and returns
a real-valued cost.

if there is no path, then it returns None

the path from start to start is [start]
```

We will present pseudocode for the algorithm in class.

Graph Building

Your server on start-up will need to load the Edmonton map data into a digraph object, and store the ancillary information about street names and vertex locations. You will also need to write a cost function for use with route finding. This cost function should match this interface.

```
c = cost_distance(e)

cost_distance returns the straight-line distance between the two
vertices at the endpoints of the edge e.
```

If you compute the straight-line distance directly using lat and long, then the distance will be in units of 100,000ths of a degree.

Server

Your server needs to provide routes based on requests from clients. For Part I, your server will be receiving and processing requests from the keyboard, by reading and writing to `stdin` and `stdout`. Even when communicating with the Arduino client we will use the same protocol described below.

All requests will be made by simply providing and latitude and longitude (in 100,000ths of degrees) of the start and end points in ASCII, separated by spaces and terminated by a newline. For example,

```
5365488 -11333914 5364727 -11335890
```

The server will process this request by computing a shortest path along Edmonton streets and then returning the waypoints (i.e., locations of the vertices along the path). The path will be returned by first printing the length of the path, followed by a new line. The latitude and longitude of each waypoint along the path is then printed, separated by a space, and terminated with a new line. For example,

```
8
5365488 -11333914
5365238 -11334423
5365157 -11334634
```

```
5365035 -11335026
5364789 -11335776
5364774 -11335815
5364756 -11335849
5364727 -11335890
```

After printing the path, the server should listen again for another route finding request.

After loading the Edmonton map data the server should only begin processing requests if it is running as the main program. For example,

```
if __name__ == "__main__":
    # Code for processing route finding requests here
```

This will allow us to test your code for `least_cost_path` and `cost_distance` separately from the server protocol.

Part II: Client

For part II, you will implement the Arduino side of the assignment (items 4-5 above). We will provide you with code for moving around with a joystick on a scrollable/zoomable map of Edmonton. You will have to implement the interface for selecting two points with the joystick, communicating these points to the server over the serial port, receiving the resulting path and displaying the path overlaid on the map. While the route is displayed, the user should be able to continue to scroll/zoom on the map. If they select new start and destination points, a new path should be retrieved from the server.

Note that not all of the path will be visible for the portion of the map being viewed. You will need to "cull" (skip and not display) portions of the path that are not visible, as well as "clip" lines that only intersect a portion of the screen. You may want to read about line clipping, for example Wikipedia's article on Cohen-Sutherland's algorithm could prove very useful.