Extending and improving the search for the shortest path in graphs

Assignment No. 2

To do before 23.04.2019

Shortest path on a weighing graph

There are two key parts to this assignment.

1. The first is to implement a classic shortest path algorithms(Dijkstra's alg, Bellman -Ford or Floyd Warshall algorithm)
2. The second is to implement the A*Search

Both these methods take a start point and a goal point. And your scope is to find the shortest path from the start to the goal, in the weighing graph.

To Do:

1. Create your map (MapGraph):
    a. The cities are given by 2 or 3 cartesian coordinates (x,y) or (x,y,z).
    b. Not all the cities are connected by an edge (the graph is not complete)
    c. You set the cost of the edges = distance (on the road) between two connected cities
    d. The cost(A,B) >= euclidean distance (A,B)

2. The biggest difference between A* Search and Dijkstra(or other algorithm) is the number of nodes that get visited on the path to finding your solution. You have to provide the number of nodes that are visited, using Djikstra's (or other) algorithm, as well as using A* Search on one (or more) of your test cases.

Assignment Details

1. MapGraph is an undirected graph that represents the road connections between different Cities. Vertices in a MapGraph are Points ((x, y,z) pairs + eventually the Name of the city) that correspond to intersections or dead ends between roads. The edges are the road segments between these cities.
2. Decide whether you will want to use an adjacency list or adjacency matrix representation by thinking about the number of neighbors each node will have. Remember that neighbors are intersections that are connected by streets. In general each intersection is connected to only a handful of other intersections.
3. You'll probably want to include a printGraph method to help you make sure your graph representation is correct before you implement the shortest path search.

Assignment Details

The methods you must support in your project are listed below with a brief description of what each method should do:

The required methods are:

a. **public MapGraph()** - Creates a new empty MapGraph object.
b. **public boolean addVertex(Point location)** Adds a vertex to the graph at the location specified by the Point. Returns true if the vertex was successfully added, and false if it was not (because the vertex representing location was already in the graph) or if the parameter object is null.

Assignment Details

c. **public void addEdge(Point end1, Point end2, double length [,String roadName])** adds an edge between two Points that are already in the graph. The roadName is the name of the road (e.g. "DN1") . The length is the length of this road segment, in km. This method should throw an IllegalArgumentException if either of the two points are not in the graph already, if any of the arguments is null, or if length is less than 0.

d. **public int getNumVertices()** returns the number of vertices in the graph

e. **public int getNumEdges()** returns the number of edges in the graph

Assignment Details

f. **public List<Point> dijkstra/BF/ FW (Point startPoint, Point goalPoint) -** Performs dijkstra's / Bellman Ford/ Floyd-Warshall algorithm to search the graph starting at the startPoint until it reaches the goalPoint and returns a list of geographic points along the shortest path from start to goal.

g. **public List<Point> aStarSearch(Point startPoint, Point goalPoint)** - Performs the A* algorithm to search the graph starting at the startPoint until it reaches the goalPoint and returns a list of geographic points along the shortest path from start to goal.

If you cannot meet all the requirements, try to solve the problem to the best of your abilities.