BIRZEIT UNIVERSITY

Design and Analysis of Algorithms (COM336)

Fall Semester 2021/2022

Project # 3

Due Date: xx/xx/xxxx

Dijkstra's algorithm

Implement the classic Dijkstra's shortest path algorithm and optimize it for maps. Such algorithms are widely used in geographic information systems (GIS) including MapQuest and GPS-based car navigation systems.

In this project you will use Dijkstra's algorithm to find route information between two cities chosen by the user. The user will be shown a route that results in the lowest price.

Maps. For this assignment we will be working with maps, or graphs whose vertices are points in the plane and are connected by edges whose weights are Euclidean distances. Think of the vertices as cities and the edges as roads connected to them. To represent a map in a file, we list the number of vertices and edges, then list the vertices (index followed by its x and y coordinates), then list the edges (pairs of vertices. For example, represents the map below:

```
6       9
City1   1000    2400
City2   2800    3000
City3   2400    2500
City4   4000    0
City5   4500    3800
City6   6000    1500
City1   City2
City1   City4
City2   City3
City2   City5
City3   City5
City3   City4
City3   City6
City4   City6
City5   City6
```

Dijkstra's algorithm. Dijkstra's algorithm is a classic solution to the shortest path problem on a weighted graph. The basic idea is not difficult to understand. We maintain, for every vertex in the graph, the length of the shortest known path from the source to that vertex, and we maintain these lengths in a priority queue. Initially, we put all the vertices on the queue with an artificially high priority and then assign priority 0.0 to the source. The algorithm proceeds by taking the lowest-priority vertex off the PQ, then checking all the vertices that can be reached from that vertex by one edge to see whether that edge gives a shorter path to the vertex from the source than the shortest previously-known path. If so, it lowers the priority to reflect this new information.

This method computes the length of the shortest path. To keep track of the path, we also maintain for each vertex, its predecessor on the shortest path from the source to that vertex. Your goal. Optimize Dijkstra's algorithm so that it can process thousands of shortest path queries for a given map. Once you read in (and optionally preprocess) the map, your program should solve shortest path problems in sublinear time. One method would be to precompute the shortest path for all pairs of vertices; however, you cannot afford the quadratic space required to store all of this information. Your goal is to reduce the amount of work involved per shortest path computation, without using excessive space.

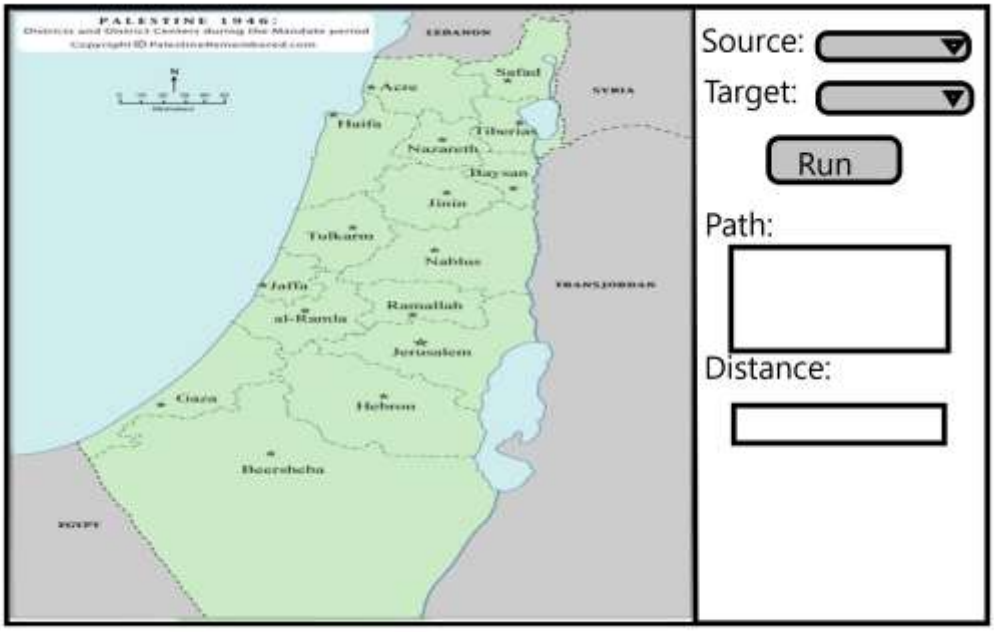
Idea. The naive implementation of Dijkstra's algorithm examines all V vertices in the graph. An obvious strategy to reduce the number of vertices examined is to stop the search as soon as you discover the shortest path to the destination. With this approach, you can make the running time per shortest path query proportional to E' log V' where E' and V' are the number of edges and vertices examined by Dijkstra's algorithm. However, this requires some care because just re-initializing all of the distances to ∞ would take time proportional to V. Since you are doing repeated queries, you can speed things up dramatically by only re-initializing those values that changed in the previous query.

Input : Palestinian Cities

Map: Palestinian Map

Output: Show the route on the map

Example for interface:



Notes:

1. You have to choose the city through mouse or keyboard.
2. The path should appear on the map.
3. Your project should include at least 50 cities.