Lab 5: Dijkstra's Shortest Path Algorithm

Due: Wednesday 4/28 at 11:59pm

In this laboratory assignment, you are going to perform an emprirical runtime analysis of the priority queue data structure and use it to implement Dijkstra's algorithm to find shortest paths in graphs. This powerful tool is applicable to a wide variety of optimization tasks, such as navigating Berkeley's road network in Figure 1. It is also the basis for more powerful search algorithms.

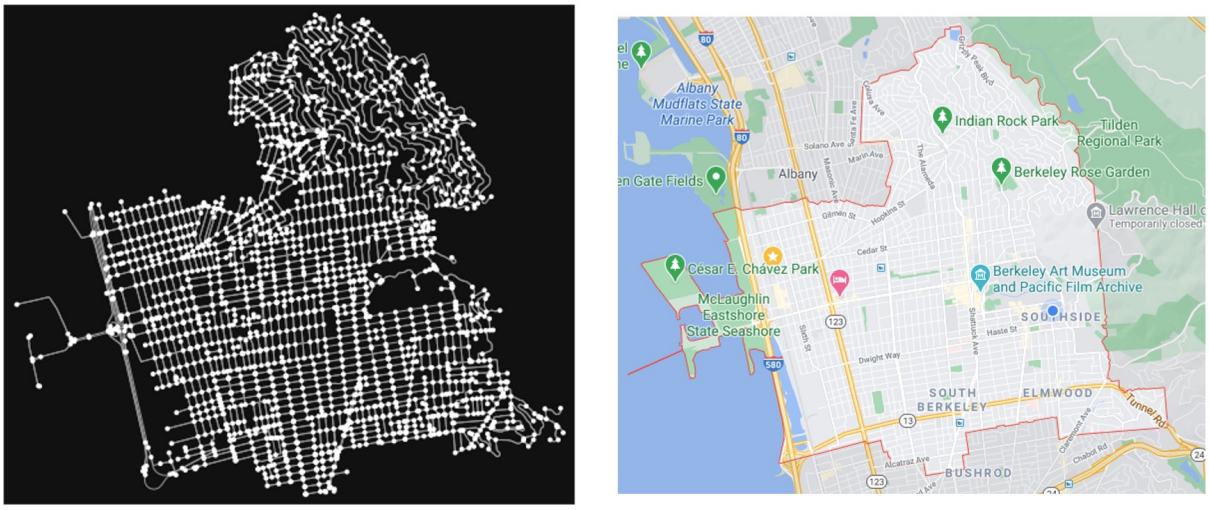

Figure 1: Left: the road network of Berkeley; Right: Google map of Berkeley

Priority Queue

You are going to utilize a PriorityQueue (PQ) data structure which has been implemented for you using a binary min heap. The PQ supports three operations:

- Insert(nodeName, priority) - adds an element to the PQ with the specified priority and node name.
- ExtractMin() - removes the element with smallest priority and returns the node name.
- DecreaseKey(nodeName, newPriority) - updates the priority of the specified element.

You are welcome to write your own PQ, use an implementation found online, or use a library. Don't forget to cite your references/libraries properly if you do so.

Problem 1. Empirical Complexity Analysis

Before we implement Dijkstra's algorithm, we want to verify that the data structure we are going to use will allow us to achieve the desired time complexity. In particular, you are going to perform an empirical analysis of the runtime of the PriorityQueue data structure by recording the runtime of the Insert(), ExtractMin(), and DecreaseKey() operations for a varying number of elements, n. Use Matlab's tic and toc functions as a stopwatch (consult the documentation). You are to do the following:

- Use $n = 10^i$ with $i = 0, 1, 2, 3$.
- Insert n items in the priority queue. Make sure to use different node numbers as duplicates will be overwritten. A good scheme would be to insert elements with their priority in ascending order. This will ensure the most efficient construction of the initial PQ. Can you explain why?
- Insert a new element and measure how much time elapses. We are interested in the worst case runtime so choose the priority of the new element accordingly.
- Extract the min element and measure how much time elapses.
- Decrease the priority of the last element in the PQ and measure how much time elapses. We are interested in the worst case runtime so choose the new priority of the element accordingly.
- Since there is some intrinsic machine performance variability, perform the above steps ~ 50 times for each n and average the recorded times (use a loop).
- Finally, create a plot of the (average) elapsed time vs. n for each operation. Use a log axis for n. Comment on your observations.

Problem 2. Dijkstra's Algorithm

Now that we are satisfied with the PriorityQueue, it's time to utilize it to find shortest paths in graphs. We're going to use the graph structures from Lab 3 for simplicity and convenience. Use the myDijkstra.m starter file on the class webpage and implement Dijkstra's algorithm using the PriorityQueue data structure. The function takes an edgelist and an origin node and returns:

- dist - an array containing the shortest distance from the origin node to all other node.
- prev - an array containing the previous node on the shortest path for each node.

The value of dist should be infinity for unreachable nodes and prev should be 0 for the origin node and the unreachable nodes. Feel free to use the visGraph function from Lab 3 to visualize the graphs and verify your algorithm's correctness for the smaller graphs.

Submission

1. Your Matlab files (.m - files) with instructions.
2. A PDF report including:
   - Attach the three plots from problem 1 and comment on them.
   - Report the dist and prev arrays for the 6 graphs using **Node 1** as the origin.

   For graph_ex.mat the dist and prev arrays are the following:

dist					
0	2	1	12	11	10

prev					
0	3	1	5	6	1