CS 170 Efficient Algorithms and Intractable Problems
February 9, 2016 Liam Leahy

HW 3

Due February 8, 11:59pm

Partner: Stephen Qu

6. (20 pts.) Travel planning

You are given a set of n cities and an $n \times n$ matrix M, where $M(i, j)$ is the cost of the cheapest direct flight from city i to city j, or ∞ if there is no such flight. (Flights cost a nonnegative integer number of dollars.) You live in city A and want to find the cheapest and most convenient route to city B. In particular, if there are multiple routings you could take with the same total cost, you'd like the routing with the fewest connections (ie, the smallest number of flights). Design an efficient algorithm to solve this problem. You can assume that the best route will require fewer than 1000 flights.

Solution:

Main Idea: You can represent this problem as a graph with n vertices and n^2 edges. The entries in the matrix are the edges and the cities are either the rows or the columns. We can construct the graph in linear time with respect to the edges and vertices and perform Dijkstra's on this graph in linear time with respect to the edges. To find the shortest path through the graph from the starting vertex (A). This is not enough and subsequently, we perform BFS in order to find the path through the maze.

Pseudocode:

Algorithm 1 Returns the shortest path from A to B, breaking ties by choosing the path with the least number of "hops".

```
1: function FIND_SHORTEST_PATH(M, A, B)
2:     G ← graph from M
3:     P ← DIJKSTRA'S(G, A, B)
4:     r ← BFS(P, A, B)
5:     return r
```

Proof: We know that Dijkstra's returns the shortest path through a graph. The graph we construct completely encapsulates the information of the problem. We want the shortest path from A to B. It is necessarily the case, given Dijkstra's functioning, that this will be the shortest path from A to B. The one caveat is the case in which there are ties. Note that the problem also wants the fewest number of "hops". Once we eliminate the extra edges that are not part of the shortest paths. In this case, we can find this by running BFS on the graph. This works because BFS will always find the "first" path through the □

Running Time: $O(n^2 \log n)$.

Justification: Dijkstra's is the dominating algorithm as iterating through the edges takes $O(n^2)$ and BFS takes $O(|V| + |E|) = O(n^2)$. It takes $O((|V| + |E|) \log |V|)$. There are n^2 edges and n vertices. Thus the runtime is as given.