Time Spent: 2 hours

Collaborators and Resources:

Problem 2

Problem 2 is asking something very similar to the problem in which we need to find the shortest distance between two vertices in a graph. This problem is generally solved by breadth-first search, however we are hoping to also discover the number of such paths. Luckily we know that the breadth-first search is going to return all cities of interest because of the fact that it will find all connected elements. We already know the runtime of BFS which is $O(|V| + |E|)$ and we want to maintain this runtime and as a result can only add linear operations. Given the fact that edges are only drawn to indicate interesting routes, and that passing through as few cities as possible is desirable we can say that the following algorithm (which is based off the BFS search we discussed in class) should suffice ...

1. Given an undirected graph G = (V,E) and nodes s,t $\in$ V
   (a) Set D[v] = $+\infty$ for all v; D[s] = 0
   # D is an array of distances, where the indices are nodes 1 through n.
   (b) Insert (s,0) into an empty queue Q
   (c) Insert (t,0) into an empty queue Q_2
   (d) Set the number of paths n from s to t to be 1
   (e) While Q or Q_2 is not empty:
   (f) If Q is not empty {
      i. Remove (u,d) from the queue Q
      # The following for loop just records the shortest distance (for each vertex) to one level at a time
      ii. for all of u's neighbors v such that D[v] = $+\infty$
      # that is, which haven't been visited

A. D[v] = d + 1 # Store the distances in an array
B. Insert (v,d+1) into Q
}

iii. Remove (u,d) from Q_2
iv. For all vertices v in the connected graph for which D[v] = D[u-1]
That is, for all the neighbors in the previous layer to our city u (t in the first case)
A. Increase n (number of paths) by the number of such vertices - 1
B. Insert (u,d-1) into Q_2

2. Return the number of paths n

Claim 1. The algorithm solves the problem of finding the total number of interesting paths through as few cities as possible (shortest paths measured in edges) in $O(|V| + |E|)$ time.

Proof. Basically what my algorithm does is it implements breadth-first search which we know has a runtime of O(n+m) because we make exactly n additions to the first queue (the number of reachable vertices n) and we add each edge twice (giving O(m)). The second part of the algorithm is intended to keep track of the number of vertices in each previous layer starting at vertex t, for each vertex in layer d computing the number of edges to vertices in layer d-1. Each new edge constitutes another distinct, shortest (because the shortest path doesn't have within-layer edges) path, and furthermore it only counts the total number of edges an additional time in the worst case (in which all paths in the connected graph are included) in the same while loop (and so the algorithm remains in O(n+m) time). □