FindMinyan Writeup

I modeled the problem as a weighted undirected graph, where the vertices were the cities, the highways between them were the edges, and the point-to-point highway durations were the edges' weights. I considered the problem as requesting the length of the shortest path by weight between vertices 1 and *n*, where at least one vertex in the path was a special "Minyan" vertex, and to find the number of such paths. I saw these as fundamentally two separate problems: first, I found the shortest path with a Minyan, and I used my results from that algorithm to find the number of such paths.

To find the length of the shortest path, I first performed a depth-first search starting at vertex 1, collecting each vertex that had a Minyan and was reachable from 1 and *n*, which I will call a Minyan for brevity. I examined each Minyan, and the paths that contain it, in its own iteration of the algorithm.

On each Minyan, I performed a modified version of Dijkstra's algorithm starting from that Minyan to obtain its shortest path tree. I will describe the modification below. For each shortest path tree, I summed the distance from Minyan to 1 and from Minyan to *n* to determine the minimum distance from 1 to *n* through the Minyan. This is the minimum because: each tree returns the shortest path from 1 to Minyan and Minyan to *n*, and thus between 1 and *n* through that Minyan. Doing this for each Minyan, I found the minimum such distance from 1 to *n* through any Minyan.

To find the number of paths, I first modified Dijkstra's algorithm. When relaxing an edge, if the edge's destination vertex had the same distance from the root Minyan as my current path to that vertex (`distTo[w] == distTo[v] + e.weight()`, in Sedgewick's notation), I added it to the shortest path tree in addition to the original edge. (I replaced the vertex's incoming edges if I found a quicker way to get to it.) This gave me a shortest path tree that contained every possible shortest path, because of the cut property: since any edge crossing a cut with the same path weight as a crossing edge in an SPT is itself part of an SPT, an SPT that contains every such edge should contain every such shortest path. Such an SPT would still be a tree, because while its branches might intertwine, they cannot form cycles, because any path containing a positive cycle could be replaced with a more efficient path without the cycle.

To find the number of shortest paths, each time I created an SPT using Dijkstra, I explored the tree with my subroutine `constructRoutes()`, to get the number of shortest paths containing the tree's Minyan but not Minyanim whose trees had already been explored. I summed the paths from each tree to get the total number, discarding the number of paths whenever I discovered a path shorter than what I had been counting.

`constructRoutes()` traverses its tree using DFS twice, once from 1 and once from *n*, stopping each time when it reaches the tree's root Minyan. The recursive function of the DFS returns, at each vertex, the number of paths from the root Minyan which end at that vertex. There are two base cases. If the vertex is the root Minyan, it returns 1, since the Minyan constitutes a single path. If the vertex is the

root Minyan from a previous iteration of the algorithm, it returns 0, since I have already counted any paths that end at this vertex and don't want to double-count them. Otherwise, I return the sum of the number of paths ending at each parent, as the paths ending at a vertex are the paths whose second-to-last node is one of its parents. I did this for every parent, even parents I had already explored, using an array to store the number of branches at each vertex so I would not have to visit a given vertex more than once. I called this method for 1 and n to determine the number of paths between each of those vertices and the root Minyan, and multiplied those numbers to get the cardinality of the set of paths between 1 and n through Minyan, since they are traveled independently, so the Product Rule from Set Theory applies. I add this to the sum of paths from all other SPTs of the shortest path length to get the total number of paths of that length through any Minyan.

My algorithm's order of growth is (unfortunately) $O(mh\lg(n))$, where h is the number of highways, n is the number of cities, and m is the number of Minyanim reachable from 1 and n. This is because my algorithm used Dijkstra's shortest path algorithm, whose efficiency is proportional to O(ElgV), for each Minyan reachable from 1. Since the edges were highways and the vertices cities, it has an efficiency of $m * h\lg(n)$. All other algorithms I used are more efficient, and so are asymptotically irrelevant.