Moshe (Judah) Goldring
Professor Leff

Find Minyan Write-Up:

How does your solution “model the problem”? & Based on the above, describe the algorithm that solves the problem. You should incorporate “well known algorithms” by reference but be sure to discuss algorithm modifications (if any).

Shortest Path:
The find minyan problem is one that can be best modeled using an edge weighted graph where the weights are the distance between cities and the edges are the highways. As we already know the best way to find the shortest distance between two vertices (in this case cities) is to use Dijkstra’s algorithm. However, this problem comes with the caveat that a person must pass through a minyan before reaching their desired city. To solve this problem, I decided to make two shortest path graphs using Dijkstra, one time having the starting point being the source vertex and once having the endpoint being the source vertex. Then for all minyanim supplied I would obtain the shortest path from the start vertex, checking if there even was a path, (using a distance to array that stored the distances to the source from any given vertex and a hasPath array that checked if a path existed) to each minyan and store that value in a map that mapped a minyan to its distance to start. Then using my other graph that used the goal vertex as the source I would do the exact same thing, however this team I would only add a value to the map if that minyan already had a value, meaning there was a path from the start vertex as well. I then looped through the distance maps values returning the minimum.

Pseudo Code:

```
Starter = new Dijkstra(Graph, start vertex)
Ender = new Dijkstra(Graph, end vertex)
For(int m : minyanim)
        If starter.hasPathTo(m)
                DistanceMap.put(m, starter.distanceTo(m))
        If ender.get(m) != null and ender.hasPathTo(m)
                DistanceMap.put(m, ender.distanceTo(m))
For(int I : Distance.values)
        Return minimum
```

Number of Paths:
To solve the number of paths I modified Dijkstras algorithm and added a paths array that counted the number of shortest paths to that vertex and every time a vertex was relaxed, meaning a shorter path was found to that vertex I set that vertex number of paths to the number of paths that led to its parent. Since Dijkstra is greedy, a vertex parent’s paths would never change, meaning I wouldn’t have to worry about there being more than the current number of shortest paths to a vertex parent. If a vertex wasn’t relaxed but the distance to a child was the same as the distance to its parent plus the weight of the current edge, I would add to the childs amount of paths the

amount of paths of its parent, since that indicated that there was another way to get to that child in the same amount of time. To count the total amount of paths, I would loop through all the minyanim that had the same shortest duration, which was figured out above, and find the total number of shortest paths from the source to a minyan and the shortest path from the goal to a minyan and multiply their total. Doing that for all minyanim and adding their sum together gave me the total paths. However, that alteration in Dijkstra did not account for multiple minyanim on the same path. Since that solution was counting that as two separate paths, I removed the total of additional minyanim on one path from the total number of paths .

Pseudo Code:

Dijkstra Alteration in relax method:

```
If(distanceTo(child) > distanceTo(parent) + weight)
        distanceTo(child) = distanceTo(parent) + weight
        path(child) = path(parent)
else if distanceTo(child) = distanceTo(parent) + weight
        path(child) = path(child) + path(parent)
```

Finding total paths:

```
Set edges = new Set
Path array for starter Dijkstra indicating the amount of shortest ways to get to each vertex
Path array for ender Dijkstra indicating the amount of shortest ways to get to each vertex
for(int m : minyamin with shortest duration)
        add m's complete path of edges to edges set
        total paths += starter.path(m) + ender.path(m)
        if other m's are a vertex in edges set
                total paths -= number of additional edges
```

The main aspects affecting order of growth is how many times Dijkstra is run