Judah Goldring
WeAreAllConnected Write-up:

1. How does your algorithm "model" the problem? Does this "modeled problem" have a name?

This problem of finding the connections and shortest duration between all pairs of vertexes, or cities, can be best described as a transitive closure question. My algorithm takes the segments provided by the user and using the x and y coordinates as indexes for an adjacency matrix. The duration of that segment is then stored at the x and y coordinate provided by the segment. Since the problem emphasized that the segments are undirected, the duration is also inserted in the opposite indexes of the x and y coordinate. Meaning store the duration at (x,y) and (y,x). To then find the transitive closure and shortest distance between all pairs, I used Floyd Warshall's algorithm which does just that. Where the value stored at each spot in the adjacency matrix being the duration it takes to get from x to y or vice versa. To then find the total distance you would loop through the adjacency matrix adding all the values stored. However, the assignment doesn't just ask for the shortest distance between all pair of cities for a given set of edges. This would be far too easy. Instead, a current set of edges is provided, which I used to create the adjacency matrix and performed Floyd Warshall on, and a set of possible edges to be added is also provided.

Pseudo Code for making original graph:

```
For(Segment : Current)
        Graph[Segment.x][Segment.y] = Segment.duration
        Graph[Segment.y][Segment.x] = Segment.duration
```

```
//Floyd Warshall
For(k = 0; k < n; k++)
        For(i = 0; i < n; i++)
                For(j = 0; j < n; j++)
                        If(Graph[i][k] + Graph[k][j] < Graph[i][j])
                                Graph[i][j] = graph[i][k] + graph[k][j]
                                Graph[j][i] = graph[i][k] + graph[k][j]
```

2. Describe (ideally by reference to a classic algorithm) a "brute force" algorithm that solves the problem. In formal terms, what is its order of growth?

A brute force algorithm to solve which possible segment would best improve the overall duration between all pairs of cities, you would usesthe original graph Floyd Warshall was run on using the current segemnts, and then loop through all possible segments adding each segment one at a time to the graph and rerunning Floyd Warshall for each segment. The one that had the total shortest duration would be the winner. However, this is exceptionally slow because Floyd Warshall is O(N^3) algorithm, so if you had a large number of possible segments, we'll call the amount of possible segments 'S,' you could will up running Floyd Warshall O(S*N^3) times and as S gets increasingly large this algorithm becomes increasingly worse.

Pseudo Code for brute force:

```
For(Segment : possibleSegments)
        Int min = Integer.MAX_VALUE
        Duration = floydWarshall(Graph, Segment) //returns the total sum of all duration
        If(Duration < min)
                Duration = min
```

3. Describe your "better" algorithm that solves the problem. Why is it an improvement of the "brute force" algorithm? In formal terms, what is its order of growth?

A better solution to the brute force algorithm would be to run a modified version of FloydWarshall on your original graph for each possible segment. This modifies version would be O(N^2) instead of O(N^3) like the regular Floyd Warshall. In this way the dominant factor would still be O(N^3) but that order of growth would only have a constant of 1, the one time you run it for the current segments, as opposed to a constant of S, the amount of times you would have to run it for each possible Segment. How do we get the Floyd Warshall down to O(N^2)? Essentially, I realized that the algorithm's outer loop goes through each vertex k, and considers all pairs of vertices (i,j) such that the path from i to j passes through vertex k, but in our case the only vertex k that we care about updating and checking is the possible segment we might wish to add to our current graph! Therefore, all we need to do is check if there's a better path from i to j through our suggested segment and we have to make sure we include our suggested duration along that path.

Pseudo Code for optimized Floyd Warshall:

```
int floydWarshall(Graph g, Segment s)
        g[s.x][s.y] = s.duration
        g[s.y][s.x] = s.duration
        //Floyd Warshall
        For(i = 0; i < n; i++)
                For(j = 0; j < n; j++)
                        If(Graph[i][s.x] + Graph[s.y][j] + s.duratiom< Graph[i][j])
                                Graph[i][j] = Graph[i][s.x] + Graph[s.y][j] + s.duratiom
                                Graph[j][i] = Graph[i][s.x] + Graph[s.y][j] + s.duratiom

        //loop through graph and return total duration
```