Results: adara2-anikp2-rohani2-abhijit3

In our GOALS.pdf, we had initially talked about creating a tier list of fantasy football players for which we could rank them based on a graph implementation. However, we realized it was difficult to correlate tier rankings to a graph. Further, it was difficult to find a use case for the traversal algorithms. As a result, we changed our strategy to focusing on a dataset provided by OpenFlights.com containing the airport name and its coordinates.

The first thing we focused on was extracting the dataset given, meaning we parsed the file into its string representation and a line-by-line vector representation. We additionally parsed the routes going from and to each airport as well as the coordinates of the airport in order to develop our adjacency matrix. From there, we utilized our airports and routes file to create an adjacency matrix by deriving connections from one vertex to another. The values in the adjacency matrix represented the weight between two nodes and we found the distance between each node by implementing a findDistance method in routes.cpp.

The first traversal that we incorporated into our project was the Breadth First Search. This algorithm works by starting at a node, searching adjacent / neighbor nodes, and then moving onto the next depth. The way we coded this algorithm was we kept track of the number of connections per level, iterated through the level, and pushed it back into a map that we returned.

Our second traversal involved Djikstra's algorithm, which generates an SPT (shortest path tree) with a given root. At each step in the algorithm, we had to find a vertex which had a minimum distance from the source. We used a priority queue to implement this algorithm and started by finding the vertex of our source point. From there, we determined the source's weight and the shortest path to another airport. Using this information combined with analyzing each path from the source, we marked the next location and updated the weights of each vertex. After updating all the weights and going through the entire graph, we returned the array of distances.

The third of our traversal algorithms was the landmark path problem. And essentially this problem was finding the shortest path from A to B that goes through C. The way we approached this problem was we found the shortest path from A to C and then added that to the shortest path from C to B. Initially we thought that we would have to have to code our own shortest path helper method, but we realized that Dijkstra's algorithm does this for us. After completing the algorithm, we verified the shortest distances via google maps.

Figures:

We used the 2 datasets from https://openflights.org/data.html, routes.dat and airports.dat to construct the nodes and edges for our graph.

Airports.dat

- This file contained 7,689 different airports from around the world
- We parsed the latitude and longitude, airport id, and 3 letter abbreviation

- Used latitude and longitude to find distance in miles from one airport to another

Routes.dat

- File included 67,663 valid connections from one airport to another respresnted by their id
- Used the connections to connect nodes for the graph and create edges
- Not all connections had valid ids so we ignored the ones that did not

Graph

- 3,199 airports have at least 1 or more connections
- Using BFS traversal we found each airport (with 1+ connections) has an average of 11.537 connections to other airports
- Estimated busiest airports counting the most connections from every airport using BFS traversal
- The top 5 airports had an average of 229.8 connections
    1. Frankfurt am Main Airport (Frankurt, Germany) - 239
    2. Charles de Gaulle International Airport (Paris, France) - 237
    3. Amsterdam Airport Schipholt (Amsterdam, Netherlands) - 232
    4. Atatürk International Airport (Istanbul, Turkey) - 224
    5. Hartsfield Jackson Atlanta International Airport (Atlanta, Georgia) -217