CS225 Final Report
Youjin Tark, Daniel Ferriss, Neil Sahai

Our final project leverages data provided by OpenFlights to build a graph that models all airports and routes provided in the datasets. This graph can be traversed with BFS, used in Dijkstra's algorithm, and used in the Landmark Path Algorithm. A user can also choose a specific airport and get a file of interesting statistics pertaining to that airport and the routes it serves.

To build the graph, data is first read from the airports and routes datasets. A map of all airports is built, with the key being the airport ID and the value being an airport data structure generated for that airport. If the file is missing any important data on an airport, such as its latitude or longitude, the airport is not added. Next, the routes are read from the file. Routes are only added if both the source and destination nodes exist in the airports map. Finally, both the airports and routes maps are iterated over and added to a graph. The graph is weighted and directed, with the weights being the distance, calculated using the Haversine formula, from the source vertex airport to the destination vertex airport.

Our first project goal was to implement the BFS traversal on our graph. BFS is a traversing algorithm that traverses the graph in a layerwise fashion. It is similar to a BFS search in trees,with some differences. In graphs we have to be aware of self loops and self cycles, whereas in trees we do not need to worry about that. Which is why in a BFS implementation we have to create some type of “visited” data structure. After we create a “visited” map, for example, we would then loop through that map and mark all the nodes as false or not visited. Then create a queue to help in traversal, and push the source into the queue. Next we will start our big while loop until the queue is empty. Within the while loop we will first print the top of the queue and push that out. After that we will loop through the adjacency list and in each iteration we will check to see if the current node has been visited or not, if it hasn't then we will visit it and push into the queue. Eventually as our loops approach the end our nodes will get printed out in BFS order.

Our Dijkstra algorithm is to find the shortest path between two airports. The idea is that even if each airport is not connected, we can figure out the final shortest path to get to that airport using the Dijkstra algorithm. Our project goal was to make the shortest path tree from a source airport. We managed to make that basic function as well as some advanced functions. In the process we figure out there are airports that source airport is not able to reach so we made a function to get the number of airports the source airport actually can get to. Secondly we made a function that returns the closest airport and farthest airport from the source airport by sorting the shortest path tree. Thirdly, we made a function that prints the shortest path from the source to the destination. It would present how many stops they will have to get to the destination as well as actual paths in sequence.

Our Landmark Path algorithm implementation leverages Dijkstra's Algorithm implementation that was earlier discussed. To find the minimum path length from A to C that intersects with B, we use Dijkstra's algorithm to calculate the shortest path from A to B and B to

C, then sum up these path lengths. If we were using an undirected graph we would be able to use Dijkstra's algorithm on path B then just sum the distance of paths from B to A and B to C, saving us some computation time. However, since our graph is directed, a path from B to A is not guaranteed to be a path from A to B.

Paragraph about statistics / interesting findings: Anyone

As we can see the statistics are about chicago o'hare, we intendly chose the source node as a o'hare since a lot of airport nodes in graph are isolated or have only one edge to the other nodes. It was hard for us to analyze the actual outcome if it was not a famous airport as a source node. To utilize these algorithms best I think we should optimize or qualify the database based on conditions such as the airport should have at least more than 3 flights going out to other airports.