CS225 Final Project

Jackson Weissing, Gaurang Dada

In this final project, we set out to find out about the possible applications of graph data structures in the real world. Using the OpenFlights dataset, our group has put together a conjunction of path finding algorithms and a traversal. Our project optimizes air travel by using algorithms like Dijkstra's and helps us find the shortest path between two airports using the Landmark Path algorithm.

Through this project, we aim to create a searchable index of routes that connect airports. Our implementation can also be helpful in determining the importance of individual airports.

This project was completed by first downloading the OpenFlights dataset. The two most used datasets were airports.dat and routes.dat. In the airport dataset, each airport is defined by an Airport ID, Airport name, latitude, and longitude. Similarly, each route is defined by an airline, stops, a source, and a destination. The way we set up our project was by allocating graph nodes to airports and using routes to serve as edges between airport nodes. In this way, we worked with a weighted, directed graph: the graph was weighted by distances between airports and directed by source and destination airports.

We implemented a Depth First Traversal on the airport nodes in this project. The implementation was straightforward, with the only tricky part being traversing unconnected airports in the graph. For traversing through connected components of the graph, we used a "DFT" function to iterate through all possible routes from an input airport. We use an "int idx" to store the IDs of all possible destination airports we can travel to using all possible routes from the input starting airport. For every unvisited destination airport, we increase count and recurse. To allow traversing through unconnected components of the graph, we implemented a "DFTDis" function which visits only unvisited nodes from the initial DFT implementation. We iterate through all airports, and for every unvisited airport, we call our initial "DFT" function again. For traversing "n" airport nodes, the time complexity of this operation was O(n).

Next, we implemented Dijkstra's algorithm to find the shortest paths between any two airports. This required a priority queue of pairs each comprising a distance between the airport and the source and the airport index. By default, a priority queue stores elements in descending order, so we needed to invert the distances because we needed to store them in ascending order. We then loop through this priority queue until we reach the destination node. We get all the edges for that node and then if those nodes haven't been visited we update the minimum distance to the node and update the corresponding previous node. Importantly, we also push the distance to the queue so as to process the queue in order of descending weight. The weights were generated using the longitude and latitude values available in each airport. These numbers were used in the haversine formula to calculate the shortest distance across the globe.

Finally, we created images using the provided PNG class. We mapped the latitude and longitude values to a mercator map. Points were placed by getting a 5x5 grid around the target pixel and painting that range with a given color. Lines were drawn by finding the equation of the line, y=mx+b, between two airports and placing enough dots to create a solid line. Below you can see the result of running the shortest path algorithm between John F Kennedy airport in New York and Vladivostok in Russia, which happens to go through Moscow.