# Final Project Report

By Allen Shen, Zekai Zhang, Jacob Albus and Jiankun Yang

For our final project, we chose to work on OpenFlights dataset and find the shortest path between two airports. We use a breadth-first search (BFS) to find the shortest unweighted path and Dijkstra's Algorithm to find the shortest weighted path. Then we do a visualization of the found shortest path by drawing it on a world map.

When we got started on the project, each of the team members first downloaded the dataset onto our own EWS instances and decided on the algorithms that we wanted to use. Then on Github, we created a repository with three branches, each representing one algorithm/visualization to avoid conflicts when committing and pushing to Github and to better organize the project development and code base. Our graph object was the common denominator between the several branches as we worked on the three algorithms.

The first of the algorithms that we implemented was a BFS for path finding. The main contents of this algorithm is stored in BFS.cpp. This is a covered algorithm from class, which finds the shortest path between two given vertices in an unweighted graph. This algorithm can find the flight path with the fewest stops. In this project, we utilized a queue and adjacency lists to find the visited airports. The queue takes in vertices that have yet to be visited. When visited they are marked as visited in a boolean list. After finishing visiting each node, we go to its nearest neighbor and repeat the process until we find the shortest path. In testing our code, our criteria for making sure this algorithm would work involved checking for valid inputs, and outputs. Specifically, we handled the case of inputting an airport code that doesn’t exist and when there exists no path between two airports (i.e. they’re in different components). In these

instances the path vector is empty. To test our code for correctness, we cross-validated the outputted path with the true shortest path. The run time complexity of the algorithm is O(V+E).

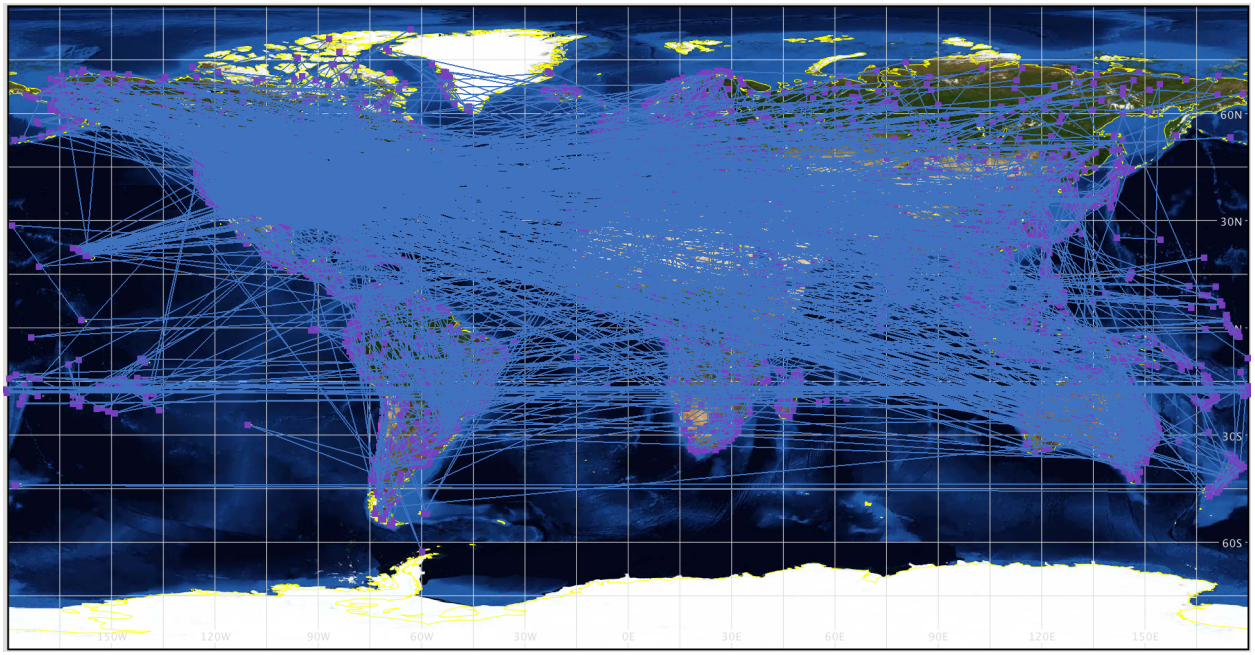
The second algorithm we used is Djikstra's algorithm. The main algorithm is stored in djikstras.cpp. It can find the shortest path between two given vertices in a weighted graph, which helps us find the flight path which minimizes total distance traveled. The algorithm uses a priority queue to go through all the vertices visited. The algorithm would visit the start node, then check all the unvisited neighbors. All the index of visited nodes will be pushed into the queue and pushed out when it's done visiting. The algorithm would store a previous node for shortest paths to every node from the start node. When the unvisited neighbors are checked, the previous node and the distance to this neighbor  node will be updated if a shorter path exists. Also, we can trace back to the start using the stored previous node. The running time of this algorithm is O(|E|+|V|log(|V|)). The test cases and error handling are similar to that of BFS with checks of non-existing and invalid airports applied. By checking the directed path such as from SFO to ORD, Dijkstra's algorithm gave the correct response. It flew directly from San Francisco to Chicago. Also by checking the path that I've taken before such as from CMI to PEK, the algorithm yielded correct results, flying to Chicago from champaign then to Beijing. Then after checking the longer paths that cross more countries, dijkstra's can be used to handle complex paths that take more airports in between.

The last one we did is visualizing a map to show all the flight paths. The function is stored in visualizer.h. It takes a vector of airport objects, a file path to a world map png, the dimensions of the png, and the name of the desired output png. The code uses the longitude and latitude of each airport and projects them onto a 2d coordinate plane, from this point it draws a square around each airport location and draws the respective routes. It is important to note that

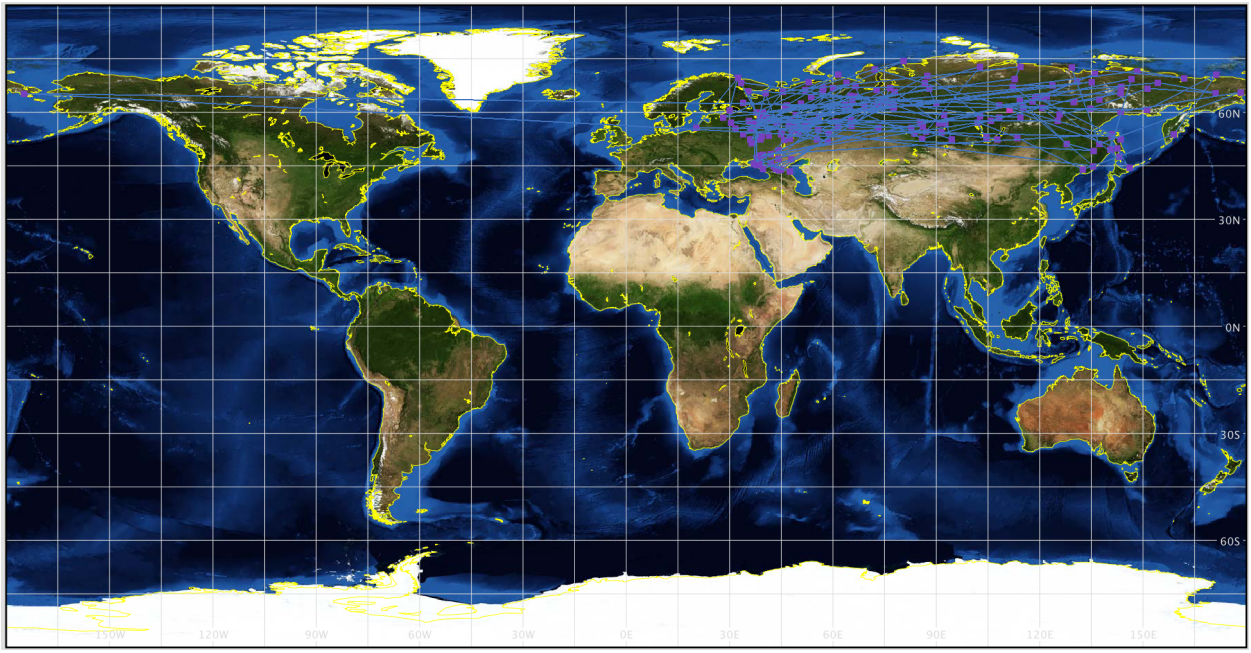
formula used for projecting the longitude and latitude onto a 2d coordinate plane relies on the fact that the map projection is equirectangular

The dataset we worked on is not a difficult one. We found that the results we get from our algorithm are generally quite straightforward. Since we have a visualization of the result, we can see the shortest path generated on the map. We also tried another algorithm called JohnsonAlgo, which could be used to find the longest path between two airports, but the downside is that this algorithm cannot handle large datasets, so we eventually gave it up after the midpoint check.

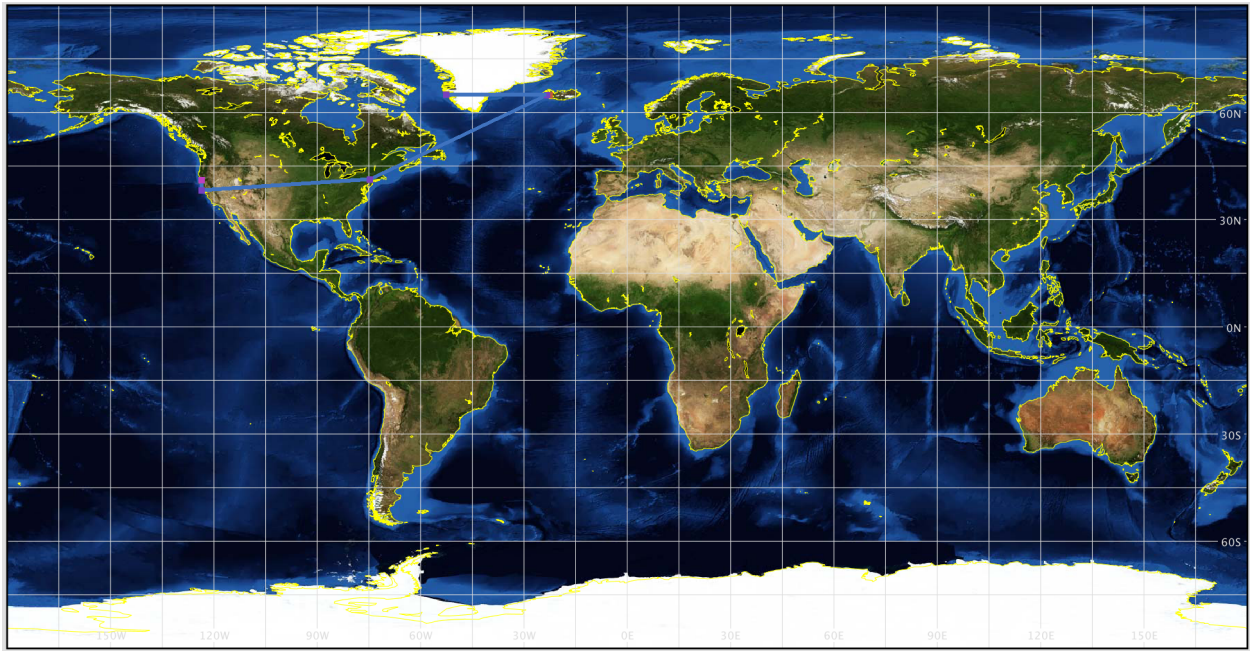
Below are the images of some flight paths and a little description of them:



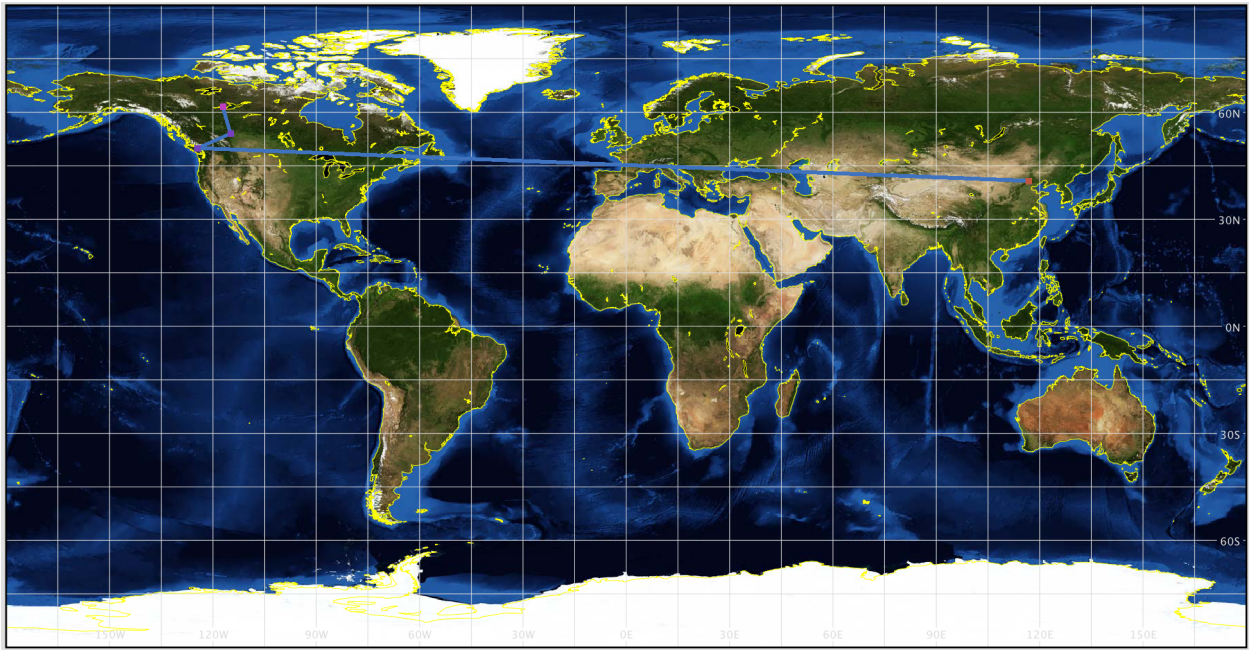

This is a visualization of all the flight routes in the dataset




This is a visualization of all the flight routes in Russia.




This is a visualization of the longest shortest flight path found using BFS which is between Redding, California and Reykjavik, Iceland. Specifically it goes from Redding to San Francisco to New York to Keflavik to Godthaab to Reykjavik.




This is a visualization of a route from Beijing, China to Hay River, Canada. Specifically, it goes from Beijing to Vancouver to Edmonton to Hay River.