Final Report

Boyou Han
Jae Lee
Lingxiao Mou
Yu (William) Zhao

Introduction

Our purpose for our project is to provide the shortest traveling route to the user who wants to travel by flight. We used the airport and route datasets from OpenFlight to create our graph implementation. When utilizing our project, the user provides the source and destination airports in which we will provide the visualizable routes and results including the shortest paths and distance.

Implementation

We constructed our weighted and directed graph by setting airports as vertices and the routes between various airports as edges. Each vertex stores the latitude, longitude, airportID and airport name. Each edge stores the distance as weight and airline as label. Our graph uses an adjacency list implementation to organize the datasets.

For our shortest path implementation, we created a priority queue, minimum heap and a hash table. Initially, we also set the source vertex's distance to be zero and all the other vertices to be infinitely large and initially all previous vertices to be empty. Then, we use Dijkstra's algorithm and BFS with the priority queue to keep updating the shortest distance from the source vertex to other vertices. We also kept updating the previous vertex to help us trace back to get the complete shortest path.

We mapped the output of the routes taken from source to destination airport onto a world map PNG image. To do this, we utilized the HSLAPixel and PNG files from the cs225 directory to implement a line drawing from the (x,y) points of the source and destination airports. To obtain the (x,y) points, we converted the latitude and longitude of each respective airport and drew the line between the two points using a linear function.

Discoveries

We obtained a deeper understanding in greedy algorithms in which we discovered the algorithm optimizes each step to get an optimized result at the end, and also became more familiar with dynamic programming and priority queue (minimum heap). The time complexity of Dijkstra's algorithm is O(V+E*log(V)), and our testing also show that it is very efficient in real world implementation. In our implementation, Dijkstra's algorithm takes ~0.8 sec to find a shortest path if using the entire OpenFlight dataset.

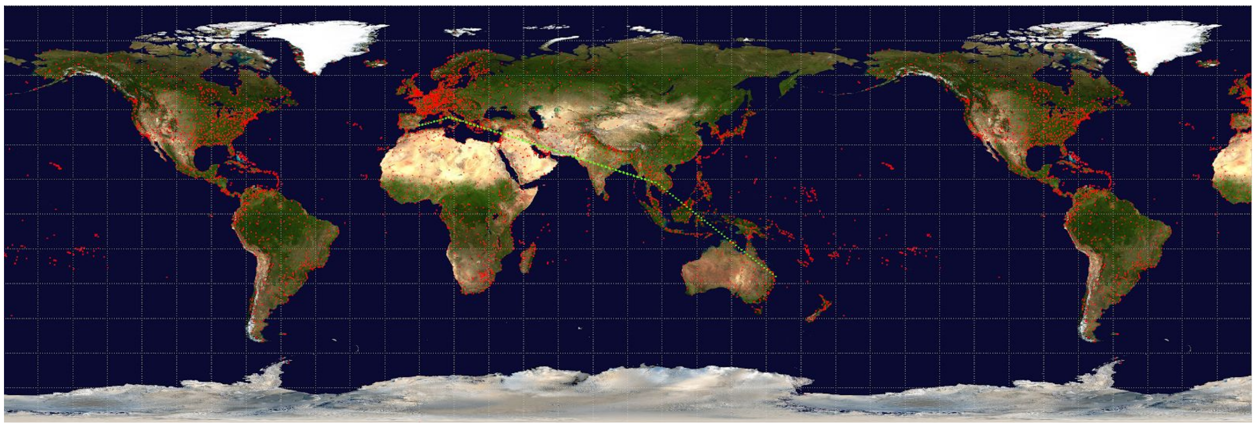
Challenges

We used Airport.txt from OpenFlights to set our vertices. However, the airport ID numbers are not fully organized in the dataset since a lot of numbers are missing. The ID ranges from 1 to 14110, but there are actually far less recorded airports in the dataset. In the end, we chose to map the Vertex with airportID directly instead of using looping through all ID numbers.

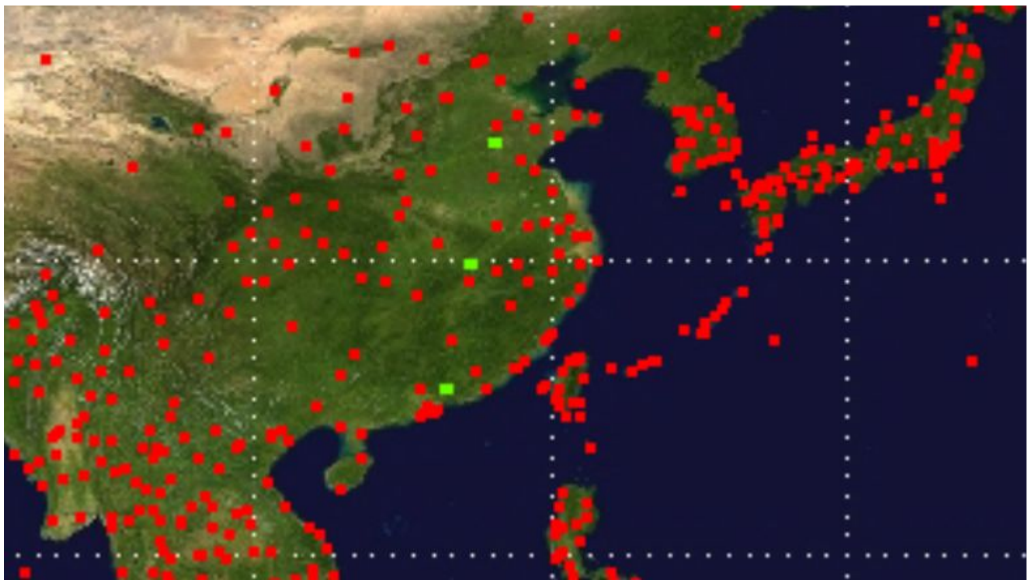
When implementing our shortest path algorithm, we also ran into some trouble understanding the implementation through Dijkstra's Algorithm, however reading 3rd-party material and discussion within the group improved our understanding. It was also difficult to directly modify the vertices in our graph, so we chose to utilize a hash table to keep track of the distances and previous vertices.

Finally, for our graphical output, line drawing was not up-to the highest standards but it served its purpose to show the routes taken from source to destination. For instance, our draw function experienced difficulty when drawing steep lines, but horizontal lines turned out to look much better.

(Good Draw)



(Bad Draw)



<u>Results</u>

For the OpenFlights database, we personally discovered that the highest number of paths using the Shortest Path algorithm from one source airport to another destination airport was fourteen paths, but it is very likely that the maximum number is much higher.

We also measured the entire time of execution for our project starting with graph creation and ending with drawing the output to a PNG. We ran a sample test case from Brisbane

International Airport to Alicante International Airport which included three paths in total. On desktop with a modern i7 processor, this took 2.425 sec, while on Macbook, the entire execution took 4.765 seconds.

We also ran the execution time of our graph construction and Shortest Path algorithm with test cases that varied in the number of paths from source to destination airport. All the tests were run on a Macbook Pro.

Number of Paths	**Time of Execution**
3	1.745 seconds
6	1.864 seconds
14	1.945 seconds

Conclusion

In the end, we successfully met all of our project goals by providing the shortest path as output string and a corresponding visual graph. We learned the usage for constructing the graph, building Dijkstra’s Algorithm, and plotting the resulting path on a world map.