For our final project, we used the OpenFlights dataset (files ‘airpots.dat’ and ‘routes.dat’) to populate our database and draw graph structures for our algorithms. The file ‘airports.dat’ provided close to 10,000 airports and ‘routes.dat’ provided 67,663 connections between those airports. Our objectives were to implement Breadth First Search (BFS), Dijkstra’s algorithm to determine the shortest path between two airports, and Kosaraju’s algorithm to determine if the airports graph is strongly connected or not. Specifically, we used Kosaraju’s algorithm to identify if an airport is capable of reaching all other airports in the dataset.

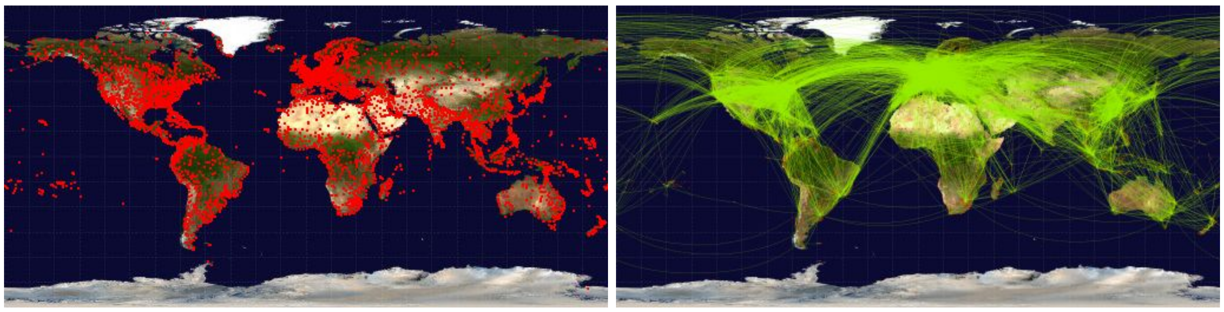
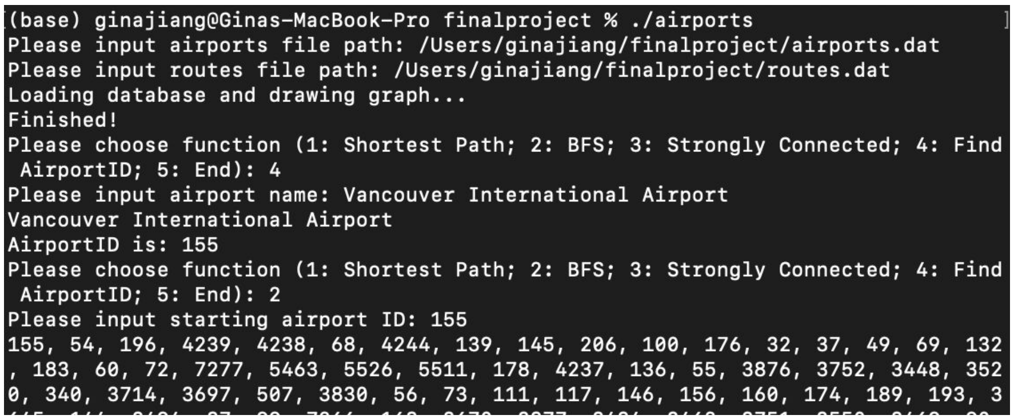


Figure 1 and 2: Left image is a visualization of all the airports in the database. The right image is a visualization of all the connections between airports in the database (https://openflights.org/data.html).

The outcome of our project met the goals we set for ourselves. Our code compiles and can execute the algorithms we detailed above. The program first prompts the user for the data files. It reads those files for airport and connection information and draws the graph to be used by the algorithms. The user can then choose which algorithm to run on the dataset.



```
(base) ginajiang@Ginas-MacBook-Pro finalproject % ./airports
Please input airports file path: /Users/ginajiang/finalproject/airports.dat
Please input routes file path: /Users/ginajiang/finalproject/routes.dat
Loading database and drawing graph...
Finished!
Please choose function (1: Shortest Path; 2: BFS; 3: Strongly Connected; 4: Find
 AirportID; 5: End): 4
Please input airport name: Vancouver International Airport
Vancouver International Airport
AirportID is: 155
Please choose function (1: Shortest Path; 2: BFS; 3: Strongly Connected; 4: Find
 AirportID; 5: End): 2
Please input starting airport ID: 155
155, 54, 196, 4239, 4238, 68, 4244, 139, 145, 206, 100, 176, 32, 37, 49, 69, 132
, 183, 60, 72, 7277, 5463, 5526, 5511, 178, 4237, 136, 55, 3876, 3752, 3448, 352
0, 340, 3714, 3697, 507, 3830, 56, 73, 111, 117, 146, 156, 160, 174, 189, 193, 3
```

Figure 1. Screenshot of executable in terminal. The database is populated with files from the OpenFlights dataset, and the user runs BFS on Vancouver International Airport after finding its ID.

BFS is an algorithm for traversing the graph from a starting vertex. For our implementation, we ask the user to input a starting airport ID to begin the traversal. If the given airport is not connected to any other airports (according to the database), it will return its own airport ID.

Dijkstra's algorithm is a method to find the shortest path between two vertices in a graph. In our program, it will return the shortest distance between two given airports in kilometers, and print out the subsequent path from the source airport to destination airport. If there is no path from the source to destination, the program will print out "Airport is not reachable."

Kosaraju's algorithm identifies whether a graph is strongly connected or not. While researching this algorithm, we learned that it uses Depth First Search (DFS) to traverse the graph. The simple algorithm uses one pass of DFS, transposes the graph, and then uses another pass of DFS. However, since we already implemented BFS, we decided to adjust the algorithm to utilize BFS. In order to use BFS, we had to ensure that the post-order property of the graph was preserved. Other than that, the steps of the algorithm were the same. Although we were still able to make it work, this situation could have been avoided if we caught it during the initial planning of our project.

All in all, our final project successfully implements the algorithms we outlined in our goals. During the development of our final project, we discovered that the OpenFlights dataset, although extensive, is not as comprehensive as we originally believed. Some airports (e.g. Chicago O'Hare International Airport) do not have any connecting flights listed for them. Therefore, the airports graph is not a strongly connected graph, and BFS only returns the initial airport ID. As a future goal, we hope to implement a visualization of our graph using a layered graph drawing or another complex graph drawing.