Results

Outcome

We successfully completed our project and met all of our goals. Our project executable is able to read the .txt file of the same format as the .txt file given on the OpenFlights.org website, and find the optimized paths from the input source to destination based on different factors, stopovers or distance. We were able to achieve an accurate output for BFS by minimizing the number of flights that can be taken from one airport to another. Hence, finding the shortest path in terms of number of stopovers. We were able to achieve an accurate output for Dijkstra's Algorithm by adding the minimum edge weight and traversing the airports along with backtracking. Lastly, we were able to create a graph which has airports as its vertices and connections as its edges to accurately visualize routes between different airports around the world.

Output

We have two types of output: one displays in the terminal and another is saved as a PNG file.

PNG Graphical Projection

The output is a graph that is written to a PNG file. When 'make' is run and the graphical projection is chosen, the output is stored in 'output-worldmap.png'. When 'make test' is run then the output test PNGs are created. It is 'output_small.png' for the small dataset (major US airports), 'output_medium.png' for a medium dataset (some major airports around the world) and 'output_large.png' for a large dataset (most airports in the world). This graph is projected onto the original map image by plotting the vertices as airports in yellow and edges as connecting routes in pink.

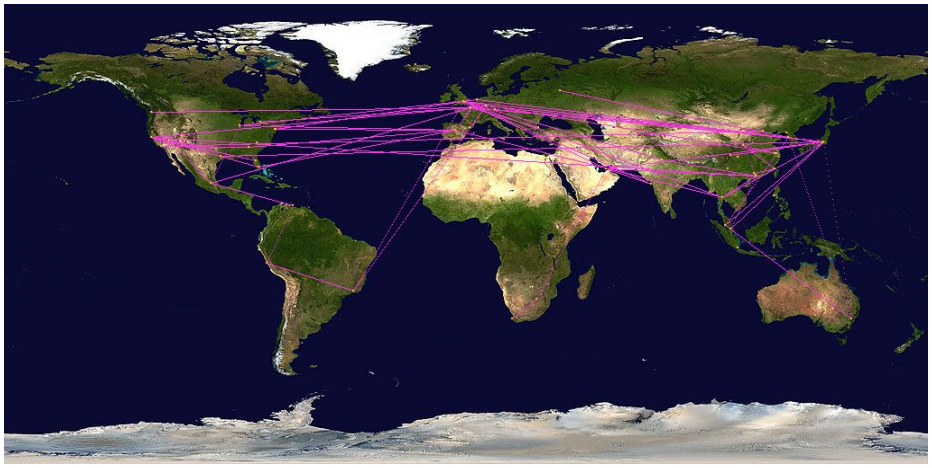

Figure 1: output_medium.png, the output of our graph creation in medium test case

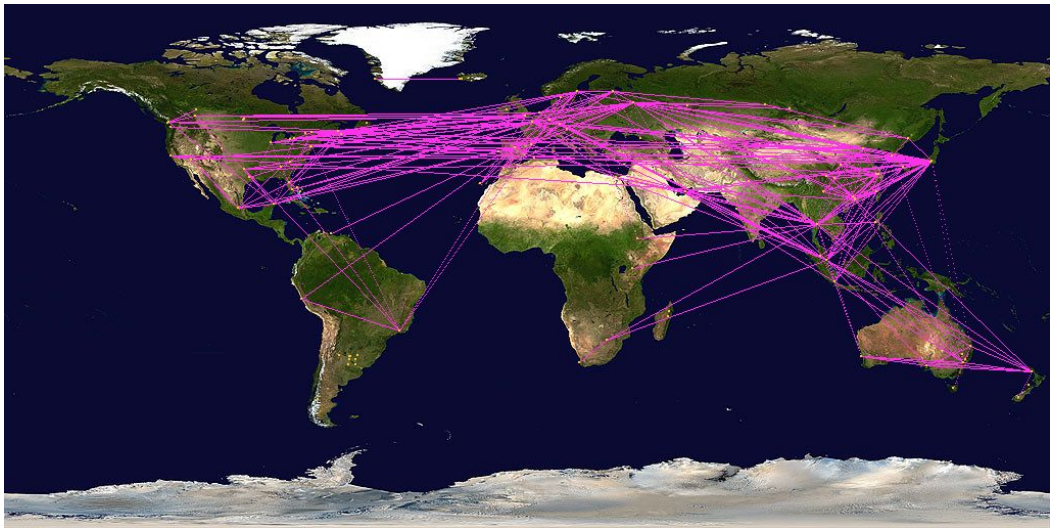

Figure 2: output_large.png, the output of our graph creation in large test case

Terminal Output for Dijkstra's Algorithm and BFS

Our terminal output asks the user for a path to the dataset file of airports and routes(of user's choosing). The user is also asked to enter the source and destination airports. For a comprehensive view of the different algorithms, we have a selection menu that allows the user to choose which algorithms to run - Graphical projection, BFS, Dijkstra, or all three. This selection menu runs in a loop until the user exits the program (with a choice number of 5). Any invalid choice is also accounted for.

For BFS, the path with the least number of stopovers and the number of flights between the source and destination node are printed in the terminal. We also print out the path from the source to the destination including all stopover airports in the terminal.

```
Input source airport code:
DFW

Input destination airport code:
BOM
 -- BFS Implementation to print the path with minimum number of stopovers --

Number of flights to take: 2
Number of stopovers: 1
Path: DFW -> AMS -> BOM
```

Figure 3: the terminal output for BFS (using data/airports.txt and data/routes.txt)

For Dijkstra's algorithm, the path with the least distance is printed out from source to destination. We also print out the total distance between the input source and destination by adding up the edge weights of the respective edges travelled.

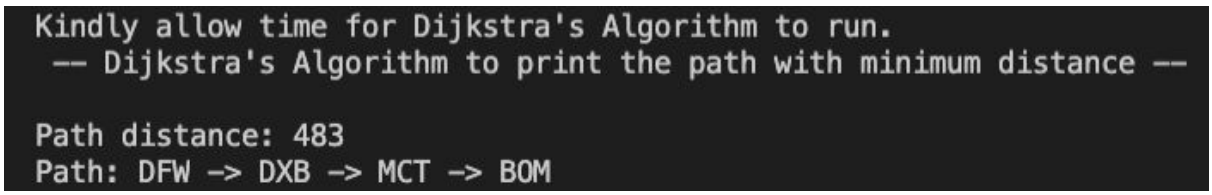

Figure 4: the terminal output for Dijkstra (using data/airports.txt and data/routes.txt

Discovery

After running and testing our program several times, we discovered some interesting facts. First, we noticed that going through busy airports which have multiple flights does not always give us the shortest path. For example, to go to Bombay from Dallas, transiting through Cairo is actually shorter than going through LHR, which is one of the busiest airports in the world (DFW→ JFK → CAI → BOM is shorter than DFW → JFK → LHR → BOM).

Additionally, we discovered that the path with the shortest distance between two of seemingly popular airports, O'hare international airport to Suvarnabhumi airport in Bangkok, actually requires 7 stopovers (ORD → JFK → CAI → DMM → DWC → MCT → BOM → HYD → BKK).

We also found a path with the shortest distance using Dijkstra's Algorithm with as many as 11 stopovers, going from Calgary, Canada to Khon Kaen, Thailand. (YYC → MSP → GRR → DTW → JFK → CAI → DMM → DWC → MCT → BOM → HYD → BKK→ KCC).

When we were trying to create the large test case for our graphical output, while ensuring that our resulting PNG will not look too messy and its routes are spread out to different continents, we realized that Africa and South America are the two continents with the least number for airports (excluding Antarctica) with respect to their continents area. This can also be verified by our graphical output created using the entire dataset. By looking at the graphical output, we also observed that although there are airports in Antarctica, there is no route to any of the airports there.

While creating the 'no path' test case in BFS, we attempted to find an airport with minimal number of connections, we found out about Pyongyang Sunan International Airport in North Korea which has only ten connections to four distinct airports. This means that if a certain airport is not connected to any of those four airports - VVO, KUL, PEK, or SHE - no flights would be able to fly to North Korea at all.

Another interesting fact is that even though we handled ICAI code inputs, the routes.txt datafile was found to not use any 4-letter ICAI codes.

In conclusion, we successfully met our project goals. Doing the project gives us the opportunity to apply the knowledge of data structures we learnt throughout the semester into creating a functioning deliverable.