Isaac Tong
Irfaan Attarwarla
Eric Jin
Pranshu Chaturvedi

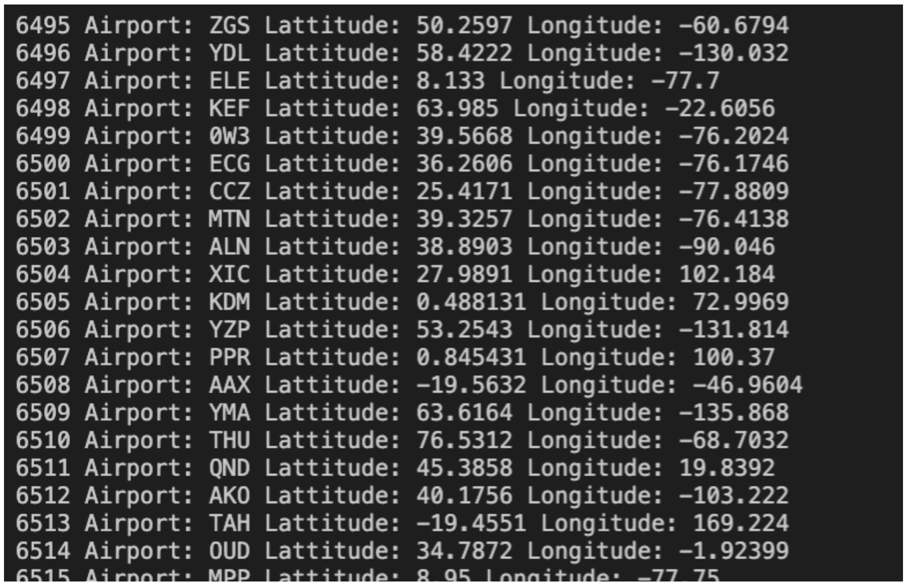
CS225 Final Project – RESULTS

Dataset Usage:

We used the OpenFlights dataset. Specifically, we used the "routes.dat" and "airlines-extended.dat" datasets. Within "routes.dat" there were 67663 airplane routes, which connected airports. Within "airports-extended.dat", there were 12668 transportation modes (including rail etc.), although we only used the data relating airports. We kept "routes.dat" the same while we formatted the structure of "airports-extended.dat" to clean the data. Specifically, we only included the relevant columns of airline, starting airport, and ending airport, and discarded unnecessary information.

Airports and Graph Printing:

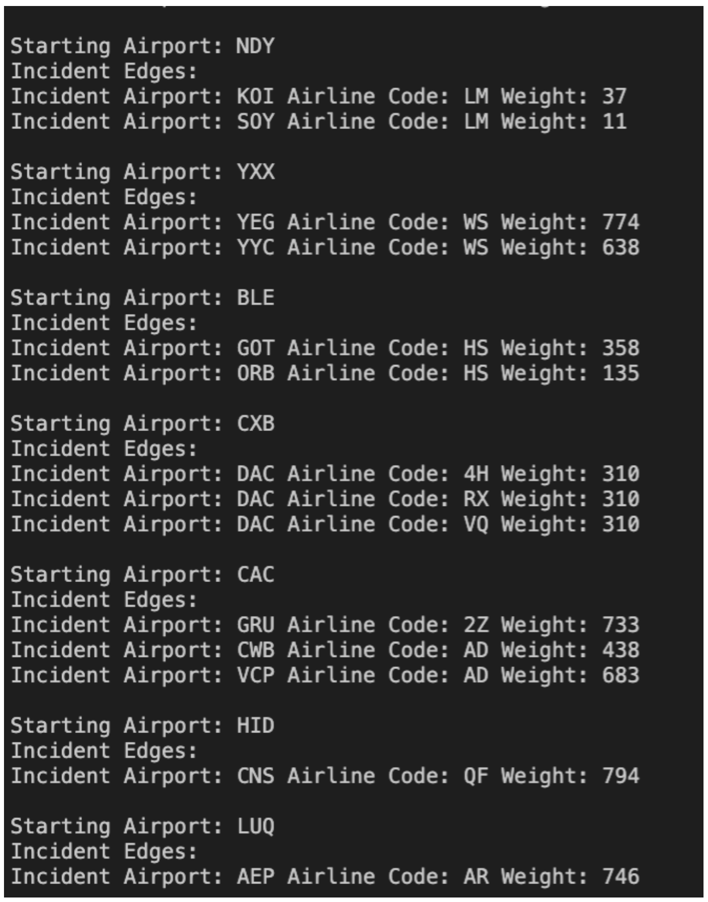
We created functions to print the airports in the dataset as well as the graph. These are printed to the console.



```
6495 Airport: ZGS Lattitude: 50.2597 Longitude: -60.6794
6496 Airport: YDL Lattitude: 58.4222 Longitude: -130.032
6497 Airport: ELE Lattitude: 8.133 Longitude: -77.7
6498 Airport: KEF Lattitude: 63.985 Longitude: -22.6056
6499 Airport: 0W3 Lattitude: 39.5668 Longitude: -76.2024
6500 Airport: ECG Lattitude: 36.2606 Longitude: -76.1746
6501 Airport: CCZ Lattitude: 25.4171 Longitude: -77.8809
6502 Airport: MTN Lattitude: 39.3257 Longitude: -76.4138
6503 Airport: ALN Lattitude: 38.8903 Longitude: -90.046
6504 Airport: XIC Lattitude: 27.9891 Longitude: 102.184
6505 Airport: KDM Lattitude: 0.488131 Longitude: 72.9969
6506 Airport: YZP Lattitude: 53.2543 Longitude: -131.814
6507 Airport: PPR Lattitude: 0.845431 Longitude: 100.37
6508 Airport: AAX Lattitude: -19.5632 Longitude: -46.9604
6509 Airport: YMA Lattitude: 63.6164 Longitude: -135.868
6510 Airport: THU Lattitude: 76.5312 Longitude: -68.7032
6511 Airport: QND Lattitude: 45.3858 Longitude: 19.8392
6512 Airport: AKO Lattitude: 40.1756 Longitude: -103.222
6513 Airport: TAH Lattitude: -19.4551 Longitude: 169.224
6514 Airport: OUD Lattitude: 34.7872 Longitude: -1.92399
6515 Airport: MPP Lattitude: 8.95 Longitude: -77.75
```

Fig 1. Sample output of the graph using printAirports() function.

As shown in figure 1. and from left to right, we print the airport number relative to the starting data of the dataset, the International Air Transport Association (IATA) airport code, and its latitude as well as longitude. This essentially prints the vertices of the graph and relevant metadata pertaining to these vertices.



```
Starting Airport: NDY
Incident Edges:
Incident Airport: KOI Airline Code: LM Weight: 37
Incident Airport: SOY Airline Code: LM Weight: 11

Starting Airport: YXX
Incident Edges:
Incident Airport: YEG Airline Code: WS Weight: 774
Incident Airport: YYC Airline Code: WS Weight: 638

Starting Airport: BLE
Incident Edges:
Incident Airport: GOT Airline Code: HS Weight: 358
Incident Airport: ORB Airline Code: HS Weight: 135

Starting Airport: CXB
Incident Edges:
Incident Airport: DAC Airline Code: 4H Weight: 310
Incident Airport: DAC Airline Code: RX Weight: 310
Incident Airport: DAC Airline Code: VQ Weight: 310

Starting Airport: CAC
Incident Edges:
Incident Airport: GRU Airline Code: 2Z Weight: 733
Incident Airport: CWB Airline Code: AD Weight: 438
Incident Airport: VCP Airline Code: AD Weight: 683

Starting Airport: HID
Incident Edges:
Incident Airport: CNS Airline Code: QF Weight: 794

Starting Airport: LUQ
Incident Edges:
Incident Airport: AEP Airline Code: AR Weight: 746
```

Fig 2. Sample output of the graph using printGraph() function.

As shown in figure 2, we can also print the graph. The sample output shows that we print the starting airport IATA code (vertices) and all the outgoing routes from the airport (edges), which includes the destination airport IATA code, the airline code, and the weight of the edge, which is the distance in kilometers between the two airports.

<u>Graphical Output:</u>

Our graphical output produces a visualization of the airports and routes onto a Mercator projection of the world map. For each vertex of the graph, or airport, we created a magenta square. The graphical output is saved to “output.png”. This square is located onto the map based on its longitude and latitude. Moreover, edges, or routes, were represented as yellow lines that links a starting airport to an ending airport. This is shown in figure 3.

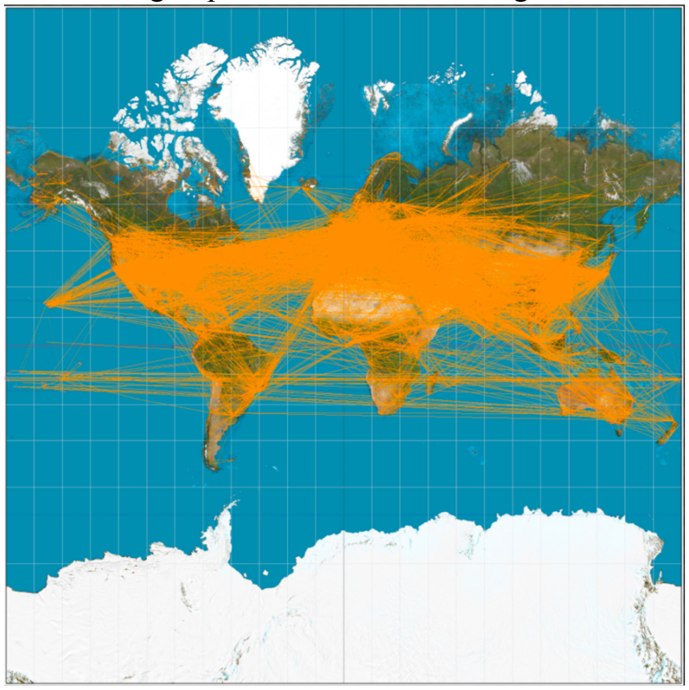


Fig 3. Graphical output using the entire routes and airports datasets.

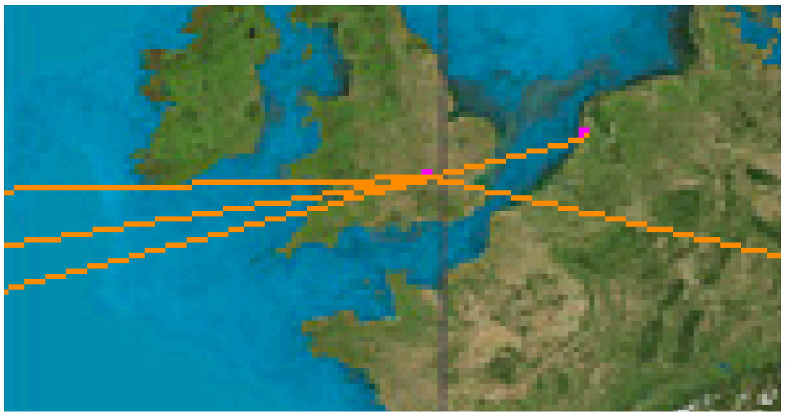


Fig 4. Closeup of the graphical output using a smaller dataset.

Figure 4 shows a closeup of the graphical airport and depicts many routes extending from London Heathrow Airport to other airports including Amsterdam Airport Schiphol to its immediate right.

As seen, the graphical output tends to get messy with tens of thousands of airport routes. Below is a clearer version with a subset of the data:

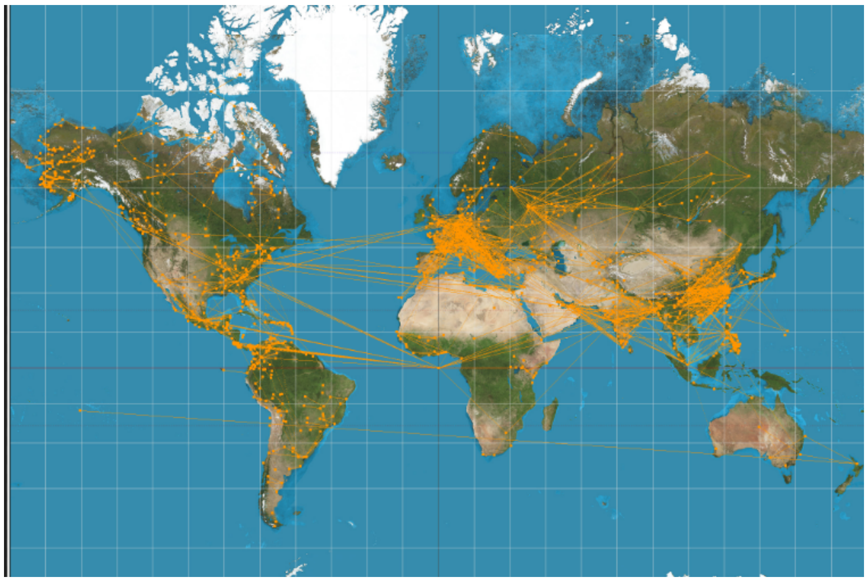


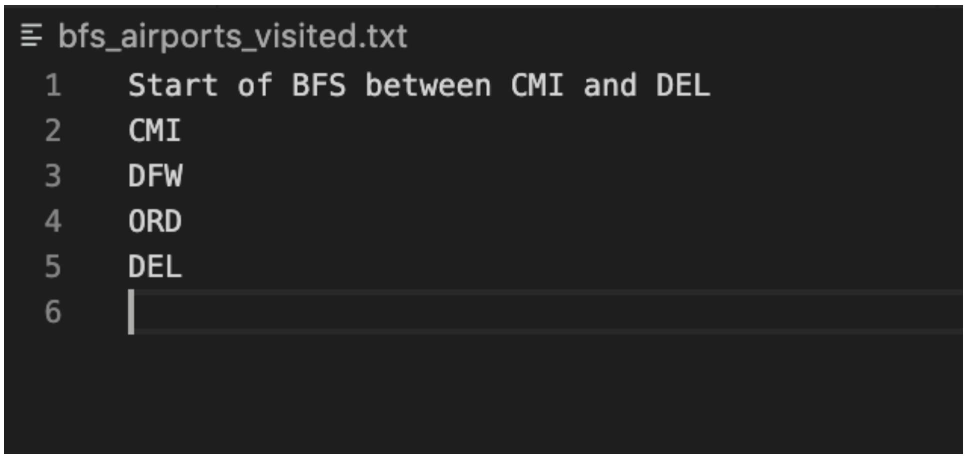
Fig 5. Graphical output run on a subset of OpenFlights data

This subset of data was randomly chosen from the entire OpenFlights dataset and so the distribution of airports and routes should be somewhat proportional to the original dataset. As shown, large clusters of airports and routes form in areas like North America, Europe and Southeast Asia.

To facilitate the creation of graphical outputs using different sizes of datasets, we created a flightsVisualizer class. This class takes in a Graph object and thus can produce a graphical output of varying complexity onto the world map based on the dataset used to produce the Graph object.

Breadth First Search (BFS) Algorithm Results:

The purpose of the BFS algorithm is to determine if a path exists between two airports and what airports are visited during the algorithm search path. The results are written to “bfs_airports_visited.txt”.


bfs_airports_visited.txt

```
Start of BFS between CMI and DEL
CMI
DFW
ORD
DEL
```

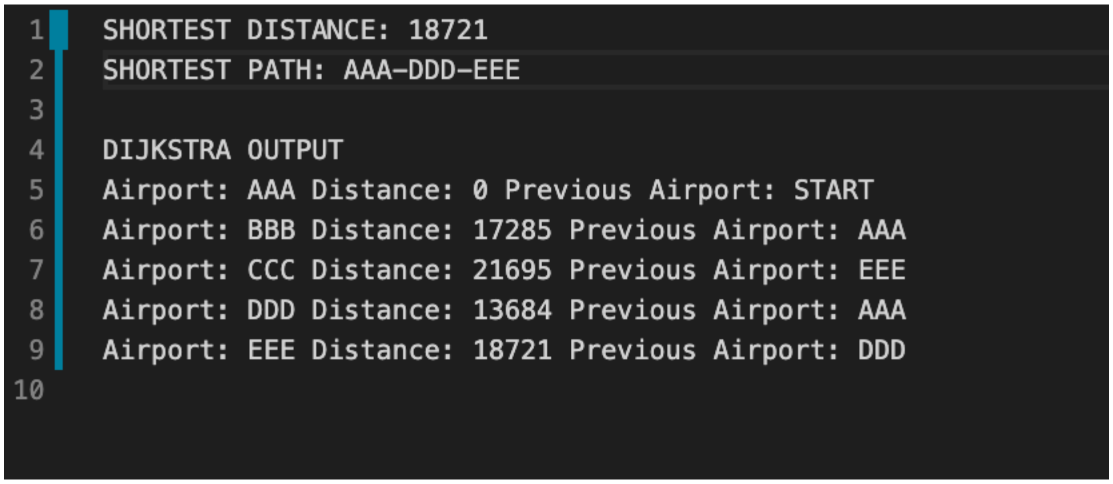
Fig 6. The output of the BFS algorithm, displaying the visited nodes from airport CMI to airport DEL

As shown in figure 6, the BFS finds a search path between the airport CMI and DEL. Lines 2 to 5 in figure 6 indicate the airports visited by the BFS algorithm from CMI to DEL. As shown there exists a path between CMI to DEL. If there is no path from the start and target airport, then the BFS algorithm implementation will still show the airports visited in the text file but the last line will not match the destination airport and thus all connected airports of the graph are searched and no path is found.

Dijkstra's Algorithm Results:

Dijkstra's algorithm is used to find the shortest path between the start airport and a target airport as well as determine the total distance travelling through the path, or edge weight. The result of this algorithm is outputted to "DijkstraOutput.txt".

For demonstration purposes, figure 5 shows Dijkstra's Algorithm running on a dataset with 5 hypothetical airports with routes connecting them.



```
1   SHORTEST DISTANCE: 18721
2   SHORTEST PATH: AAA-DDD-EEE
3
4   DIJKSTRA OUTPUT
5   Airport: AAA Distance: 0 Previous Airport: START
6   Airport: BBB Distance: 17285 Previous Airport: AAA
7   Airport: CCC Distance: 21695 Previous Airport: EEE
8   Airport: DDD Distance: 13684 Previous Airport: AAA
9   Airport: EEE Distance: 18721 Previous Airport: DDD
10
```

Fig 7. Dijkstra's output on the five airport dataset.

As shown in figure 7, our program outputs the shortest distance between the starting and target airports in kilometers, the shortest path connecting the airports, as well as the Dijkstra's Algorithm calculations of the entire graph. In this hypothetical example, the output shows the paths "AAA" (indicated by START) to the target airport "EEE". "SHORTEST DISTANCE" and "SHORTEST PATH" only pertain to the start and target airports. However, all paths and distances between every airport and the starting airport are calculated.

For example, in line 7 of figure 5., the destination airport is "CCC", its distance to the starting airport "AAA" is 21695km and the previous airport in the path from starting airport "AAA" to "CCC" is "EEE". If you continue to follow the previous path, you will see that the entire path to "CCC" is: AAA-DDD-EEE-CCC. This was only a hypothetical case, but our use of Dijkstra's algorithm extends to the entire dataset of airports and routes.