CS 225 Final Project: Take Flight Results

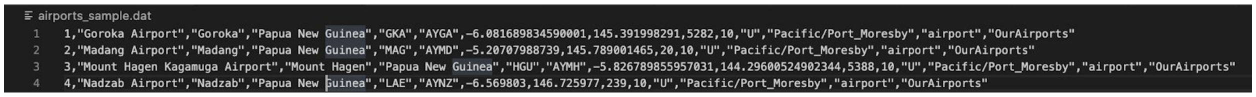
Overview of Project

Our final project, Take Flight, takes in a dataset of airports and routes from the OpenFlights dataset to create a graph of airports and utilise a breadth-first search traversal to find the shortest route between two given airports and the landmark path between three given airports.

Processing the Data Set

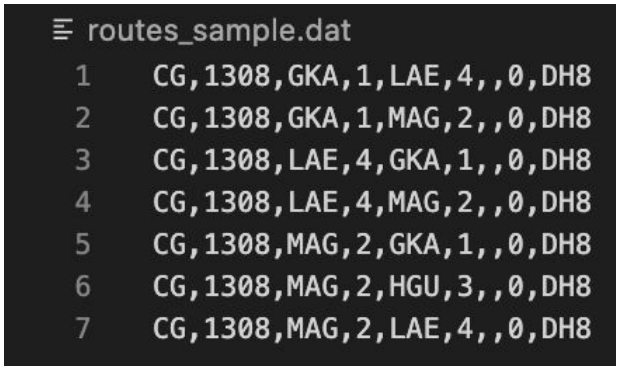
Given a dataset of airports, each airport is turned into an Airport class allowing for easy retrieval of the airport's ID, name, longitude and latitude. Each airport is then added as a vertex to a graph to be traversed. We then connected the vertices according to the possible routes given in the routes dataset. Each connection (edge) between two airports (vertices) in the graph is given a weight that indicates the distance between the two airports.

Using a subset from the airports and routes dataset from the OpenFlights dataset as shown in Figure 1 and 2, we can connect each airport using the given routes and create a graph of airports and connections. Our expected graph for this subset is shown in Figure 3. We can check the graph in our program by printing out the neighbours of each airport vertex in the graph. As shown in Figure 4 the graph output is equal to our expected graph.



```
airports_sample.dat
1,"Goroka Airport","Goroka","Papua New Guinea","GKA","AYGA",-6.081689834590001,145.391998291,5282,10,"U","Pacific/Port_Moresby","airport","OurAirports"
2,"Madang Airport","Madang","Papua New Guinea","MAG","AYMD",-5.20707988739,145.789001465,20,10,"U","Pacific/Port_Moresby","airport","OurAirports"
3,"Mount Hagen Kagamuga Airport","Mount Hagen","Papua New Guinea","HGU","AYMH",-5.826789855957031,144.29600524902344,5388,10,"U","Pacific/Port_Moresby","airport","OurAirports"
4,"Nadzab Airport","Nadzab","Papua New Guinea","LAE","AYNZ",-6.569803,146.725977,239,10,"U","Pacific/Port_Moresby","airport","OurAirports"
```

Figure 1: Subset of Airports from Airports Dataset



```
routes_sample.dat
CG,1308,GKA,1,LAE,4,,0,DH8
CG,1308,GKA,1,MAG,2,,0,DH8
CG,1308,LAE,4,GKA,1,,0,DH8
CG,1308,LAE,4,MAG,2,,0,DH8
CG,1308,MAG,2,GKA,1,,0,DH8
CG,1308,MAG,2,HGU,3,,0,DH8
CG,1308,MAG,2,LAE,4,,0,DH8
```

Figure 2: Subset of Routes from Routes Dataset

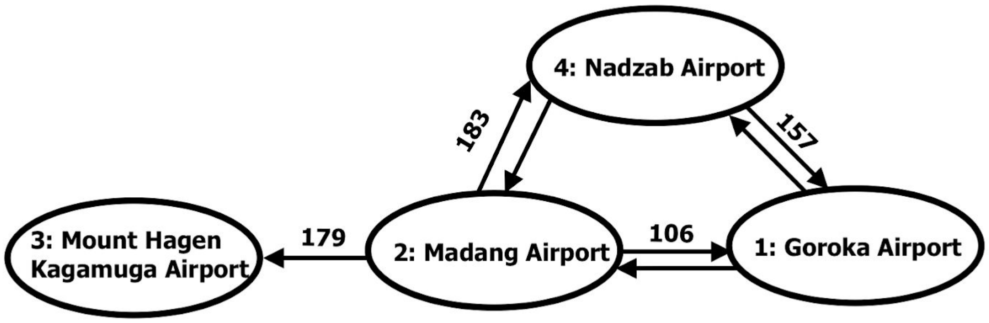
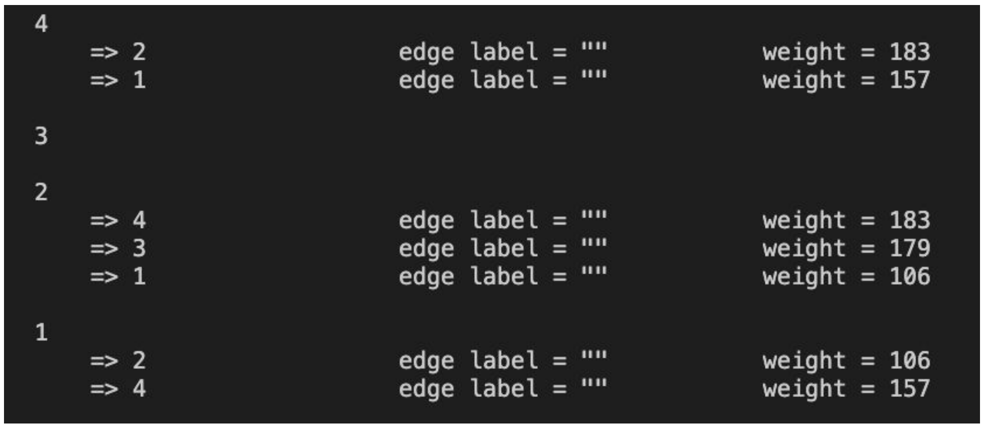


Figure 3: Expected Graph from Airports and Routes Subset



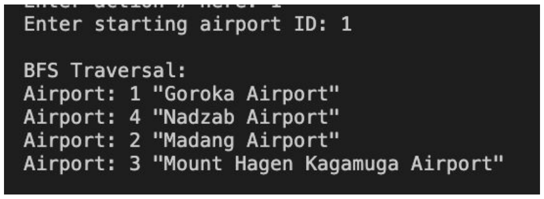
```
4
    => 2                    edge label = ""          weight = 183
    => 1                    edge label = ""          weight = 157

3

2
    => 4                    edge label = ""          weight = 183
    => 3                    edge label = ""          weight = 179
    => 1                    edge label = ""          weight = 106

1
    => 2                    edge label = ""          weight = 106
    => 4                    edge label = ""          weight = 157
```

Figure 4: Program Output of Created Graph from Airports and Routes Subset

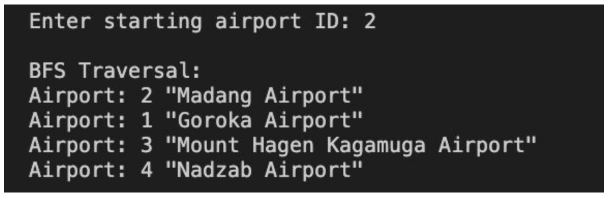
Traversing the Graph: BFS Traversal

Using the created graph of airports and connections, we can traverse the graph using a breadth-first search traversal. The breadth-first search traversal first starts with the starting airport (root). Then each neighbouring airport connected to the starting airport on a given level is traversed. This traversal through each connected neighbouring airport is continued on each level until all the connected airports have been visited once.

From our graph created using our sample subset data, we can see the output of each breadth-first search traversal of all the airports in the graph shown in Figure 5 and 6.

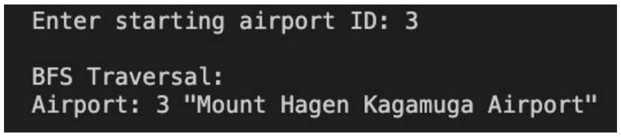


```
Enter starting airport ID: 1

BFS Traversal:
Airport: 1 "Goroka Airport"
Airport: 4 "Nadzab Airport"
Airport: 2 "Madang Airport"
Airport: 3 "Mount Hagen Kagamuga Airport"
```

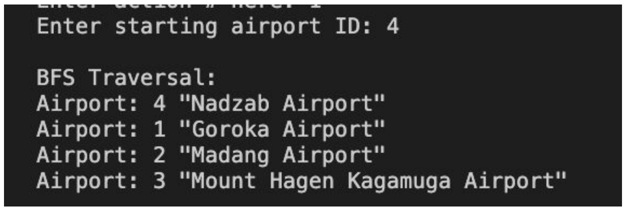


```
Enter starting airport ID: 2

BFS Traversal:
Airport: 2 "Madang Airport"
Airport: 1 "Goroka Airport"
Airport: 3 "Mount Hagen Kagamuga Airport"
Airport: 4 "Nadzab Airport"
```

Figure 5: Output of BFS Traversal of Airports 1 and 2



```
Enter starting airport ID: 3

BFS Traversal:
Airport: 3 "Mount Hagen Kagamuga Airport"
```



```
Enter starting airport ID: 4

BFS Traversal:
Airport: 4 "Nadzab Airport"
Airport: 1 "Goroka Airport"
Airport: 2 "Madang Airport"
Airport: 3 "Mount Hagen Kagamuga Airport"
```

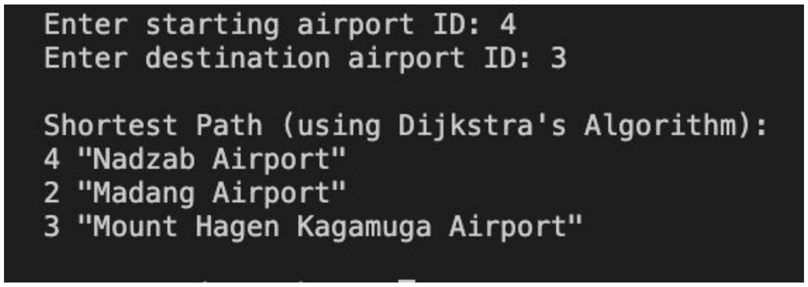
Figure 5: Output of BFS Traversal of Airports 3 and 4

Since the Airport 3 does not have any routes to go to any other airports, its traversal is empty. However, since Airport 2 has a route to go to Airport 3, the traversals of the other airports contain Airport 3.

Finding the Shortest Path: Dijkstra's Algorithm

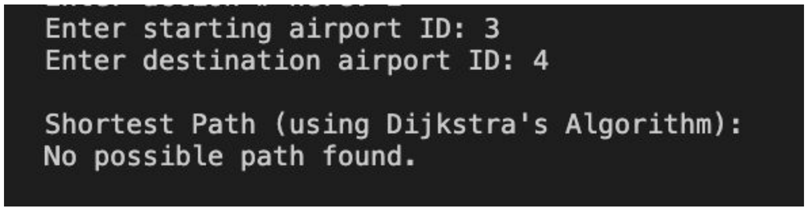
To find the shortest path between two airports, we implemented Dijkstra's Algorithm. We can use a queue to keep track of every visited airport and the distances to subsequently find the shortest path. While the queue is not empty, we iterate through every neighbour of the current airport to find the distance between the current airport and its neighbouring airport. If the distance between the two airports is less than the current distance stored, we insert the neighbour into the queue to check in the future.

Using the graph of the sample subset of airport and route data, we find the shortest path between any two given airports in the graph. Figure 6 shows the output of the shortest path from Airport 4 to 3. When reversing this route, however, where we want to find the shortest path from Airport 3 to 4, there is no shortest path found (as shown in Figure 7). This is because there are no routes going out of Airport 3 as given in the subset of routes data we used.



```
Enter starting airport ID: 4
Enter destination airport ID: 3

Shortest Path (using Dijkstra's Algorithm):
4 "Nadzab Airport"
2 "Madang Airport"
3 "Mount Hagen Kagamuga Airport"
```

Figure 6: Output of Shortest Path Between Airports 4 and 3



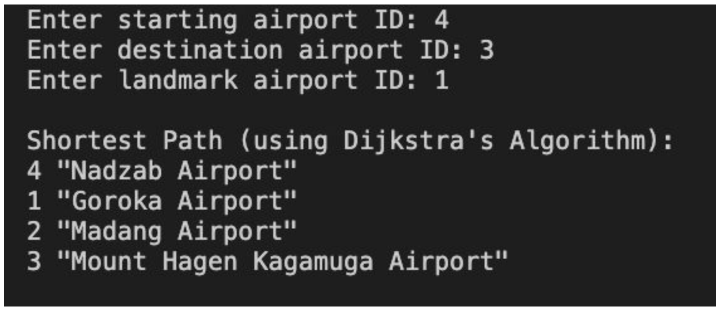
```
Enter starting airport ID: 3
Enter destination airport ID: 4

Shortest Path (using Dijkstra's Algorithm):
No possible path found.
```

Figure 7: Output of Shortest Path Between Airports 3 and 4

Finding the Landmark Path

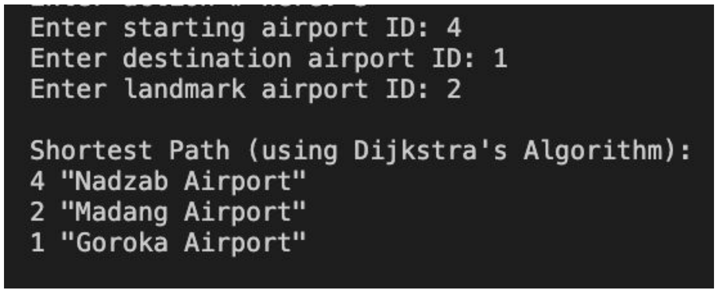
The landmark path is the shortest path between two airports that goes through a given landmark airport. To find the landmark path, we used a priority queue that stores the remaining airports to visit ordered by distance. We used the Dijkstra Algorithm to find the shortest path between the starting airport and landmark airport and the landmark airport and destination airport to find the shortest route between two airports that goes through the landmark airport.

Using the graph of the sample subset of airport and route data, we can find the shortest path between any two given airports in the graph that goes through the landmark airport. We can see the sample outputs of the landmark paths in Figure 8, 9, and 10.



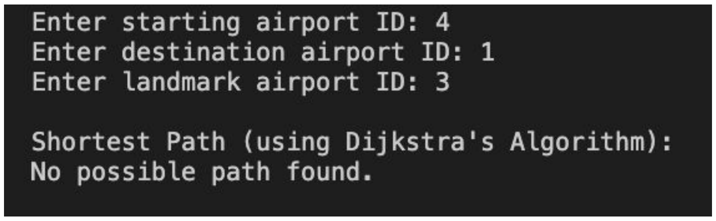
```
Enter starting airport ID: 4
Enter destination airport ID: 3
Enter landmark airport ID: 1

Shortest Path (using Dijkstra's Algorithm):
4 "Nadzab Airport"
1 "Goroka Airport"
2 "Madang Airport"
3 "Mount Hagen Kagamuga Airport"
```

Figure 8: Output of Landmark Path between airport 4 to 1 through airport 3



```
Enter starting airport ID: 4
Enter destination airport ID: 1
Enter landmark airport ID: 2

Shortest Path (using Dijkstra's Algorithm):
4 "Nadzab Airport"
2 "Madang Airport"
1 "Goroka Airport"
```

Figure 9: Output of Landmark Path between airport 4 to 1 through airport 2



```
Enter starting airport ID: 4
Enter destination airport ID: 1
Enter landmark airport ID: 3

Shortest Path (using Dijkstra's Algorithm):
No possible path found.
```

Figure 10: Output of Landmark Path between airport 4 to 3 through airport 1

Final Results & Discoveries

We made the results of our project displayable via terminal by providing the user with different input options for what they want to see (i.e. BFS Traversal, Shortest Path between two airports, Landmark Path between three airports, Graph, etc.), and printing the corresponding output.

For us, it was interesting to see which airports were connected to one another, which airports didn't have routes, and which airports did have routes yet could not reach one another. For example, #762 "Fassberg Air Base" has no connected routes. On the other hand, #1 "Goroka Airport" & #5439 "Neerlerit Inaat Airport" both have routes, but neither can be used to reach the other.

Although the routes data we had was outdated, it was exciting to think that the algorithms we implemented could be applied in real life to quickly find the fastest routes to take to get from one place to another. If we expanded upon this project in the future, we could add more practicality to the program by incorporating more data, such as information about airlines, to provide the user with more detailed information about what flights they should take to get to their desired location.