INTRODUCTION

Our main goal was to create a data structure that holds the airports and routes between them and then implement algorithms that would help us compute the shortest route one could take from Airport A to Airport B.

The first algorithm we implemented on our dataset was BFS, or the Breadth First Search Algorithm. Our implementation uses a queue and given a starting source airport ID, the algorithm creates a graph structure with all the destination airports that are reachable from the source airport (meaning it outputs all airports connected to the source). Additionally, from our implementation we are able to determine the total number of routes it takes to reach each valid destination airport from our starting source node. This means we would be able to display that all airports in our graph are within X edges, or routes, of the starting nodes, or airports, that we provide.

BFS ALGORITHM RESULTS AND OBSERVATIONS

The BFS algorithm successfully traverses through the dataset to create a graph structure showing the pathways described by the route and airport datasets. When running the algorithm on our full dataset, we obtain a written "BFS.txt" file showing the location of several airports from the selected source airport, ordered in the number of routes from said airport. A visual representation of our algorithm is shown in this text file, located in "BFS.txt", and is updated every time BFS is run on the dataset when picking a starting airport. After implementing this algorithm on our dataset, we found some interesting results. For example, we observed that Source airport ID: 2276 which is associated with the airport "Taiwan Taoyuan International Airport" The maximum number of routes it would take to reach any destination airport that has a path connected to this airport would be 7. Additionally, after running BFS implementation on several other source ID airports, we noticed that the number of destination airports that are 2 or 3 flight stops away tends to make up the highest proportion of the number of destination airports. This can been seen as true even for source airport ID: 2276, which we observed that in the data from "BFS.txt":

102 destination airports that are 1 flight stop away

1066 destination airports that are 2 flight stops away

1054 destination airports that are 3 flight stops away

348 destination airports that are 4 flight stops away

61 destination airports that are 5 flight stops away

15 destination airports that are 6 flight stops away

4 destination airports that are 7 flight stops away

As shown in the data above, destination airports that are 2 or 3 flight stops away from the source airport make up a significantly larger percentage compared to those of other amounts of flight stops.

To test our BFS implementation, we created multiple test cases of graphs such as a linear path, a circular cyclic path, a tree, as well as a K4 dense graph among others. In all of these cases, we would test that only destination nodes that are connected to the source node are set to be visited and that the edges of the graph are correctly set to be undiscovered, discovery edges, and cross edges.

DIJKSTRA'S ALGORITHM RESULTS AND OBSERVATIONS

The second algorithm we implemented on our dataset was Dijkstra's Algorithm. Our implementation calculates all possible shortest paths from one starting airport to all destination airports that are reachable, or has a valid path present between them. In the implementation of our algorithm, we use a map of paths that uses the airport ID as its key with a mapped value of a two dimensional vector storing all the possible shortest paths (each path being a vector storing the order of airports traversed) starting from a source airport ID of our choice to all the vertices present in our graph structure. We can also use the map output from our implementation to find all shortest possible paths from a given source ID to a given destination airport ID of our choice.

Our implementation of Dijkstra's algorithm was successful in calculating the shortest paths from a starting airport to all destination airports assuming there was a valid path between them. We are able to generate visual representations of our algorithm implementation in the text file,"dijkstra.txt", with an example for all shortest paths from source airport ID: 50, associated with "Arviat Airport". Below, we show a screenshot of paths connected to two of the possible destination airports from source airport 50. Upon examining our data, an interesting fact that we found was that the direct flight was not always the shortest distance path between two airports- sometimes connecting flights span a shorter distance. In addition to this, sometimes connecting flights actually add up to the same distance span as a direct flight between two airports. This was illuminating to see because we were under the assumption that direct flights would always be the shortest path, but the data shows otherwise.

```
1   ALL shortest paths from source airport: 50
2
```

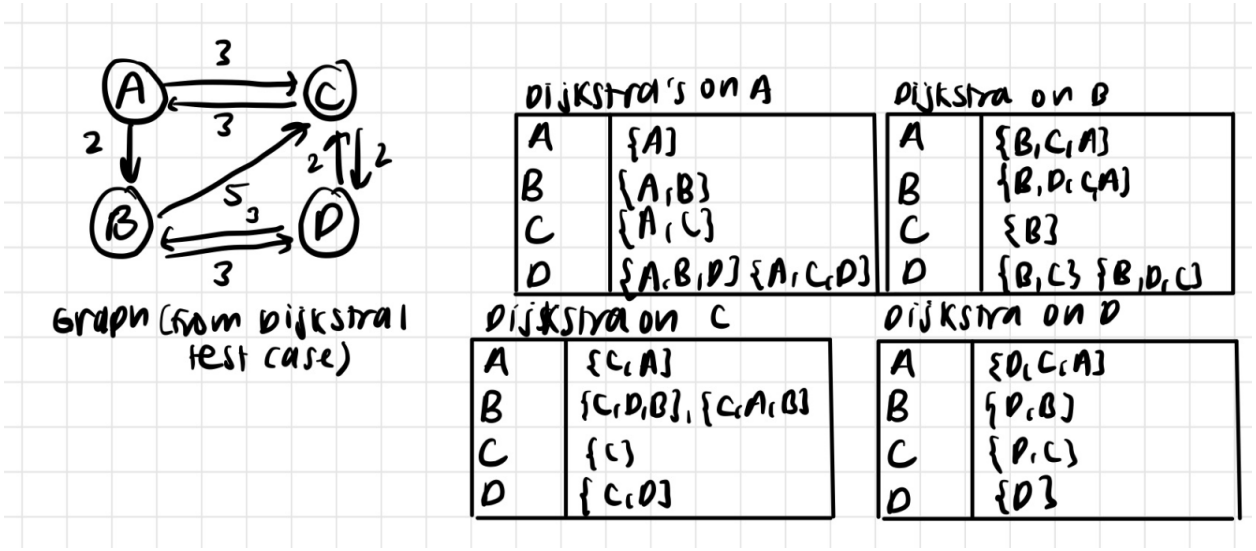
…

```
914   destination: 625
915       path: 50,187,160,193,16,636,625,
```

…

```
3117   destination: 2195
3118       path: 50,132,55,154,80,161,188,189,507,340,1682,11051,2195,
3119       path: 50,132,55,154,80,161,188,125,189,507,340,1682,11051,2195,
3120       path: 50,132,55,154,80,161,188,189,507,302,340,1682,11051,2195,
3121       path: 50,132,55,154,80,161,188,125,189,507,302,340,1682,11051,2195,
3122       path: 50,132,55,154,80,161,188,189,507,2057,11051,2195,
3123       path: 50,132,55,154,80,161,188,125,189,507,2057,11051,2195,
3124       path: 50,132,55,154,80,161,188,189,507,340,2057,11051,2195,
3125       path: 50,132,55,154,80,161,188,125,189,507,340,2057,11051,2195,
3126       path: 50,132,55,154,80,161,188,189,507,302,340,2057,11051,2195,
3127       path: 50,132,55,154,80,161,188,125,189,507,302,340,2057,11051,2195,
```

To test our implementation of Dijkstra's Algorithm, we created a couple simple tests such as the one below, which makes sure that even if there are multiple shortest paths for a source to a destination, both paths are conserved. This graph specifically is the graph for the test case "dijkstra 1", which shows that all shortest path from Dijkstra's are easily seen as correct in the test case assertions.



Additionally, we had another test involving a graph in which for some source nodes, not all other nodes had reachable paths and thus were not destination nodes for that source. This means that the vector of paths in their mapped value would be a size of 0 in this case.

BETWEENNESS CENTRALITY ALGORITHM RESULTS AND OBSERVATIONS

The last algorithm that we implemented was the Betweenness Centrality. It is a measure of centrality in a graph based on shortest paths, that is, for every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that the sum of the weights of the edge is minimized. The betweenness centrality for each vertex can be calculated according to the below equation:

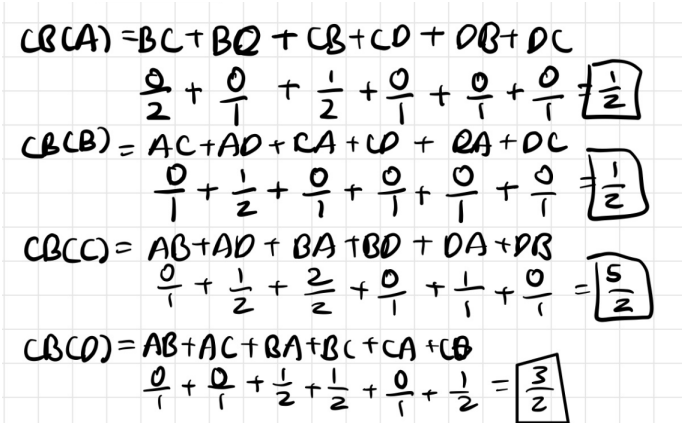
Betweenness (BC) $\sum_{s,t \in V} \frac{\sigma(s,t|u)}{\sigma(s,t)}$ fraction of shortest paths between all other nodes that run through node *u*

In our algorithm implementation, given a centrality node, we used Dijkstra's in order to compute all the possible shortest paths of each of the pairs that do not include the centrality node. The total number of shortest paths between a pair would give us the denominator and the number of times any of paths cross our centrality node would give us our numerator value for that pair. Adding all the pairs of numerators divided by denominator values calculated would result in the centrality value for our chosen source airport.

If a centrality value for a particular source airport ID is less than another source airport ID then this would mean that the number of shortest paths that pass through the centrality node of the source ID with a lower centrality node value would be lower than the source ID with a higher centrality node value. For example,

when running our algorithm on “routes_limited.txt”, we found that the centrality of source airport 2275 is 342, and 0 for any other airports, which makes sense because that specific routes dataset involves only routes relation to source airport 2275.

To test our implementation, we also created test cases using the same graphs as the ones for the previous Dijkstra’s test cases. For example, here is the result of the betweenness centrality value of each of the individual nodes in the graph previously mentioned above. In the test cases we made sure that the output of the betweenness centrality would match the value as we manually calculated below.



Although not pictured, one other edge test case graph we did have to consider was one in which there were some pairs of nodes that are not connected by a path. The denominator would become 0, so we made sure that in our implementation, that the pair would not be added to the centrality completely to avoid dividing by 0 error.

CONCLUSION

We set out to make a program that uses the OpenFlights airports and routes datasets to determine the shortest path from one airport to another, which is a simple mission. Over the course of the project, by successfully implementing all three of our chosen algorithms with a pulled and parsed dataset, we realized how complex data analysis really is. Through BFS, we were able to create a graph structure that contained every airport and routes connected to that airport. Through Dijkstra’s, we were able to successfully traverse our graph structure in order to determine the shortest paths from a source airport to any destination airport, given there was a path between them. Through Betweenness Centrality, we were able to determine the fraction of shortest paths that run through a given airport. Combining all the knowledge from our algorithms, we were able to successfully complete our original mission of being able to find the shortest path from one airport to another. In this process, we also learned a lot about the data we were using as well- we learned interesting facts such as realizing that the shortest path is not always a direct flight as we previously assumed, we discovered the efficiency of huge airports such as the Taiwan international airport, and we gained a greater understanding of data processing and management as a whole.

> All project goals are met and:
>
> There are clear descriptions, figures, or tables of each method's output on the full target dataset. (**Note:** Figures can be stored as plain images in your git repo and the link provided in the .md)
>
> There is a written discussion of the projects findings that makes and proves a claim that each method was successful.

1. A written report. In addition to your code, your Github repository must contain a `results.md` file which describes:
    - The output and correctness of each algorithm – You should summarize, visualize, or highlight some part of the full-scale run of each algorithm. Additionally, the report should briefly describe what tests you performed to confirm that each algorithm was working as intended.
    - The answer to your leading question – You should direct address your proposed leading question. How did you answer this question? What did you discover? If your project was ultimately unsuccessful, give a brief reflection about what worked and what you would do differently as a team.