Final project writeup:

Dataset:

- Global Air Transportation Network, Airports, Airlines, and Routes: https://www.kaggle.com/datasets/thedevastator/global-air-transportation-network-mappin g-the-wo/data
- For my project I chose to use the Global Air Transportation Network dataset which is a comprehensive collection of information on airports, airlines, and their routes worldwide. It contains detailed data on airport names, locations, codes, airline information, routes, codeshare details, aircraft types, and more, all in the form of csv's. The dataset has been meticulously compiled by researchers globally and covers over 60000 flights routes worldwide. These routes form a connected graph that spans the majority of the world. It is an interesting one to analyze as planning airline routes is a very relevant issue and being able to determine areas that lack efficient access could be very useful.

My Project:

- When it comes to analyzing my dataset I chose to do one of the recommended methods, that being average distances.

- Instructions: Average distances: Instead of calculating whether nodes are reachable within a fixed number of steps, pick a large random sample from your nodes and calculate the distance to all other nodes in the graph. What is the average distance between all node pairs in your calculation?
- To carry out the instructions, broadly put I had to put the data into a usable format so I read the relevant csv's and built an adjacency list out of them then ran a breadth first search on 1000 randomly selected airports to determine how they are connected to every other airport in the data set. Then out of all the pairs generated from the 1000 random samples I averaged the total distance to find the average distance between airports across the entire random sample. Below is a breakdown of each step individually.

I separated my code into four modules: airports.rs, bfs.rs,graph.rs, and main.rs each plays a certain role in assembling the final output. Here is the breakdown of what each does and how they play into the process described above.

Airports.rs:

- This module is used to get one of the critical parts of assembling the graph adjacency list. It reads the airports.csv file which contains a list of the vast majority of airports that are used in routes csv the ones that are not detailed in airports.csv are removed in a later step and location data. Anyways the file is read and for each airport the coordinates are retrieved and stored for later use in calculating the distance between airports. That is the main purpose of this module grabbing the coordinates of airports. It also introduces the Geoutils crate which I used to calculate the kilometer distance between points later. I found it here: https://docs.rs/geoutils/latest/geoutils/

Graph.rs:

- In this module the weighted adjacency list is assembled. Routes.csv containing all the airplane routes is read and an adjacency list is built by looping through the route pairs and finding all the adjacent airports to each individual airport. From there the airports.rs module is used to grab the coordinate data for the respective data and then the distance from the source airport to all its adjacent airports is calculated with geoutils and inputted in the adjacency list alongside the each neighbor so that that the list now has tuples with the neighbors and there distance from the source airport.

Bfs.rs:

- This is the main function of the code, everything else has been preparing the data for being put through this basically. The breadth first search takes the adjacency list generated by graph.rs and runs it search finding a path between one specified airport and every other airport that it can reach. It samples a number specified in main.rs. As the search goes through the entire possible paths until it reaches its destination it also stores the visited nodes in a queue for later. Once the successful path is found it goes back and looks at each node that it visited on the way and grabs the airport identifier and the distance of each step. In the end this finds every possible path between the specified airport and the others, storing both the path it took and the total physical distance to.

main.rs:

- This module basically uses all the others and actually runs them then finds the average distance and also tests them. First it creates a text file that all the output is stored in. Then it assembles the adjacency list using the mods above and writes it to the text file; this is the first viewable part of the output as the entire adjacency list is outputted to the text file. Next the code runs the breadth first search on a 1000 nodes finding the distance to every other airport that is psychically achievable for each node, skipping over the ones labeled as infinite as they are not connected, of which there are relatively few, the graph is pretty well connected. It writes each of these connection lists and their total distance to the text file output.txt. Lastly the main.rs find the average total distance by adding up the distance between all airport pairs that are sampled and dividing by the number of pairs thus finding the average distance for the sample.

Tests in main.rs:

1. Test test_bfs:
    - This test validates the functionality of the BFS algorithm by applying it to a small sample dataset. The sample dataset consists of a HashMap representing an adjacency list, containing a few airports and their connections with distances. The BFS algorithm is executed from a specified starting node ("A"), traversing the

graph to find the shortest path distances to other nodes. After running the BFS algorithm, the distances calculated for each node from the source node ("A") are compared with the expected distances. Expected distances and paths are predefined for each node based on the graph structure. The test asserts that the actual distances and paths obtained from the BFS algorithm match the expected values.

2. Test test_bfs_unreachable_airports:
    - This test assesses the BFS algorithm's handling of unreachable airports within the dataset. A sample dataset, similar to the previous test, is constructed with additional airports that are not directly connected to the starting node ("A"). The BFS algorithm is executed from the starting node ("A") as before, traversing the graph to find distances to other nodes. However, in this case, certain airports are intentionally made unreachable from the starting node to simulate disconnected portions of the graph. After running the BFS algorithm, the distances and paths obtained for each node are compared with the expected values. If a node is unreachable, its absence from the BFS results is verified, and it is asserted that the node should not be reachable from the starting node. This test ensures that the BFS algorithm correctly handles cases where certain nodes cannot be reached from the starting node and accurately identifies such unreachable nodes.

That is all of the code in my project here is a description of the output:
All of the output for my code is put into a text file that my main.rs code assembles called output.txt. Within output.txt is the entire adjacency list. The route and distances between the sampled nodes and all the other reachable airports and then lastly the average distance between pairs.

Adjacency list sample:

```
Airport FMA: [("AEP", 925.4673819999999), ("AEP", 925.4673819999999)]
Airport AYP: [("LIM", 340.118352), ("LIM", 340.118352), ("LIM", 340.118352), ("LIM",
340.118352)]
Airport KKR: [("AXR", 54.184007), ("PPT", 357.93180099999995)]
```

Connections sample:

```
Path: ["MAQ", "DMK", "SUB", "JHB", "SZB", "USM", "BKK", "BLR"]
Distance from MAQ to HRI: 10033.76 kilometers
Path: ["MAQ", "DMK", "SUB", "JHB", "SZB", "USM", "BKK", "MLE", "HRI"]
Distance from MAQ to ZRH: 15056.94 kilometers
Path: ["MAQ", "DMK", "SUB", "JHB", "SZB", "USM", "BKK", "ZRH"]
Distance from MAQ to MKQ: 7372.37 kilometers
Path: ["MAQ", "DMK", "SUB", "MDC", "SOQ", "FKQ", "KNG", "NBX", "DJJ", "MKQ"]
```

Average for 1000 sampled nodes:

Average distance between every reachable airport within sampled pairs: 50874.91 kilometers

Analysis:

The average distance is very high, much higher than it would be in real life mostly due to the inefficiencies of a breadth-first search with weights. It would be interesting to see the results had something like dijkstra's algorithm been implemented and the most efficient route found by distance, maybe something to do for future analysis. I hand checked the distances for some that I picked randomly and they were accurate, just inefficient. It is interesting just how connected the airports are. Very few relatively had to be tossed out due to lack of connections. The biggest limitation was some of the airports did not have location data. I would have liked for my code to have a more relevant impact but it nonetheless is accurate for what it aims to do.