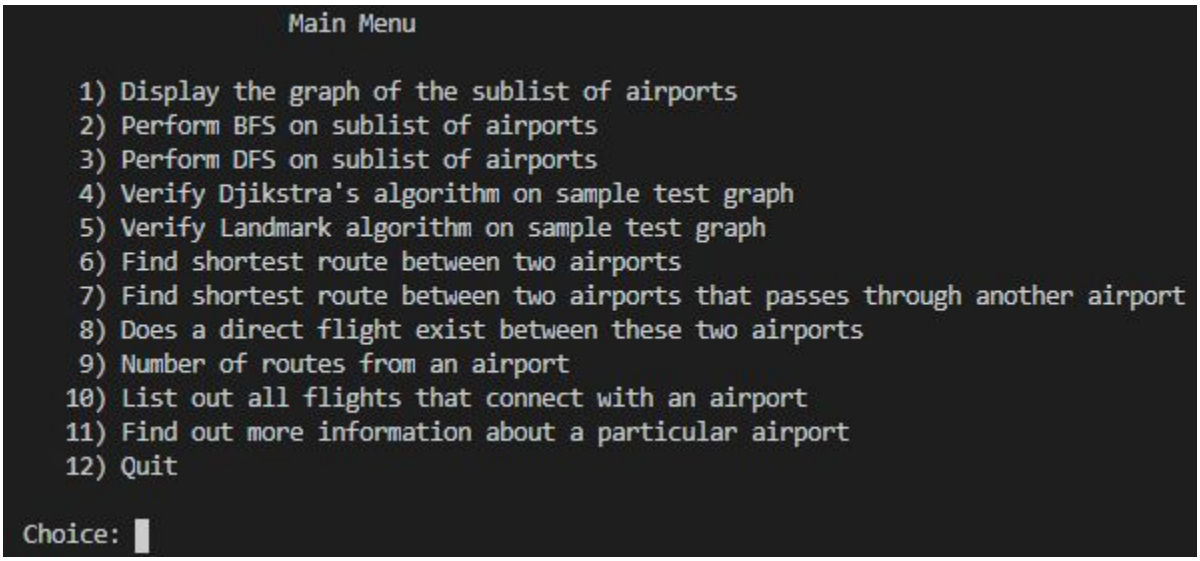
Tejus Kurdukar (tkurdu2)
Edward Guo (eguo4)
Gloria Xiao (gloriax2)
Calvin Lee (lclee3)

Final Project Report

The purpose of this project was to process the OpenFlights airports and routes datasets obtained from www.openflights.org using an efficient graph implementation. We adopted an adjacency list implementation with edge weights representing physical distance. The implementation included several useful algorithms to extract useful information from the dataset, notably DFS, BFS, Djikstra's algorithm, and the Landmark Path algorithm. We were successful in accomplishing these goals. To make the functions more accessible to the users, we implemented a menu interface to test and demonstrate the capability of our project, shown below.



The user can input a choice of number from 1 to 12, each of which corresponds to a function that will be executed. After the function execution, the menu is displayed again, allowing users to call another function. The smaller subgraphs used for testing purposes are drawn out and available in the repository for verification. If the user inputs an airport that does not exist, the program will prompt the user to enter a different airport. The user can also look up the country, city, longitude, latitude, IATA code, and ICAO code of any airport they want. Entering the number 12 in the menu screen will terminate the program.

In general, we were able to create a graph-like data structure that efficiently accesses the information of each airport along with its respective routes. By implementing a structure that takes airports as vertices and routes as edges, we can clearly see how each airport is related to the others. This ease of access allows us to easily navigate the desired information of an airport, like how far it is from another airport, whether there is a direct route between two airports, etc. Our implementations of the traversal, Dijkstra's, and landmark path algorithms were executed with this in mind. Each algorithm was coded based on the given pseudocode in class for the BFS and

DFS traversals and on Wikipedia: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm for Dijkstra's algorithm and modified it for the Landmark Path algorithm.

Code was tested using small subsets of the OpenFlights dataset containing ten routes and ten airports; the BFS and DFS algorithms refer to this as the "sublist" in the menu options. The sublist is used to test BFS and DFS by displaying the graph vertices and edges in the console such that the user can write the graph on their own and confirm that the traversals functioned as expected. We used enums to be able to label edges and vertices. Vertices labeled 0 are unvisited, and vertices labeled 1 are visited. Edges labeled 0 are unvisited, edges labeled 1 are discovered, and edges labeled 2 are cross/back edges. As expected, labels of 0 never appeared, and every vertex was labeled 1 in our test output. Labels of 1 and 2 appeared in the appropriate places when testing for edges, and the function accurately returns a minimum spanning tree. Djikstra's and Landmark were tested in similar ways but used different subsets to test their functionality, accounting for a test of unconnected components.

We worked collaboratively on the projects using Zoom Meetings to actively discuss the code while programming. Google Drive was used to manage all non-code related documents such as the contract, project goals, and development log for the final project. Originally, we worked on the code by having one person share their screen while everyone else would vocally comment on what to do. Eventually, we figured out how to get live share working on Visual Studio Code, which allowed us to actively edit together in a Google Drive like manner.

There were some drawbacks to our approach and certain aspects of our project that we would change given more time. One flaw in our project was that we did not fully parse the data due to entries in the OpenFlights set with missing information. Some destination and source airports had information labeled "\N." If we had a chance, we would have sought a more complete and updated dataset, though our implementation processed OpenFlights to the best ability that it could. Another drawback was how we implemented Dijkstra's algorithm. By using the standard library priority queue, we were limited to certain constraints, like the inability to update the contents of our queue. To work around this issue, we continuously push duplicates of an airport when we update its information. In order to implement a cleaner approach, we would create our own priority queue structure that allows for updates on the queue's contents. There were some aspects to our code that could have been refined. For example, an adjacency matrix could have been used in addition to our adjacency list to make lookup runtime faster for options in our menu that require areAdjacent. A major flaw in our implementation was our over-reliance on maps; we used five total maps in our graph file to circumvent const and private access issues where we could not find a better solution. While they function accurately, they create an unnecessary amount of clutter, and we would have found more optimized ways to run our graph given more time.

Although the Dijkstra algorithm works correctly in the sense that it outputs the path with minimum distance, the paths it outputs are not very practical in the real world. It will often return a path with half a dozen connections for airports far apart. Paths like these are time-consuming

and do not make sense financially and environmentally. To make Dijkstra's output more meaningful, we need to factor in flight time, connection time, and cost.

We conclude this report with some neat facts. One interesting fact we found out is that the University of Illinois Willard Airport is actually in the dataset, and it has a direct flight to O'Hare International Airport and Dallas/Fort Worth International Airport. We also found it interesting to see the kinds of long routes and connections for some airports and the number of direct flights for some big airports. For example, to get to Recife, Brazil from Chicago, you have to get through seven other airports (if only considering direct flights from one airport to another). O'hare has a total of 206 direct flights, Delhi's international airport has 97, the Newark airport has 153, Toronto's international airport has 147, etc. In conclusion, our project was largely successful and achieved the goals we set out to accomplish and we discovered some interesting results.