Teammates:

Tim Li - tli94, Tim Long - ttlong4, yuhanw11, Alex Wei

Project Objective Summary:

The intention of our project was to create a representation of the shortest path between two airports. When you input the source airport and destination airport, the program would output the shortest path between these two airports.

Direction: In the beginning of the project, we decide other teammates send their code to one person, which this person will put all the code into main.cpp, but soon we find there's some issue. The biggest issue is compiling errors which that code is unable to run at ews. When we try to fix the problem of main.cpp, we find that since all the code is in one file, it's really hard to debug and upload the new file, so we decide to break down the main.cpp into several different files. And this time, the code work properly.

Implementing dataset: We successfully implement the csv dataset in Openflight website by creating an airport.h file to get all the information of airports and using readAirport function to create a map of airports id to airports called airlines. This process in our program is done separately from each algorithm, so we don't need to create a new map during each algorithm.

Algorithm: In the beginning of the project, we decided to use BFS traversal, A-star algorithm and Floyd-Warshall algorithm.

For BFS traversal, we will first get the source airport id into the queue, and the first node inside the queue is the current airport we go through. Then check if the current airport is the destination. If it isn't, then mark this airport as visited and go to check the next airport. We provide a test case for BFS in our program, just un-command test_bfs inside the main function, and the test case proves that BFS traversal works properly in the program.

For A star algorithm, it is similar to BFS traversal, but this time we will use a map called open, a set called close and g represent the cost(distance) of traversal. Beside this, there's 2 helper functions called build_path and find_min. We provide a test case for A star traversal in our program, like BFS traversal, just un-command test_astar inside the main function, and the test case proves that A star algorithm l works properly in the program.

For the Floyd-Warshall algorithm, we meet some trouble when we first implement this algorithm. Since Floyd-Warshall will create a matrix and compare every row and column to get the shortest distance and repeat this process many times, the time complexity for

Floyd-Warshall is O(v^3). Even though we already know the time complexity for the Floyd-Warshall algorithm is long, the actual time complexity for Floyd-Warshall is still longer than what we expected. We considered replacing the Floyd-Warshall algorithm with another algorithm, but after discussion with group members and project mentors, we decided to create a small dataset that includes some airports called airlines-small.csv. After using the small dataset, the running time of Floyd-Warshall low down into an acceptable range. We also prove a test case for Floyd-Warshall, in the test case, it shows that Floyd-Warshall implants well in the small dataset.

Overall: Though we met all of our goals, we still have some points that need to improve. The first one and the most important one is to have a good and comprehensive plan before starting. Our initial plan has many defects in the details. Our first goal is to build a search engine on a dataset, but we soon discovered that a search engine for a dataset is really hard to implement graph algorithms inside the code, so we decided to switch to shortest distance between 2 airports. After changing the shortest distance between 2 airports, we don’t have a clear understanding on the running time for the algorithm, which causes some trouble in the late development. The second point that we need to improve is organizing our code. Since it’s our first time on a group project, at the beginning we put all of our code inside main.cpp. This causes a huge trouble in debugging and maintaining our code since most of us have different styles of code. Next time, we will first build a comprehensive plan and a clear rubric for style of code before we start.