We have accomplished the our goals described in our GOALS document. Specially, we successfully parsed data we needed into Airport objects and a directly graph. With a helper function, we calculated the distance for each route and used it as the weight for the corresponding edge. We also succeeded in implementing three algorithms, namely BFS, Dijkstra's Shortest Path, and Strongly Connected Component. Since we came up with many test cases to test our algorithms extensively, we are confident that our codebase is bug free.

For the BFS, we took advantage of Iterator class and queue. In this way, we were able to traversal the graph vertex and vertex, making it easier to demo and test it. The result we got from the strongly connected component was most intriguing. Before implementing it, we assumed most airports would be in the same component. However, it turned out half the airports were a single component by itself. That meant a lot of airports were not connected with each other. There was one component that contained the other half of the airports.

Throughout the process we met several issues, like the parsing data did not really fit the requirements of the function, sources and destination are not in the airport and dataset or something they are equal. Some "Airports" have 3 digits but some others have 4 digits, we need to determine the position of them since they have different locations.
For the Dijkstra algorithm, it originally could only return the length of the shortest path, however, the data we need actually is the travelled distance between two airports, so we create a struct called "Path" which can return the travelled distance between two airports. Also, at first we did not know what we should return if there are no connection lines between to destinations so we get an error, but finally we know that we should return intermax for this kind of situation.