- 11/21 - 11/29:
    - Pseudocode
        - Strongly Connected
        - Dijkstra's Shortest Path Algorithm
    - Discussed
        - Parsing/Storing of Data
            - How we want to store airports and their respective data (location, connections, etc.) for easy access for our implementation

This week we had multiple discussions over Zoom to discuss our method of parsing and of storing the data from the OpenFlights database. We then split the work between the three of us into data organization, an implementation of the strongly connected components concept, and Dijkstra's shortest path algorithm. We also hashed out the finer details of implementing maps and or dictionaries and their stored information. By our mid-point progress check, we were able to develop some pseudocode for the two algorithms and intend to ask for more insight regarding the BFS traversal.

- 11/30 - 12/7:
    - Completed:
        - Parsing/storing data in database
            - Useful variables and functions
        - Main.cpp for user input
        - Makefile
        - Strongly connected: brute force (disjoint sets) -> Kosaraju's
        - Figured out how our parts connect
        - Drew a graph of the connected airports to use for strongly connected, BFS, and shortest path
        - BFS Traversal through the graph
        - Shortest path using Dijkstra's
    - Began Debugging

We had our mid-point check in the beginning of the week and were provided important information and insight regarding our code. For data parsing, suggestions were made for considering the addition of an airport class. We were also encouraged to use more established algorithms, like Kosaraju's, to determine the strongly connected airports in the database, instead of using disjoint sets. We implemented both of these suggestions. Along the week, we continued to meet to discuss how our parts are entwined and decided to create a graph of all the airports so that the strongly connected, shortest path, and BFS traversal functions can all effectively utilize it. The foundation of these DrawGraph files were built upon lab_ml's edge.h, graph.h, graph.cpp, NimLearner.h, and NimLearner.cpp. We then completed the makefile, the main.cpp file for user input, the parsing/storing in database.cpp, the graph drawing of the connected airports, the strongly connected using Kosaraju's, the BFS traversal, and the shortest path using Dijkstra's. We have thus begun the debugging process.

- 12/7 - 12/11:
    - Tons of debugging
    - Writing test cases to confirm functionality
    - Completed the presentation recording, README, and the RESULTS file

This week was centered mainly on creating test cases in test-database.cpp for each of our program's algorithms. By creating these test cases with manual data inputs, we were able to go through each algorithm individually to test their functionality and debug any errors that were found. There were a few significant bugs that were detected and addressed. One of them concerns the anomalies existing in the OpenFlights dataset and how our code reads in the data. In the provided airport names, we found that commas were sometimes included, which was an issue as we identified our delimiter to be a comma. After completing the debugging process, we wrapped up our project by creating and recording our presentation as well as completing our README documentation and RESULTS file.