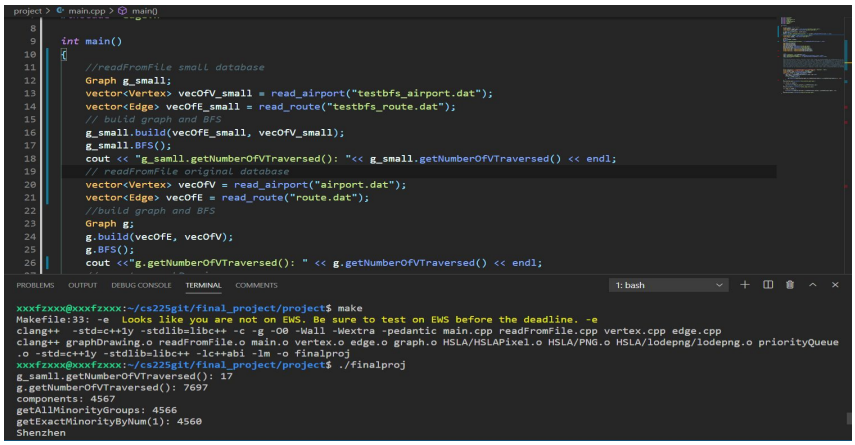
Final Project Result

I. Data Structure

We successfully implemented the graph using an adjacency list. Vertices are stored in an array and indexed by their Airport ID. Each vertex stores a deque of incident edges (route). Finally, there is a deque of all edges (routes), representing all the airlines in our flight graph.

II. BFS Traversal

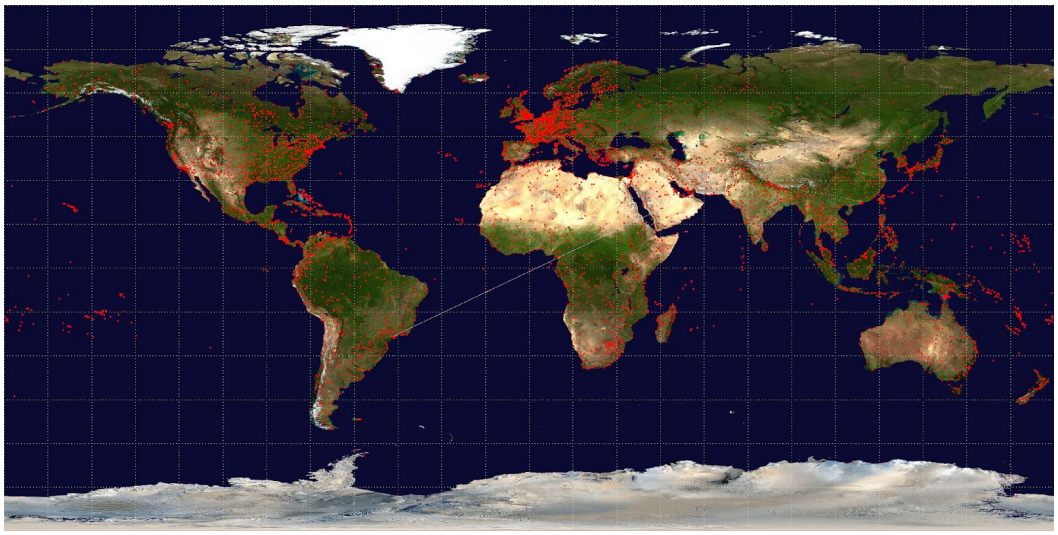
We implemented BFS Traversal using the pseudo-code provided in lecture. This method is nearly correct after we tested it.



When we tested BFS on a small sample of data, it worked and printed out the correct number of vertices (17) contained in file “testbfs_airport.dat”. However, when it came to the large data file, the output was 1 off the correct number. The correct number of vertices in “airport.dat” is 7698 while the output showed 7697.

III. Dijkstra’s Algorithm

Our implementation of Dijkstra’s Algorithm runs reasonably. The outputs were reasonable for both simpler and more difficult shortest paths. A hard example is given in the figure below: From SZX to Sao Paulo (Hard Shortest Path)



IV. Uncovered Algorithms

A. Project onto map based on data

Basically we use the line drawing algorithm and CS225 HSLAPixel to draw a straight line on the Cartesian Coordinate System. We assume that the airline is a straight line on the map. However, there are some cases where we need to draw two disconnected lines that cross the borders of the picture because we are drawing lines on a 2D world map instead of a 3D sphere. We map a coordinate system on the map where the upper- left corner is the origin. Then reflect the latitude and longitude to the x-axis and y-axis relatively in order to get the pixel on the world map picture. We classify four cases and draw the disconnected lines separately. If the airline needs to cross the Paciffic Ocean, the line crosses the border and thus is disconnected. We color airports on the map based on the incident edges number. Airports with no incident edges appear in red. Those with a small number of incident edges appear in green. The more incident edges an airport has, the hue is larger. Our result graph is shown in readme.md. The output image of all air routes is shown in Figure IV. 1. The output image of

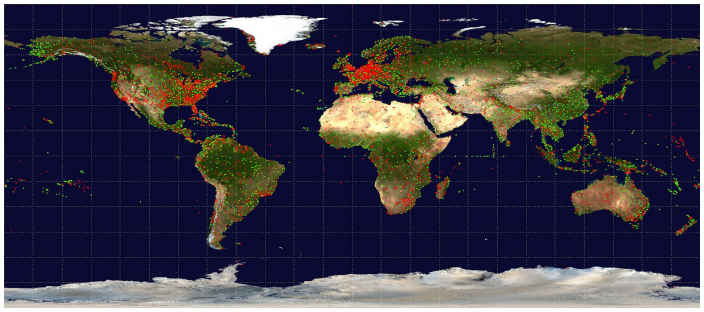


Figure IV. 1

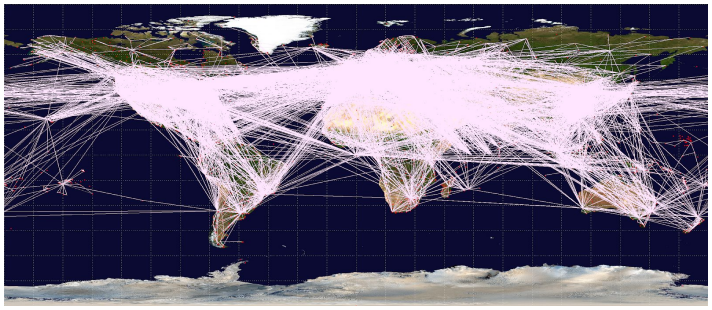


Figure IV. 2

B. Clustering Finder

Based on BFS, finding the minority groups of cities according to certain cities or certain quantities. We have functions such as `getAllMinorityGroups`,`getExactMinorityByNum`, `getExactMinorityByV`.

V. Test cases

Our test cases cover all the possible cases in class readFromFile and graph.
Test cases are all passed, which ensure that our algorithms' correctness.

VI. References

priorityQueue.{h, cpp}: lab_heaps
Catch Test: Final Project Demo