Alex Li

Praval Telagi

Kevin Zhao

CS 225 Goals

Data Set - OpenFlights

Open flights is an open source data set of flight routes and airports. The data is currently dated from 2014 and has no path for updates but is still interesting for historical study. As a project you would load the data from at least the routes and treat this as a directed graph.

One key question in this case is going to be is what do you use for weights for the graph. There are several possible choices.

- Count the number of routes on some edge and use (1/number of routes). This would give a measure of throughput on the route.
- Use the airport data to compute the distance and use that as the weight.
- Augment the graph with a separate node for each airline at the airport and the airport itself. Then add edges between each airline node at the airport. You can then have a standard weight for flights and for moving from airlines to airports.

Any Shortest Path algorithm to find preferred routes.

You can find this data set and descriptions here
https://openflights.org/data.html

Traversals

1. BFS (Breadth First Search)

Covered Algorithms

1. Shortest Path
    a. Dijkstra's Algorithm

Complex or Uncovered Options

1. Landmark Path (shortest path a-b through c)

Goals

1. ~~We are planning to implement Dijkstra's Algorithm and the Landmark Path to find the shortest running time and shortest flight path between airports using the OpenFlights dataset.~~
2. ~~We plan to compare both implementations to demonstrate the efficiency and effectiveness of one algorithm over the other when finding the shortest path between multiple airports.~~
3. ~~We will perform statistical analysis to determine if the difference between the two algorithms are significant.~~
4. We will write a very in-depth readme file that shows how to set up the environment required to run the program.
5. We will update our development documentation log weekly.
6. We will ensure that we follow Google style guide for C++ and comment code.
7. We will also write comprehensive and exhaustive unit tests using Catch2.
8. We will ensure that there are no memory leaks by running Valgrind.
9. In our results file, we will formulate conclusions based on the statistical analysis.

Goals Clarified

1. We will ask the user to input an origin and any amount of destination IDs.
    a. If the ID(s) are invalid, we will ask the user to try again.
        i. If there is only 1 destination ID, we will perform Dijkstra's search.
        ii. If there are more than 1 destination IDs, we will perform landmark search.
    b. For every instance the user inputs an ID, we will return the airport name associated with that ID.
    c. Afterwards, we will print out the path consisting of all airports along the shortest path from origin to destination
    d. We will also print out the total distance (in kilometers) of this path.
    e. We will catch errors
        i. user input
        ii. invalid results
2. We will write a very in-depth readme file that shows how to set up the environment required to run the program.
3. We will update our development documentation log weekly.
4. We will ensure that we follow Google style guide for C++ and comment code.
5. We will also write comprehensive and exhaustive unit tests using Catch2.
6. We will ensure that there are no memory leaks by running Valgrind.