CS225 Final Project Proposal: OpenFlights

NETIDs: zihengc2, haoyu9, yuen9, rmday2

Leading Question:

In this project, we want to explore the OpenFlights dataset to gain insight into how humans can be optimally interconnected via air travel. (https://openflights.org/data.html). Using the algorithms described below, a real life application would be generating a route visiting the 7 world wonders or a shortest path from one wonder to another, visiting all of the national parks within the United States, etc. We would use the nearest airport to each corresponding location and construct a route based on the shortest path.

Dataset and Implementation:

Our airports and routes data will be stored in a graph. In our implementation, we would treat the airports as nodes in our directed graph. To calculate the weight of the edges, we would first parse the airports, then calculate the distances between each airport using the airports dataset, assuming the flight travels the shortest distance. During our implementation, we can possibly create util files to help with data loading and graph creation. We will download all of our data from OpenFlights (airports.dat and routes.dat). We would store it on our GitHub repository initially. Routes.dat contains 67663 routes between 3321 airports on 548 airlines, each piece of flight information contains possibly the airline and ID, source airport and ID, destination airport and ID, codeshare, stops and equipment. Airports.dat contains the name, ID, location of each airport. In our project, we would utilize the source and destination airports and their locations to compute the distance before using it as a weighted edge connecting the airports. If an error occurs due to invalid or uninterpretable data, we would treat it as faulty and it would be ignored. After the airport data is stored in a desired form, we can implement a method to generate the page rank matrix. With the help of C++'s linear algebra library, we can derive a vector that contains the information about the "importance" of all the airports. Then we can pick out the top N airports, traverse through our graph to retrieve the information and map them into a world map.

Graph Algorithms:

In this project, we will create a graph that can hold airports and the routes between them. We will use a DFS traversal with an iterator to traverse our graph structure and visit the nodes that represent airports. We will then use Djikstra's algorithm to find the distance between relevant nodes by choosing a start point, an end point, and possibly some landmarks to visit along the path, which would be handled by the Landmark Algorithm. The output can be a vector of vertices that represent the shortest path. We intend to use AVL trees and possibly other graph structures like adjacency matrix and adjacency lists. We might use other improved structures if we encounter them later on in the class.

Algorithm description:
Using Djikstra's shortest path algorithm, we will be able to find the shortest path between any two airports, passing other airports as needed. In addition, we would also use the Landmark Path, which would calculate the shortest path when we want to pass a collection of airports. Compared to Dijkstra's algorithm, although both computes the shortest paths, Dijkstra's gives a path that's shortest relative to the start and end points, with or without any passing airports. The Landmark Path returns a shortest path that contains target airports. Together, we would be able to achieve goals like creating various efficient tourist routes.

We also want to take advantage of the Page Rank algorithm to find the most important airports around the world and map them on the world map. The first input of this implementation should be a node from the direct graph we are using to store the information of all the airports. The second input should set the range of the airports that we want in the adjacency matrix. The implementation will first create the adjacency matrix. Then, with the tools in the C++'s linear algebra library it would be easy to figure out the eigenvalues of the adjacency matrix, which is the result we want.

Timeline:
Below are our timeline goals. These are the tasks that we hope to accomplish by the provided dates.

April 19th: Algorithm Skeleton set up properly: Djikstra's, Landmark Path; have Page Rank completed and verify with tests
April 26th: have Djikstra's written and verify with tests
May 3rd: have Landmark Path and Graphic Output completed and verify with tests
May 5th : Final presentation completed
May 5th : have project report completed