CS 225 Final Project

Open Flights Dataset

By: Ryan Day, Jonathan Yuen, Jack Chen, Kevin Zhou

In this project, we set out to use the Open Flights dataset in conjunction with some path-finding algorithms and a traversal in order to explore the possible real-world applications of graph data-structures. Our project allows us to see how humans can be optimally connected via air travel by using Djikstra's algorithm, and the Landmark Path algorithm to find the shortest path between two airports, as well as a Page Rank algorithm to find the most important airports.

A summary of our project and its results can be found in this presentation.
Slides: CS 225 Final Project Presentation.pdf
Video: https://youtu.be/r_ltZNUqKkw

To complete this project, we first downloaded all of the OpenFlights airport and routes data. They are CSV files, each line representing an airport or flight. For each airport, this dataset stores an Airport ID, Airport name, and longitude and latitude coordinates. We parsed each line character by character so that if a line is missing data or contains NULL, we would disregard it. We then convert this data to a vector of strings to construct airport objects. Each airport object contains an unordered_map to host the adjacent airports and its path. When inserting edges, we first calculate the weight of the edge with the coordinates of the two airports. In our design, the airports are the vertices, and the edges are the flights with weight being the distance. When inserting redundant edges, only the first one is inserted to keep the graph simple. In conclusion, we created a graph class that supports the insertion of vertices into the graph, the creation and insertion of edges and the fetching of vertices, adjacent vertices, their data, and the weight between them.

For our traversal, we implemented three Breadth First Search functions, following the basic algorithms presented in lecture. The first BFS function traverses through the whole graph given a starting airport, the second one traverses up to a given move, and the last one traverses until the given destination airport is reached. All of these functions utilize the queue data structure, and their outputs are vectors of strings of the name of the passing airports. For testing, we started with small datasets and checked the traversal by hand. We then tested the BFS_moves by comparing the length to the number of moves and the first element of the vector with the starting airport. To test BFS_dest, we compared the starting and ending airports to the start and end of the output vector. The running time of our BFS traversals are O(m+n), which is relatively fast.

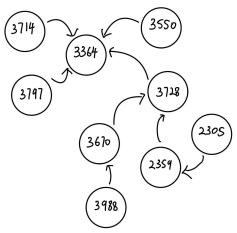
The first algorithm we implemented was Djikstra's algorithm for finding the shortest path between two airports. This function outputs a list of vertices that follow, in order, the shortest path between two airports, as well as the path length. We used a priority queue to maintain the airports that we were finding on the way to locating the shortest path, as well as a map to keep track of the airports and from where they were flown to. We also maintained unordered maps to keep track of airports that were visited. What we found interesting after running this algorithm on multiple combinations of airports, is that even if a path with fewer stops is available, it is sometimes more efficient from a flight distance perspective to stop at more airports along the way. Each edge in the algorithm run process is viewed at most two times, and each node is viewed at most two times (as you add it to the queue and then remove it). Since we used a priority queue as a data structure for keeping the vertices as we visit them, it takes constant time to queue the node and log time to query it. So the total run time is $O(|E|+|V|\log(|V|))$.

Another algorithm we implemented was the Landmark Path algorithm for finding the shortest path between two nodes while enforcing that a landmark is visited along the way. We thought of this as an extension of a regular shortest-path finding algorithm, but with the modification of including different stops. The shortest path between two airports with a landmark is just the shortest path between the initial stop and the landmark, and then the shortest path between the landmark and the final destination. This function has a similar output as our Djikstra's algorithm and returns a list of vertices and a path length.

The last algorithm we implemented was a Page Rank algorithm. Page Rank algorithm in this project is used to evaluate the importance of the airports. Our implementations of this algorithm produce two vectors. One vector contains the ID of each airport and the other includes the evaluation of each airport. A function which can extract the most important airports can utilize those two vectors to reveal the ID of the most important airports. The first step of this algorithm is extracting the adjacent matrix out of a graph object. To do this, we first check through the vertices of the unordered map of the graph, extracting the ID of the airports and put them into a vector which will be used as a reference later. Then, the flights of each airport are reached. The weight of each flight is acquired and stored in the adjacent matrix in the order specified by the ID list. Before the Page Rank algorithm to be performed, the matrix should be adjusted to fulfill the requirements. A function will first normalize the column vectors of the matrix to ensure the sum of each column vector is one. If an airport has no outgoing edge, the sum should be zero. In this case, each entry of the column vector will be adjusted to 1/vector.size. After the normalization procedure, the adjacent matrix will be further modified according to the description of Page Rank algorithm. Each value will be set to the product of itself with the damping factor plus (1 - damping factor) / vector.size.

Then, a starting vector can be generated with random numbers. Calculating the product of the matrix and the starting vector will generate another vector which reveals the future state of

the airport of the current state, which is specified by the starting vector. Keep multiplying the vector with the adjacent matrix, the result vector will approach a stable state, which then can be used to rank the importance of the airports.



To validate our implementation, a subset of the flight date was gathered. The graph constructed based on this subset can be represented as shown.

Since this subset is pretty simple, we can intuitively conclude that the 3364 is the most important airport simply because it has the most incoming edge and has no outgoing edge. The result of our implementation is consistent with this intuition. After running the implementation with the whole data set, our most important airport is Paris-Charles de Gaulle Airport. This result seems also inline with the reality. Thus, we conclude that our implementation of the Page Rank algorithm meets our expectations.