CS 225 Fall 2022 Final Project OpenFlights

By Harry Xu, Dylan Zhuang, Logan Cheng, and Andy Xu

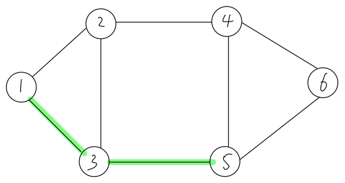
The goal of this project is to gain insight into how people can travel between airports in the most optimal way possible using the OpenFlights dataset found at https://openflights.org/data.html In this project, we explored the functionalities of the shortest path between airports, the fewest stopovers between airports, as well as the importance/popularity of an airport through the implementation of Dijkstra's algorithmBFS traversal, BFS traversal, and the Pagerank algorithm. Intrigued by this topic, we are excited to present the fruition of our work and how it can be used in real-life applications.

Data Parsing:

To begin with, we started this project with data parsing. We approached this with the implementation of a graph data structure. The constructor of the graph takes two inputs: the **airports.dat** and the **routes.dat** file. Both these files contain detailed information about each airport and the routes that connect two airports, respectively. The constructor is designed to filter out the irrelevant data about the airports and routes and only keep what we need moving forward. The data that we are working with contains: the airport IDs, the latitude of an airport, the longitude of an airport, and the starting airport ID along with the destination airport ID of a route. After filtering out the irrelevant data, the constructor starts to convert the data in the CSV files into int by character which we will be storing for further use. One of the obstacles we faced was that some of the data in the CSV files were invalid while others were repetitive of previous data. We overcame this obstacle by populating a vector by inserting the airport ID and the index from the CSV files. The constructor is also responsible for populating two map data structures, with the first map matching the airport ID with a list of pairs of its destinations and their distance and the second map matching the airport ID with a pair of its latitude and longitude. To find the distance between two airports, we implemented a findDistance function that calculates the distance using the latitudes and longitudes of two airports.

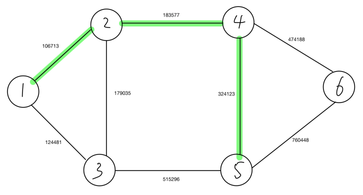
BFS Traversal:

We then implemented the BFS (Breadth First Search) traversal to iterate through every airport in order to find the route from a starting airport to a destination airport. Following the basic principles of BFS we learned in class, our BFS utilizes the queue data structure and keeps track of all visited airports using a vector of Booleans. The input of this function are two airport IDs which represents the starting airport and the destination airport in that order. The BFS function is designed to determine where a path exists between two airports. To elaborate, it first checks if the starting airport ID exists and is present in the dataset. Moving on, the function searches each adjacent airport through the traversal process and repeats this process until it finds the destination airport we were looking for. The BFS function lastly outputs a vector of integers that keeps track of the airports' IDs along the path from the starting airport to the destination airport. The running time of our BFS function is also relatively fast since its O(m+n). To test the BFS function, we wrote a smaller database by ourselves because the dataset in the Openflight website is too large and it is hard for us to find the correct path on our own. Therefore, we created a graph(the picture on the right) with six nodes, which represents the airports, to test if the BFS works perfect. In the testcase, the desired path is from point 1 to point 5 through point 3. After running the BFS function, it outputs the same vector as we want.



Dijkstra’s Algorithm:

Next up, we implemented the Dijkstra’s algorithm to find the shortest possible path from a starting airport to a destination airport. Dijkstra’s algorithm is a good match for this certain task because it can be easily applied to how we choose to represent our data. We approached this task by treating each airport as nodes on a graph, the routes that connect two airports as the edge that connects the two nodes representing distinct airports, and the distance between two airports as the weight of the edge. This way we ensure that Dijkstra’s algorithm works smoothly. During the implementation of this algorithm, the function input the ID of the starting airport and the destination airport and output a pair of vectors that hold integers and doubles represents the shortest route between airports. To achieve our goal, we initiates and utilizes a vector of integers that maintains the previous airport visited. We are also aware that not all airports have a route to other airports and in that case we simply return an empty pair. This function makes use of a priority queue since it has the functionality such that its first element is always the greatest of the elements it contains. With this functionality, we are able to keep track of all the airports on the shortest path to the destination airport. One discovery we made after implementing this algorithm is how some routes are shorter than other routes that have fewer stopovers. The importance of distance, which are represented by the weights of the edges, manifests in this interesting discovery. To conclude, our implementation of the Dijkstra’s algorithm has a good running time of O(|E| + V|log(|V|)). To test the Dijkstra’s Algorithm, we used the same dataset as the test case for BFS. The graph on the right contains all possible routes between the start point(1) and the destination point(5). The numbers on the edges represent the distance between two nodes. In the testcase, the desired route is from point 1, go through point 2, 4, and finally arrived at 5 with the shortest total distance. Finally, by implement our Dijkstra algorithm, the output of our algorithm successfully match the testcase.



Pagerank Algorithm:

The last algorithm we implemented is the pagerank algorithm. One of our goals of this project is to find the importance/popularity of an airport. The graph constructor has also initiated an adjacency matrix which this algorithm will use to evaluate the importance of each airport. The adjacency matrix stores the weight of each edge, which is the representation of the routes, in the order of the airport ID list. Moreover, a helper function called normalize would serve to normalize the extracted adjacency matrix before any addition operation takes place. What this helper function achieves is to make sure that the numerical total of each column is equal to one. The Pagerank algorithm then initiates a vector that is populated by randomized numbers. The product of the adjacency matrix and that vector will be used to initiate another vector which represents the future states of the airport. The process will be repeated the same way until the newly initiated vector is stable enough to rank the importance/popularity of the airport. Then, we can transform this vector to the rank of the airport. To test the pagerank, we use the data from CS357, there is a adjacent matrix and the result after pagerank. To use the page rank algorithm, we will type an airport id, then the program will print out the rank of the importance of this airport. The pagerank algorithm has running time of O(n^2).

Result:

All the functions that we proposed in the project proposal work correctly with the target runtimes we had in mind. We have successfully created a project that answers our leading question. There is a written discussion of the projects findings that makes and proves a claim that each method was successful. Given a dataset of airports and routes, our project finds the shortest distance between two airports provided by the user, if a path exists. We used the Euclidean distance between latitudes and longitudes of airports to assign edge weights and create our graph. We run BFS and Dijkstra's algorithm to find the shortest distance between the source and destination airports and generate outputs as discussed above.