CSE450 Project Report

Kevin Shannon - 1213551857

Preface

The goal of this programming assignment was to find the maximum flow of people from LAX to JFK through a subset of airports and airlines on January 6th 2020. This document will outline the data collection, data preprocessing, data structures, and implementation of solution to this problem. The source code is written in python3. Running 'src/airline_maximum_flow.py' will compute and output a maximum flow of 6124.

Data Collection

Finding data from January 6th 2020 proved to be an undertaking as the airline's websites don't have flight itineraries for flights that happened in the past. Fortunately, the Bureau of Transportation Statistics has exactly the data we're looking for in their Airline On-Time Statistics database (Detailed Statistics Departures). For each airline and airport I queried their database for a csv containing all flights on January 6th 2020. It comes with Tail numbers, departure times, origin/destination airports, departure time, and total elapsed time in minutes. While it isn't quite in our desired format, it gets us almost all the way there.

The other missing piece of data needed is the knowledge of the capacity of each plane in our itinerary. From the tail number of the plane it would be nice to be able to query another online database for the configuration of that plane to get the exact capacity of each plane, given that airlines have multiple configurations for the same types of planes so one AA A321-200 may not have the same number of seats as another AA A321-200. There are some sites that claim to have this information but by inspection I found that their data was unreliable and could never be 100% correct as airlines frequently change their seat configurations and make no effort to relay these changes outside their internal systems. For these reasons I chose to make a best approximation by assuming that the capacity of each aircraft in each airline's fleet is a fixed number. To find these numbers I used the respective airline's fleet information on wikipedia for each aircraft that they flew (American Airlines fleet, Delta Air Lines fleet, United Airlines fleet). If one aircraft had several variants, I used the most popular variant if the information was available and if not I went with the most conservative number.To get the aircraft type from the tail number I used the Federal Aviation Administration's N-Number Inquiry (lookup aircraft by N-Number). Searching each tail number by hand would be costly for ~600 flights so I instead made a web crawler to make all these requests for me and strip out the aircraft information. I stored the findings in a pickled dictionary at `process/plane_lookup`.

Data Preprocessing

There will be a necessary step of preprocessing the data we collected once to get it into a more useful form. The desired format is Origin/Destination Airports, Local Departure/Arrival times, and the plane capacity. Three of these are present and the other two will have to be extrapolated. Transforming tail numbers to aircraft type and then doing a lookup of an airline's aircraft capacity from the data collected. Local Arrival Time will be computed from the Actual Elapsed time in minutes with the difference in time zones coming from a matrix of time differentials for each city in `process/time_shift`. Capacity will be computer by Also in this step extraneous flights that don't satisfy the requirements will be removed. Running `process/process_data.py` will do all of these preprocessing steps and output a single master csv to `process/processed_data/itinerary.csv`. This will be our input data for creating the topology of our graph in the next step.

Data Structures

To capture the spatial and temporal nature of this problem I chose to represent the problem as a graph where each vertex is a flight. Forward edges represent possible next flights someone from that flight could make. The weight of said edge is the capacity of the flight. To keep the structure of the maximum network flow problem two special nodes must be added; one at the beginning and one at the end. The capacity of the edges coming out of the start vertex will be the same as the weights on the flights flying out of LAX, this represents the number of people that could get on the plane and the outgoing edge out of LAX would represent how many people could go to the next flight.

To store the graph data I chose to make two main classes to store Vertex and Edge data. A Vertex stores a list of edges and a label and flight information. An Edge stores vertex information of where it starts and stops, flow, and capacity. For the Flight data I chose to store the Vertex objects in a dictionary where each key was an airport and the value was a list of all flights that departed from that airport sorted by departure time. This choice made edge computation very efficient as only one vertex needed to be found to find all of them.

Implementation

The problem now becomes what is the maximum flow of the network that we have created. This question is easily answered through the use of the Ford-Fulkerson algorithm. I chose to implement the algorithm as it was presented in class. For the search method I used depth-first-search as it was easiest to implement. I chose to make the implementation non-application specific because I wanted to use it on test graphs to test its correctness and it is a good programming practice.

Conclusion

The output after running the Ford-Fulkerson on our graph was 6124. This number represents the maximum number of people that could possibly get to JFK from LAX through a subset of airlines and airports on January 6th 2020. This number seems reasonable and while it probably isn't exact as estimations had to be made on the capacity of some planes, it is a good approximation. Given that in our Simplified NAS model only 7821 people flew into JFK this means 78% of arriving passengers would be coming from LAX which is quite impressive.