CS 225 - Final Project Report

Tara D'Souza, Anirudh Avadhani, and Jacob Rubin

Our group successfully processed the OpenFlights airport and flight route datasets into a graph structure of airport vertices connected by flight route edges. In implementing our graph structure we chose to include every unique route between two airports (no different airline repeats) and stored the travel distance in that edge as its weight. The graph edges are also directed as a viable flight route from airport A to airport B. This does not necessarily mean there is a viable flight route from airport B to airport A.

The OpenFlights airport data is organized as comma separated entries, separated by line for each airport (Fig 1). These entries contain the airports database OpenFlights code, name, city, country, IATA code, ICAO code, latitude, longitude, altitude, timezone, daylight savings time, port type and data source. Of these entries, the OpenFlights code through the airports longitude were used. Additionally, some airports contained a second city name, for which separate data processing code was used. The route data is organized in the same way with entries for airline IATA or ICAO code, OpenFlights airline code, source airport IATA/ICAO code, source airport OpenFlights code, destination airport IATA/ICAO code, destination airport OpenFlights code, codeshare, number of stops, and equipment. Of these entries, the OpenFlights airline code, codeshare, and equipment data were not used. Processing the data from their respective dat files involved using ifstream to read the file lines, and stringstream to read comma separated entries. Data was cleaned of extra whitespace and characters such as quotation marks and then stored into either a Vertex or Edge object which was then added to the graph structure. Additionally, airport OpenFlights codes were mapped to their IATA and ICAO codes as well as their airport names for easy code/name conversion. This was useful as certain route data was missing OpenFlights codes for its airports and lookup by code or airport name was needed. As flight routes were processed, the travel distance was calculated from the start and end latitudes and longitudes using the Haversine (great circle) distance formula. In the final graph structure, there is only one edge for each unique combination of source and destination airport and it stores the route with the least number of stops.

The graph structure is implemented as a hash table (std::unordered_map) of Vertex objects mapped by their corresponding OpenFlights airport code (Fig 2). Each vertex internally stores an adjacency list of Edge objects representing outgoing flights. Additionally, each vertex stores a hash table of the same Edge objects mapped by the destination airport code for quick check of existing edges when processing data into the structure. The structure also features helper/access functions for retrieval, addition, and removal of Vertices and Edges using input of object, name or any form of code. Additionally, helper functions were written for printing and writing graph structure data as well as updating the data source file.

The traversal implemented by our graph structure is a breadth first search. This traversal is most useful for our dataset as it will traverse nearby nodes first, or nearby airports in the case of our graph. The traversal makes use of each vertex's adjacency list to expand outward from a

starting point, adding neighbors to a queue and removing from the queue to find the next vertex to visit. The traversal is stored and returned as a vector of Vertex objects. See Figure 3 for an example traversal.

The first algorithm implemented by our graph structure is Dijkstra's shortest path. This algorithm makes use of a priority queue to traverse outward from a starting point, storing a linked path which is updated when a shorter path is found. The final path is then traced backwards from endpoint to startpoint and then reversed and returned. This path is also stored as a vector of vertices. See Figure 4 for a sample output.

The second algorithm implemented by our graph structure is the landmark path. This algorithm finds the shortest path between a startpoint and an endpoint which includes a midpoint. This was done by combining the shortest path from start to mid and the shortest path from mid to end using the previously written Dijkstra shortest path. See Figure 5 for a sample output.

The graph structure and each of its key algorithms was tested using a catch framework. Key values were crosschecked with the input data and manual calculations / pathing. Additionally, output / structure data was written to output files for manual checking.

In conclusion, we found that the graph structure was most useful for network datasets such as the OpenFlight dataset and increased our overall understanding of how graphs can be implemented. We made some interesting discoveries throughout our project. For example, we used our implementation to find the largest airports in the set. We counted both the greatest number of unique routes, and the greatest number of total routes, with repetition for multiple airlines. We found that the airport with the most total flights was Frankfurt Main Airport and the airport with the most unique flights was Hartsfield Jackson Atlanta International Airport. We also discovered that not all flights were bidirectional which is something we had to account for in our implementation.

Figures

```
1,"Goroka Airport","Goroka","Papua New Guinea","GKA","AYGA",-6.081689834590001,145.391998291,5282,10
2,"Madang Airport","Madang","Papua New Guinea","MAG","AYMD",-5.20707988739,145.789001465,20,10,"U","
3,"Mount Hagen Kagamuga Airport","Mount Hagen","Papua New Guinea","HGU","AYMH",-5.826789855957031,14
```

Figure 1

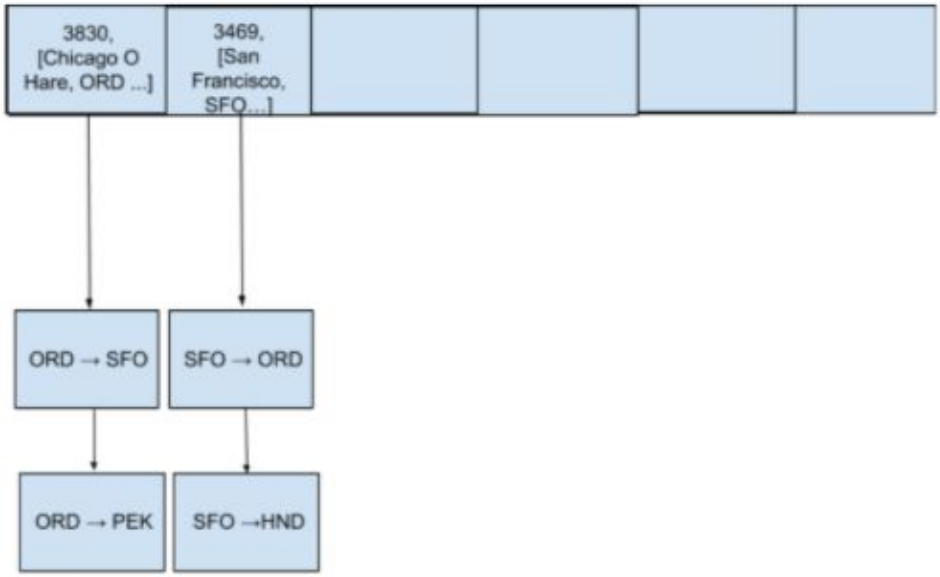

Figure 2

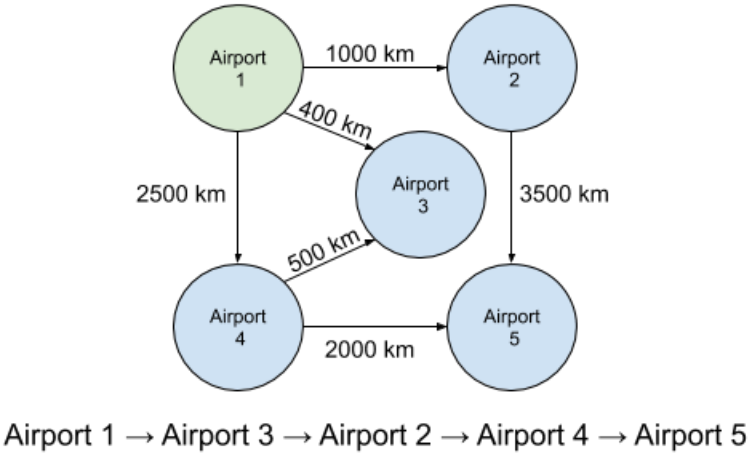

Figure 3

```
SHORTEST PATH BETWEEN [ORD, Chicago O'Hare International Airport] AND [HND, Tokyo Haneda International Airport]

3830, ORD, Chicago O'Hare International Airport, Chicago, United States, 41.9786, -87.9048
156, YVR, Vancouver International Airport, Vancouver, Canada, 49.1939, -123.184
2359, HND, Tokyo Haneda International Airport, Tokyo, Japan, 35.5523, 139.78

TOTAL DISTANCE: 10390km
```

Figure 4

```
SHORTEST PATH BETWEEN [ORD, Chicago O'Hare International Airport] AND [HND, Tokyo Haneda International Airport] THROUGH SELECTED MIDPOINT

3830, ORD, Chicago O'Hare International Airport, Chicago, United States, 41.9786, -87.9048
3520, DCA, Ronald Reagan Washington National Airport, Washington, United States, 38.8521, -77.0377
3797, JFK, John F Kennedy International Airport, New York, United States, 40.6398, -73.7789
3992, KIX, Kansai International Airport, Osaka, Japan, 34.4273, 135.244
2359, HND, Tokyo Haneda International Airport, Tokyo, Japan, 35.5523, 139.78

TOTAL DISTANCE: 12911.2km
```

Figure 5