Alex Li
Praval Telagi
Kevin Zhao

Results

For our final project, we elected to explore the OpenFlights data. We implemented graph algorithms to create a program that generate flight paths for a user input origin and destination(s). This program would be helpful for travelers who are interested in planning out a potential trip.

We were successful in creating a functional program to generate flight path. To briefly describe our creation: the program parsed through the airport and air routes data provided by OpenFlights to create a map of all documented airports around the globe and connect them with routes. We implemented Dijkstra's algorithm to find the shortest path between the user's specified origin and destination point. We merged multiple Dijkstra results to create a Landmark Path algorithm intended to provide a sequence of paths for users who intended to travel to multiple destinations.

Throughout the process of working on the project, our group made multiple interesting discoveries about both the data and programming practices. Our data set contained nearly 7,700 airports and over 67,000 routes, which made us believe that every airport on the map would be connected in some fashion. However, when testing our program, we discovered that many airports were uncharted by air routes so not all destinations had a valid path. Specifically, we found that there were in total 4470 connected components in the graph. (We discovered this result by implementing a disjoint set to count unions between airports on the graph. Each set is counted as a connected component, and within each set, there is a path between any two airports).

Another discovery we made while testing functionality of our code revolved around the storage of numeric data. As shown in figure 1, when we searched a path from Chicago to Los Angeles, the program returned a flight sequence from Chicago to Colorado Springs to LA. Intuitively, this result did not make sense to our group, as a straight path between two points should be shorter than any combination of paths. Upon investigation, we found that our edge weight, which indicated distances between airports, were stored as an int, as supposed to a double. As such, we lost precision within our numeric computation of distances, thus altering the outcome of the weight comparison used by Dijkstra's algorithm. When we adjusted the data type stored by our graph, the correct result returned, as shown in figure 2.

```
Please enter your origin airport id
3830
Origin Airport: "Chicago O'Hare International Airport"
How many destination airports would you like?
1
Please enter your destination airport id
3484
Destination #1 : "Los Angeles International Airport"
Full Path of Trip:
"Chicago O'Hare International Airport" -> "City of Colorado Springs Municipal Airport" -> "Los Angeles International Airport"
Total Distance: 2801
Bon Voyage! :)
```

Figure 1

```
Please enter your origin airport id
3830
Origin Airport: "Chicago O'Hare International Airport"
How many destination airports would you like?
1
Please enter your destination airport id
3484
Destination #1 : "Los Angeles International Airport"
Full Path of Trip:
"Chicago O'Hare International Airport" -> "Los Angeles International Airport"
Total Distance: 2802
Bon Voyage! :)
```

Figure 2

Overall, we have successfully implemented powerful graph algorithm to create a functional and useful program using the airport dataset. While we are satisfied with our product,

we have thought of a few ways the usability can be further enhanced. One possible improvement would be to find an API that provide data about flight fares on airline routes so the program can help the user find the not just the shortest/fastest route but also the cheapest possible route.