COSC 320 – 001

Analysis of Algorithms

2020 Winter Term 2

Project Topic Number: #5

Pathfinder 5000

Group Lead: Sebi Unipan

Group Members: Sebi Unipan, Andrew Dunn, Victor Frunza, Radhi Sharma

Dataset.

The dataset consisted of three different parts:

1. Airline – This dataset contains the airline name, its alias, the airline code, and the callsign (which are basically unique identifiers used to distinguish between airlines.) along with this information the dataset also includes the country from which the airline originated from.
2. Airport – This dataset contains a list of the airports and their respective cities or countries in which they are located. Each Airport in the dataset is represented by a name, city and country it is located in. Additionally, both the longitude and latitudes were provided for each airport in the dataset. This specific set of data was then used to calculate the actual distance from one airport to another.
3. Routes – This dataset contains the different routes/paths available from the various airports listed in the dataset above. We utilized this particular set of data in conjunction with the airport dataset to create methods that would help us determine whether or not there was an existing route between airports. This dataset was important as it helped us construct a graph with existing flights and routes.

Implementation. Explain how you implemented the algorithm and tested it. All the subtle details should be included. This is just an explanation and you do not need to copy paste your implementation here. It can be as short as one paragraph. Include links if required

Graph Creation: Before implementing the algorithm, we had to create a network of flights that would represent our graphs. In this scenario our airports represented our nodes/vertices, and the flights represented our edges. We then created two classes, Airport and Flight with the necessary attributes and methods required to retrieve specific data from the respective datasets. We then created the FlightNetworkGenerator where we include methods that would generate various different flight networks utilizing the information from the dataset.

Algorithm: We decided to use Dijkstra's algorithm to calculate the best path. To do this we created a method called, bestFlight that would take in three parameters required to yield the shortest path. The method would take in a graph (FlightNetwork), a source and destination (both represented using an Airport). For this particular instance we initialized a priority queue to implement Dijkstra's algorithm. What the algorithm does is that it maintains a minimum priority queue of airports, and their corresponding distances. As we iterate through the graph, the algorithm compares all the values present within the queue and then extracts the minimum value from the priority queue each time. Additionally, it also examines the edges adjacent to the source, so if there is an airport/node adjacent to

the source, and if that particular edge yields the shortest path it will be added to the min priority queue.

Github link: https://github.com/sunipan/320-Project/blob/main/src/BestPath.java

Results.

The result was that our implementation runs almost instantaneously as the graph we created for canada which consists of about 237 airports and 1692 routes between the airports. Since we were not able to set up graphs to plot the performance we cannot get an accurate result or estimation of our runtime but with the paths available and number of nodes the implementation runs as expected from the pre-calculated runtime. We used an ArrayList to store the outgoing flights for each airport which definitely affects the amount of memory used by the algorithm, but not it's speed. Choosing to use a min-heap priority queue to order the airports for iteration helps the performance of the algorithm by a lot.

Unexpected Cases/Difficulties.

Difficulties that occurred with the implementation of our straightforward algorithm was getting it to work on everyone's computer as we used different IDEs. Things like file directories and certain functions to read the dataset into the airport objects really brought a lot of difficulties. Graphing the performance of our implementation has also been really tough as we had a limited time with creating graphs as well as finding a proper way to test our algorithm. The team member who was responsible for creating simple graphs had their PC die on them last minute and unfortunately with the time given we were not able to plot out our graph's performance in Java. A lot of time was also allocated for cleaning up the dataset to ensure it worked perfectly with the flight network generator. For the next implementation we plan to have several graphs as well as a well defined plot to analyze our runtimes.

Task Separation and Responsibilities.

Sebi Unipan was responsible for implementing the priority queue for the ordering of the airports in the graph and doing the write up for the final document

Victor Frunza was responsible for establishing our dataset and implementing a flight network generator to be used with our algorithm

Andrew Dunn was responsible for implementing our straightforward algorithm for finding the best way to reach the target airport with the lowest cost.