Data Structures and Algorithms
End Semester Project
<TRAVEL MANAGER>

Submitted By

Areeb Tariq (00907)

Arslan Ahmed (04160)

Hafiz Usman Mahmood (01317)

BEE – 4C

4th January, 2014

Problem Being Addressed

Many times people have to travel in case of emergency from one place to another. They have to search for flight schedules of different airlines and best land routes if within the country.

It takes much of their time in getting required information and then comparing it to decide the **quickest or cheapest option** for them. Our project is a solution to this problem.

Abstract

Our aim is to provide **travel guide** to the citizens. We will analyze two basic points in this project:

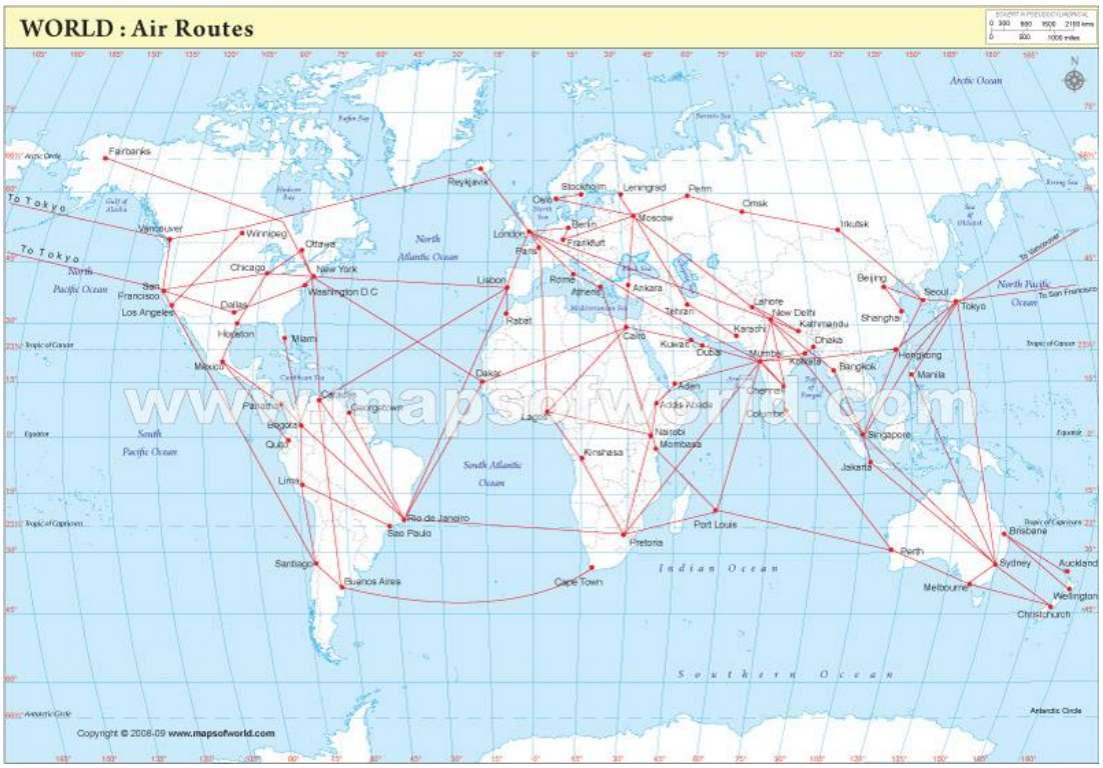
- Quickest Flight from our source city to our desired destination city
- Cheapest Flight from our source city to our desired destination city

Features

- Provide the quickest possible route.
- Fares also displayed with each route
    - alongwith Weather Conditions, Distance, Duration, Departure and Arrival Time.
- Sorting by fares also possible.
- Contact numbers of the services also displayed for timely reservation.

Scope

Our project is limited to only few cities of the world as it is just a prototype project. By adding more cities and more flights, we can implement it at commercial level.



BASIC TERMS

Nodes, Vertex = Cities

Edges = Flights (with weights as distance, fare, time, weather)

GRAPH IMPLEMENTATION

The main purpose of the project is the implementation of data structures and algorithms to real life problems. Our problem was related to the mapping of airline routes, in which different cities of the world were connected through flights.

In our case, we used graphs to implement the project because it was the best data structure related to this problem. The main reason behind selecting **GRAPHS** as our data structure was that in graphs, we have vertices connected to each other through edges, which is same like the connection of cities across the world through different flights.

We could have used **binary trees** to implement the project but the main reason we did not chose them was that in the binary trees, we have certain nodes which are not directly connected to each other, but have a connection in between them through their parent node, i.e. if we have a certain binary tree of height 4, and if we want to go from the left most leaf node to the right most leaf node, then we would have to traverse through whole of the tree by first going to the root

node and then traversing down the tree to reach the right most leaf node. Now, if we would have implemented this on the real life travel manager, then it would be a very inefficient way of expressing the results as it will cost the travel company lots of extra fuel.

In the graphs, however, majority of the nodes, if not all, are connected to each other in some way or the other. Suppose, we have a graph of 30 nodes connected to each other and we want to go from one node to another node, than there is a huge possibility that the nodes might be connected directly to each other which results in a much less traversal time.

DIJKSTRA ALGORITHM EMPLOYED

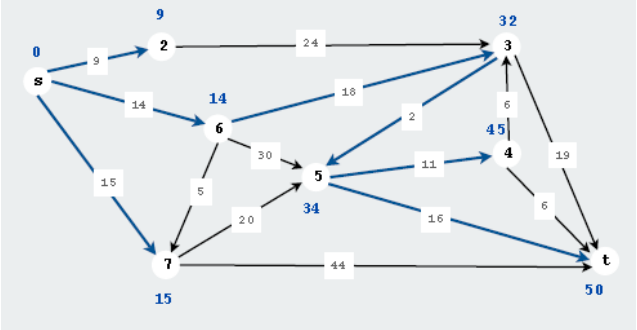
The main purpose of our project was to find out the shortest path between two cities in the graph w.r.t fare and duration both. We needed a travel manager, which help the passenger to get to his/her destination in the shortest possible time or in minimum cost.

For example,

One of the possible way of finding the shortest route between Islamabad and London would have been to enumerate all the paths between these two cities, add up all the path lengths and then calculate the shortest path among them.

However, in the above approach, we could also have got the path between these two cities which are simply not worth considering. For example, going from Islamabad to London via Dubai is a thing that can be considered as an optimal path but there could also be a path through Moscow, which is an extremely poor choice because Moscow is hundreds of miles away from the main path. Hence, we applied the algorithm that would calculate the minimum distance of destination from the source.

Hence, we applied **Dijkstra's Shortest Path Algorithm** for finding out the shortest path between to cities. The main reason was that it is a uniform algorithm, which means that it should be used when we have no information about the graph and we cannot estimate the distance from one node to the other.



There are a number of other algorithms for finding out the shortest path between the source and destination which are as follows:

(i) **Bellman-Ford Algorithm**

The Bellman-Ford Algorithm is also used for finding out the shortest distance between two nodes, if they have **negative weights** on the edges. Though seems practical on the simulations, but in real life, we find out that there are no negative distances between the cities. Also, the time complexity of this algorithm in worst case scenario is $O(|V||E|)$ which is certainly not required because as the number of vertices (cities) and edges (flights) increase, the running time of the algorithm increases.

(ii) **Breadth First Search**

The breadth first search (BFS) algorithm is also widely used for finding out the shortest path between two nodes. If the **weights on the edges are same**, then BFS is a better choice than Dijkstra's Algorithm but we see that in real life, it is almost impossible that all the cities will have same distances between them. Hence, Dijkstra's Algorithm felt to be a better choice in our case.

(iii) **A* Algorithm**

The A* algorithm is also used for finding out the shortest path between two nodes in a graph. It has a time complexity of $O(|E|)$. This is a good competitor of Dijkstra's Algorithm but does not give optimal solution for finding out the shortest path. It makes certain **assumptions**, known as heuristic estimate that gives an estimate of the shortest route from one point to another.

From the above description of the alternative algorithms, we find out each algorithm has its own drawbacks. Bellman-Ford is slower, BFS works for same edge weights while A* makes assumption due to which it is possible that we may not find out the optimal path from one city to another.Hence, Dijkstra's Algorithm seemed to be right choice in our case.

STL VECTORS USED

We implemented the graph using **adjacency list**. The main reason for implementing graph using adjacency list was that our graph is sparse. For a **sparse graph**, adjacency list is a better choice than the adjacency matrix.**Adjacency Matrix** is well suited for **dense graphs** with E approaching to V^2. In real world, none of the airline operates such flights which connect all other airports of the world. Mainly because distance is too much or fuel, relaxation time etc counts in. Hence we used STL vectors to implement the adjacency list.

The main advantage of using **STL vectors** to implement adjacency list using vectors is that it storage operation corresponds to the complexity of $O(|V| + |E|)$. In the worst case scenario, there could be $C(V, 2)$ edges, thus consuming $O(V^2)$ space. However, adding the

vertex and edge to the adjacency list is easier causing only $O(1)$ time. Removing the vertex or edge takes $O(E)$ time. This time is because of the fact, when we need to remove a certain vertex or and edge, it is required to traverse through the whole list, find out the vertex or edge and then remove it.

OVERALL IMPLEMENTATION

As far as the solution of our project is concerned, it is correctly finding out the shortest path between the source and the destination city. The Dijkstra's Algorithm employed has a time complexity of $O(n^2)$, which in this case is $O(V^2)$ corresponding to the number of vertices (cities) in the graph.

In the main code we wrote for Dijkstra's Algorithm (apart from the helper function), there is a FOR loop which make them to run in Big-O time of$O(n)$.

After this, the helper function is called. In the helper function, there is a FOR loop which checks if a particular city is visited or not. This for loop hence runs in a linear time as it has to traverse through all the cities listed in vectors. Further, our recursive function contains more FOR loops. It is for iterating through all the cities which gets them marked as visited. This FOR loop further contains nested FOR loops which is the main part in determining the time complexity of our code.

The first FOR loop encountered in the group of nested loops goes into the adjacency list of the given source city and updates all the information corresponding to the flights originating from that city. Now, if we have a city at the end of our adjacency list, then we will have to traverse through the whole list which will make it run in linear time $O(n)$. The statements inside the FOR loop are dependent on the loop due to which there will appear some constants in the Big-O notation mentioned above but they can be neglected.

The second FOR loop checks for the non-visited cities with least duration or fare. It check all the flights from the source by iterating through them. This loop runs in $O(n)$ time. Similar to the previous case, this FOR loop also contains some statements like assigning the city with shortest duration/fare from the source as temporary city. This will only include some constants in the Big-O running time but overall, the running time of second FOR loop remains linear.

After considering the effect of all these FOR loops, we find out that the nested FOR loops have a Big-O running time of $O(n^2)$. Hence, the total time taken by Dijkstra's Algorithm for finding the shortest path from source to destination would be:

$$O(n + n + n^2)$$

$$= O(2n + n^2)$$

On dropping the linear term, as the dominant term is the quadratic factor, the time complexity is $O(n^2)$, which in this case is $O(V^2)$.

ACTUAL RUNNING TIME ON LAPTOP

The Dijkstra Algorithm takes as low as only **22milliseconds** (for direct flights) to traverse through the graph and find the shortest distance from source city to the destination city for a graph containing 32 cities and approximately more than 100+ flights.

PROBLEMS FACED DURING PROJECT

The main problem we face while implementing the algorithm was the optimization of running time of the code. Since, we were implementing the graph using the STL vectors, no specific help was provided on the internet in this regard. The codes on the internet were all implemented for the binary heaps and priority queues.

The code for Dijkstra we first wrote had a time complexity of almost $\boldsymbol{O(n^4)}$ due to which our code ran in **5 seconds,** though we were able to find out the shortest route between source and destination. But this was not considered an optimal solution to our problem. Hence, we debugged the code and removed some extra statements and FOR loops due to which we were able to achieve a running time in milliseconds.

DISCUSSION

We can **further optimize** the running time of our project by implementing graphs using Fibonacci heap or binary heap as a priority queue. If we implement the graph using the binary heaps, the time complexity would be $\theta(|E| + |V|\log|V|)$. The Fibonacci heap can further improve this time to $O(|E| + |V|\log|V|)$.

RESULTS AND CONCLUSION

In short, our travel manager displays all needed info to the user like arrival time, departure time, duration, distance, fare of the flights. Moreover, user can sort the flights based on cost/duration whether he wants to go quick or whether he prefers economy.He also gets the Contact numbers of concerned airports and the weather at that time. All of this info is displayed in a very short time, in tabular form which is easy to read.

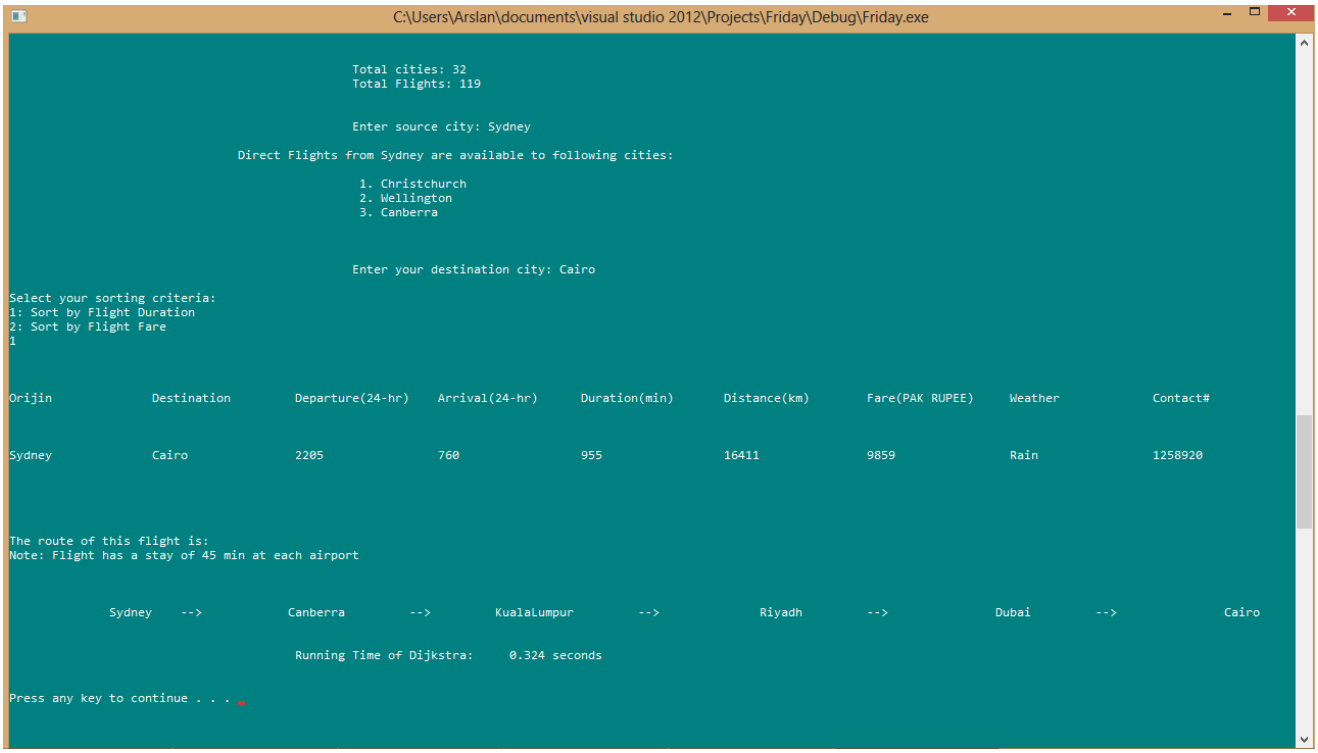
Our complete project involves map of cities linked with flights. We have used STL Vector implementation throughout our project, and path sorting algorithm used was Dijkstra. Running Time of our algorithm was $O(n^2)$, where $n = number\ of\ cities$.If we had used fibonaci heaps with priority queues, we could achieve $O(|E| + |V|\log|V|)$.Indeed, it was a fun solving a real world problem by coding this project.

Project Level

It is of level 3 project according to criteria mentioned in the class by the instructor. We have used two data structures-Graphs, and STL vectors (not studied in class). Making use of two data structures makes it Level 3 project.

SCREEN SHOTS

```
1   /* ... */
13  #include ...          //library to set width and allignment of colums
19  #define INF INT_MAX    //define infinity value, it will be used to initialize distances of nodes at start of algo
20  using namespace std;
21
22  //forward declaration of City class
23  class City;
24
25  //An object of this class represents a flight from one city to another
26  class Flight { ... };
78
79  //Visit Status of a city
80  enum Status { ... };
84
85  //Different cities of the world are objects of class City
86  class City { ... };
185
186 //Helper function to reduce no. of Lines Of Code in Dijsktra() method
187 void helper(string source, string destination, vector<City*>cities, int sortingCriteria) { ... }
277
278 //Gives the path of flight from source to destination
279 void pathfunction(string destination,string source,  vector<City*> cityList, vector<string>& route) { ... }
299
300 //Class having all the cities
301 class Graph { ... };
476
477 void main() { ... }
```


C:\Users\Arslan\documents\visual studio 2012\Projects\Friday\Debug\Friday.exe
Total cities: 32
Total Flights: 119
Enter source city: Sydney
Direct Flights from Sydney are available to following cities:
1. Christchurch
2. Wellington
3. Canberra
Enter your destination city: Cairo
Select your sorting criteria:
1: Sort by Flight Duration
2: Sort by Flight Fare
1
Orijin Destination Departure(24-hr) Arrival(24-hr) Duration(min) Distance(km) Fare(PAK RUPEE) Weather Contact#
Sydney Cairo 2205 760 955 16411 9859 Rain 1258920
The route of this flight is:
Note: Flight has a stay of 45 min at each airport
Sydney --> Canberra --> KualaLumpur --> Riyadh --> Dubai --> Cairo
Running Time of Dijkstra: 0.324 seconds
Press any key to continue . . .