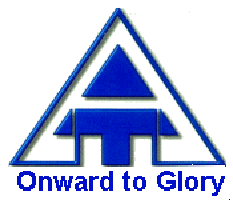
**ADS MINI PROJECT REPORT**

FLIGHTS SCHEDULING



**ARMY INSTITUTE OF TECHNOLOGY**

**(DEPARTMENT OF COMPUTER ENGINEERING)**

**SE COMPUTER SEM-II 2018-19**

SUBMITTED BY :-

DHEERAJ KUMAR (3237)

ABSTRACT

Aeroplanes preserve an expense during their flight as well as in standby circumstance in the airport. Each and every flight operated is a multi-thousand dollar gamble on whether or not passengers will purchase tickets. Aside from the expense, the total time of flight is a noteworthy issue for passengers who need to board a flight. So, airlines put tremendous effort into properly assembling their puzzle flights so they can provide the shortest connection time possible. It’s just natural that most routes require connections so airways need to attract connecting passengers in order to stay in business. Our work will assist to maintain their flights and make maximum use of their flights. Max flow Algorithm helps us to find out the best possible solution to tackle this problem. With Max flow Algorithm, This work ensures suitable time maintenance. Our fundamental goal is to preserve an airline schedule in such a way so that the aeroplanes continue to be in flight mode for a maximum period of time making use of the max flow algorithm. With the help of this project we will demonstrate the use of Max flow Algorithm to make a flight scheduling program.

* Software and Hardware Requirement:-
* Software:
  + Ubuntu or Linux for the operating system
  + Gedit: to write the program
  + G++ compiler: to compile the program
* Hardware:
  + RAM : 4 GB
  + HARD DISK: 500GB
* Platform used:-
  + OS: This project is compiled and executed on the Ubuntu operating system.
  + COMPILER: This program is compiled with the help of g++ compiler.
  + LANGUAGE: The program is written in the c++ language
  + HARDWARE: 4GB RAM with 1TD hard disk

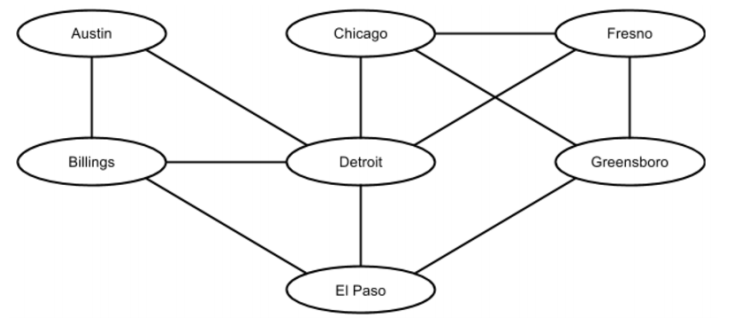
INTRODUCTION

In this project, you will determine all possible flight plans for a person wishing to travel between two different cities serviced by an airline (assuming a path exists). You will also calculate the total cost incurred for all parts of the trip. For this project, you will use information from two different input files in order to calculate the trip plan and total cost.

* **Origination and Destination Data** – This file will contain a sequence of city pairs representing different legs of flights that can be considered in preparing a flight plan. For each leg, the file will also contain a dollar cost for that leg and a time to travel. For each pair in the file, you can assume that it is possible to fly both directions.
* **Requested Flights** – This file will contain a sequence of origin/destination city pairs. For each pair, your program will determine if the flight is or is not possible. If it is possible, it will output to a file the flight plan with the total cost for the flight. If it is not possible, then a suitable message will be written to the output file. The names of the two input files as well as the output file will be provided via command line arguments.

**Flight Data**

Consider a flight from Dallas to Paris. It’s possible that there is a direct flight, or it may be the case that a stop must be made in Chicago. One stop in Chicago would mean the flight would have two legs. We can think of the complete set of flights between different cities serviced by our airline as a directed graph. An example of an undirected graph is given in Figure 1.



In this example, a line from one city to another indicates a flight path both ways between the cities. The price and flight time is the same for both directions. If we wanted to travel from El Paso to city Chicago, we would have to pass through Detroit. This would be a trip with two legs. It is possible that there might not be a path from one city to another city. In this case, you’d print an error message indicating such. In forming a flight plan from a set of flight legs, one must consider the possibility of cycles. In Figure 1, notice there is a cycle involving Chicago, Fresno, and Greensboro. In a flight plan from city X to city Y, a particular city should appear no more than one time. The input file for flight data will represent a sequence of origin/destination city pairs with a cost of that flight. The first line of the input file will contain an integer which indicates the total number of origin/destination pairs contained in the file.

**Sample Data**

* **Flight Data**

Here is an example of a flight data input file (it is not one that goes with Figure 1):

**4**

**Dallas|Austin|98|47**

**Austin|Houston|95|39**

**Dallas|Houston|101|51**

**Austin|Chicago|144|192**

The first line of the file will contain an integer indicating how many rows of data will be in the file. Each subsequent row will contain two city names, the cost of the flight, and the number of minutes of the flight. Each field will be separated with a pipe (shift­\ on most keyboards).

* **Requested Flight Plans**

A sample input file for requested flight plans is shown below. The first line will contain an integer indicating the number of flight plans requested. The subsequent lines will contain a piped limited list of city pairs with a trailing character to indicate sorting the output of flights by time (T) or cost (C). Your solution will find all flight paths between these two cities (if any exists) and calculate the total cost of the flights and the total time in the air.

**2 Dallas|Houston|T Chicago|Dallas|C**

* **Output File**

For each flight in the Requested Flight Plans file, your program will print the three most efficient flight plans available based on whether the request was to order by time or cost. If there are fewer than three possible plans, output all of the possible plans. If no flight plan can be created, then an error message should be output. Here is an example:

**Flight 1: Dallas, Houston (Time)**

**Path 1: Dallas ‐> Houston. Time: 51 Cost: 101.00**

**Path 2: Dallas ‐> Austin ‐> Houston. Time: 86 Cost: 193.00**

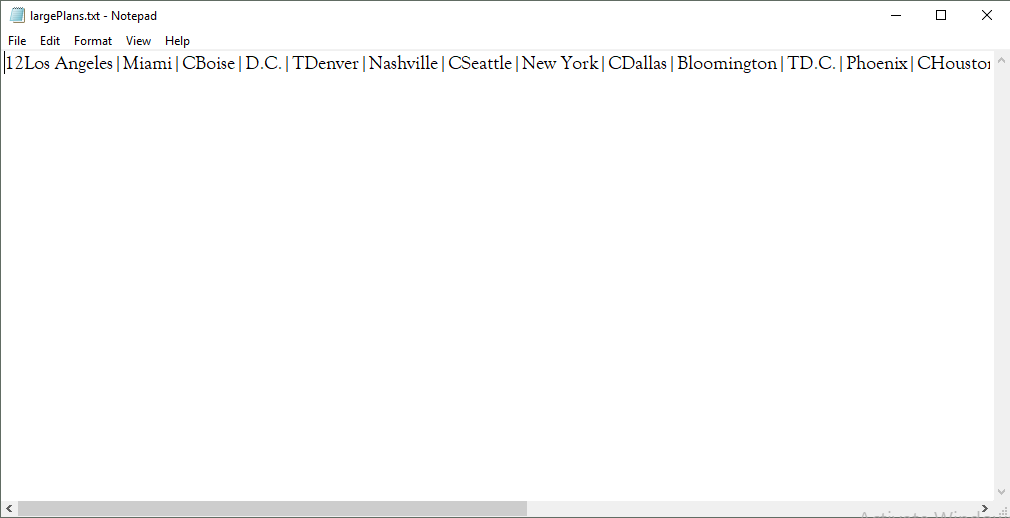
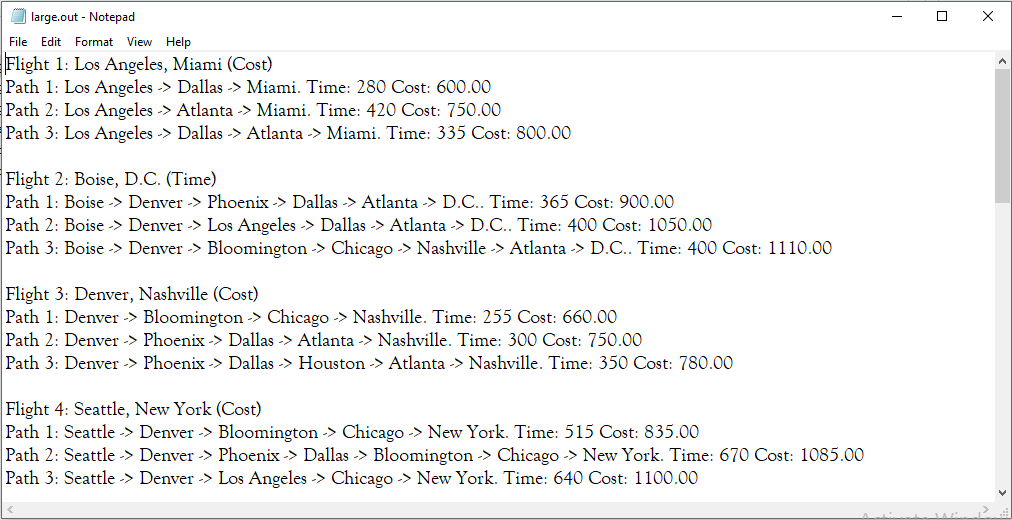
**Flight 2: Chicago, Dallas (Cost)**

**Path 1: Chicago ‐> Austin ‐> Dallas. Time: 237 Cost: 242.00**

**Path 2: Chicago ‐> Austin ‐> Houston ‐> Dallas. Time: 282 Cost: 340.00**

**Snapshots:-**



* **Aims and objectives**

Aims and objectives of this research work are pointed out below:

* + - To Save Money.
    - To save Cost.
    - To save time.
    - To utilize aeroplane resources properly.
    - To make great use of airline resources, thus it will help to make much more profit.
    - To use the airline crew in a very efficient way. So a lot of money can save.
    - To give the customer more information about the flight.
    - To arrange airline in a very effective manner.

ALGORITHM AND DATA STRUCTURE USED:

**Algorithm**:

Step 1: Take information from the authority.

Step 2: Make a node for each and every flight of the airline scheduling.

Step 3: Give capacity of every link from one node to another. The capacity is given by the time of the flight comes to the airline.

Step 4: Start from the node of the current flight. Make the flow 0.

Step 5: Do BFS from the node and find the available node from the current node.

Step 6: Then check for more flow if there are possible.

Step 7: If there is no more flow at all, then return.

Step 8: If there is more flow, then update the network.

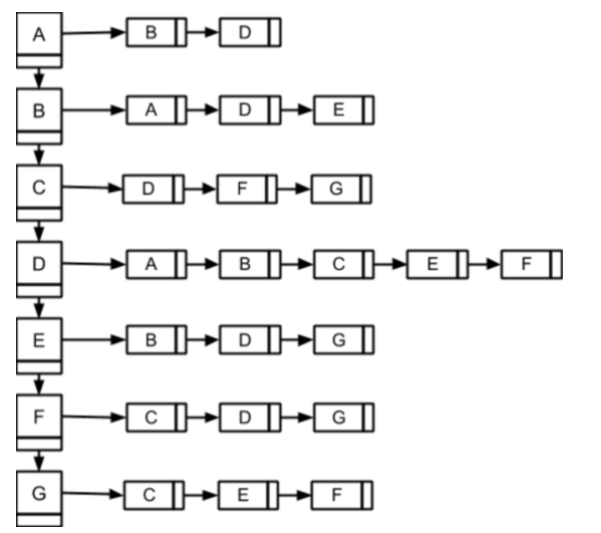
Step 9: Goto step 5.

Step 10: End.

Data structure used:

* **LinkedList** : it is the user made linked list with the help of STL
* **AdjacencyList** : it is the user made Adjacency List with the help of class LinkedList in STL
* **Stack** : it is the user made stack with the help of STL
* **Vector**: it is the user made stack with the help of STL
* **Graph** : it is implemented with the help of AdjacencyList

Implementation Details and Requirements In order to store the structure representing flights serviced by the company, you will implement a simple adjacency list data structure. Essentially, it will be a linked list of linked lists. There will be one linked list for every distinct city. Each list will contain the cities (and other needed info) that can be reached from this city. Figure 2 is an example representation of an adjacency list for the graph in Figure.



The larger squares on the left represent the list cities (with one node for each city). The list to which each node is pointing represents a city from which you can get to from the parent node. For example, from city A, it is possible to fly to cities B and D. To solve this problem, you’ll need to implement an exhaustive search of all flights. To achieve this, you’ll implement an iterative backtracking algorithm (using a stack). As you are calculating the flight path, you will use the stack to “remember” where you are in the search. The stack will also be used in the event that you've gone down a path that does not lead to the destination city. This algorithm method will be discussed in lecture, and you are encouraged to do some of your own research. For this project, you may NOT use any of the STL container classes or associated algorithms. You MAY use string objects if you’d like. Therefore, you must implement a linked list class and a stack class. Your stack class could make use of the linked list class. Additionally, your implementation should be object­oriented in both design and implementation. Minimize the amount of code you have in your main method. Note that implementing a single class in which you have multiple methods does not make your solution object oriented in design.

**Executing Your Program:**

The final version of your program will be run from the command line with the following arguments:

**./a.out <flight data file name> <paths to calculate file name> <output file name>**

CONCLUSION AND FUTURE SCOPE

This research idea will save money and make the airline industry more profitable and reliable. It will help the authority to facilitate their work more easily. It will also help the customer to get their desired flight in time and thus save their time. In future customer management system may be included for promoting them in different categories (e.g. frequent flyers, regular flyers etc.). Airline reservation system may make the system more interactive. And also the crew scheduling system can also be incorporated so that they can be used more effectively.

REFRENCES

* + ilab-australia.org/jms/wp-content/uploads/2018/09/28.-P.ICT\_.012.pdf
  + <https://github.com/gpiskas/Flights_Scheduling_Max_Flow>