**Open Ended Project Report**

**On**

Air Travel Made Cheap

Submitted for the partial fulfillment of Bachelor of Engineering

by

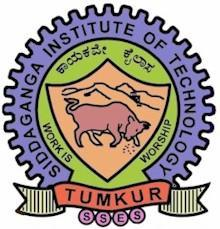
**Arup Das (1SI15CS015)**

**Under the guidance of**

Mrs. [S Thejaswini](javascript:openMyPopUp%20('facultyprofile/15092017thejaswini%20S.pdf');) M. Tech

Assistant Professor,

Department of CSE, SIT



**Department of Computer Science and Engineering**

**Siddaganga Institute of Technology, Tumkur – 572103**

(An Autonomous Institution, Affiliated to VTU, Belagavi & Recognized by AICTE, New Delhi)

**2016-2017**

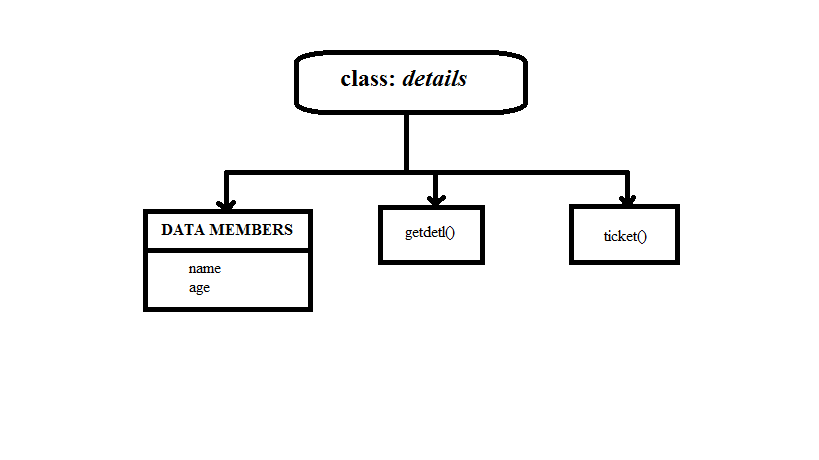
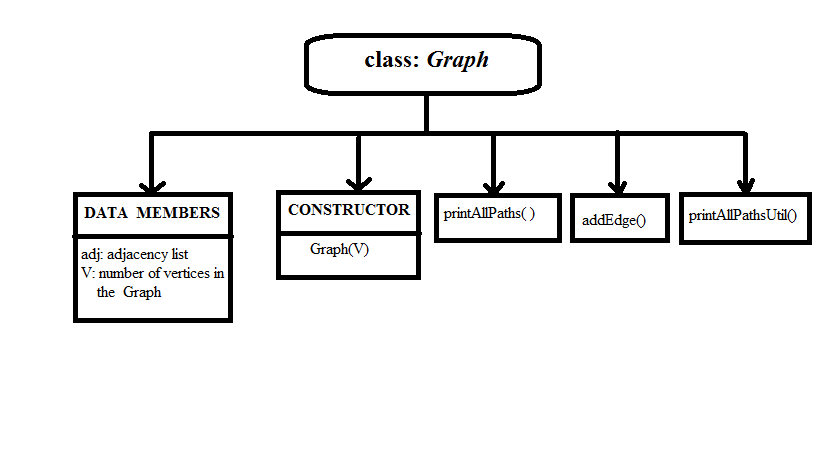
Introduction:

***Air Travel Made Cheap*** is an attempt to simulate a travel company providing services like booking flight tickets. What it all seeks from the customer is its travel from and to location to provide the cheapest path of journey.

The company has successfully established its flying routes to Jammu , Delhi , Mumbai , Bangalore , Trivandrum and Kolkata. A customer can either choose to go for a single trip or a round trip. Based on the locations chosen within the limits of the flying routes a cheapest path is determined. After the determination of the cheapest path a ticket is generated for the customer.

The undertaken project extensively implements the concepts of analysis and design of algorithms in the framing of the core logic of the program keeping in view the time and space efficiency.

Architecture Block Diagram and its description:



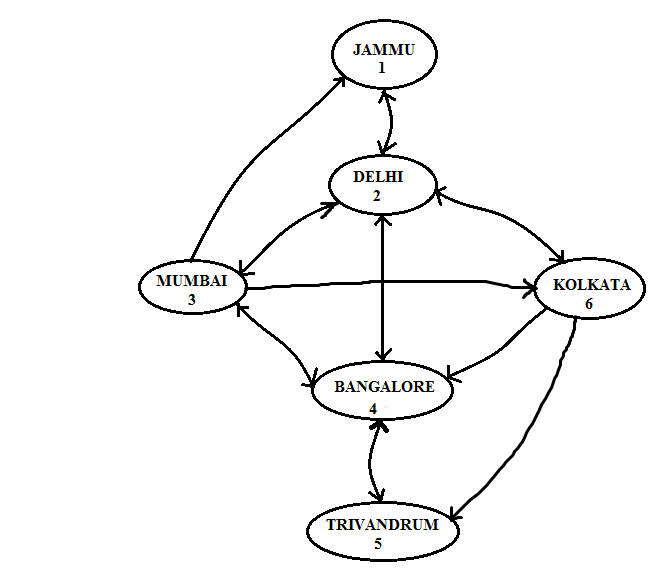
In the following project class details contains two data members name and age of type protected. The data member name stores the name of the passenger booking the flight and his/her age is stored in the data member age. It contains two public member functions getdetl() and ticket().

The getdetl() stores the name and age of the passenger. The ticket() generates the flight ticket of the passenger on successful determination of cheapest path.

To find the cheapest path between the source and the destination every end of a flying route is considered to be a vertex of the weighted graph. The graph is weighted with respect to the cost of travel from one vertex to the other. To implement this idea, the class Graph has been used.

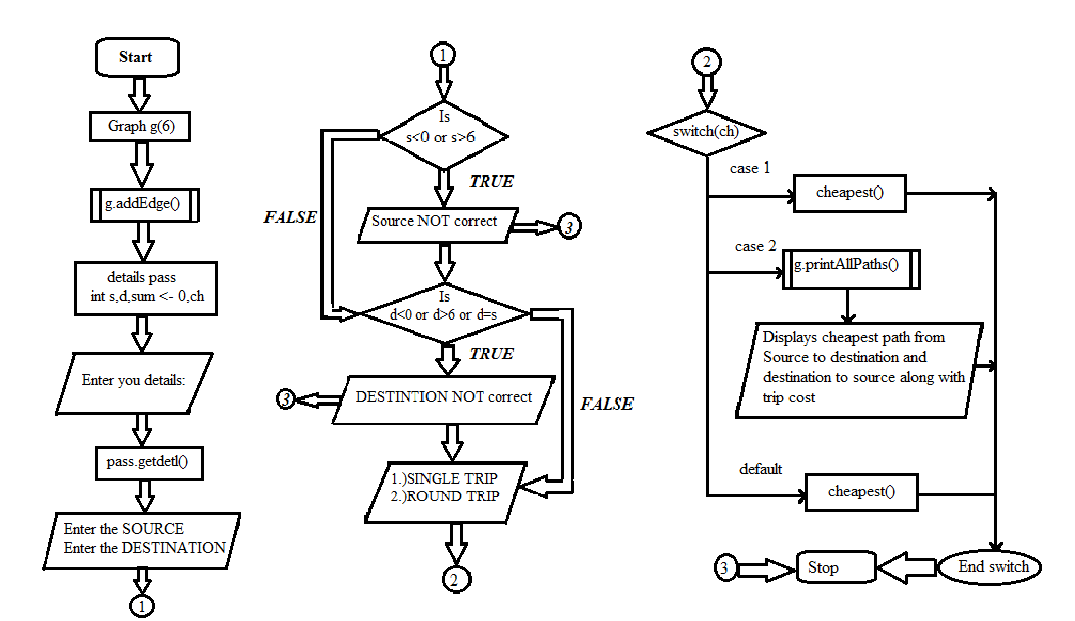
The class Graph contains two private data members adj and V.The data member adj keeps a track of all the neighboring vertices of a vertex by maintaining an adjacency list for each vertex. On the other hand, the data member V contains the number of vertices present in the graph.

It contains three member functions namely printAllPathsUtil(),addEdge() and printAllPaths().The function addEdge() is used to establish an edge that is, a route between two vertices during the creation of the graph. The graph is created by invoking the constructor graph() which initializes a graph with V vertices. The graph created is shown in the below figure.

****

The function printAllPaths() is used to print all the travel paths possible between the source and the destination vertex. It calls the function printAllPathsUtil() which determines all the possible paths through DFS traversal technique.

Flowchart:



Data Structure used:

The given project uses depth – first search traversal technique to compute the cheapest path between the source vertex and the destination vertex. The traversal occurs in a recursive manner thereby storing the recursive calls in a stack. The last vertex pushed into the stack is popped out first. Hence the data structure primarily used in this project is stack.

The project also makes use of a linear linked list to maintain the adjacency list of each vertex.

Algorithm design technique

Algorithm: g (n)

//Algorithm for computing the cheapest path between cities

//Input: Takes no of vertices =n , as input

//Output: Prints the cheapest cost of travel and its path

While edges are there in the directed graph g

addEdge(n,m); //edge from n to m;

int s , d , ch , sum←0;

getdetl(); //function to get the details of passenger;

Show the details of the travel

Take value of s and d

s←source ; s←s-1;

d←destination; d←d-1;

checking for the correctness of the inputs;

choose amongst the options and ch←choosen\_option //single trip or round trip

if ch=1

cheapest();

if ch=2

{

printAllPaths(s,d);

for i←0 to siz do

print path of all flights;

sum+=m;

print sum;

printAllPaths(d,s);

for i←siz to 0 do

print path of all flights;

print m;

sum+=m;

print sum; //The Total cost of the round trip;

}

ticket(s,d); //prints the generated ticket ;

return 0;

*/\*The supporting functions are as follows\*/*

Algorithm:addEdge(n,m)

//Adds an edge to the existing graph

//Input: The starting vertex ‘s’ and ending vertex ‘d’

//Output: An added directed edge to the graph

Edge n to m is added to list

adj[n]← edge n to m; //adj[n] is the adjacency list of n;

Algorithm:getdetl( )

//Takes the information of traveler

//Input: No explicit input, takes the name and age of the user inside function

//Output: Stored data of the user

name ← users\_name\_as\_input;

age←users\_age\_as\_input;

Algorithm:cheapest( )

//Finds the cheapest path from source to destination

//Input: source and destination nodes

//Output: The cheapest path with cost

//Print cheapest paths

for i←siz to 0 do

print path of source to destination;

print path’s cost

Algorithm:printAllPaths(d,s)

//Finds all the paths from source to destination

//Input: source and destination nodes

//Output: All the paths

Creating Boolean Matrix visited[];

Creating integer array path[];

path\_index ← 0;

Initialize visited[] to False;

printAllPathsUtil(s, d, visited, path, path\_index);

Algorithm:printAllPathsUtil(s,d,visited,path,path\_index)

//Finds all the paths from source to destination

//Input: source,destination,visited and path arrays and path\_index

//Output: All the paths with subsequent cost

visited[u] ←true;

path[path\_index] ← u;

path\_index++;

if u =d

{

price=0;

int c=0;

for i ←0 to i<path\_index do

{

If i=path\_index-1

{

If price<m

{

m←price;

for j←0 to j<path\_index do

arr[c++]←path[j];

siz=c;

}

}

else

{

int x ←path[i];

int y ←path[i+1];

display edge;

price+=cost[x][y];

}

}

}

else

{

visit the adjacent nodes of list recursively

if node is not visited

{

Mark it as visited and call

printAllPathsUtil with unvisited node;

}

}

Technique Used

The Decrease -and –Conquer design technique along with the recursive approach is used

in the designing of the algorithm. The large problem of traversal is broken or decreased

in an instance of smaller size and the solution is obtained for it recursively.

Implementation

Code:

#include<iomanip>

#include<iostream>

#include <list>

#include <time.h>

#include <cstring>

using namespace std;

//Making a 2-D array to store the weights of the edges

intcost[6][6]= {{0,1000,0,0,0,0},

{1000,0,3000,4000,0,2500},

{3000,2500,0,4000,0,4000},

{0,4000,4000,0,2500,0},

{0,0,0,2000,0,0},

{0,2500,0,2000,5000,0},

};

int price;//Global variable to store cost if each traversal of list

intarr[6],siz,m=99999;

string cities[]={"Jammu","Delhi","Mumbai","Banglore","Trivandrum","Kolkata"};

// A directed graph using adjacency list representation

class Graph

{

intV; // No. of vertices in graph

list<int> \*adj; // Pointer to an array containing adjacency lists

// A recursive function used by printAllPaths()

void printAllPathsUtil(int , int , bool [], int [], int&);

public:

Graph(int V); // Constructor

void addEdge(int u, int v);

void printAllPaths(int s, int d);

};

Graph::Graph(int V)

{

this->V = V;

adj = new list<int>[V];

}

void Graph::addEdge(int u, int v)

{

adj[u].push\_back(v); // Add v to u’s list.

}

// Prints all paths from 's' to 'd'

void Graph::printAllPaths(int s, int d)

{

// Make an array of boolean type to indicated visited & unvisited nodes

bool \*visited = new bool[V];

// Create an array to store paths

int \*path = new int[V];

intpath\_index = 0; // Initialize path[] as empty

// Initialize all vertices as not visited

for (inti = 0; i< V; i++)

visited[i] = false;

for(inti=0;i<6;i++)arr[i]=0;

// Call the recursive helper function to print all paths

printAllPathsUtil(s, d, visited, path, path\_index);

}

// A recursive function to print all paths from 'u' to 'd'.

// visited[] keeps track of vertices in current path.

// path[] stores actual vertices and path\_index is current

// index in path[]

void Graph::printAllPathsUtil(int u, int d, bool visited[],

intpath[], int&path\_index)

{

// Mark the current node and store it in path[]

visited[u] = true;

path[path\_index] = u;

path\_index++;

// If current vertex is same as destination, then print

// current path[]

if (u == d)

{

price=0;

int c=0;

for (inti = 0; i<path\_index; i++)

{

if(i==path\_index-1)

{

cout<< cities[path[i]] << " ";

cout<<" = Rs "<<price;

if(price<m)

{

m=price;

for(int j=0;j<path\_index;j++)

arr[c++]=path[j];

siz=c-1;

}

}

else

{

int x=path[i];

int y=path[i+1];

cout<< cities[path[i]] << " -> ";

price+=cost[x][y];

}

}

cout<<endl;

}

else // If current vertex is not destination

{

// Recur for all the vertices adjacent to current vertex

list<int>::iterator i;

for (i = adj[u].begin(); i != adj[u].end(); ++i)

if (!visited[\*i])

printAllPathsUtil(\*i, d, visited, path, path\_index);

}

// Remove current vertex from path[] and mark it as unvisited

path\_index--;

visited[u] = false;

}

class details

{

protected:

char name[30];

int age;

public:

void getdetl();

void ticket(ints,int d);

};

void details::getdetl()

{

cout<<"Enter your name :";

cin>>name;

cout<<"Enter your age:";

cin>>age;

}

void details::ticket(ints,int d)

{

cout<<"\a";

cout<<"\n\n\n\t\t\t====T.I.C.K.E.T====\n";

cout<<"|-------------------------------------------------------------------------------|\n";

cout<<"| NAME: ";

cout<<name<<setw(32-strlen(name))<<"|\tSOURCE: ";

cout<<cities[s]<<setw(33-cities[s].size())<<"|";

cout<<"\n|\t\t\t\t|\t\t\t\t\t\t|\n|\t\t\t\t|\t\t\t\t\t\t|\t\n| AGE: "<<age<<"\t\t\t|\tDESTINATION: "<<cities[d]<<"\t\t\t|\n";

cout<<"|\t\t\t\t|\t\t\t\t\t\t|";

cout<<"\n|-------------------------------------------------------------------------------|";

}

void cheapest()

{

cout<<"\n\n\t\tCHEAPEST PATH =";

for(inti=0;i<=siz;i++)

{

if(i==siz)

cout<<cities[arr[i]]<<" ";

else

cout<<cities[arr[i]]<<" -> ";

}

cout<<" = Rs "<<m;

}

// Driver program

intmain()

{

// Create a graph given in the above diagram

Graph g(6);

g.addEdge(0, 1);

g.addEdge(1, 0);

g.addEdge(1, 2);

g.addEdge(1, 3);

g.addEdge(1, 5);

g.addEdge(2, 0);

g.addEdge(2, 1);

g.addEdge(2, 3);

g.addEdge(2,5);

g.addEdge(3, 1);

g.addEdge(3, 2);

g.addEdge(3, 4);

g.addEdge(4, 3);

g.addEdge(5, 1);

g.addEdge(5, 3);

g.addEdge(5, 4);

details pass;

ints,d;

int sum=0;

intch;

cout<<"\t\tWELCOME TO CHEAP-AIR-TRAVELS\n\t\t============================\n\n";

cout<<"Please Enter your Details:\n";

pass.getdetl();

cout<<"\n\nThe Following are the cities to which Flights are available.\nKindly Select your SOURCE and DESTINATION cities.\n\n";

cout<<"1.)JAMMU\t\t4.)BANGLORE\n2.)DELHI\t\t5.)TRIVANDRUM\n3.)MUMBAI\t\t6.)KOLKATA\n\n";

cout<<"Enter the SOURCE:";

cin>>s;

s=s-1;

cout<<"Enter the DESTINATION:";

cin>>d;

d=d-1;

if(s<0 || s>6 ||s==d)

{

cout<<"Source NOT correct";

return 0;

}

else if(d<0 || d>6 || d==s)

{

cout<<"Destination NOT correct";

return 0;

}

cout<<"\n\n\n\tNOW WHAT TYPE OF TRIP WOULD YOU LIKE TO MAKE?\n\t---------------------------------------------\n";

cout<<"\t1.)SINGLE TRIP\t2.)ROUND TRIP\n"<<endl;

cin>>ch;

switch(ch)

{

case 1: g.printAllPaths(s, d);

cheapest();

break;

case 2:

cout<<"\n\t\tFlights from Source to Destination are:\n\t\t--------------------------------\n\n";

g.printAllPaths(s, d);

cout<<"\n\n\t\tCHEAPEST PATH FROM S TO D = ";

for(inti=0;i<=siz;i++)

{

if(i==siz)

cout<<cities[arr[i]]<<" ";

else

cout<<cities[arr[i]]<<" -> ";

}

sum+=m;

cout<<"= Rs "<<sum;

cout<<"\n\n\n\t\tAll flights from Destination to Source are:\n\t\t-------------------------------\n\n";

m=99999;

g.printAllPaths(d, s);

cout<<"\n\n\t\tCHEAPEST PATH FROM D TO S = ";

for(inti=0;i<=siz;i++)

{

if(i==siz)

cout<<cities[arr[i]]<<" ";

else

cout<<cities[arr[i]]<<" -> ";

}

cout<<" = Rs "<<m;

sum+=m;

cout<<"\n\t\t\n\n\t\t|======================|\n\t\tTOTAL COST OF TRIP="<<sum<<endl;

cout<<"\t\t|======================|"<<endl;

break;

default:g.printAllPaths(s, d);

cheapest();

break;

}

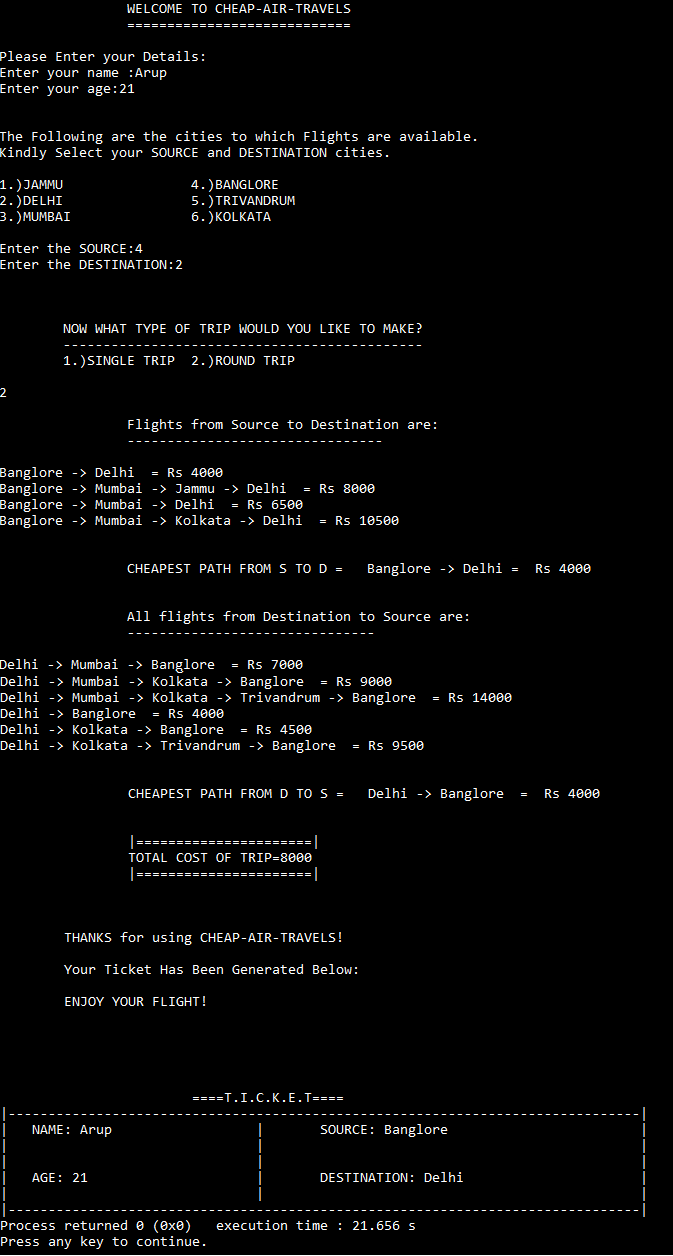
cout<<"\n\n\n\tTHANKS for using CHEAP-AIR-TRAVELS!\n\n\tYour Ticket Has Been Generated Below:\n\n\tENJOY YOUR FLIGHT!\n\n"<<endl;

pass.ticket(s,d);

return 0;

}

OUTPUT



Investigate and Analyze the of Efficiency of Algorithm

Input size: The number of vertices in the graph i.e. V

Basic operation: Basic operation of the algorithm is comparison

Efficiency: The efficiency of algorithm doesn’t depend upon the order of input

.It only depends upon the number of input.

Mathematically:

Cworst(n) = Cmarking + Cscanning

Since, vertex marked and explored exactly once so

Cmarking= O(|V|); here V is no of vertices in graph

And, scanning in case of adjacency list takes

Cscanning = O(|E|); where E is the no of edges hence

Total efficiency is

Cworst(n) = O(|V|) + O(|E|)

= O(|V|+|E|) for Adjacency list

How solution of your problem statement is useful to the society or its application in real-world scenario

With the help of this solution it would become really easy for the users in real world to

book flight tickets. The algorithm provides a way to easily find a connecting air route between the major cities in India.

Majorly what a user looks for while booking a flight is the cost, this algorithm focusses on that fact, it provides the user with the information of all the flights from his choice of source city to his choice of destination city along with the cost of each flight.

It serves as a ready reference for the user to look upon and decide. Moreover, with the option of cheapest path, the user can easily find the most cost efficient way of going from the source to the destination.

Along with that the option of round trip is also provided which helps in preplanning the trip to the city and with cheapest round trip, suitable cost efficient option is provided which gives an estimate of the total travelling expenditure beforehand.

Hence, this solution in real life practically helps the user in saving money on air travel and hence the name cheap air travel.

REFERENCES

* www.GeeksforGeeks.com
* www.nptel.com
* Anany Levitin: Introduction to The Design & Analysis of Algorithms. Ed2. Pearson Education. 2007.