VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Electronics Engineering (SENSE)

PROJECT BASED LEARNING (J Component) - REPORT

COURSE CODE / NAME	CSE2003 – Data Structures and Algorithms	
PROGRAM / YEAR	B. Tech (ECE)	
LAST DATE FOR REPORT SUBMISSION	April 23rd	
DATE OF SUBMISSION	23/04/2022	
TEAM MEMBERS DETAILS	**REGISTER NO.**	**NAME**
	19BEE1001	B CHAITHANYA KUMAR REDDY
	19BEE1027	P GOWTHAM
	19BEE1103	P V N S SAI SARAN
	19BEE1156	N BHUVAN
J TITLE	**FLIGHTS SEARCHING SYSTEM**	
COURSE HANDLER'S NAME	**Dr. JAYAVIGNESH.T**	**REMARKS**
COURSE HANDLER'S SIGN		

FLIGHTS SEARCHING SYSTEM

Team Members:

- B CHAITANYA KUMAR REDDY (19BEE1001)
- P GOWTHAM (19BEE1027)
- P V N S SAI SARAN (19BEE1103)
- N BHUVAN (19BEE1156)

ABSTRACT:

This project is basically a searching system to check whether there is a direct flight route or not between two cities. The database is created for the cities using an adjacency list and a matrix is created for distances between the cities to implement Floyd-Warshall algorithm. From this database, we are going to find whether there is any direct flight between the two cities which are taken as input from the user, which is case insensitive for a better interactive experience to the users. If any direct flight is available between two cities the program will display that there is a direct flight available between two cities. If there is no direct flight between the two cities the program prompts that there is no direct flight available between the two cities and then checks for the shortest path to the destination through a connected-flight from the boarding city. To design this program, techniques like Floyd-Warshall Algorithm, searching algorithms for linked-list and arrays and a Heuristic route search have been used to find the best possible shortest paths between the cities in the database. Both the algorithm and the greedy approach are used to find the shortest path and distance and they have been separated into two different programs so that we can compare the pros and cons of each approach and draw out a conclusion to use whichever is better for the current situation. The main purpose of this project is to make travelling convenient and easier as searching for a connected-flight through the shortest path might be troublesome for some passengers who travel frequently, or people who are new to travelling by flights and this also saves a lot of time for them. So, creating a program which can find direct or connected flights between two cities might be helpful for people who travel through flights often.

Keywords:

1) Adjacency List

2) Adjacency Matrix

3) Floyd - Warshall Algorithm

4) Heuristic Search Method

5) Traversal algorithms of Linked List and 2D - array.

Introduction to the problem:

Many people around us travel through flights and take air routes on a daily basis, they spend a lot of time choosing a path which would take a shorter travel time and distance from their source to the destination. For example, if a person wants to travel from Allahabad to goa and there is no direct flight available from Allahabad to Goa, one should decide an intermediate connecting city to which they can go from Allahabad and from that connecting city they can go to their destination. An application that would help them conserve their time and energy in searching for a path is necessary to overcome this problem.

The application will ask passenger to enter the city name from which the passenger wants to travel from and the destination to which passenger wants to go, after that the application will display whether a direct flight connection is available between two cities which are present in the database, if available it would prompt that there is a direct flight available and also give the total distance between the source and destination and if there is no direct connection between the two cities, the application is designed in such a way that it displays the shortest path showing the intermediate cities and also the distance of shortest path from boarding city to destination.

Details about the related work:

Based on business purpose, there are two types of flight search engines: independent and commercial. Independent flight search engines do not sell tickets. They merely perform the search, then redirect potential clients to the respected airline website to continue with the booking process. Commercial engines are usually owned by travel agents which have agreements with the airlines to allow clients to book directly on their website.

For our project we created a model of commercial flight search engine which allows clients to book directly on their website and also shows the shortest path and distance of shortest path if there is no direct flight connection.

Modules of our project:

Module 1:

This module contains the database of around 60 cities of India which have been selected from an online airlines map network. The database is created using an adjacency list (for Heuristic Search method) and adjacency matrix (for Floyd Warshall algorithm) which is a representation of graphs. The database has been divided into separate modules so that in future, if there are any changes in routes in flights the database can be updated separately without disturbing the whole program.

. Module 2:

This module contains taking the input from the user, basic design of output window like colour of the output screen, checking whether the given input is a valid city or whether both theinput are same or not, and display that there is a direct flight available, in case of adjacency list, by traversing through the adjacency list and in case of adjacency matrix, by using if condition created in database.

Basic design of output window, verifying input and checking for direct flight connection when adjacency list is used -

Taking input from user and checking for direct flight connection when adjacency matrix is used-

Module 3:

This module contains implementation of Floyd - Warshall Algorithm and greedy approach which is used to find out the shortest path and distance of shortest path between the cities if there is no direct flight.

Floyd- Warshall Algorithm

Heuristic Search Approach

Motivation on the project:

There are a lot of flight search engines currently available on the Internet. If one tries to search in Google using Keywords such as “flight booking”, he/she would get numerous online travel agents from where the flight query can be posted. Most of these search engines share the same characteristic, that is they behave as meta- search engines that simply forward the query to several other websites and aggregate the result from each website.

The first weakness is that the current flight search engines have not yet offered all available routes. They show that the flight is available only if there is a direct flight application. The problem is that not everyone has the patience to look for such a route. Aside from frequent flyers who may know about the existence of this route. Hence common people will never know the shortest path between the cities they entered if the direct flight connection does not exist.

It would be nice to have search engine which shows the shortest path if the direct flight connection does not exist for instance Imagine yourself being someone who travels a lot through flights or someone who is new to flights, you would wish to have something that can reduce your time to search the flights and its routes or help you take a shortest path to

destination if it involves an indirect path to the destination, i.e., through intermediate cities until the destination.

Many people face this problem and mostly rely on other people or check through all the available flights which takes a lot of time, so this problem motivated us to find an approach to make this process easier and responsive. Our application helps you in finding the shortest path to the destination by checking through various paths and gives you the shortest path and also displays the distance between the source and destination.

Proposed work:

Our idea is to create a database of 61 cities using both adjacency list and adjacencymatrix. To check for a direct flight connection in case of adjacency list we used a traversal algorithm of linked list and in case of adjacency list we used a simple if condition. If there is no direct flight connection, we used two different algorithms which are Floyd Warshall and greedy approach algorithms to find the shortest path and the distance of shortest path between the two cities. We used adjacency or distance matrix for Floyd Warshall algorithm because the algorithm requires it.

1st Program:

In this program, first we have created a Adjacency list using and array and creating a linked list from each element of the array, each node of the linked list contains two integers one is data, which store the city number (each city has been allotted a unique number) and another is weight which stores the distances between the parent city (current city in array from which linked list starts) and city which is denoted by its number in data. In this format we create the adjacency list of all the 61 cities which is our database containing all the information about the cities and the distances between cities. This database is stored as a separate function in the program.

Coming to the main function, first we take an array to store the city names in the same order as created in the database. Then we take the input from the user and convert into lower

case and verify it with city names array and then store the index number of source city and then we clear the output screen when a key is pressed and ask for the destination city input and the same process and continued to store the destination city index and then verify if both source and destination are same or not. If the city entered is not present in our 61 cities database or if both the city names are the same then we would ask them again to enter the input. As the inputs are checked by converting in lower case, taking input is case insensitive. Then we would create a pointer which would traverse through the adjacency list. From taking the source city as parent city and go to that index in the array and from then by using the pointer we check through the linked list and if it matches with second city we would display that there is a direct path (as only direct connections of parent city are stored in that particular linked list) and display the distance which is stored as weight in the node of the linked list.

On the other hand, when there is no direct path from the source to the destination, the program displays that there is no direct path and finds the indirect path. To find this indirect path Heuristic Route Search technique is used. What is a Heuristic Search? A Heuristic is a technique to solve a problem faster than classic methods, or to find an approximate solution when classic methods cannot. This is a kind of a shortcut as we often trade one of optimality, completeness, accuracy, or precision for speed. To find the indirect path we take two pointers and one pointer searches from source city which is taken as parent city and from that particular linked list for each next node another pointer points takes the city which is pointed by first pointer as parent city and from that linked list it points every next node checking if the name of city is equal to destination city. If its equal and if its distance is less than other paths then it prints out the total path and total distance between the cities, it also shows the intermediate city in the given path.

2nd Program:

The Floyd Warshall algorithm is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights. A single execution of the algorithm will find the lengths of shortest paths between all pairs of vertices. This algorithm requires a distance matrix (adjacency matrix) for its implementation. To print the shortest path the algorithm is implemented with a little variation.

First thing we do is, we take two 2D matrices. These are adjacency matrices. The size of the matrices is going to be the total number of vertices. For our graph, we will take 61*61 matrices. The distance matrix (adjacency matrix) is going to store the minimum distance found so far between two vertices. At first, for the edges, if there is an edge between u-v and the weight is w, we will store: d[u][v] =w. For all edges that don't exist, we are going to put infinity (considered as 10000 in the code). The other matrix is for the path which is called path matrix. The path matrix is for regenerating the minimum distance path between two vertices.

So initially, if there is a path between u and v, we are going to put path[u][v] =u. This means the best way to come to vertex-v from vertex-u is to use the edge that connects v with u. If there is no path between two vertices, we are going to put 100 there indicating there is no path available now.

To apply Floyd Warshall algorithm, we are going to select a middle vertex k. Then for each vertex i, we are going to check if we can go from i to k and then k to j, where j is another vertex and minimize the cost of going from i to j. If the current d[i][j] is greater than d[i][k] +d[k][j], we are going to put d[i][j] equals to the summation of those two distances. And the path[i][j] will be set to path[k][j], as it is better to go from i to k, and then k to j. All the vertices will be selected as k.

To print the path, we will check the path matrix. To print the path from u to v, we will start from path[u][v]. We will keep changing v=path[u][v] until we find path[u][v] = u and push every value of path[u][v] in a stack. After finding u, we will print u and start popping items from the stack and print them. This works because the path matrix stores the value of vertex which shares the shortest path to v from any other node.

Experimentation:

The main part of algorithms used to search the shortest distance and path are mentioned below:

Implementation of Heuristic Route Search Method:

```
// Heuristic Search Method
    struct graph *temp1,*temp2;
    temp1 = a[city1indx];
    int citymindx,mDist = 5000;
    int c = 0;
    while(temp1->next!=NULL){
       temp1 = temp1->next;
       temp2 = a[temp1->data];
       while(temp2->next != NULL){
          temp2 = temp2->next;
          if(temp2->data == city2indx && (temp2->weight + temp1->weight)<mDist){
             citymindx = temp1->data;
             mDist = (temp2->weight + temp1->weight);
             c = 1;
          }
       }
    }
```

Implementation of Floyd-Warshall Algorithm:

```
//Floyd – Warshall Algorithm
int i,j,k;
    for (int k = 0; k < 61; k++) {
       for (int i = 0; i < 61; i++) {
          for (int j = 0; j < 61; j++) {
             if(d[i][j] > d[i][k] + d[k][j])
             {
                d[i][j] = d[i][k] + d[k][j];
                path[i][j] = path[k][j];
             }
          }
       }
    }
```

Both the programs give the same output as both of them are designed to output the shortest path and distance between the cities. So the output given below is for both the algorithms.

SOURCE CODE FOR THE PROJECT -

```
#include<iostream>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<conio.h>
#include<vector>
#include<list>
using namespace std;
struct graph{
  int data;
  struct graph* next;
};

/*Adjacency list-of twenty cities-
0-srinagar->{delhi,leh}
1-leh->{srinagar,jammu,delhi}
2-jammu->{delhi,leh}
3- kullu->{delhi}
4- amirtsar->{delhi}
5- dharmshala->{delhi}
6-chandigarh->{delhi}
7-dehradun->{delhi,lucknow}
8-delhi-
>{srinagar,jammu,amritsar,chandigarh,kullu,dharmshala,dehradun,leh,guwahati,bagdogra,kolkata,patna,ranc
hi,bhubaneshwar,varanasi,lucknow,allahabad,agra,khajuraho,raipur,jabalpur}
9-jodhpur->{udaipur,delhi}
10-jaipur->{delhi,mumbai}
11- agra->{delhi,varanasi,mumbai,gwalior}
12- lucknow->{varanasi,delhi,dehradun,mumbai}
13-bagdogra->{guwahati,kolkata,delhi}
14-udaipur->{jodhpur,delhi,mumbai}
15-gwalior->{agra,mumbai}
16-allahabad->{delhi,mumbai}
17-patna->{delhi}
18- bhuj->{mumbai}
19- ahmedabad->{mumbai,delhi,hyderabad}
20-indore->{mumbai,delhi}
21- bhopal->{mumbai,delhi}
```

22- khajuraho->{varanasi,delhi}
23- varanasi->{lucknow,khajuraho,agra,delhi,mumbai}
24-jamnagar->{mumbai}
25-vadodara->{delhi}
26-ranchi->{delhi}
27- kolkata->{agartala,aizawl,silchar,bagdogra,dimapur,dibrugarh,hyderabad,delhi,port blair,chennai,mumbai,bengaluru}
28- surat->{delhi}
29-mumbai->{pune,aurangabad,goa,ahmedabad,bhuj,jamnagar,hyderabad,indore,bhopal,nagpur,raipur,udaipur,bengaluru,mangalore,visakhapatnam,allahabad,varanasi,bhubhaneshwar,lucknow,jaipur,agra,delhi,kolkata,gwalior,kochi,thiruvananthapuram,kozhikode,coimbatore}
30-pune->{mumbai,goa,hyderabad,bengaluru,delhi}
31-aurangabad->{mumbai,delhi}
32-nagpur->{raipur,hyderabad,mumbai,delhi}
33-raipur->{nagpur,mumbai,delhi}
34- jabalpur->{delhi}
35- bhubaneshwar->{visakhapatnam,port blair,chennai,mumbai,delhi}
36-hyderabad->{vijayawada,visakhapatnam,pune,goa,bengaluru,chennai,nagpur,mumbai,tirupur,ahmedabad}
37-goa->{pune,mumbai,hyderabad,kozhikode,kochi}
38- ijayawada->{hyderabad}
39- isakhapatnam->{bhubaneshwar,hyderabad,chennai,mumbai,delhi}
40-tirupar->{hyderabad}
41- mangalore->{bengaluru,mumbai,kozhikode}
42- bengaluru->{mangalore,chennai,tiruchirapalli,kochi,hyderabad,pune,mumbai,kolkata,delhi}
43-chennai->{bengaluru,tiruchirapalli,hyderabad,kochi,madurai,kozhikode,visakhapatnam,bhubaneshwar,port blair,kolkata,delhi}
44-kozhikode->{coimbatore,kochi,mangalore,thiruvananthapuram,goa,mumbai}
45-coimbatore->{kozhikode,mumbai}
46- tiruchirapalli->{chennai,bengaluru}
47- kochi->{agatti,thiruvananthapuram,kozhikode,goa,bengaluru,chennai,mumbai,delhi}
48-agatti->{kochi}
49- madurai->{chennai}
50- thiruvananthapuram->{kochi,kozhikode,mumbai}
51-port blair->{chennai,bhubaneshwar,kolkata}
52-guwahati->{bagdogra,libabari,delhi,imphal}
53-libabari->{guwahati}
54- dibrugarh->{dimapur,kolkata}

```
55- tezpur->{silchar}
56- silchar->{tezpur,kolkata}
57- dimapur->{dibrugarh,kolkata}
58-imphal->{aizawl,guwahati}
59-agartala->{kolkata}
60-aizawl->{imphal,kolkata}
*/
int main()
{
   struct graph *a[61],*t1,*t2;
   char
cityname[61][20]={"srinagar","leh","jammu","kullu","amritsar","dharmshala","chandigarh","dehradun","del
hi","jodhpur","jaipur","agra",

"lucknow","bagdogra","udaipur","gwalior","allahabad","patna","bhuj","ahmedabad","indore","bhopal","khaj
uraho","varanasi",

"jamnagar","vadodara","ranchi","kolkata","surat","mumbai","pune","aurangabad","nagpur","raipur","jabalpu
r","bhubaneshwar",

"hyderabad","goa","vijayawada","visakhapatnam","tirupar","mangalore","bengaluru","chennai","kozhikode",
"coimbatore",
                    "tiruchirapalli","kochi","agatti","madurai","thiruvananthapuram","port
blair","guwahati","lilabari","dibrugarh",
                    "tezpur","silchar","dimapur","imphal","agartala","aizawl"};

  // linked list of city 0-srinagar-

   t1=(struct graph*)malloc(sizeof(struct graph));
   a[0]=t1;
   t1->data=0;
   t2=(struct graph*)malloc(sizeof(struct graph));
   t1->next=t2;
   t2->data=1;
   t1=(struct graph*)malloc(sizeof(struct graph));
   t2->next=t1;
   t1->data=8;
   t1->next=NULL;

   // linked list for 1-leh
```

```
t1=(struct graph*)malloc(sizeof(struct graph));
a[1]=t1;
t1->data=1;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=0;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=2;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t2->next=NULL;

 // linked list for 2-jammu

t1=(struct graph*)malloc(sizeof(struct graph));
a[2]=t1;
t1->data=2;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=1;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=8;
t1->next=NULL;

//  linked list for 3-kullu

t1=(struct graph*)malloc(sizeof(struct graph));
a[3]=t1;
t1->data=3;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t2->next=NULL;

//  linked list for city 4-amritsar
```

```
t1=(struct graph*)malloc(sizeof(struct graph));
a[4]=t1;
t1->data=4;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t2->next=NULL;

//  linked list for city 5-dharmshala

t1=(struct graph*)malloc(sizeof(struct graph));
a[5]=t1;
t1->data=5;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t2->next=NULL;

// linked list for city 6-chandigarh

t1=(struct graph*)malloc(sizeof(struct graph));
a[6]=t1;
t1->data=6;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t2->next=NULL;

//  linked list for the city 7-dehradun

t1=(struct graph*)malloc(sizeof(struct graph));
a[7]=t1;
t1->data=7;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=12;
t1->next=NULL;
```

```
//  linked list for the city 8-delhi

t1=(struct graph*)malloc(sizeof(struct graph));
a[8]=t1;
t1->data=8;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=11;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=7;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=10;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=6;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=15;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=5;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=4;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=3;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=1;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=0;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=12;
t1=(struct graph*)malloc(sizeof(struct graph));
```

```
t2->next=t1;
t1->data=9;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=14;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=16;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=22;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=23;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=17;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=19;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=20;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=21;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=13;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=25;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=26;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=28;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
```

```
t2->data=52;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=27;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=32;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=33;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=31;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=35;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=34;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=29;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=30;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=36;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=39;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=38;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=43;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=42;
```

```
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=47;
t2->next=NULL;

// linked list for the city 9-jodhpur

t1=(struct graph*)malloc(sizeof(struct graph));
a[9]=t1;
t1->data=9;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=14;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=8;
t1->next=NULL;

//  linked list for the city 10-jaipur

t1=(struct graph*)malloc(sizeof(struct graph));
a[10]=t1;
t1->data=10;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=29;
t1->next=NULL;

// linked list for the city 11-Agra

t1=(struct graph*)malloc(sizeof(struct graph));
a[11]=t1;
t1->data=11;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t1=(struct graph*)malloc(sizeof(struct graph));
```

```
t2->next=t1;
t1->data=15;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=23;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=29;
t1->next=NULL;

//  linked list for the city 12-lucknow

t1=(struct graph*)malloc(sizeof(struct graph));
a[12]=t1;
t1->data=12;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=23;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=8;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=7;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=29;
t1->next=NULL;

//  linked list for the city 13-bagdogra

t1=(struct graph*)malloc(sizeof(struct graph));
a[13]=t1;
t1->data=13;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=52;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=27;
```

```
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t2->next=NULL;

//  linked list for the city 14-udaipur

t1=(struct graph*)malloc(sizeof(struct graph));
a[14]=t1;
t1->data=14;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=9;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=8;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=29;
t2->next=NULL;

//  linked list for the city 15-gwalior

t1=(struct graph*)malloc(sizeof(struct graph));
a[15]=t1;
t1->data=15;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=11;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=29;
t1->next=NULL;

//  linked list for the city 16-allahabad

t1=(struct graph*)malloc(sizeof(struct graph));
a[16]=t1;
t1->data=16;
t2=(struct graph*)malloc(sizeof(struct graph));
```

```
t1->next=t2;
t2->data=8;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=29;
t1->next=NULL;

//  linked list for the city 17-patna

t1=(struct graph*)malloc(sizeof(struct graph));
a[17]=t1;
t1->data=17;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t2->next=NULL;

//  linked list for the city 18-bhuj

t1=(struct graph*)malloc(sizeof(struct graph));
a[18]=t1;
t1->data=18;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=29;
t2->next=NULL;

//  linked list for the city 19-ahmedabad

t1=(struct graph*)malloc(sizeof(struct graph));
a[19]=t1;
t1->data=19;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=29;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=8;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
```

```
t2->data=36;
t2->next=NULL;

//  linked list for the city 20-indore

t1=(struct graph*)malloc(sizeof(struct graph));
a[20]=t1;
t1->data=20;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=29;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=8;
t1->next=NULL;

//  linked list for the city 21-bhopal

t1=(struct graph*)malloc(sizeof(struct graph));
a[21]=t1;
t1->data=21;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=29;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=8;
t1->next=NULL;

// linked list for the city 22-khajuraho

t1=(struct graph*)malloc(sizeof(struct graph));
a[22]=t1;
t1->data=22;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=23;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=8;
```

```
t1->next=NULL;

//  linked list for the city 23-varanasi

t1=(struct graph*)malloc(sizeof(struct graph));
a[23]=t1;
t1->data=23;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=12;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=22;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=11;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=8;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=29;
t2->next=NULL;

// linked list for the city 24-jamnagar

t1=(struct graph*)malloc(sizeof(struct graph));
a[24]=t1;
t1->data=24;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=29;
t2->next=NULL;

// linked list for the city 25-vadodara

t1=(struct graph*)malloc(sizeof(struct graph));
a[25]=t1;
t1->data=25;
t2=(struct graph*)malloc(sizeof(struct graph));
```

```
t1->next=t2;
t2->data=8;
t2->next=NULL;

// linked list for the city 26-ranchi

t1=(struct graph*)malloc(sizeof(struct graph));
a[26]=t1;
t1->data=26;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t2->next=NULL;

// linked list for the city 27-kolkata

t1=(struct graph*)malloc(sizeof(struct graph));
a[27]=t1;
t1->data=27;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=59;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=60;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=56;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=13;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=57;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=54;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=36;
```

```
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=8;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=51;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=43;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=29;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=42;
 t1->next=NULL;

// linked list for the city 28-surat

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[28]=t1;
 t1->data=28;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=8;
 t2->next=NULL;

// linked list for the city 29-mumbai

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[29]=t1;
 t1->data=29;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=30;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=31;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
```

```
t2->data=37;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=19;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=18;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=24;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=36;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=20;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=21;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=32;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=33;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=14;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=42;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=41;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=44;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=39;
```

```
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=45;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=47;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=50;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=16;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=23;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=35;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=12;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=15;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=10;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=11;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=27;
t1->next=NULL;

// linked list for the city 30-pune
```

```
 t1=(struct graph*)malloc(sizeof(struct graph));
 a[30]=t1;
 t1->data=30;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=29;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=37;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=36;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=42;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=8;
 t2->next=NULL;

// linked list for the city 31-aurangabad

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[31]=t1;
 t1->data=31;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=29;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=8;
 t1->next=NULL;

// linked list for the city 32-nagpur

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[32]=t1;
 t1->data=32;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
```

```
 t2->data=33;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=36;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=29;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=8;
 t1->next=NULL;


// linked list for the city 33-raipur

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[33]=t1;
 t1->data=33;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=32;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=29;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=8;
 t2->next=NULL;

// linked list for the city 34-jabalpur

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[34]=t1;
 t1->data=34;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=8;
 t2->next=NULL;

// linked list for the city 35-bhubaneshwar
```

```
t1=(struct graph*)malloc(sizeof(struct graph));
a[35]=t1;
t1->data=35;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=39;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=51;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=43;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=29;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t2->next=NULL;


//  linked list for the city 36-hyderabad

t1=(struct graph*)malloc(sizeof(struct graph));
a[36]=t1;
t1->data=36;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=38;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=39;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=30;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=37;
t2=(struct graph*)malloc(sizeof(struct graph));
```

```
t1->next=t2;
t2->data=42;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=40;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=43;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=32;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=29;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=19;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=27;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=8;
t1->next=NULL;


//  linked list for the city 37-goa

t1=(struct graph*)malloc(sizeof(struct graph));
a[37]=t1;
t1->data=37;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=30;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=29;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=36;
```

```
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=44;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=47;
 t2->next=NULL;

//  linked list for the city 38-vijayawada

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[38]=t1;
 t1->data=38;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=36;
 t2->next=NULL;

// linked list for the city 39-visakhapatnam

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[39]=t1;
 t1->data=39;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=35;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=36;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=43;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=29;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=8;
 t1->next=NULL;
```

```
// linked list for the city 40-tirupar

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[40]=t1;
 t1->data=40;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=36;
 t2->next=NULL;

// linked list for the city 41-mangalore

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[41]=t1;
 t1->data=41;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=42;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=44;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=29;
 t2->next=NULL;

// linked list for the city 42-bengaluru

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[42]=t1;
 t1->data=42;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=41;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=43;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=46;
```

```
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=47;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=36;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=30;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=29;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=27;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t2->next=NULL;

//  linked list for the city 43-chennai

t1=(struct graph*)malloc(sizeof(struct graph));
a[43]=t1;
t1->data=43;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=42;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=46;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=36;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=47;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=49;
```

```
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=44;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=39;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=35;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=51;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=27;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=8;
t2->next=NULL;

//  linked list for the city 44-kozhikode

t1=(struct graph*)malloc(sizeof(struct graph));
a[44]=t1;
t1->data=44;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=45;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=47;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=41;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=50;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=37;
```

```
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=29;
 t1->next=NULL;

//  linked list for the city 45-coimbatore

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[45]=t1;
 t1->data=45;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=44;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=29;
 t1->next=NULL;


// linked list for the city 46-tiruchirapalli

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[46]=t1;
 t1->data=46;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=43;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=42;
 t1->next=NULL;

 // linked list for the city 47-kochi

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[47]=t1;
 t1->data=47;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=48;
```

```
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=50;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=44;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=37;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=42;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=43;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=29;
 t1=(struct graph*)malloc(sizeof(struct graph));
 t2->next=t1;
 t1->data=8;
 t1->next=NULL;

// linked list for the city 48-agatti

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[48]=t1;
 t1->data=48;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
 t2->data=47;
 t2->next=NULL;

// linked list for the city 49-madurai

 t1=(struct graph*)malloc(sizeof(struct graph));
 a[49]=t1;
 t1->data=49;
 t2=(struct graph*)malloc(sizeof(struct graph));
 t1->next=t2;
```

```
t2->data=43;
t2->next=NULL;

//linked list for the city 50-thiruvananthapuram

t1=(struct graph*)malloc(sizeof(struct graph));
a[50]=t1;
t1->data=50;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=47;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=44;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=29;
t2->next=NULL;

//linked list for the city 51-port blair

t1=(struct graph*)malloc(sizeof(struct graph));
a[51]=t1;
t1->data=51;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=43;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=35;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=27;
t2->next=NULL;

//linked list for the city 52-guwahati

t1=(struct graph*)malloc(sizeof(struct graph));
a[52]=t1;
t1->data=52;
```

```
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=13;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=53;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=58;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=8;
t1->next=NULL;

//linked list for the city 53-lilabari

t1=(struct graph*)malloc(sizeof(struct graph));
a[53]=t1;
t1->data=53;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=52;
t2->next=NULL;

//linked list for the city 54-dibrugarh

t1=(struct graph*)malloc(sizeof(struct graph));
a[54]=t1;
t1->data=54;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=57;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=27;
t1->next=NULL;



//linked list for the city 55-tezpur
```

```
t1=(struct graph*)malloc(sizeof(struct graph));
a[55]=t1;
t1->data=55;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=56;
t2->next=NULL;

//linked list for the city 56-silchar

t1=(struct graph*)malloc(sizeof(struct graph));
a[56]=t1;
t1->data=56;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=55;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=27;
t1->next=NULL;

//linked list for the city 57-dimapur

t1=(struct graph*)malloc(sizeof(struct graph));
a[57]=t1;
t1->data=57;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=54;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=27;
t1->next=NULL;

//linked list for the city 58-imphal

t1=(struct graph*)malloc(sizeof(struct graph));
a[58]=t1;
t1->data=58;
t2=(struct graph*)malloc(sizeof(struct graph));
```

```
t1->next=t2;
t2->data=60;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=52;
t1->next=NULL;

// linked list for the city 59-agartala

t1=(struct graph*)malloc(sizeof(struct graph));
a[59]=t1;
t1->data=59;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=27;
t2->next=NULL;

// linked list for the city 60-aizawl

t1=(struct graph*)malloc(sizeof(struct graph));
a[60]=t1;
t1->data=60;
t2=(struct graph*)malloc(sizeof(struct graph));
t1->next=t2;
t2->data=58;
t1=(struct graph*)malloc(sizeof(struct graph));
t2->next=t1;
t1->data=27;
t1->next=NULL;

//Data Base Done

//Designing the first window of the application.

system("color 80");
printf("\n\t\t**************************************************");
printf("\n\t\t*                                                *");
printf("\n\t\t*                                                *");
printf("\n\t\t*                                                *");
printf("\n\t\t*     WELCOME TO THE FLIGHT SEARCH SYSTEM        *");
```

```
printf("\n\t\t*                                        *");
printf("\n\t\t*                                        *");
printf("\n\t\t*    It is a place where you search for          *");
printf("\n\t\t*    direct flight between the two cities        *");
printf("\n\t\t*    of India and also it provides the           *");
printf("\n\t\t*    shortest route that user can opted in       *");
printf("\n\t\t*    case no direct flight is available.         *");
printf("\n\t\t*                                        *");
printf("\n\t\t*                                        *");
printf("\n\t\t*                                        *");
printf("\n\t\t*                                        *");
printf("\n\t\t*    Hope you will enjoy this application.       *");
printf("\n\t\t**************************************************");
printf("\n\n\t\tpress any to continue. ....");
getch();
system("cls");

//Reading the 1st-city name from the user...

char cityname1[15],cityname2[15];
int city1indx,city2indx;
while(1)
{
   printf("\n\n\n\tEnter the city name(starting point)...");
   scanf("%s",strlwr(cityname1));
   int c=0;
   for(int i=0;i<=60;i++)
   {
      if(strcmp(strlwr(cityname1),cityname[i])==0)
      {
         c=1;
         city1indx=i;
         break;
      }
   }
   printf("\n\n\n\tMessage:");
   if(c!=1)
   {
      printf("\n\n\t\tSorry such city doesnt exist please re-enter the cityname.");
      printf("\n\n\t\tpress any key to continue...");
```

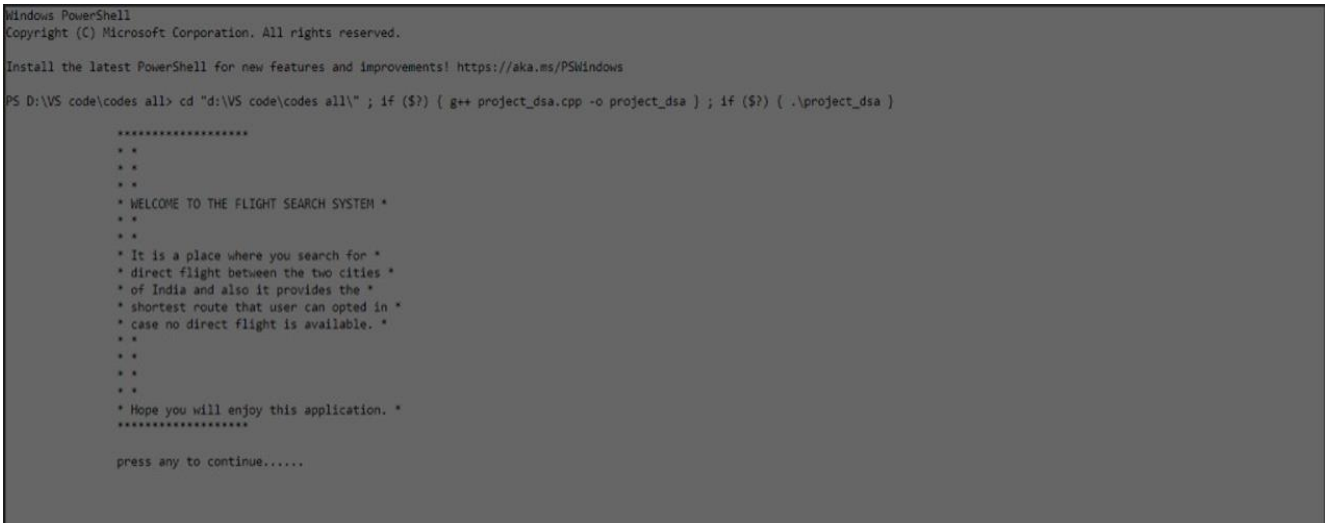
```
        getch();
        system("cls");
    }
    else
    {
        printf("\n\n\t\tcity name accepted.");
        printf("\n\n\t\tpress any key to continue...");
        getch();
        system("cls");
        break;
    }
}
int c;

//reading the 2-city name from the user...

while(1)
{
    printf("\n\n\n\tEnter the city name(destination point)...");
   scanf("%s",strlwr(cityname2));
   if(strcmp(strlwr(cityname1),strlwr(cityname2))==0)
    {
    printf("\n\n\tMessage:");
    printf("\n\n\t\tTwo city names cannot be same.");
    printf("\n\t\tplease re-enter the destination.");
    printf("\n\n\t\tpress any key to continue...");
    getch();
    system("cls");
    continue;
    }
    for(int i=0;i<=60;i++)
    {
        if(strcmp(strlwr(cityname2),cityname[i])==0)
        {
            city2indx=i;
            c=1;
            break;
        }
    }
    if(c==1)
```

```
    {
        printf("\n\n\tMessage:");
        printf("\n\t\tcity name is accepted.");
        printf("\n\n\t\tpress any key to continue...");
        getch();
        system("cls");
        break;
    }
    else
    {
        printf("\n\n\tMessage:");
        printf("\n\t\tSorry such city doesnt exist please re-enter the cityname.");
        printf("\n\t\tpress any key to continue...");
        getch();
        system("cls");

    }
}
printf("\n\n\tStarting Point-%s\t\tDestination Point-%s",cityname1,cityname2);
printf("\n\n\n\n\tMessage:\n\n");
c=0;
struct graph *t;
t=a[city1indx];
while(1)
{
    t=t->next;
    if(t->data==city2indx)
    {
        c=1;
        break;
    }
    if(t->next==NULL)
    {
        break;
    }
}
if(c==1)
{
    printf("\tYes the direct flight is available between the two mentioned cities.");
}
```

```
else
{
   printf("\tSorry no direct flight is available.\n\n");
   printf("\tPress any key to know the shortest path between the cities...");
   getch();
   system("cls");

   // find the shortest path.


   list<int> queue;
   int pred[61];
   bool visited[61];
   for(int i=0;i<61;i++)
   {
      visited[i]=false;
      pred[i]=-1;
   }
   visited[city1indx]=true;
   queue.push_back(city1indx);
   while(!queue.empty())
   {
      int u=queue.front();
      t=a[u];
      queue.pop_front();
      while(1)
      {
         if(visited[t->data]==false)
         {
            visited[t->data]=true;
            pred[t->data]=u;
            queue.push_back(t->data);
            if(t->data==city2indx)
            {
               break;
            }


         }
         if(t->next==NULL)
            break;
         t=t->next;
```

```
}
            if(t->data==city2indx)
            {
               break;
            }

         }
         vector<int> path;
         int crawl = city2indx;
         path.push_back(crawl);
         while(pred[crawl]!= -1)
         {
            path.push_back(pred[crawl]);
            crawl = pred[crawl];
         }
         printf("\n\n\n\tMessage:\n\n");
         printf("\n\tThe Shortest Path the User can take to reach the destination:\n\n\t");
         for(int i = path.size()-1; i >= 0; i--)
         {
            string s=cityname[path[i]];
            cout<<s;
            if(i==0)
               break;
            cout<<" --> ";
         }
      }
      printf("\n\n\n\n\n\n\tHope you have enjoyed this application..");
      printf("\n\n\tpress any key to exit.");
      exit(0);
      return 0;
   }
```

The welcome page of the programs is as follows:



After pressing a key, it clears the output window and asks for the input as shown below:

Then taking input for second part is shown below:

If the input is not proper then it asks to re - enter the output as shown below:

If both the cities are same it asks to re-enter the output as shown below:

Then it checks for direct flight and if available it outputs as:

If direct flight is unavailable, then it prompts that there is no direct path available:

Then it searches for the alternate path and displays the shortest route and distance from the source to the destination as shows as below:

By using Heuristic Search Method:

By using Floyd-Warshall method:

Conclusion:

We have taken two different approaches to solve the same shortest path problem because each problem has its own advantage and disadvantage:

The first approach Heuristic search is a lot faster than Floyd-Warshall algorithm as its time complexity is O(n^2) compared to Floyd-Warshall algorithm's complexity which is O(n^3), but the main drawback of the heuristic search method is that it cannot give accurate result. In this case, it cannot give a result if there is more than one connecting city.

On the other hand, the Floyd-Warshall algorithm gives accurate results for any number of connecting cities but it is comparatively slower than Heuristic route search. Generally, if the database is in a way that there is only one connecting city which is mostly common then the first method can be used as it is faster, else if there is more than 1 connecting city the second method is used. In our database, almost all have only one connecting city but in rare cases there are two connecting cities, so both the methods have been developed to overcome this problem.

References:

The project is done by referring various online materials and classroom materials, some of the online materials referred are linked below:

https://www.geeksforgeeks.org/graph-and-its-representations/

https://www.geeksforgeeks.org/pointer-array-array-pointer/

https://codeforwin.org/2015/09/c-program-to-create-and-traverse-singly-linked-list.html

https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/

https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/

https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/

https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

https://www.programiz.com/dsa/floyd-warshall-algorithm

https://en.wikipedia.org/wiki/Shortest_path_problem

https://www.geeksforgeeks.org/dynamic-programming/