Artificial Intelligence Project Report 2021

Course Instructor – Dr. Poulami Dalapati

Problem 10 - Find the optimal way to get from one place in Jaipur to another place, considering uncertain travel times due to traffic and road condition.

Team members –

Samarth Agarwal 19UCS063

Animesh Jain 19UCC079

Yash Kedia 19UCS256

Harsh Jain 19UCC092

Atharv Goyal 19UCS109

Problem approach -

We are considering **Jaipur to be 20\*20 matrix** and marked its various co-ordinates as high traffic areas of Jaipur and user can choose from the location list the source and destination he wants to visit and our algorithm will try to find the path to the destination if there exists (there could be no path as some roads are blocked due to construction) and the path will would be such the user encounters minimum traffic (not in all cases but it tends to minimize it). We are using

**A\* Search algorithm** to find the path and have optimized so that it guides us to the **nearest lowest traffic path.**

Class Car which holds all the variable and member functions of a car,

like **Source, Destination, and Path followed by the Car**.

class Car{

public:

    pair<int,int>cur\_pos;//stores current position of car

    pair<int,int>dest;//stores destination of each car

    stack<Pair>path;//stores the path followed by each car

    void pos(pair<int,int>&pos)//Randomly calculating source and destination of cars

    {

        c++;

        srand(c);

        int a=rand()%20;

        int b=rand()%20;

        pos.first=a;

        pos.second=b;

        return;

    }

};

Here **pos() function** is calculating random source and destination of each car we are using to generate a traffic before taking input from the user.

int traffic[ROW][COL];//matrix to store traffic at each instant of time,like live traffic

void initializeTraffic() //Initializing Traffic matrix as 0 on all roads

{

    for(int i=0;i<ROW;i++)

    {

        for(int j=0;j<COL;j++)

            traffic[i][j]=0;

    }

}

We are declaring the traffic array and initializing traffic as **0** at all points.

In **main()** function –

/\* Description of the Grid-

    1--> The road is not blocked

    0--> The road is blocked \*/

    int grid[ROW][COL];

    for(int i=0;i<ROW;i++)

    {

        for(int j=0;j<COL;j++)

        {

            grid[i][j]=fillGrid();

        }

    }

We are initializing the **map of Jaipur with 0 and 1** where

**0 denotes the road is blocked** and **1 denotes the road is unblocked.**

unordered\_map<pair<int, int>, string, hash\_pair>m;//map to store co-ordinated of different places of Jaipur on the grid

This map stores various places of Jaipur and its coordinate on the grid.

We have used a Hash function **hash\_pair** which hashes the name of the place to its coordinate.

vector<Car>T;//Stores each car that is on road

    priority\_queue<ppPair,vector<ppPair>,TrafficCompare>openList;

    for(int i=0;i<500;i++)//Initializing traffic of Jaipur with 500 Cars

    {

        Car a;

        a.pos(a.cur\_pos);

        a.pos(a.dest);

        grid[a.cur\_pos.first][a.cur\_pos.second]=1;

        grid[a.dest.first][a.dest.second]=1;

        int ans=aStarSearch(grid, a.cur\_pos,a.dest,a,openList);

        traffic[a.cur\_pos.first][a.cur\_pos.second]+=1;

        UpdateTraffic(T,i);//it is updating traffic of all the previous cars which have already started

        T.push\_back(a);

    }

Here we are initializing the traffic by moving **500** Cars , We assume each car is starting its journey after a delay of unit time and each car which has already started its journey moves to the next position as each unit time passes by.

After each unit time we are updating the traffic of each previous car by removing the traffic on its last position and moving the traffic onto its next position.

The next Car calculates its optimal path on the basis of current traffic matrix at that instant of time.

//Function to update traffic and car position of every previous car that has started its journey as each unit time passes by.

void UpdateTraffic(vector<Car>&T,int i)

{

    int j=0;

    while(j!=i)

    {

        updatePosition(T[j]);//call to the function that updates cur\_pos and traffic matrix

        j++;

    }

    return;

}

The main() function passes the Car vector T and this function traverses each car to update its position and traffic onto the next.

 void updatePosition(Car&a)//Updating position of each car as time passes by each unit time

    {

        if(a.cur\_pos.first==a.dest.first && a.cur\_pos.second==a.dest.second)

        {

            return;//if car is at destination no need to move it

        }

        traffic[a.cur\_pos.first][a.cur\_pos.second]>0?traffic[a.cur\_pos.first][a.cur\_pos.second]-=1:0;//decreasing traffic as now the car is moving ahead

        if(a.path.empty())

            return;

        a.cur\_pos=a.path.top();

        a.path.pop();

        traffic[a.cur\_pos.first][a.cur\_pos.second]+=1;//increasing traffic at new position

        return;

    }

This function updates the position of the car onto its next position and also decreases the traffic on its last position and increases the traffic on its current position.

We are **using A\* search algorithm** –

int aStarSearch(int grid[][COL], Pair src, Pair dest,Car&a,priority\_queue<ppPair,vector<ppPair>,TrafficCompare>openList)

Here grid denotes the roads of Jaipur ,src is the source for the car,dest is the destination.

Car a holds all the information of the car we need to move from src to dest.

In **A\* search algorithm**, **g(x)** is the distance from **one point to another(in our case its 1)** + **the traffic at that coordinate**. h(x) is a function which is already computed for each of the nodes and hence this value is directly used. Now in A\* we do not visit all the nodes. So, we start from the source node, and then extract all the neighbors, then for each neighbor add its g value (distance from parent to this neighbor) to the g value of the parent, then add its h value to this, giving you the f value (f = g + h). Then add this all these neighbors to a priority queue with f values. These nodes are now OPEN to calculate and the source node is CLOSED to calculate. (Initially the source node is OPEN and CLOSED list is empty).

**THE CAR CAN MOVE IN ALL THE 8 DIRECTIONS namely EAST,WEST,SOUTH,NORTH,**

**NORTH EAST, NORTH WEST, SOUTH EAST, SOUTH WEST.**

openList.push(make\_pair(make\_pair(0.0,euclidean\_dist(src.first,src.second,dest.first,dest.second)), make\_pair(i, j)));

Here **openList** stores the f value,the Euclidean distance of that point and the coordinate of the point and the destination its headed to.

//closedList denotes the visited array.

    bool closedList[ROW][COL];

    memset(closedList, false, sizeof(closedList));

It denotes all the coordinates we have already visited.

/Compare function for elements to enter the priority queue according to non decreasing traffic order

struct TrafficCompare{

bool operator()(ppPair a,ppPair b)

{

    int x=traffic[a.second.first][a.second.second];

    int y=traffic[b.second.first][b.second.second];

    if(x>y)

        return true;

    else if(x<y)

        return false;

    //if traffic is same at two positions then it is sorted according to distance from destination

    return a.first.second>b.first.second;

}

};

This function helps us to insert neighbours of each coordinate according to their increasing traffic and if the traffic is same it is inserted according to its distance from the destination.

// A function to store the path of each Car

void tracePath(cell cellDetails[][COL], Pair dest,Car&a)

{

    int row = dest.first;

    int col = dest.second;

    stack<Pair> Path;

    while (!(cellDetails[row][col].parent\_i == row

            && cellDetails[row][col].parent\_j == col)) {

        a.path.push(make\_pair(row, col));

        int temp\_row = cellDetails[row][col].parent\_i;

        int temp\_col = cellDetails[row][col].parent\_j;

        row = temp\_row;

        col = temp\_col;

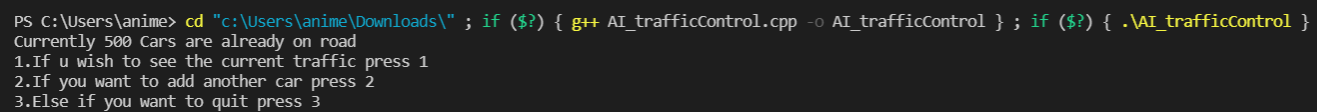
    }

    return;

}

This function calculates the path of each car after it has found its optimal path to destination and stores it in the path stack of its Car class.

During Execution of CODE



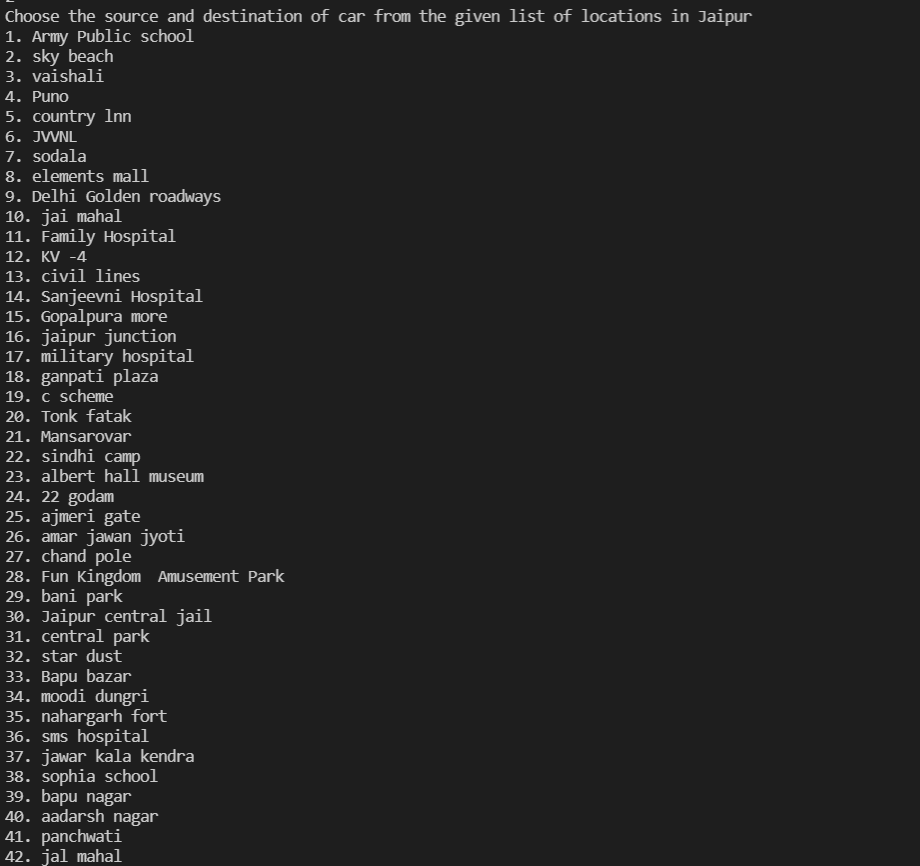
Now, we have 3 options to choose from.

1)By pressing 1,we get the cureent traffic matrix.

2)By pressing 2,we can add another car on the roads of Jaipur.

3)Exit command.

Suppose we want to add a new Car, Whose source is from **Jal mahal** to **JVVNL**

****



Here **42 represents Jal mahal and 6 represents JVVNL** as chosen by the user.

Now,our Algorithm checks whether there exists an optimal path between the source and destination chosen by the user, if there exists a path then the program outputs as follows……



Now , if you want to view the path that the Car has followed,Press 1.

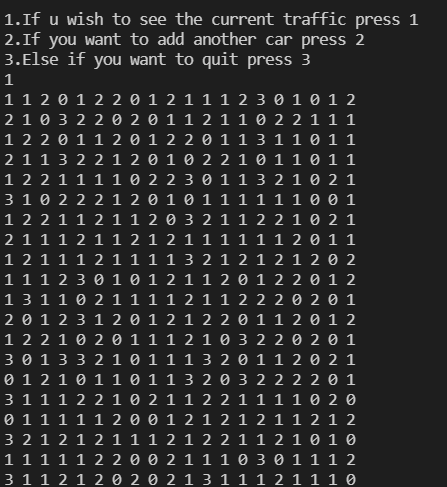
Now, the complete route will be printed.



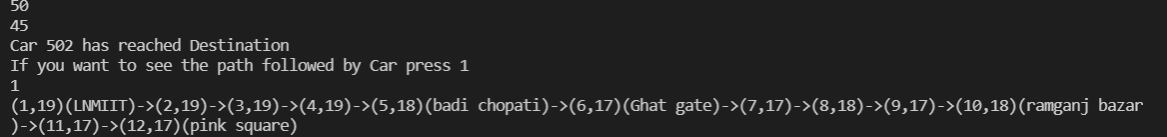
Now, if the user wants to exit He/She can Press 3.

If He/She wants to see the current traffic after entering his/her car press 1.

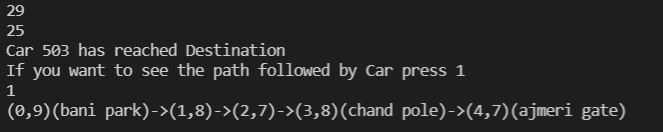
Now **LIVE traffic** is as follows—



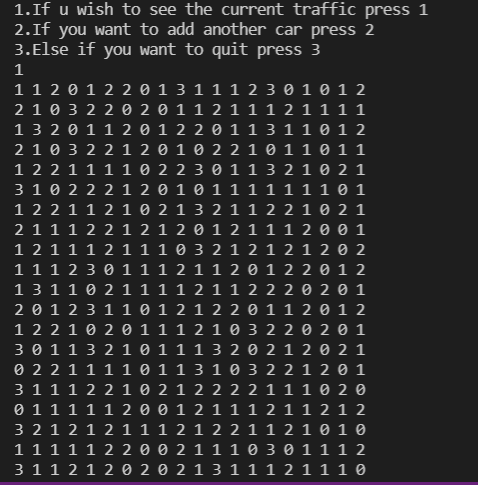
Suppose we want to add more Cars, and want to travel **from LNMIIT to pink square**

****

Now, suppose we want to move one more Car from **Bani park to Ajmeri gate**…….



After Adding 3 Cars we have now total 503 Cars running on the grid, Now let us choose to **see the LIVE traffic status**->



SO, as a **Conclusion** we can see that as more and more cars are added to the grid the traffic grid also keeps on updating timely and thus the **A\* search algorithm** we are using helps us to find the most optimal way to get to the desired destination.

Thank You!!