18CSC204J - DESIGN AND ANALYSIS OF ALGORITHMS

MINI PROJECT REPORT

Register Number : RA2011033010041

Name of the Student : SANSKRITI SINHA

Semester / Year : 4th Semester/2nd Year

Department and Section : CSE – SWE T1

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

S.R.M. NAGAR, KATTANKULATHUR -603 203

BONAFIDE CERTIFICATE

Register No. : RA2011033010041

Certified to be bonafide record of the work done by Sanskriti Sinha of CSE SWE T1, B.tech degree course in the Mini project of 18CSC204J - Design and Analysis of Algorithms in SRM Institute of Science and Technology, Kattankulathur during the academic year 2021-22.

Date: **Lab Incharge:**

Head of Department

Submitted for University Examination held in_____SRM Institute of Science and Technology, Kattankulathur.

Examiner-1 **Examiner-2**

ACKNOWLEDGEMENT

I am over helmed in all humbleness and gratefulness to acknowledge my depth to all those who have helped me to put these ideas, well above the level of simplicity and into something concrete.
I would like to express my special thanks of gratitude to my professor Mrs. A. Jackulin Mahariba who gave me the golden opportunity to do this wonderful project on the topic “Rat in a maze”, which also helped me in doing a lot of Research and I came to know about so many new things. I am thankful to her.
Any attempt at any level can‘t be satisfactorily completed without the support and guidance of my parents and friend.
I would like to thank my parents who helped me a lot in gathering different information, collecting data and guiding me from time to time in making this project, despite of their busy schedules, they gave me different ideas in making this project unique.

~ Sanskriti Sinha

TABLE OF CONTENTS

CONTRIBUTION TABLE

S.No.	Process	Contributed By
1.	Development of Algorithm	Sanskriti Sinha
2.	Implementation	Sanskriti Sinha
3.	Documentation	Sanskriti Sinha

PROBLEM DEFINITION

A rat in a maze is a very popular backtracking problem. A Maze is given as N*N binary matrix of blocks where source block is the upper left most block i.e. maze[0][0] and destination block is lower rightmost block i.e. maze[N-1][N-1]. A rat starts from a source where its stuck and has to reach the destination. The rat can move only in two directions : forward and down. So, we need to find all paths or directions from where a rat can move.

In the maze matrix, 0 denotes that the road or path which is blocked or dead and 1 means that the block can be used in the path from source to destination.

PROBLEM EXPLANATION WITH DIAGRAM

So, basically we are given a maze(2D matrix) with obstacles, starting from (0,0) you have to reach (n-1, n-1). If you are currently on (x, y), you can move to (x+1,y) or (x,y+1) only two directions are possible. You can not move towards block.

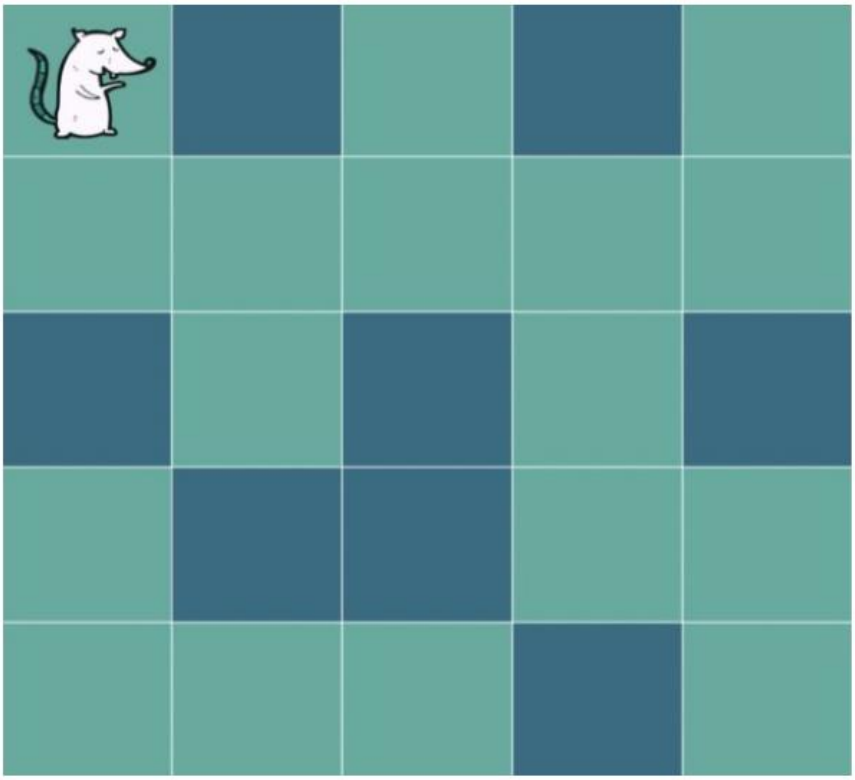
Idea: Try all the possible paths to see if you can reach (n-1,n-1).

Given Input:

0 denotes path is blocked, 1 denotes free path. Next n lines contain m numbers (0 or 1) .

Required Output:

Print 1 if rat can reach (n-1,m-1) Print 0 if it can not reach (n-1,m-1).



DESIGN TECHNIQUE USED

The rat in a maze problem is solved using **BACKTRACKING** design technique.

So, first let's understand what is Backtracking??

Backtracking is a design technique or algorithm technique for solving recursive problems or finding a solution by trying to build every possible solution incrementally and removing those solutions or steps that fail to satisfy the condition or constraints of the problem at any point of time.

So, it tries to find a bunch of solution which has some small checkpoints from where the problem can backtrack if no feasible solution is found for the problem.

In backtracking, we use recursion to explore all the possibilities until we get the best result for the problem .

The backtracking algorithm is applied to some specific types of problems. For instance, we can use it to find a feasible solution to a decision problem. It was also found to be very effective for optimization problems.

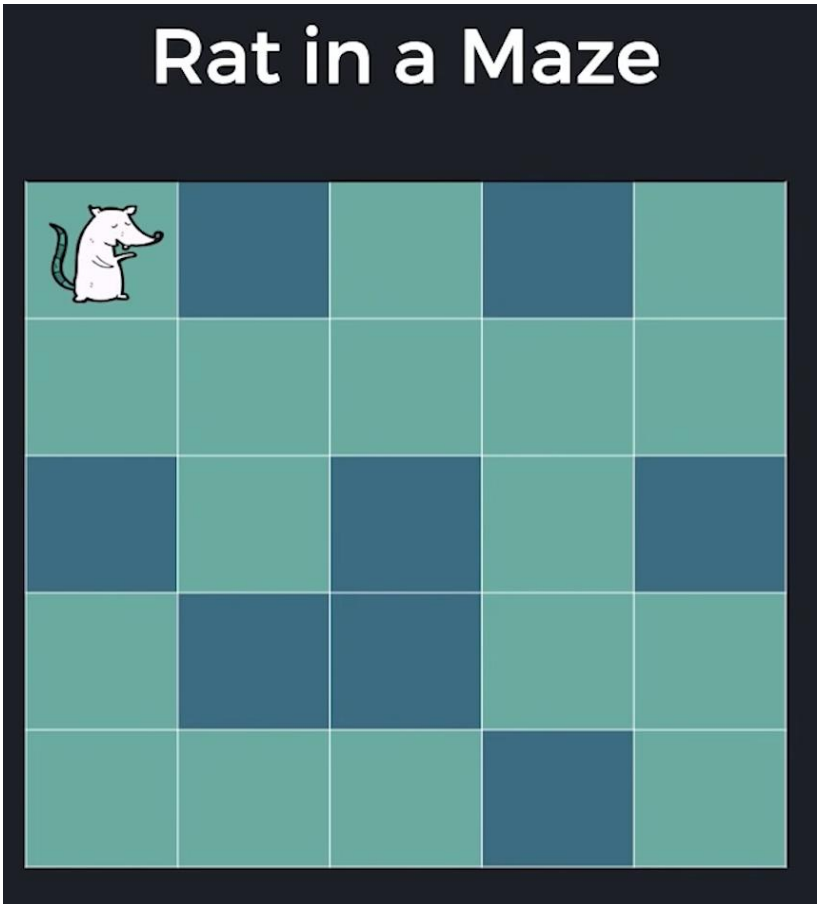
Backtracking remains a valid and vital tool for solving various kinds of problems, even though this algorithm's time complexity may be high, as it may need to explore all existing solutions.

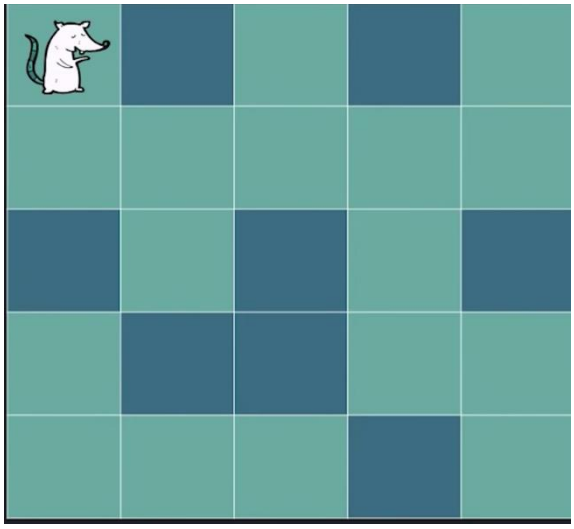
ALGORITHM FOR THE PROBLEM

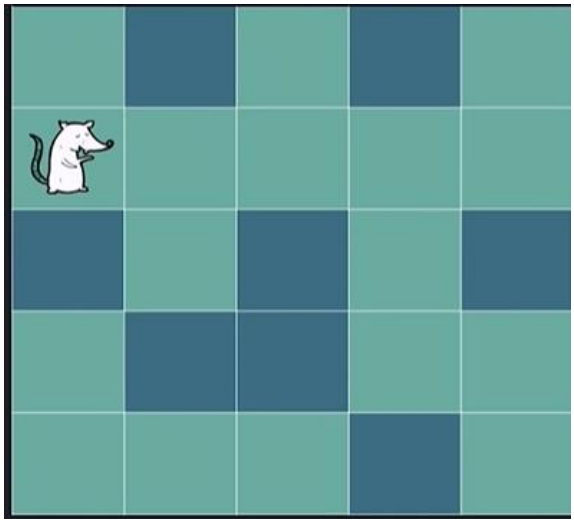
```
Begin
Algorithm isSafe(int** arr,int x,int y,int n)
{
        if(x<n && y<n && arr[x][y]==1) then
            return true;
        else
            return false;
}
Algorithm ratinMaze(int**arr,int x,int y,int n,int** solArr)
{
        if(x==n-1 && y==n-1) then
            solArr[x][y]=1;
            return true;
        if(isSafe(arr,x,y,n)) then
            solArr[x][y]=1;
            if(ratinMaze(arr,x+1,y,n,solArr)) then
                return true;

            if(ratinMaze(arr,x,y+1,n,solArr)) then
                return true;

            solArr[x][y]=0;
            return false;

        return false;
}
End
```

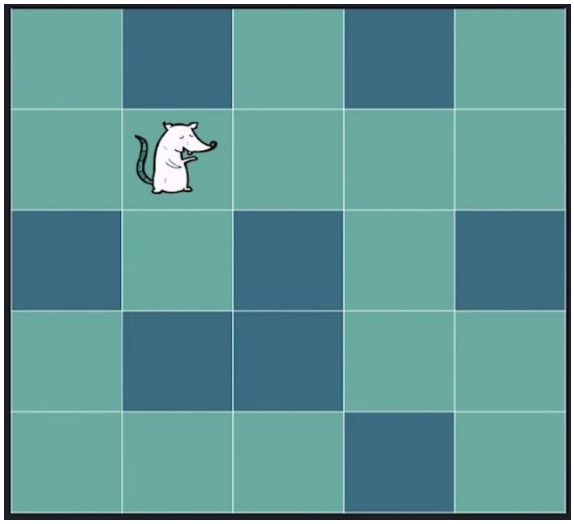
EXPLANATION OF ALGORITHM WITH EXAMPLE

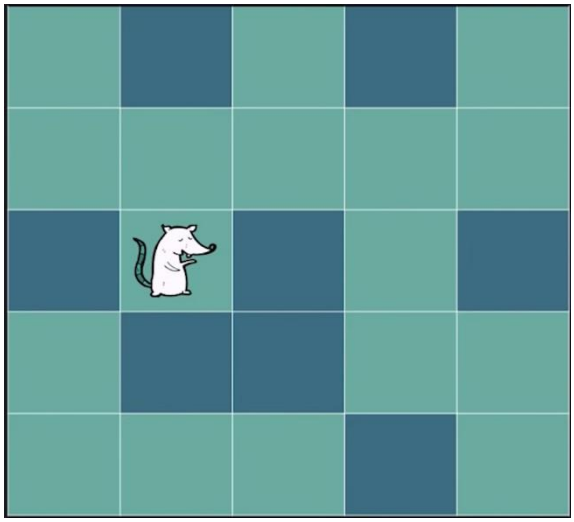
DRY RUN :

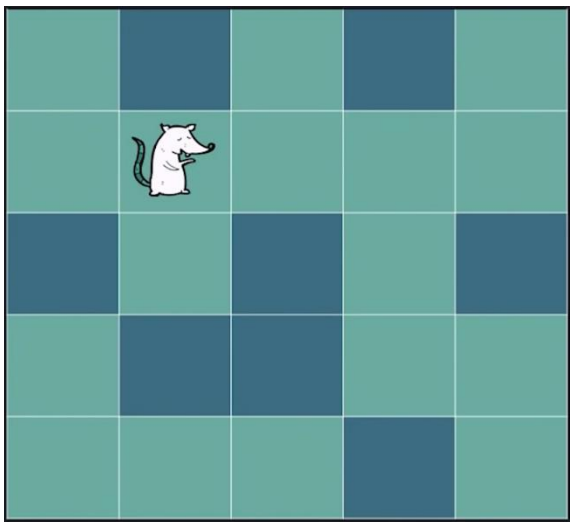


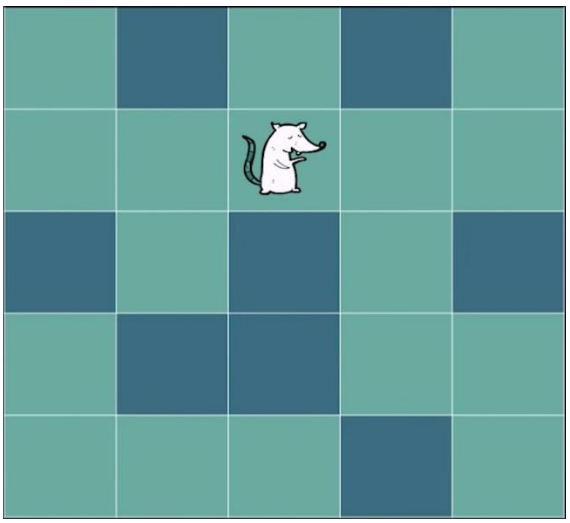


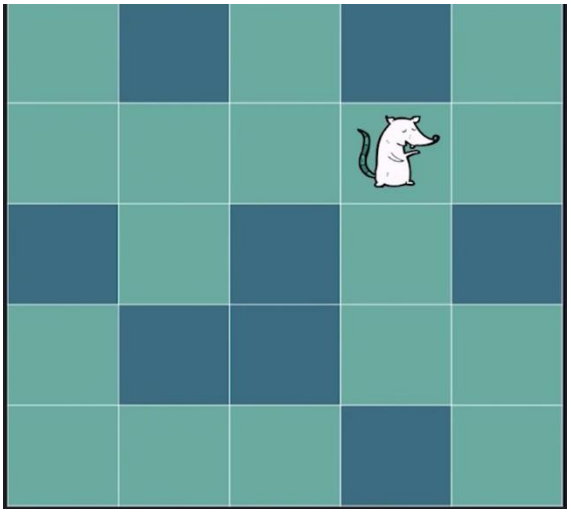


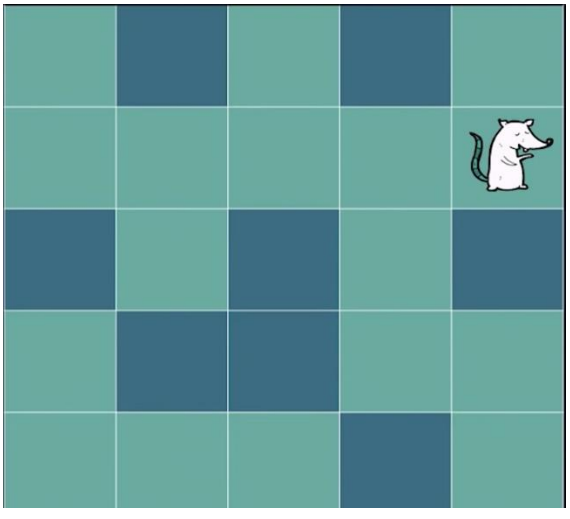


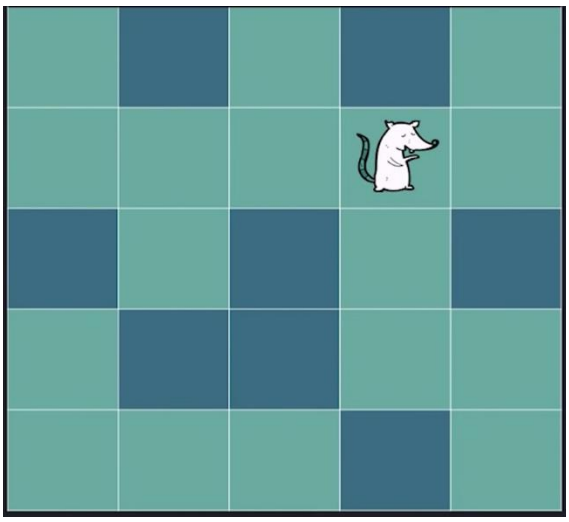


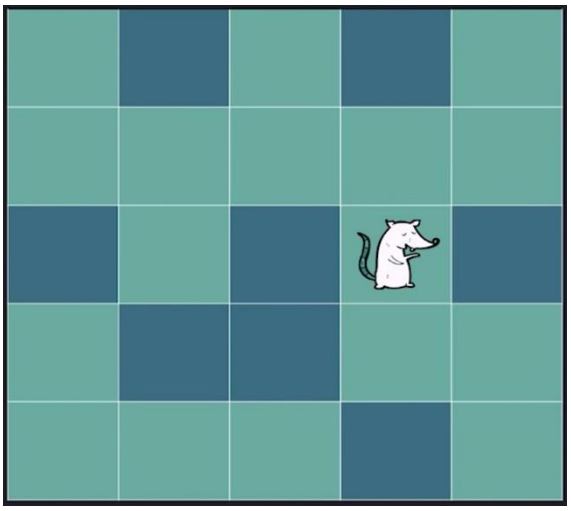


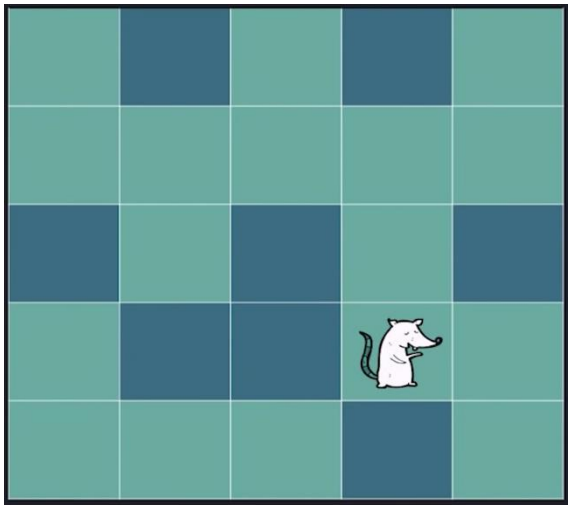


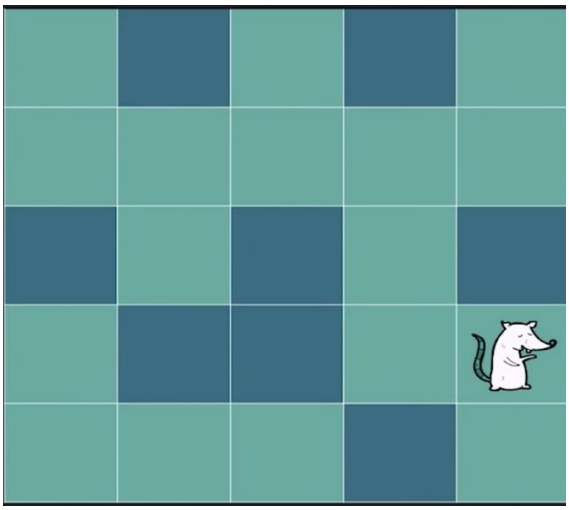




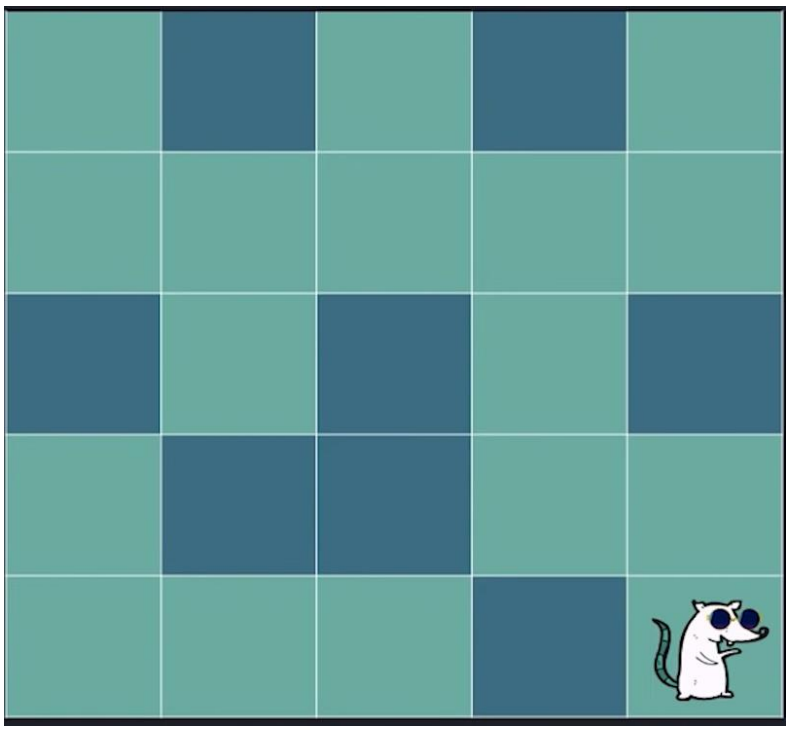








FINAL :



EXPLANATION OF ALGORITHM

So, in this problem a rat is trapped in a maze. All dark colors are blocked and rat can't go there. Also, it has a constraint that it can either move forward or in right direction. We have one function named **"isSafe"** it will tell us whether it's possible to go on a block or not, it will check whether its blocked or we can go. In parameters of this function we have dynamically allocated an array, we are passing x and y which are coordinates of a matrix (e.g. (0,0) or (1,0) etc.), then we are passing "n" which is size of matrix.

Here, we have given an *if* condition which checks that x and y should be less than n or it is out of given matrix and array should not be blocked which is equal to 1, so it will return True else False.

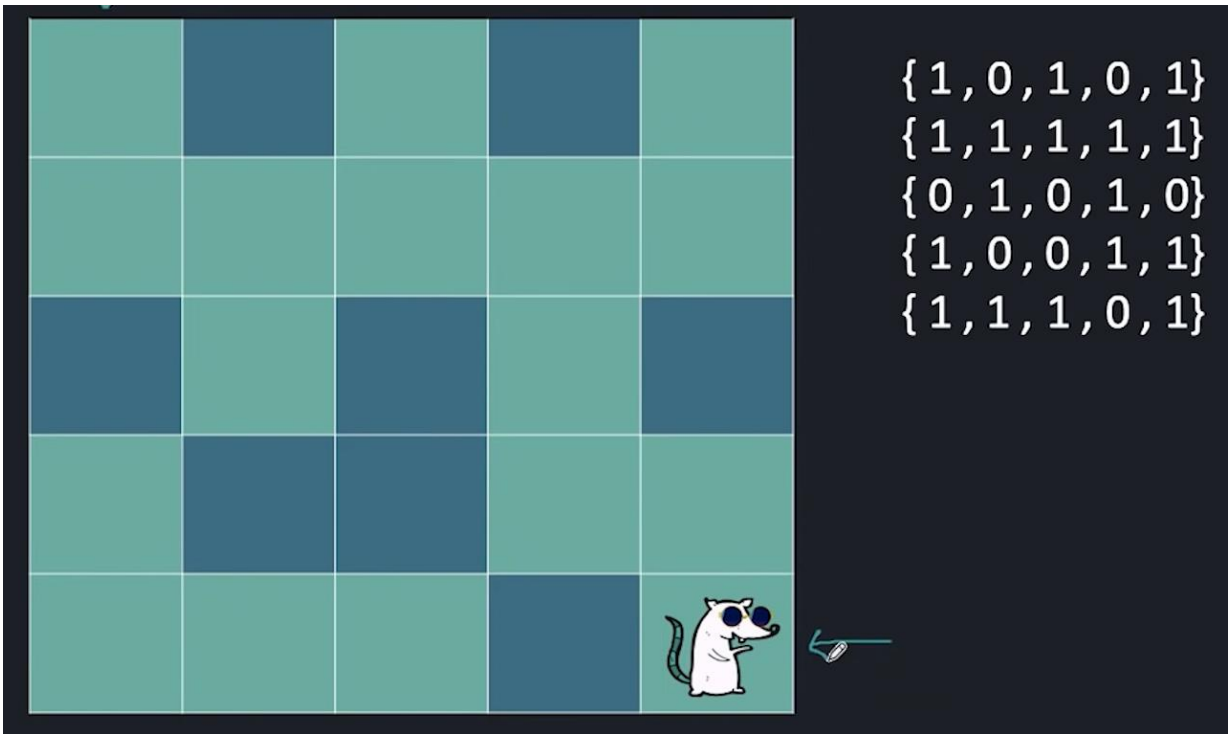
Now, we have another function called **"ratinMaze"** , it has a parameters as an array which is dynamically allocated and it is our input, then we are passing x and y which are coordinates of a matrix which will tell us our position in matrix, like where we are and then we are passing "n" which is size of matrix and **"solArr"** which will give solution of given input. In this function we have two *if* conditions and other *if* condition is an nested if.

Here, first *if* condition is our **Base case**, so whatever we want rat to be at that is our base condition and if x ==n-1 and y == n-1, so first solArr is set to 1 like solArr[x][y]=1 and return true.

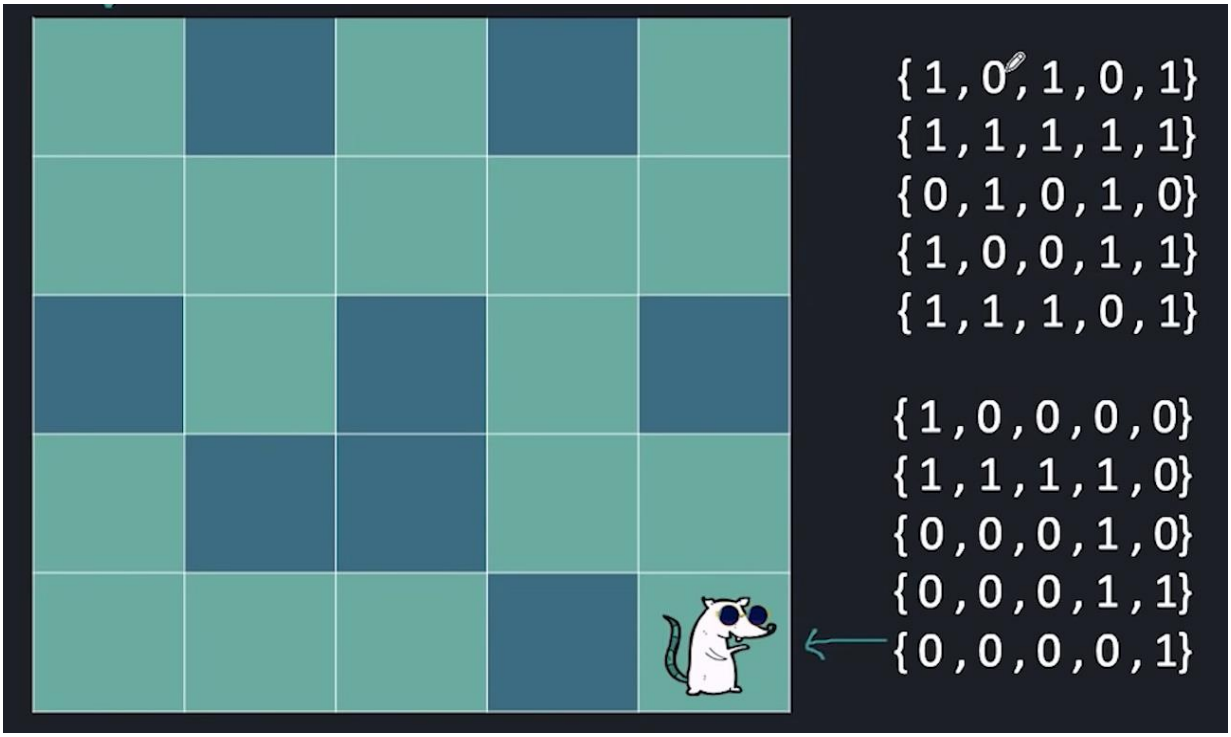
And another *if* condition will check that whatever we are standing on whether its safe or not and hence we'll call **isSafe** function, so if its safe then solArr[x][y]=1 and will return true and now we'll recursively call **ratinMaze** function where we'll pass two directions forward and towards right and if it gives us the solution we'll return true and if its false then it indicates we have chosen wrong path and we will do **BACKTRACK** so we have set that position as 1 and since it is not giving the desired output we will go back and backtrack and set is as 0 so, solArr[x][y]=0 and will return false and if the position is also wrong then it will return false.

INPUT AND OUTPUT

INPUT :



OUTPUT :



CODE AND IMPLEMENTATION

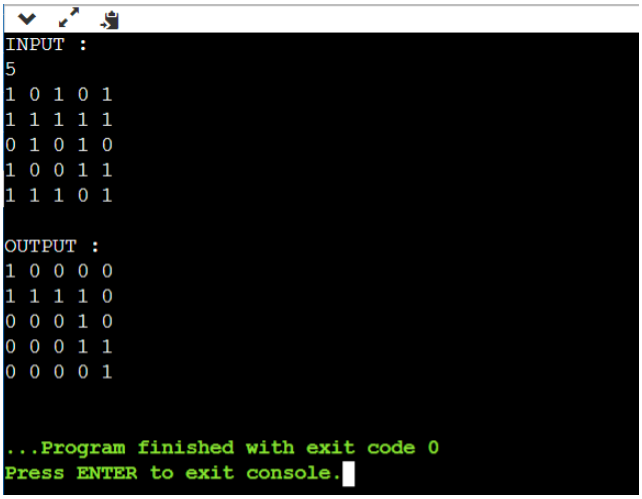
CODE :

```cpp
#include <iostream>
using namespace std;

bool isSafe(int** arr,int x,int y,int n){
    if(x<n && y<n && arr[x][y]==1){
        return true;
    }
    return false;
}

bool ratinMaze(int**arr,int x,int y,int n,int** solArr){

    if(x==n-1 && y==n-1){
        solArr[x][y]=1;
        return true;
    }

    if(isSafe(arr,x,y,n)){
        solArr[x][y]=1;
        if(ratinMaze(arr,x+1,y,n,solArr)){
            return true;
        }
        if(ratinMaze(arr,x,y+1,n,solArr)){
            return true;
        }
        solArr[x][y]=0;
        return false;
    }
    return false;
}

int main()
{

```

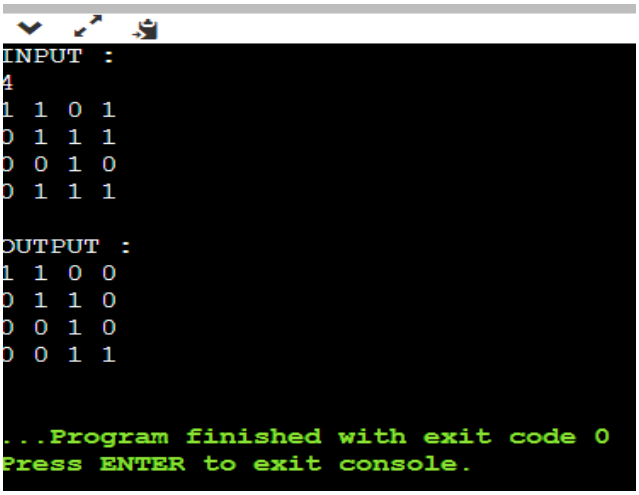
```cpp
int main()
{

    int n;
    cout<<"INPUT :\n";
    cin>>n;
    int** arr = new int*[n];
    for(int i=0;i<n;i++){
        arr[i]= new int[n];
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            cin>>arr[i][j];
        }
    }
    cout<<endl;
    cout<<"OUTPUT :\n";
    int** solArr=new int*[n];
    for(int i=0;i<n;i++){
        solArr[i]=new int[n];
        for(int j=0;j<n;j++){
            solArr[i][j]=0;
        }
    }
    if(ratinMaze(arr,0,0,n,solArr)){
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                cout<<solArr[i][j]<<" ";
            }cout<<endl;
        }
    }
    return 0;
}
```

INPUT & OUTPUT :

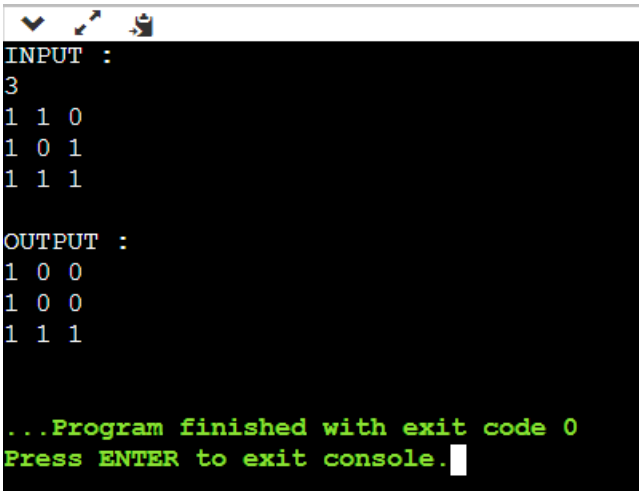
CASE 1:



CASE 2:



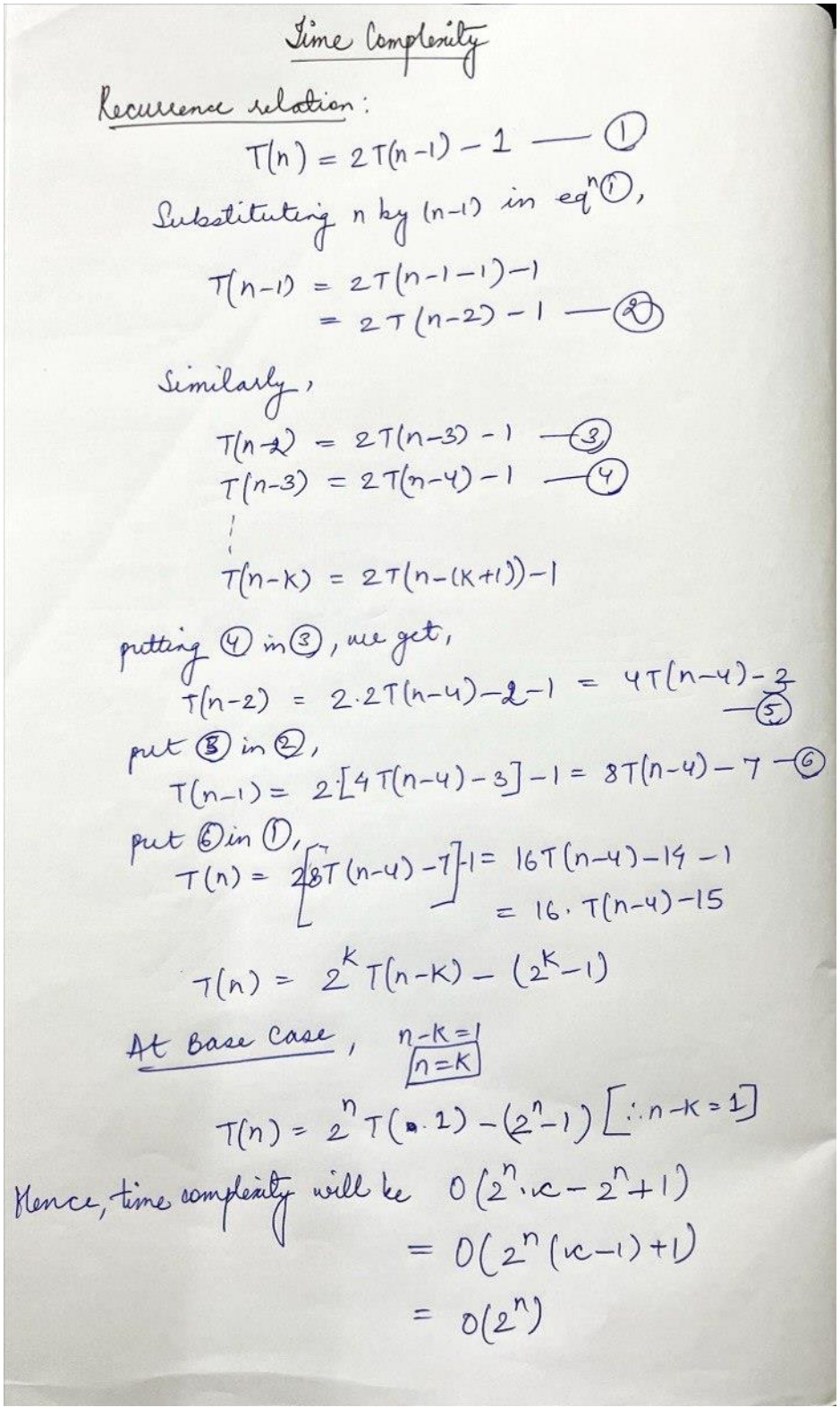
CASE 3:



COMPLEXITY ANALYSIS

Space Complexity : O(2^n)

Time Complexity : O(2^n)



Time Complexity

Recurrence relation:

$$T(n) = 2T(n-1) - 1 \quad \text{— (1)}$$

Substituting n by (n-1) in eq^n (1),

$$T(n-1) = 2T(n-1-1) - 1$$
$$= 2T(n-2) - 1 \quad \text{— (2)}$$

Similarly,

$$T(n-2) = 2T(n-3) - 1 \quad \text{— (3)}$$
$$T(n-3) = 2T(n-4) - 1 \quad \text{— (4)}$$
$$\vdots$$
$$T(n-K) = 2T(n-(K+1)) - 1$$

putting (4) in (3), we get,

$$T(n-2) = 2 \cdot 2T(n-4) - 2 - 1 = 4T(n-4) - 3 \quad \text{— (5)}$$

put (5) in (2),

$$T(n-1) = 2[4T(n-4) - 3] - 1 = 8T(n-4) - 7 \quad \text{— (6)}$$

put (6) in (1),

$$T(n) = 2[8T(n-4) - 7] - 1 = 16T(n-4) - 14 - 1$$
$$= 16 \cdot T(n-4) - 15$$

$$T(n) = 2^K T(n-K) - (2^K - 1)$$

At Base case, $n-K=1$

$$\boxed{n=K}$$

$$T(n) = 2^n T(n-n) - (2^n - 1) \quad [\because n-K=1]$$

Hence, time complexity will be $O(2^n \cdot n - 2^n + 1)$

$$= O(2^n(n-1)+1)$$

$$= O(2^n)$$

CONCLUSION

The Rat in a maze problem is implemented using Backtracking design technique in C++. The outcome is evaluated, algorithm and code is implemented, output is obtained. The discussion is on how or what are the paths or options a rat is having to come out of a maze, to reach from source to destination. Also, space complexity and time complexity is obtained.

REFERNCES

1. Backtracking – Rat in maze by Apna College

2. Rat in maze – geekforgeeks

3. Rat in maze by codingninjas

4. www.quora.com