18CSC204J
DESIGN AND ANALYSIS OF ALGORITHM

RAT IN A MAZE

A PROJECT REPORT

Submitted by

AYUSH DOGRA [RA2011003011175]
YASH VEER SINGH [RA2011003011176]
PATIL HITESH REDDY [RA2011003011163]

Under the guidance of

Ms. M. REVATHI

(Assistant Professor, Department of Computer Science & Engineering)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

JULY 2022

SRM UNIVERSITY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled "**RAT IN A MAZE via BACTRACKING ALGORITHM** "is the Bonafide work of

AYUSH DOGRA [RA2011003011175]
YASH VEER SINGH [RA2011003011176]
PATIL HITESH REDDY [RA2011003011163]

, who carried out the project work under my supervision.

Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

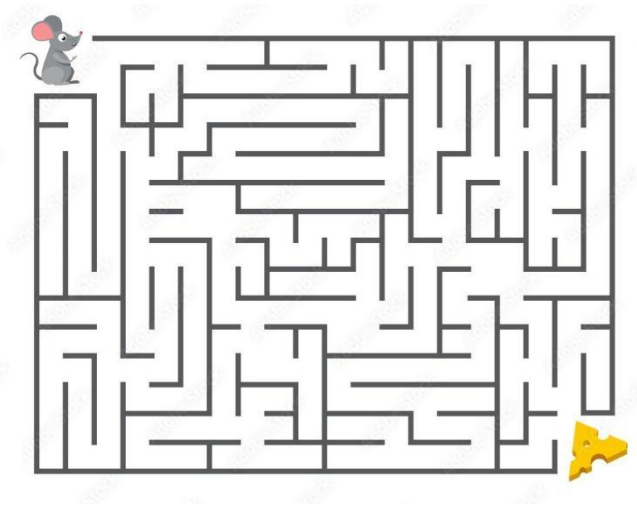
Signature of the Internal Examiner

Signature of the External Examiner

TABLE OF CONTENTS

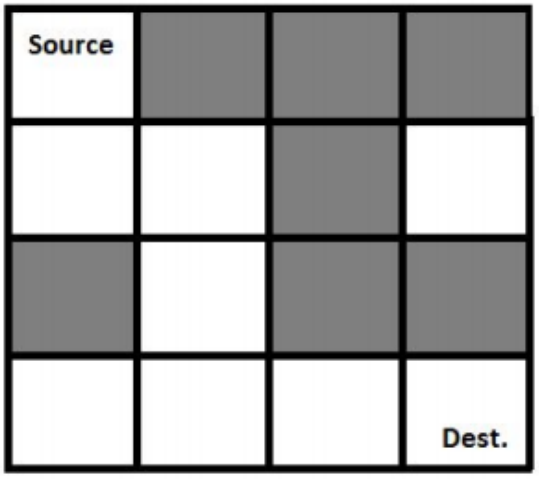
Introduction

You may remember the maze game from childhood where a player starts from one place and ends up at another destination via a series of steps. This game is also known as the *rat maze problem*



Problem Statement & Objectives

A maze is in the form of a 2D matrix in which some cells/blocks are blocked. One of the cells is termed as a source cell, from where we have to start. And another one of them is termed as a destination cell, where we
have to reach. We have to find a path from the source to the destination without moving into any of the blocked cells. A picture of an unsolved maze is shown below, where grey cells denote the dead ends and white cells denote the cells which can be accessed.

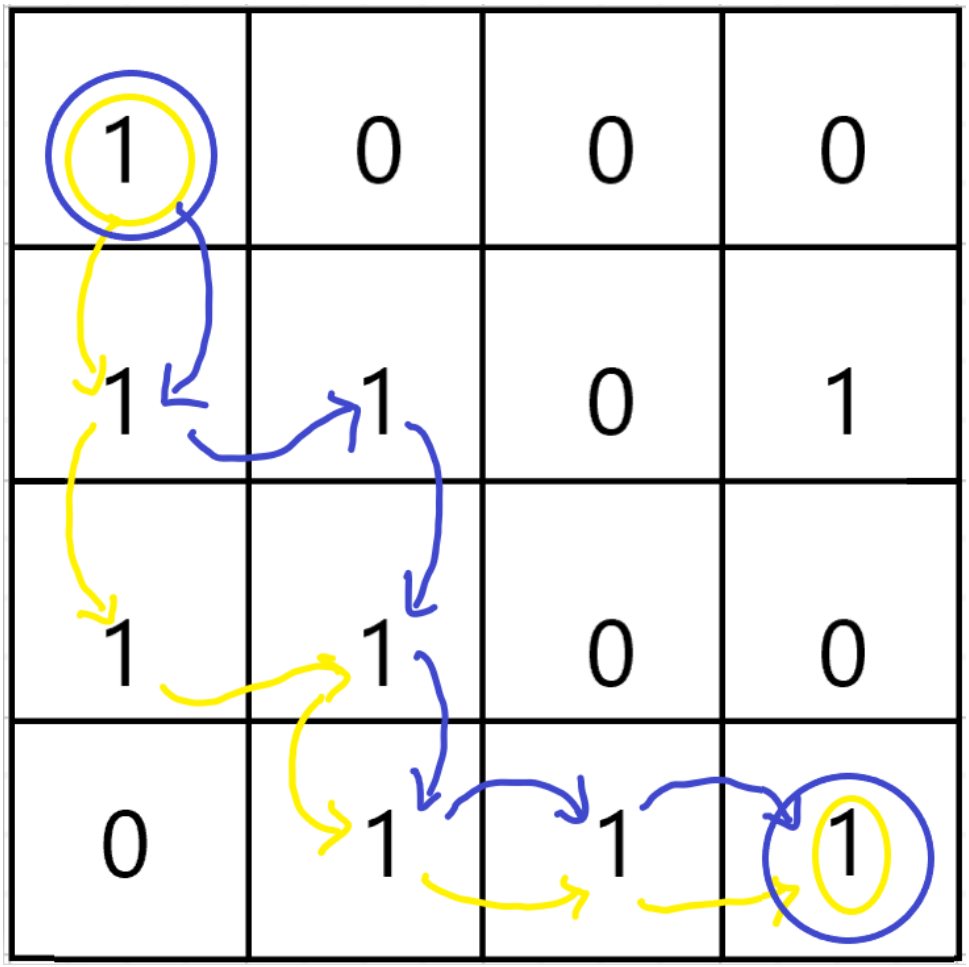


Objectives

A rat starts at a position (`source`) and can only move in two directions:

1. Forward
2. Down

The goal is to reach the destination.



Solving Rat in Maze Problem

- **Backtracking**

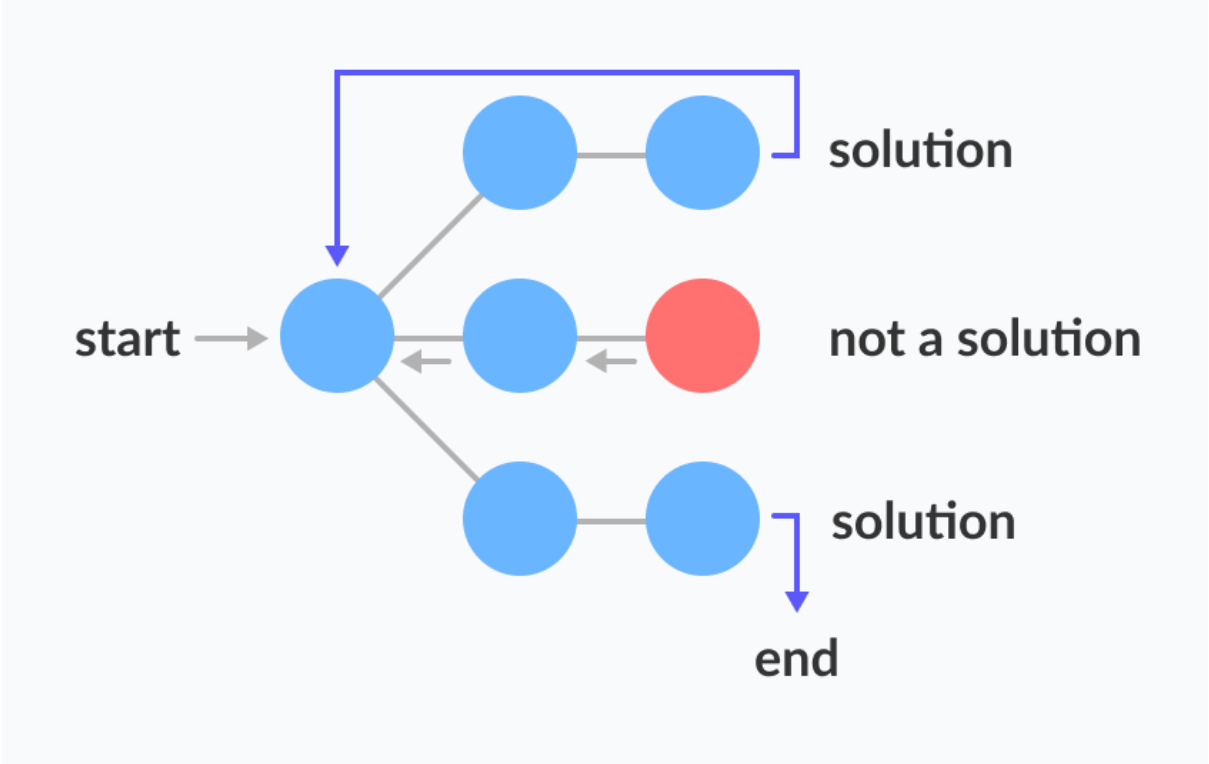
A backtracking algorithm is a problem-solving algorithm that uses a **brute force approach** for finding the desired output.
The term backtracking suggests that if the current solution is not suitable, then backtrack and try other solutions. Thus, recursion is used in this approach.
This approach is used to solve problems that have multiple solutions.

State Space Tree

A space state tree is a tree representing all the possible states (solution or nonsolution) of the problem from the root as an initial state to the leaf as a terminal state.

Backtracking Algorithm

```
Backtrack(x)
    if x is not a solution
        return false
    if x is a new solution
        add to list of solutions
    backtrack (expand x)
```



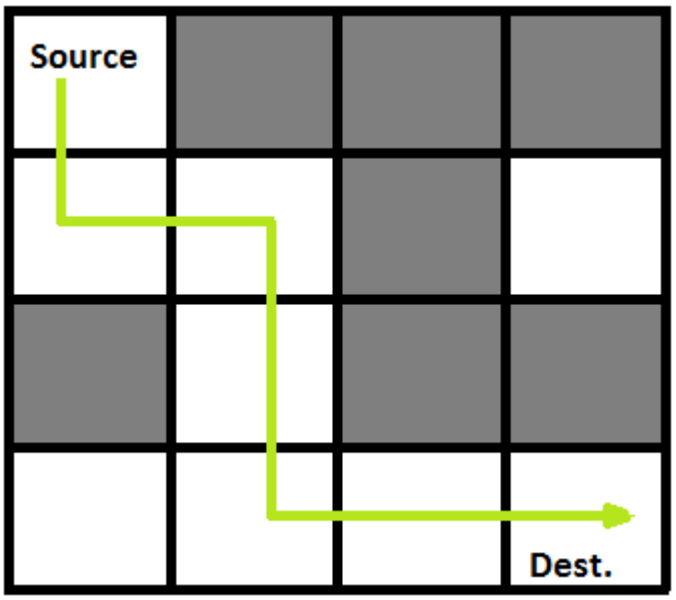
Using Backtracking Algorithm

Approach:

Initially, we will push a node with indexes i=0, j=0 and dir=0 into the stack. We will move to all the direction of the topmost node one by one in an anti-clockwise manner and each time as we try out a new path we will push that node (block of the maze) in the stack. We will increase dir variable of the topmost node each time so that we can try a new direction each time unless all the directions are explored ie. dir=4. If dir equals to 4 we will pop that node from the stack that means we are retracting one step back to the path where we came from.
We will also maintain a visited matrix which will maintain which blocks of the maze are already used in the path or in other words present in the stack. While trying out any direction we will also check if the block of the maze is not a dead end and is not out of the maze too.
We will do this while either the topmost node coordinates become equal to the food's coordinates that means we have reached the food or the stack becomes empty which means that there is no possible path to reach the food.

Algorithm

- Create a solution matrix, initially filled with 0's.
- Create a recursive function, which takes initial matrix, output matrix and position of rat (i, j).
- if the position is out of the matrix or the position is not valid then return.
- Mark the position output[i][j] as 1 and check if the current position is destination or not. If destination is reached print the output matrix and return.
- Recursively call for position (i-1,j), (I,j-1), (i+1, j) and (i, j+1).
- Unmark position (i, j), i.e output[i][j] = 0.



Example Input:

Following is a binary matrix representation of the above maze. Where 1 represent open path and 0 represent blockage

Grid:

```
{{1, 0, 0, 0}
{1, 1, 0, 1}
{0, 1, 0, 0}
```

```
{1, 1, 1, 1}}
```

Output:

Following is the solution matrix (output of program) for the above input matrix.
{1, 0, 0, 0}
{1, 1, 0, 0}
{0, 1, 0, 0}
{0, 1, 1, 1}
All entries in solution path are marked as 1.

CODE

```
// C++ program to solve Rat in a Maze problem using

// backtracking

#include <stdio.h>

#include <stdbool.h>

// Maze size

#define N 4


bool solveMazeUtil(int maze[N][N], int x, int y,int
sol[N][N]);


// A utility function to print solution matrix
sol[N][N]

void printSolution(int sol[N][N])

{

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++)

            printf(" %d ", sol[i][j]);
```

```
        printf("\n");

    }

}


// A utility function to check if x, y is valid
index for

// N*N maze

bool isSafe(int maze[N][N], int x, int y)

{

    // if (x, y outside maze) return false

    if (x >= 0 && x < N && y >= 0 && y < N &&
maze[x][y] == 1)

        return true;

    return false;

}


// This function solves the Maze problem using
Backtracking.

// It mainly uses solveMazeUtil() to solve the
problem. It

// returns false if no path is possible, otherwise
return

// true and prints the path in the form of 1s.
Please note

// that there may be more than one solutions, this
function

// prints one of the feasible solutions.
```

```
bool solveMaze(int maze[N][N])

{

    int sol[N][N] = { { 0, 0, 0, 0 },

                      { 0, 0, 0, 0 },

                      { 0, 0, 0, 0 },

                      { 0, 0, 0, 0 } };

    if (solveMazeUtil(maze, 0, 0, sol) == false) {

        printf("Solution doesn't exist");

        return false;

    }

    printSolution(sol);

    return true;

}


// A recursive utility function to solve Maze
problem

bool solveMazeUtil(int maze[N][N], int x, int y, int
sol[N][N])

{

    // if (x, y is goal) return true

    if (x == N - 1 && y == N - 1 && maze[x][y] == 1)
{

        sol[x][y] = 1;

        return true;

    }
```

```
    // Check if maze[x][y] is valid
    if (isSafe(maze, x, y) == true) {
        // Check if the current block is already
part of
        // solution path.
        if (sol[x][y] == 1)
            return false;
        // mark x, y as part of solution path
        sol[x][y] = 1;
        /* Move forward in x direction */
        if (solveMazeUtil(maze, x + 1, y, sol) ==
true)
            return true;
        // If moving in x direction doesn't give
solution
        // then Move down in y direction
        if (solveMazeUtil(maze, x, y + 1, sol) ==
true)
            return true;
        // If none of the above movements work then
        // BACKTRACK: unmark x, y as part of
solution path
        sol[x][y] = 0;
        return false;
    }
    return false;
```

```
}


// driver program to test above function
int main()
{
    int maze[N][N] = { { 1, 0, 0, 0 },
                       { 1, 1, 0, 1 },
                       { 0, 1, 0, 0 },
                       { 1, 1, 1, 1 } };
    printf("\n\nOUTPUT/SOLUTION PATH\n\n");
    solveMaze(maze);
    printf("\n\nThe 1 values show the path for
rat\n\n");
    return 0;
}
```

OUTPUT:

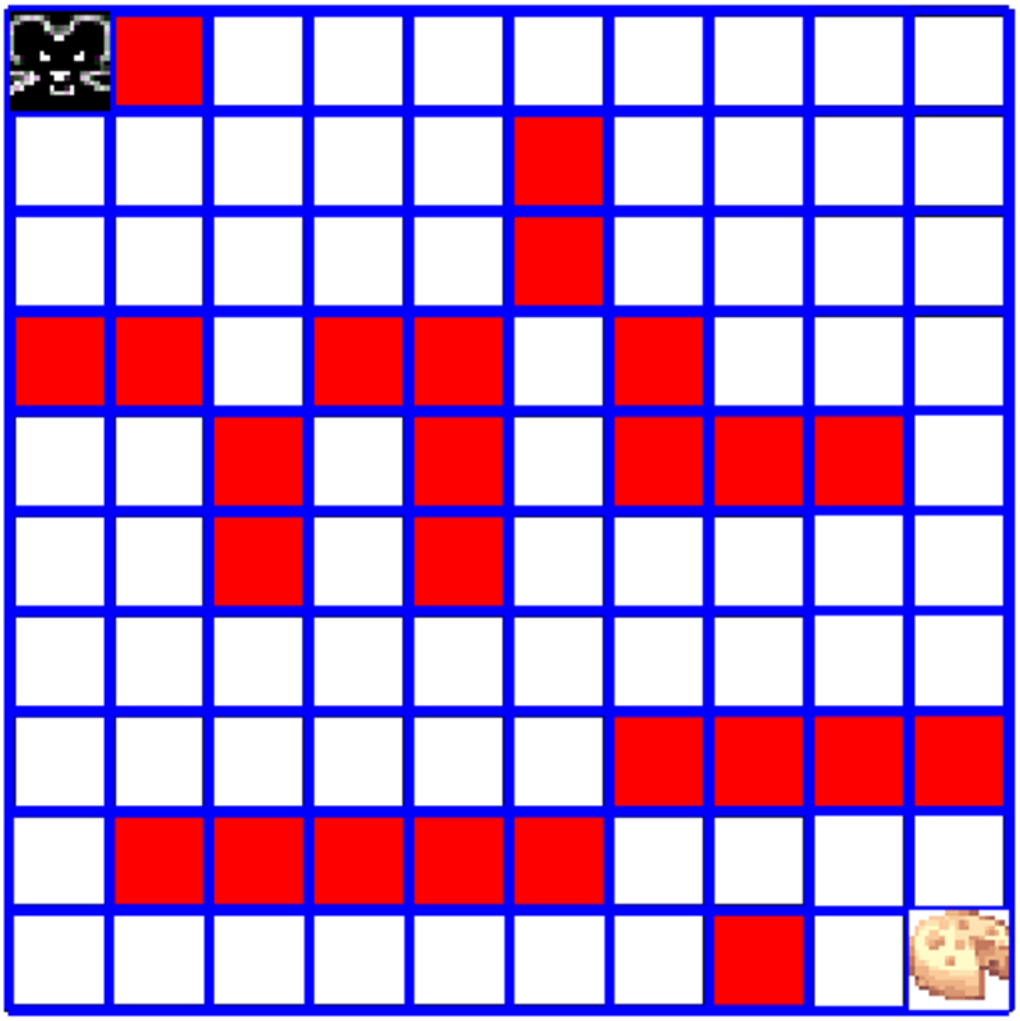
```
OUTPUT/SOLUTION PATH

 1  0  0  0
 1  1  0  0
 0  1  0  0
 0  1  1  1


The 1 values show the path for rat
```

COMPLEXITY ANALYSIS:

- Time Complexity: O(2^(n^2)).
  The recursion can run upper-bound 2^(n^2) times.
- Space Complexity: O(n^2).
  Output matrix is required so an extra space of size n*n is needed.

Conclusion:

- Using Backtracking algorithm , rat in maze problem can be solved easily as it's very intuitive to code, it is a step-by-step representation of a solution to a given problem, which is very easy to understand and it has got a definite procedure.
- A backtracking algorithm makes an effort to build a solution to a computational problem incrementally. Whenever the algorithm needs to choose between multiple alternatives to the next component of the solution, it simply tries all possible options recursively step-by-step.



References

1. https://www.geeksforgeeks.org/rat-in-a-maze-backtracking-2/
2. https://www.codingninjas.com/blog/2020/09/02/backtracking-rat-in-a-maze/