Data Structures and Algorithms: Mini Project

// Roll Numbers: A038 & A008

// Name: Atharva Rajale & Mahwish Dadan

Question:

A008	A038	Backtracking	Stack

Topic:

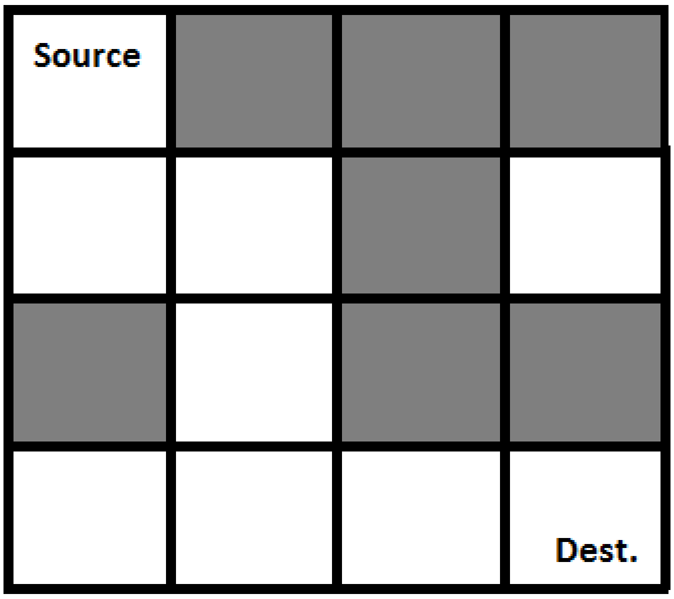
Robot in a Maze (Using Backtracking and Recursion)

Situation:

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 robot starts from source and must reach the destination. The robot can move only in two directions: forward and down.

In the maze matrix, 0 means the block is a dead end and 1 means the block can be used in the path from source to destination.

The example maze we will be using in this program:



Note: Gray blocks are dead ends (value = 0).

Matrix Representation of the above maze:

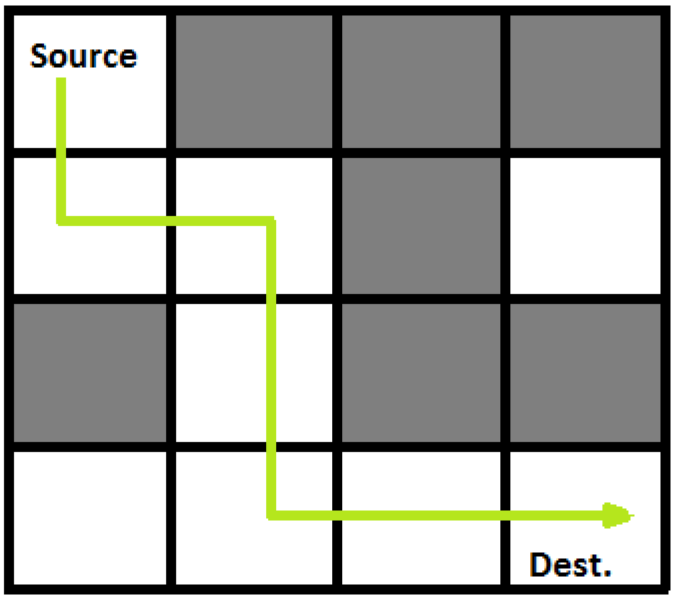
{1, 0, 0, 0}

{1, 1, 0, 1}

{0, 1, 0, 0}

{1, 1, 1, 1}

The Solution for the maze would be:



Solution Matrix (that the program will generate for us):

{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:

```
#include <stdio.h>

#define N 4

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

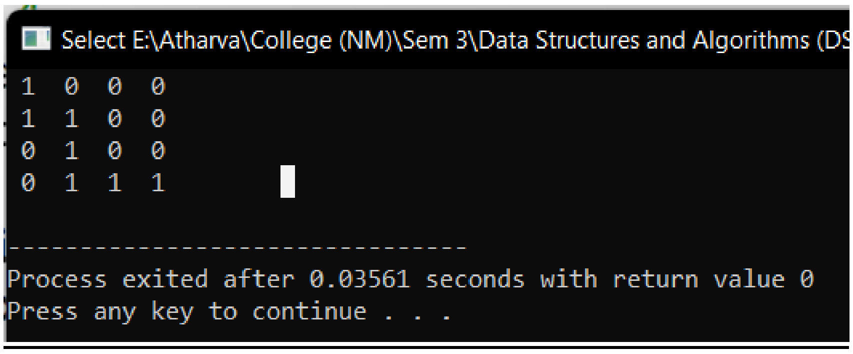
```
    }
}
bool isSafe(int maze[N][N], int x, int y)
{
 if (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1)
        return true;
    return false;
}
bool solveMaze(int maze[N][N])
{
    int sol[N][N] = { { 0, 0, 0, 0 },
                      { 0, 0, 0, 0 },
                      { 0, 0, 0, 0 },
                      { 0, 0, 0, 0 } };

    if (solveMazeUtil(maze, 0, 0, sol) == false) {
        printf("Solution doesn't exist");
        return false;
    }
    printSolution(sol);
    return true;
}
 bool solveMazeUtil(int maze[N][N], int x, int y, int sol[N][N])
{
if (x == N - 1 && y == N - 1) {
        sol[x][y] = 1;
        return true;
    }
if (isSafe(maze, x, y) == true) {
```

```
            sol[x][y] = 1;
 if (solveMazeUtil(maze, x + 1, y, sol) == true)
            return true;
 if (solveMazeUtil(maze, x, y + 1, sol) == true)
            return true;
  sol[x][y] = 0;
      return false;
   }
   return false;
}
int main()
{
   int maze[N][N] = { { 1, 0, 0, 0 },
                   { 1, 1, 0, 1 },
                   { 0, 1, 0, 0 },
                   { 1, 1, 1, 1 } };

   solveMaze(maze);
   return 0;
}
```

Output:



Select E:\Atharva\College (NM)\Sem 3\Data Structures and Algorithms (DS

```
 1  0  0  0
 1  1  0  0
 0  1  0  0
 0  1  1  1

--------------------------------
Process exited after 0.03561 seconds with return value 0
Press any key to continue . . .
```

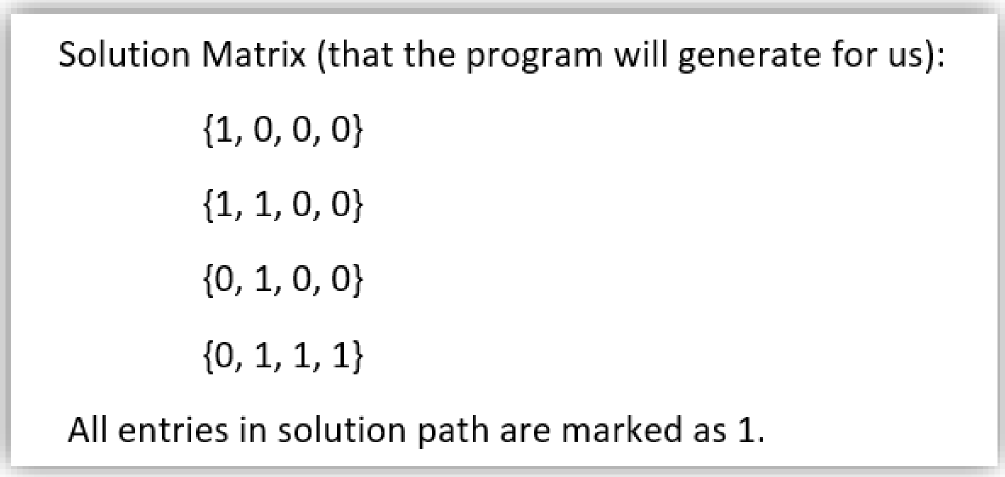
```
1  0  0  0

1  1  0  0

0  1  0  0

0  1  1  1
```

We can verify that this output is correct by cross checking it with our solution from earlier:



Solution Matrix (that the program will generate for us):

{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.

As we can see, the program's output matches the answer from earlier, therefore our program is functional and correct.

References:

geeksforgeeks.org

stackoverflow.com

codesdope.com

simple.wikipedia.org/wiki/Backtracking