**RAT IN A MAZE**

**PROBLEM DEFINITION:**

Maze game is a well-known problem, where we are given a grid of 0’s and 1’s, 0’s corresponds to a place that can be traversed, and 1 corresponds to a place that cannot be traversed (i.e. a wall or barrier); the problem is to find a path from bottom left corner of grid to top right corner; immediate right, immediate left, immediate up and immediate down only are possible (no diagonal moves). We consider a variant of the maze problem where a cost (positive value) or profit (negative value) is attached to visiting each location in the maze, and the problem is to find a path of least cost through the maze.

**METHOD FOR SOLVING THE PROBLEM:**

BACKTRACKING:

A backtracking algorithm tries to construct a solution to a computational problem incrementally, one small piece at a time. Whenever the algorithm needs to decide between multiple alternatives to the next component of the solution, it recursively evaluates every alternative and then chooses the best one. Backtracking is a systematic method to iterate through all the possible configurations of a search space.

We chose backtracking method because we can stop anytime if we find moving forward is not possible/not useful. We first start with the source cell and move in a direction where the path is not blocked. If taken path makes us reach to the destination then the puzzle is solved else, we come back and change our direction of the path taken. We are going to implement the same logic in our code also.

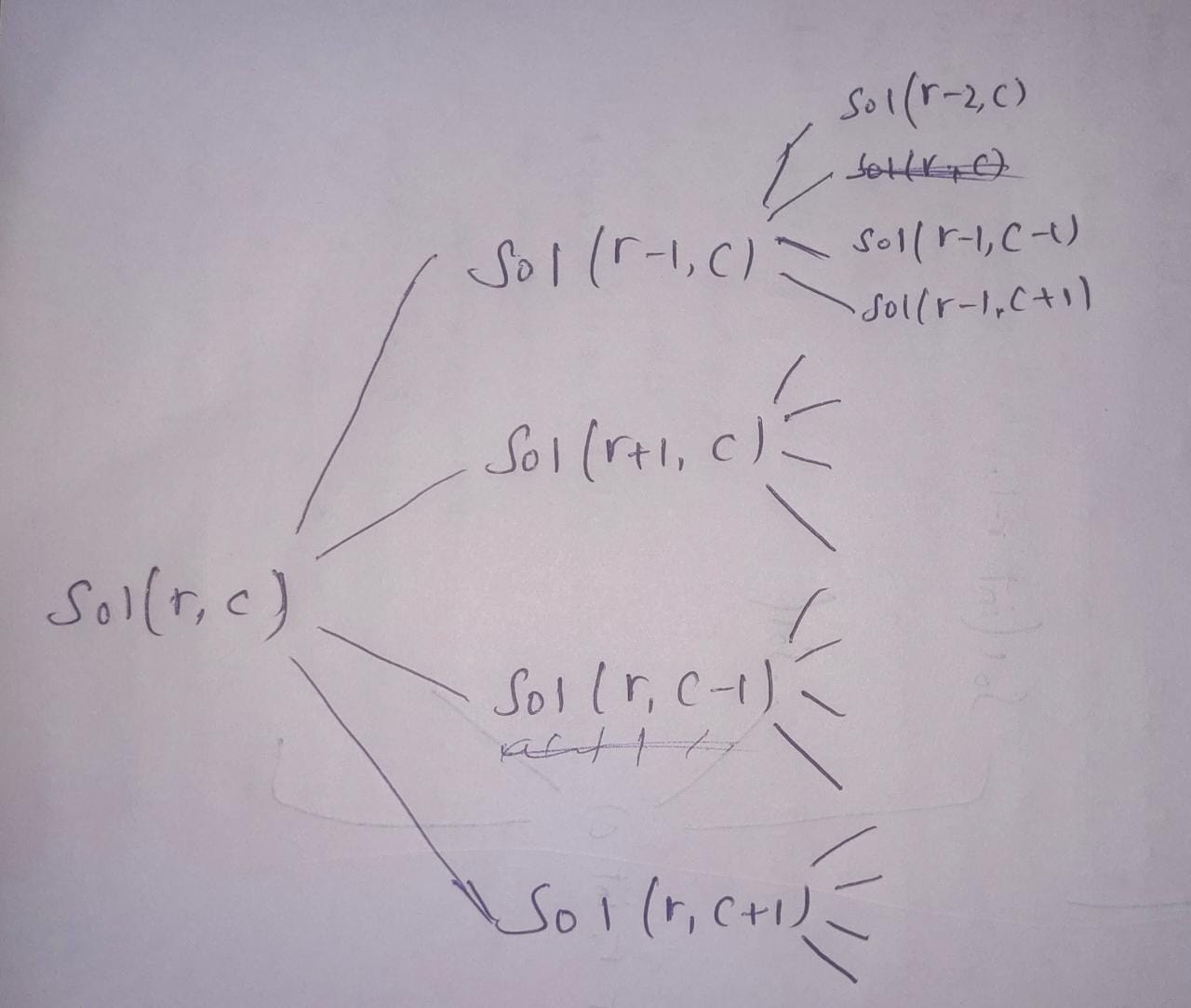
**ANALYSIS:**

TIME COMPLEXITY:

Since we are transversing cells of whole 2D array, for that part time complexity will be O(MAX^2)

And each cell can have atmost 3 unvisited cells. So the function which splits in three calls will have an exponential time complexity of O(3^n)

So the time complexity = O(3^(MAX^2))



SPACE COMEPLEXITY:

Since we have to display the output 2D array the space required will be O(MAX^2).

And if maximum depth in each recursion is DEPTH, then the total space required will be O(DEPTH\*(MAX^2))

**PROGRAM:**

#include <iostream>

using namespace std;

#define MAX 5

int ansMatrix[MAX][MAX];

int solution(int r, int c,int mat[MAX][MAX])

{

if ((r == 0) && (c == MAX - 1))

{

ansMatrix[r][c] = 1;

return true;

}

if (r >= 0 && r < MAX && c >= 0 && c < MAX && mat[r][c] == 0 && ansMatrix[r][c] == 0)

{

ansMatrix[r][c] = 1;

if (solution(r - 1, c, mat))

{

return true;

}

if (solution(r + 1, c, mat))

{

return true;

}

if (solution(r, c - 1, mat))

{

return true;

}

if (solution(r, c + 1, mat))

{

return true;

}

ansMatrix[r][c] = 0;

return false;

}

return false;

}

int main()

{

int i, j, x = 0;

int mat[MAX][MAX];

for (i = 0; i < MAX; i++)

{

for (j = 0; j < MAX; j++)

{

cin>>mat[i][j];

}

}

for (i = 0; i < MAX; i++)

{

for (j = 0; j < MAX; j++)

{

ansMatrix[i][j] = 0;

}

}

if (solution(MAX - 1, 0,mat) == true)

{

cout << "The solution path:\n";

for (i = 0; i < MAX; i++)

{

for (j = 0; j < MAX; j++)

{

if (ansMatrix[i][j] == 0)

{

cout << "1"

<< " ";

}

else

{

cout << "0"

<< " ";

}

if (ansMatrix[i][j] == 1 && i % 2 == 0)

{

x = x + 1;

}

else if (ansMatrix[i][j] == 1 && i % 2 == 1)

{

x = x + 2;

}

}

cout << "\n";

}

cout << "\n";

cout << "The total cost of followed way is:" << x;

}

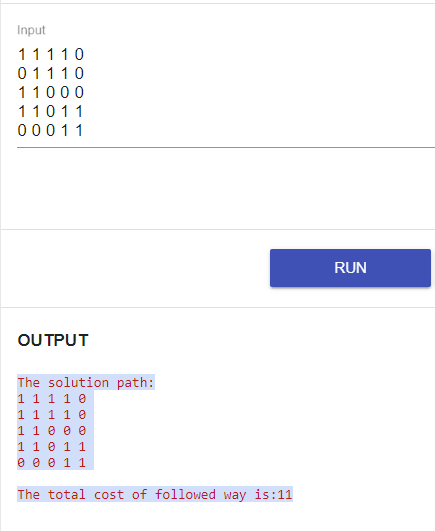
else

cout << "There is no possible way";

return 0;

}

**SOLUTION:**

****