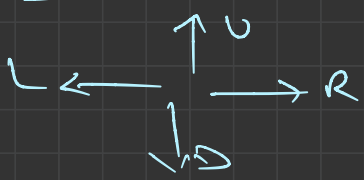
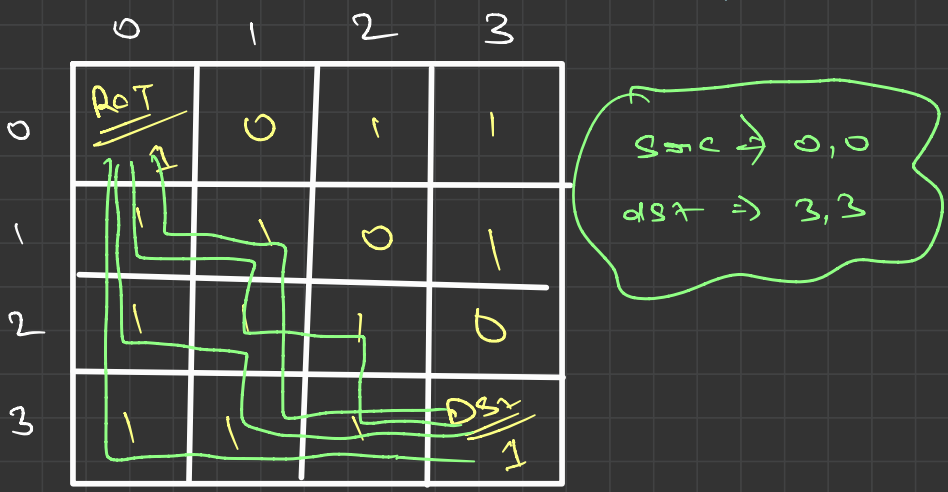
Rat in a maze → Print all possible paths

Possible movements

0 → Closed path
1 → Open path

Move Direction





Base Case

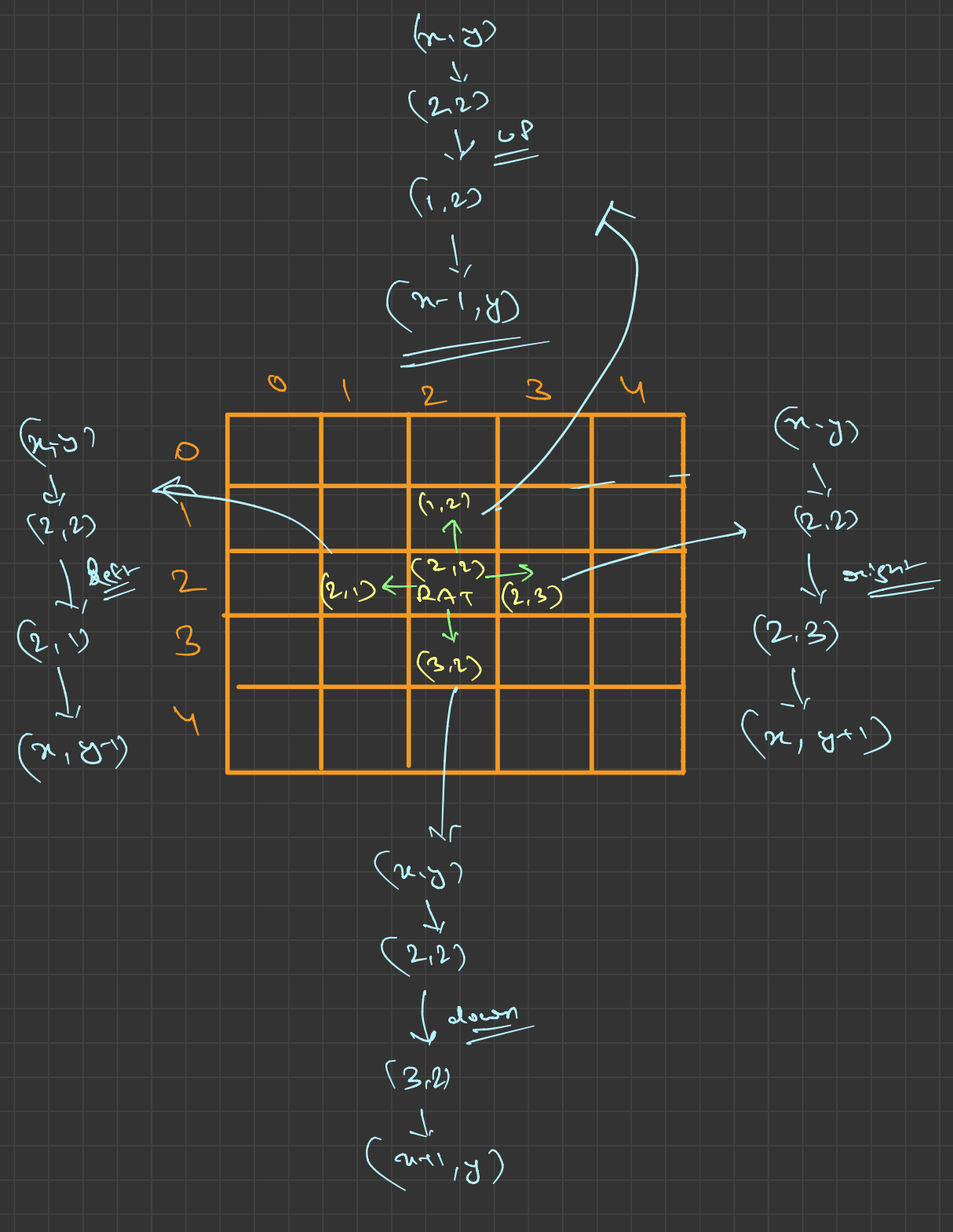
when dst found → (3,3)

Call recursion four times

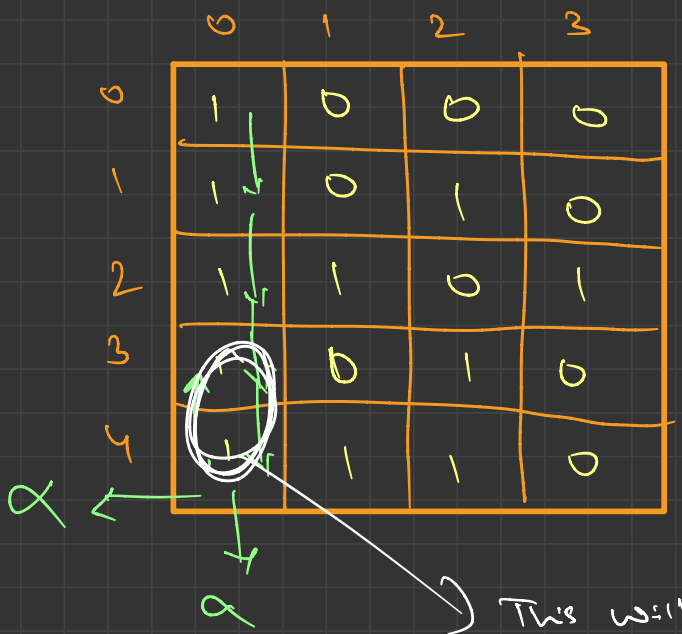
↳ solve only one case

↓

rec → up
rec → right
rec → down
rec → left


(x,y)
(2,2)
up
(1,2)
(x-1,y)
0 1 2 3 4
(x,y)
(2,2)
left
(2,1)
(x,y-1)
(x,y)
(2,2)
right
(2,3)
(x,y+1)
(1,2)
(2,1)
RAT
(2,3)
(3,2)
(x,y)
(2,2)
down
(3,2)
(x+1,y)

Possibility → Infinite loop



This will create a infinite loop

↓

We dont go the same position that we covered or visit

↓

Mark the position visited

↓

Make an illusion boal 2D - array

```
void printAllPath(int maze[][4], int row, int col, int srcX, int srcY,
string &output, vector<vector<bool > > &visited){
    // dest coord -> [row-1][col-1]

    // base case
    if (srcX == row-1 && srcY == col-1)
    {
        // reached destination
        cout << output << endl;
        return;
    }

    // solve 1 case -> remaining rec handle
    // edge case ->
        // -> path closed
        // -> out if bound
        // -> check is pos already visited

    // up
    int newX = srcX - 1;
    int newY = srcY;
    if (isSafe(srcX, srcY, newX, newY, maze, row, col, visited))
    {
        // mark visited
        visited[newX][newY] = true;
        // call rec
        output.push_back('U');
        printAllPath(maze, row, col, newX, newY, output, visited);
        // backtracking
        visited[newX][newY] = false;
        output.pop_back();
    }

    // down
    newX = srcX + 1;
    newY = srcY;
    if (isSafe(srcX, srcY, newX, newY, maze, row, col, visited))
    {
        // mark visited
        visited[newX][newY] = true;
        // call rec
        output.push_back('D');
        printAllPath(maze, row, col, newX, newY, output, visited);
        // backtracking
        visited[newX][newY] = false;
        output.pop_back();
    }

    // left
    newX = srcX;
    newY = srcY - 1;
    if (isSafe(srcX, srcY, newX, newY, maze, row, col, visited))
    {
        // mark visited
        visited[newX][newY] = true;
        // call rec
        output.push_back('L');
        printAllPath(maze, row, col, newX, newY, output, visited);
        // backtracking
        visited[newX][newY] = false;
        output.pop_back();
    }

    // right
    newX = srcX;
    newY = srcY + 1;
    if (isSafe(srcX, srcY, newX, newY, maze, row, col, visited))
    {
        // mark visited
        visited[newX][newY] = true;
        // call rec
        output.push_back('R');
        printAllPath(maze, row, col, newX, newY, output, visited);
        // backtracking
        visited[newX][newY] = false;
        output.pop_back();
    }
}
```

```
// function -> that will handle all the edge cases
// edge case ->
    // -> path closed
    // -> out if bound
    // -> check is pos already visited
bool isSafe(int srcX, int srcY, int newX, int newY, int maze[][4], int row, int col,
vector<vector<bool > > &visited){
    if (
        (newX >= 0 && newX < row) && (newY >= 0 && newY < col) &&
        maze[newX][newY] == 1 &&
        visited[newX][newY] == false)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

```
int main(){

    int maze[4][4] = {
        {1,0,0,0},
        {1,1,0,0},
        {1,1,1,0},
        {1,1,1,1}
    };

    int row = 4;
    int col = 4;

    int srcX = 0;
    int srcY = 0;

    string output = "";

    // create visited 2d array
    vector<vector<bool > > visited(row, vector<bool>(col, false));

    // if rat position -> src -> 0
    if (maze[0][0] == 0)
    {
        cout << "no path exist" << endl;
    }
    else
    {
        visited[srcX][srcY] = true;
        printAllPath(maze, row, col, srcX, srcY, output , visited);
    }

    return 0;
}
```