$\Rightarrow$ mat[9][9] , partially filled , fill this Sudoku.
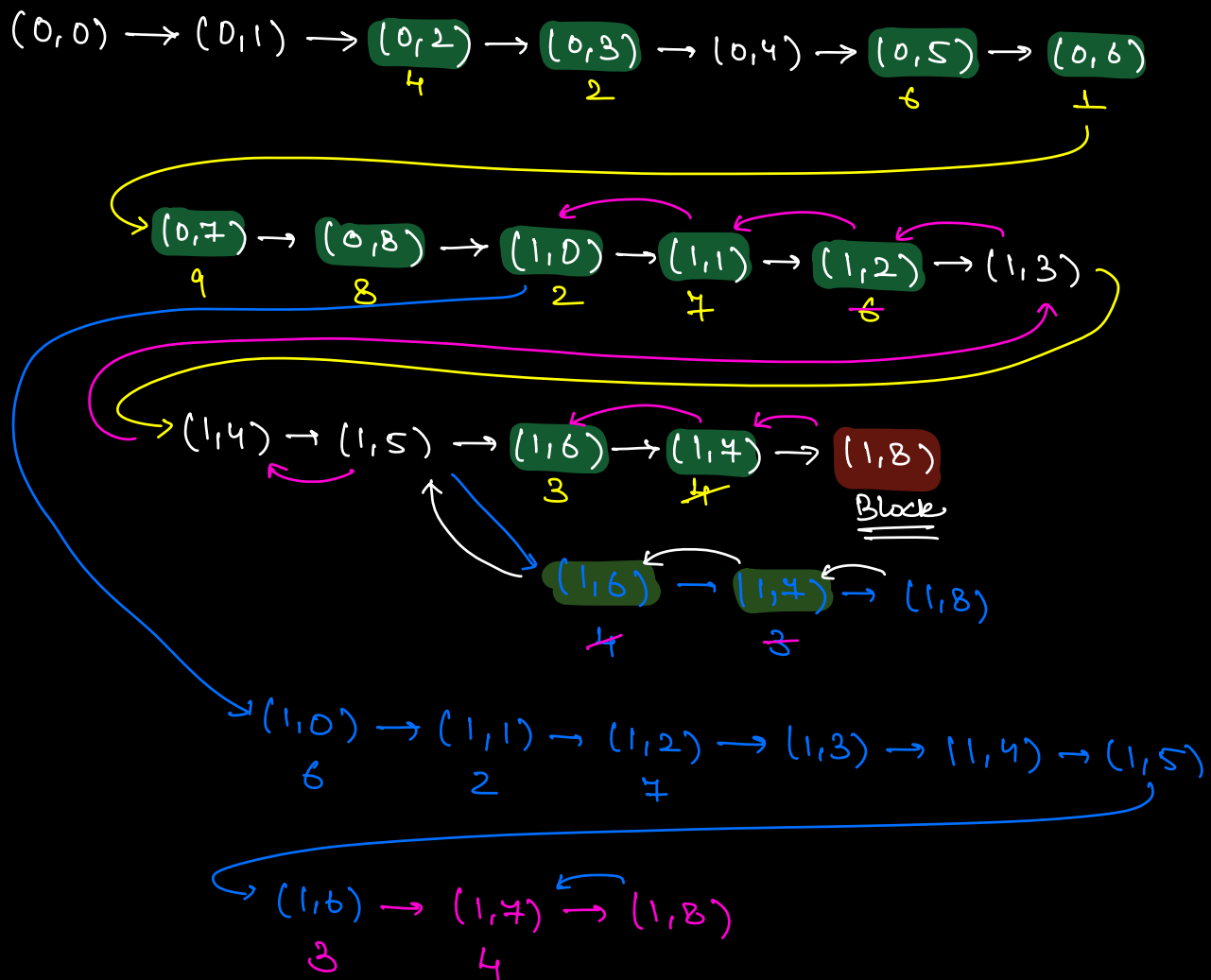   [1-9]

Rules :

1. In a row, number can't repeat.

2. In a col , number can't repeat.

3. In a 3×3 box, number can't repeat.

Start index of
3×3 box

4,4 $\rightarrow$ 3,3	Nearest multiple of 3
5,2 $\rightarrow$ 3,0	$\leq i$.
2,7 $\rightarrow$ 0,6	
7,7 $\rightarrow$ 6,6	$r \rightarrow r - r \% 3$
4,1 $\rightarrow$ 3,0	$c \rightarrow c - c \% 3$

$(0,0) \longrightarrow (0,1) \longrightarrow (0,2) \longrightarrow (0,3) \longrightarrow (0,4) \longrightarrow (0,5) \longrightarrow (0,6)$

4   2   6   1

$(0,7) \longrightarrow (0,8) \longrightarrow (1,0) \longrightarrow (1,1) \longrightarrow (1,2) \longrightarrow (1,3)$

9   8   2   7   6

$(1,4) \longrightarrow (1,5) \longrightarrow (1,6) \longrightarrow (1,7) \longrightarrow (1,8)$

3   4   Block

$(1,6) \longrightarrow (1,7) \longrightarrow (1,8)$

~~4~~   ~~3~~

$(1,0) \longrightarrow (1,1) \longrightarrow (1,2) \longrightarrow (1,3) \longrightarrow (1,4) \longrightarrow (1,5)$

6   2   7

$(1,6) \longrightarrow (1,7) \longrightarrow (1,8)$

3   4

$\Rightarrow$ Backtracking

     ↳ Try all the possibilities.

$\Rightarrow$ Recursion + Backtracking

$\Rightarrow$ function

     mat[][], ⓝ

	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	9	10	11	12	13	14	15	16	17
2	18	19	20	21	22	23	24	25	26
3	27	28	29	30	31	32	33	34	35
4	36	37	38	39	40	41	42	43	44
5	45	46	47	48	49	50	51	52	53
6	54	55	56	57	58	59	xe		
7									
8									80

9×9

$x = 24 \quad \nearrow \quad r = 24 / 9$
$\qquad\qquad \searrow \quad c = 24 \,\%\, 9$
$\qquad \downarrow$
$2 \times 9 + 6$

$x = 50 \Rightarrow r = 50/9 = 5$
$\qquad\qquad\qquad c = 50 \,\%\, 9 = 5$

$x \longrightarrow r = x / 9$
$\qquad\qquad c = x \,\%\, 9$

```
void   sudokusolver (mat[][] , x ) {
    if ( x == 81 ) {
        print (mat[][])
        return;
    }
  3
    r = x/9 ,   c = x%9 ;
    if (mat[r][c] != 0 ) {
        // filled  cell.
        sudokusolver ( mat , x+1);
  3
    else { // mat[r][c] = 0 ⇒ Blank cell
        for ( i = 1; i <= 9; i++ ) {
            if ( isValid (mat, r, c, i )) {
                mat[r][c] = i;
                ⇒ sudokusolver ( mat , x+1 );
                mat[r][c] = 0
            3
        3
    3
  3

  3
```

```
bool    isValid ( mat[][] , r, c , d ) {
        // this fun^ checks if it is possible to put
        // i at index r,c.

        // row & col
        for( j = 0; j < 9; j++) {
            if ( mat [r][j] == d ) return false;
            if ( mat [j][c] == d )  return false;
        }

        // Start index of 3×3 Box
        x = r - r%3
        y = c - c%3
        for ( i = x ; i < x + 3 ; i++) {
            for( j = y ; j < y + 3 ; j++) {
                if ( mat [i][j] == d) {
                    return false;
                }
            }
        }

        return true;
}
```

$$TC: \quad 9 \times 9 \times 9 \times 9 \cdots 9 \Rightarrow 9^{81}$$

$$\underbrace{\quad\quad\quad\quad\quad\quad\quad\quad}_{81 \text{ times}}$$
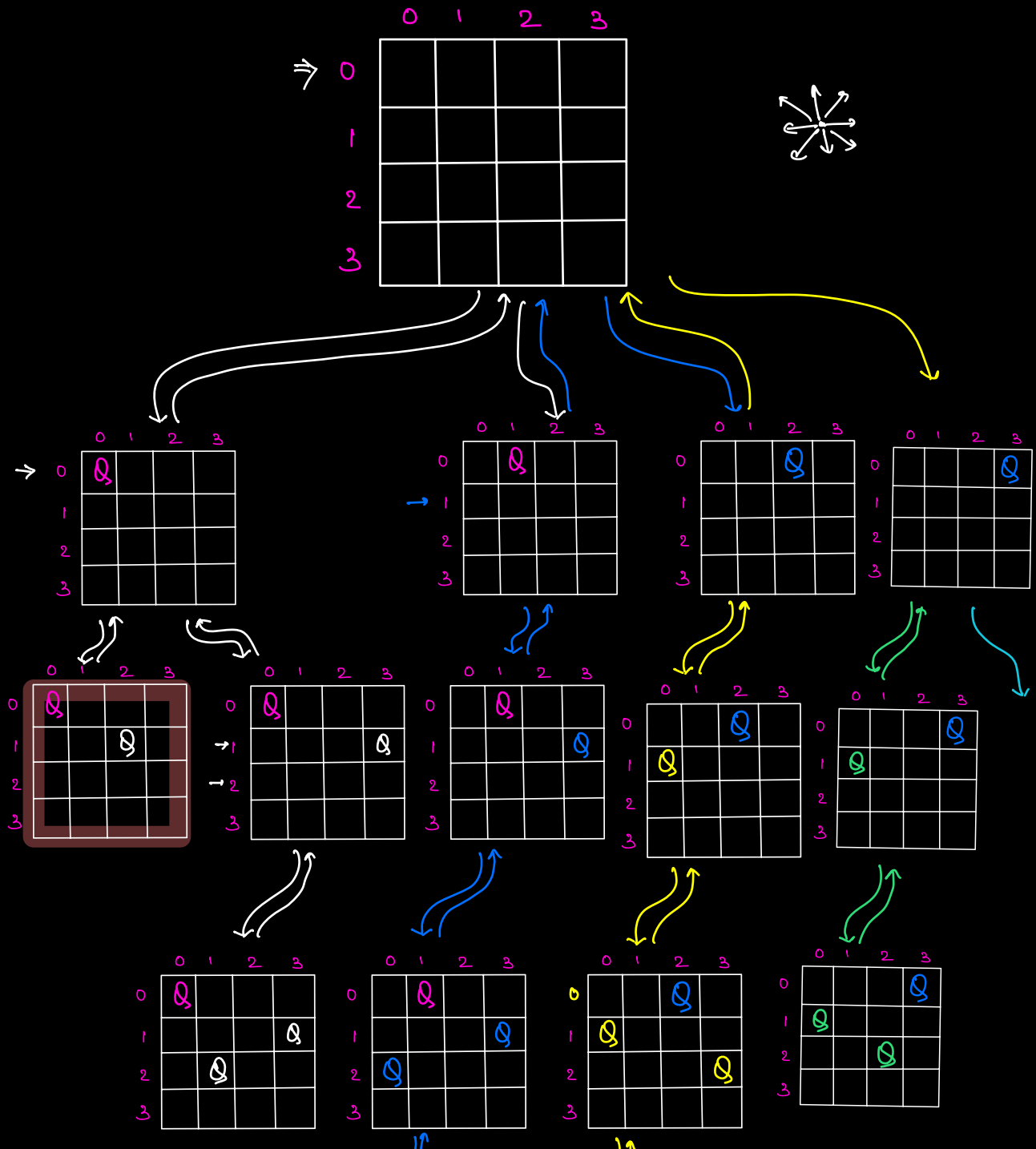
Upper Bound
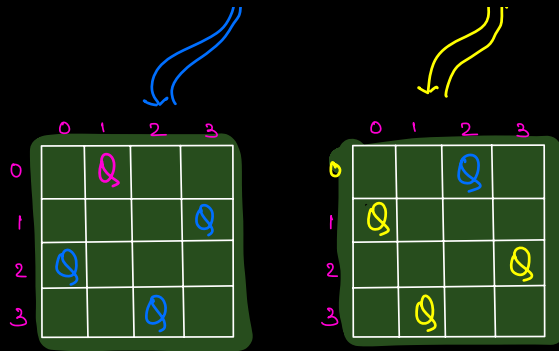
SC : 9×9

Q. N-Queen.

Given N×N matrix, print all valid possibilities of placing N queens in the given matrix.

⟹ 1 row can have only 1 queen.

```
void  nQueens ( mat[][] , N, (i) ) {
    if( i == N ) {
        print (mat)
        return ;
    3

    for( j = 0; j < N; j++ ) {
        // Check if we can place a Queen i,j
        if ( check (mat, i, j ) ) {
            mat[i][j] = 1;
            nQueens ( mat , i+1 );
            mat[i][j] = 0;
```
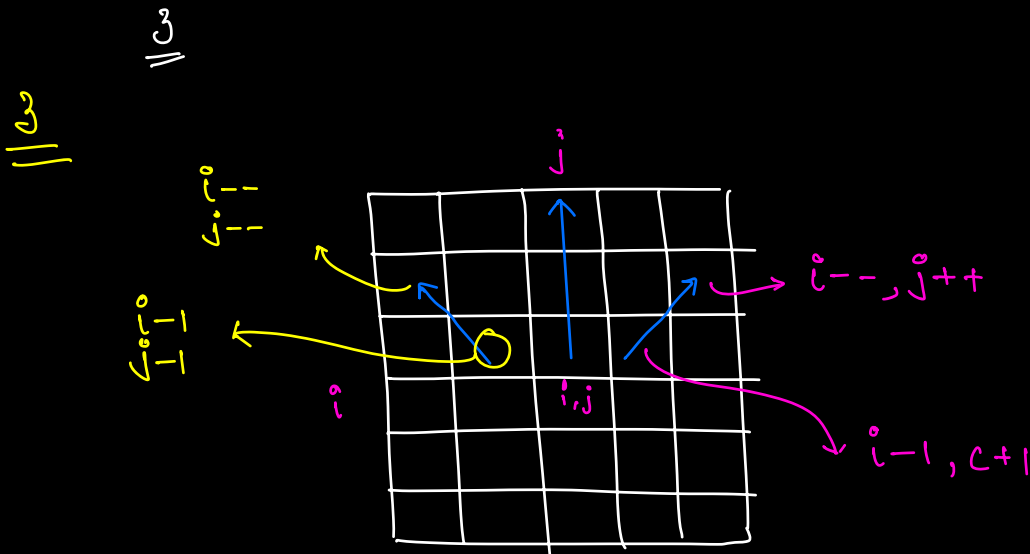
→ row index

↓ ↓
i,j

3

3

3

i--
j--

j:i-1
-1

i

j

i,j

i--, j++

i-1, c+1

```
bool   check ( mat[][], i, j )  →  O(N)

       for ( r=0; r < i; r++ ) {
              if ( mat[r][j] == 1 )           O(N)
                    return false;
       3

       r = i-1, c = j+1
       while ( r >= 0 && c < N ) {
              if ( mat[r][c] == 1 )           O(N)
                    return false;

              r--
              c++
       3

       r = i-1, c = j-1
       while ( r >= 0 && c >= 0 ) {
              if ( mat[r][c] == 1 )           O(N)
                    return false;

              r--
              c--
       3
       return true;
3

HW
→ check TC  of  N-Queens.
```
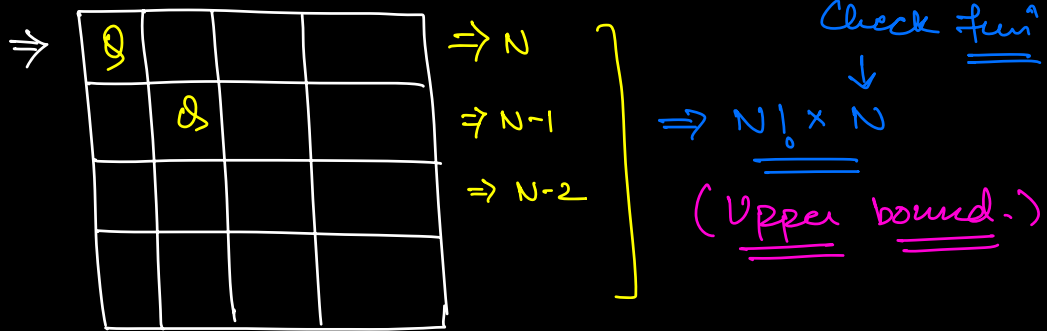
$\Rightarrow$

$\Rightarrow N$

$\Rightarrow N-1$

$\Rightarrow N-2$

Check fun^n

$\Rightarrow \underline{N! \times N}$

(Upper bound.)

Q.3 Rat in a Maze

$N \times M$
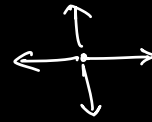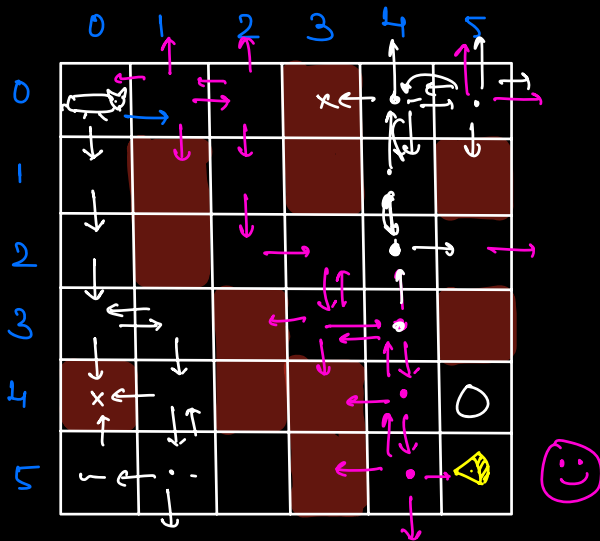
→ One cell can't be visited more than once.

(0,0)

T   L D   R (0,1)

(-1,0)   (0,-1)   (1,0)   R
  X   X   (1,1) X
         T   L   D
     0,0   (1,-1)   (2,0)   R
                          T   L   D   2,1
                              (3,0)   X
                           T   L   D   R
                         X   X   4,0   (3,1)   R
                                   X   D   X
                              T   L
                         X   3,0   4,1   R   4,2
                                   X   L D   X
                                        T   L   R
                                   X   X   5,1   S,2
                                         T   L   D   X
                                   X   X   (X)   X
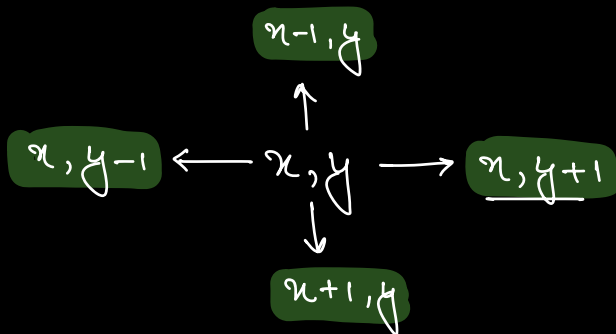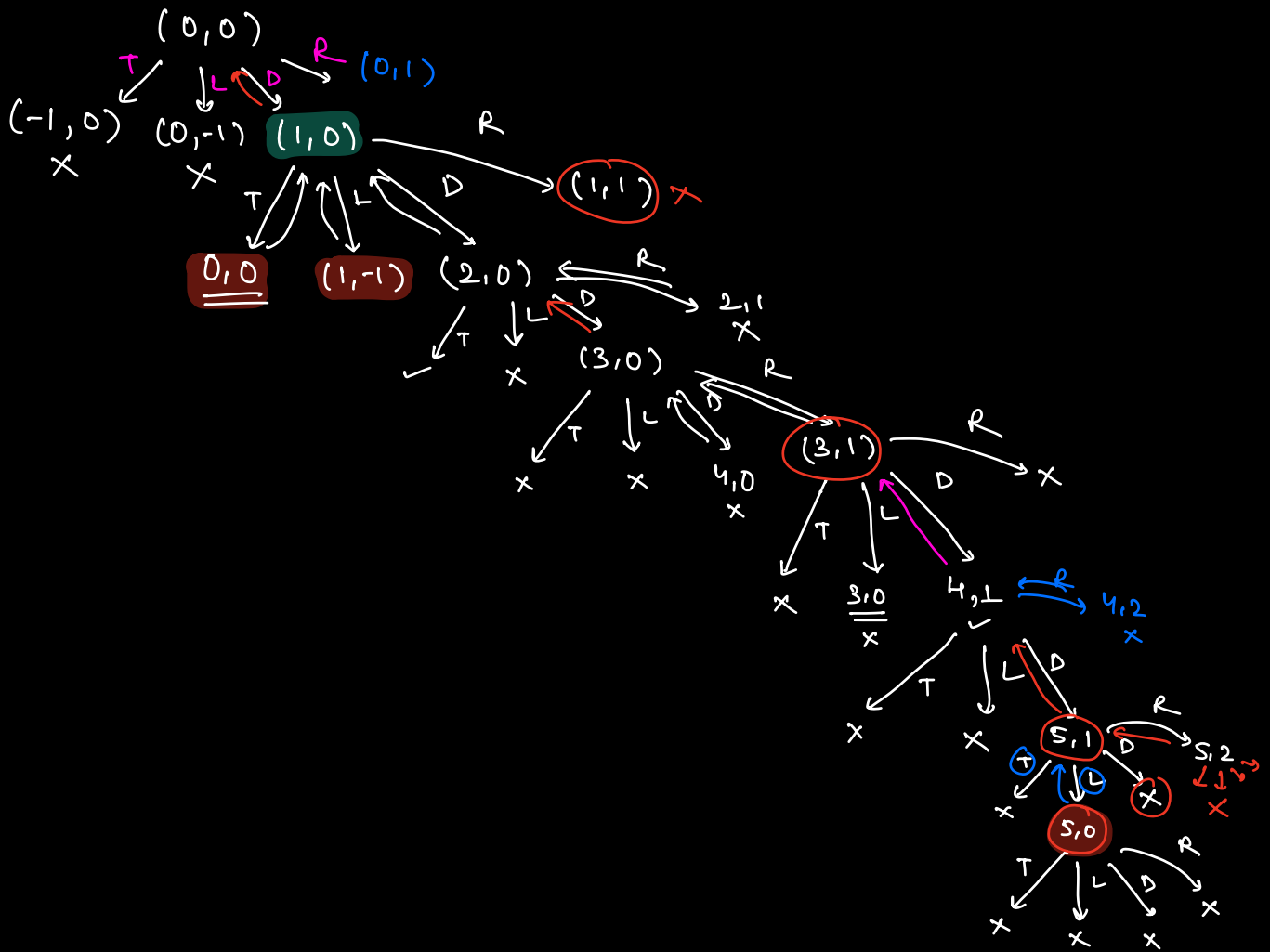   5,0
                                     T   L   D   R
                                     X   X   X   X

$$mat[i][j] = \begin{cases} 0 \Rightarrow \text{free} \\ 1 \Rightarrow \text{visited} \\ 2 \Rightarrow \underline{\text{Blocked}}. \end{cases}$$

list < pair < int , int > >

```
Void    maze (mat[][] , N, M, i,j , list <pair<int,int>> Path)
{
        if (i<0 || i>N || j<0 || j>=M) {
              return;
        }
        if ( mat[i][j] == 1 || mat[i][j] == 2 )
              return;
        if (i == N-1 && j == M-1) {
              print (Path)
              return;
        }
        if (mat[i][j] == 0) {

              mat[i][j] = (1);
              Path.add ( {i,j} );
              maze (mat, N, M, i-1, j, Path);
              maze (mat, N, M, i, j-1, Path)
              maze (mat, N, M, i+1, j , Path)
              maze (mat, N, M, i, j+1, Path)

              mat[i][j] = 0;
              Path.remove ();

                    3

      3

                TC :  O(N×M)
                      ↳Each cell  is  visited exactly
                      once.

                SC:    O(NM)              ——— * ———
```