Todays Content:

- → Sudoko → (Searching in 2D Matrix)
- → Rat in a maze ↳ Search in 2D Matrix

Sudoko: Given 9*9 matrix fill Sudoko

	0	1	2	3	4	5	6	7	8
0	5	3			7				
1				1	9	5			
2		9	8					6	
3	8		2		6				3
4	4			8		3			1
5	7		3		2				6
6		6					2	8	
7				4	1	9			5
8			1		8			7	9

Q) mat[9][9] valid sudoko, fill sudoko

- → In a row data cannot repeat
- → In a column data cannot repeat
- → In all 3*3 boxes data cannot repeat
- → We can only fill 1-9

cells: Start point of box →

x	y		x-x%3	y-y%3
1,	4	→	0	3
4	8	→	3	6
7	1	→	6	0
8,	5	→	6	3

	0	1	2	3	4	5	6	7	8
0	5	3	4	2	7	6	1	9	8
1	6	2		1	9	5	X	X	X
2		9	8					6	
3	8		2		6				3
4	4			8		3			1
5	7		3		2				6
6		6					2	8	
7				4	1	9			5
8			1		8			7	9

(0,0) → (0,1) → (0,2) → (0,3) → (0,4) → (0,5) → (0,6) → (0,7)
4 2 6 1 9

(0,8) → (1,0) $\xrightarrow{2}$ (1,1) → (1,2) → (1,3) → (1,4) → (1,5) → (1,6) $\xrightarrow{3}$ (1,7) $\xrightarrow{4}$ (1,8)
8 7 6 pre pre pre

(1,6) $\xrightarrow{4}$ (1,7) $\xrightarrow{3}$ (1,8)

(1,0) $\xrightarrow{6}$ (1,1) $\xrightarrow{2}$ (1,2)

	0	1	2	3	4	5	6	7	8
0	0 5	1 3	2 6	3 -	4 7	5	6	7	8 2
1	9	10 4	11	12 1	13 9	14 5	15	16 =	17
2	18	19 9	20 8	21	22	23	24 7	25 6	26
3	27 8	28	29 2	30	31 6	32 =	33	34	35 5
4	4			8		3			1
5	7		3		2				6
6		6					2	8	
7				4	1	9			5
8			1		8			7	80 = 9

→ Not possible

$9 \times 9 \times 9 - 9 \cdot 9 \to 9^{9}$

$9! \times (9!)^{9}$

→ 32 → row : 32/9 → 3
↳ col : 32 % 9 → 5

16 → row : 16/9 → 1
col : 16 % 9 → 7

27 → row : 27/9 → 3
col : 27 % 9 = 0

2D index → r, c

1D: $x \longrightarrow x/9 \quad x\%9$

Back Tracking :

↳ parameters : mat[][], x, → curr index is 1D

↳ function calls : At mat[i][j] → 1, 2, 3, 4 … 9
↳ 9 function calls.

↳ Void

↳ How to know which cell data is filled, not filled
↳ If cell is empty data is 0

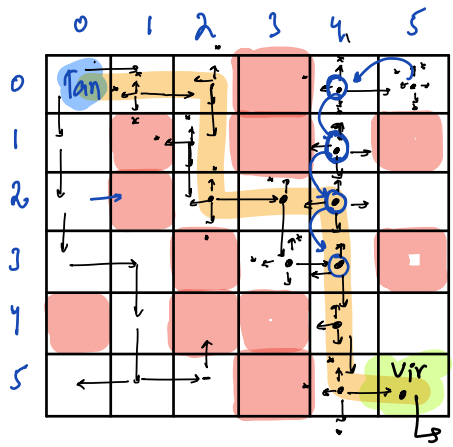
```
void printSudoko( int mat[][], int n){ TC→ SC→ O(81)
    if(n==81) {                                  ↳ O(1)
        iterate & print mat[]   } Base condition
        return                                   ↳ Todo
    }
    int r= n/9, c= n%9
    // we need to fill mat[r][c]
    if( mat[r][c] != 0) { // goto next cell
        printSudoko( mat, n+1)
    }
    else {
        for(int i= 1; i <= 9; i++){   → Wether we can place data i, in cell mat[r][c]?
            if ( valid( mat, r, c, i))){
                mat[r][c] = i          → make changes
                printSudoko( mat, n+1)  index
                mat[r][c] = 0          → revert changes
            }
        }
    }
}
```

```
bool valid( int mat[][], int r, int c, int d){
    for(int j=0; j<9; j++){
        if( mat[r][j] == d) {return false} -> To check r^th row
        if( mat[j][c] == d) {return false} -> To check c^th col
    }
    int x = r - r%3, y = c - c%3
    for(int i=x; i<x+3; i++){
        for(int j=y; j<y+3; j++){
            if( mat[i][j] == d) {return false}
        }
    }
    return True
}
```

10 -> 15

10 -> 25:

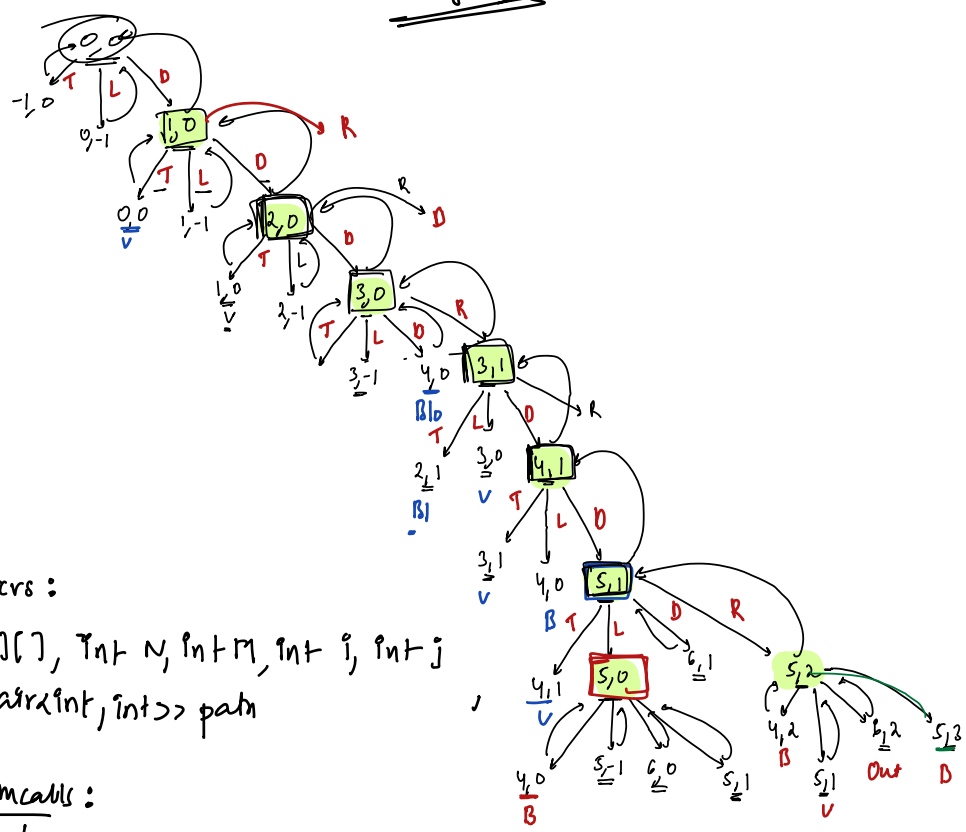
Rat in Maze :



↳ always at BR

1) cell ← ↑ ↓ →

2) Every cell we can un once



Parameters :

mat[][], int N, int M, int i, int j
list<pair<int, int>> path

↳ function calls :

↳ 4 calls :

$(i-1, j)$

$(i, j-1) \leftarrow (i, j) \rightarrow (i, j+1)$

$(i+1, j)$

→ mat[i][j] = 2 → Blocked

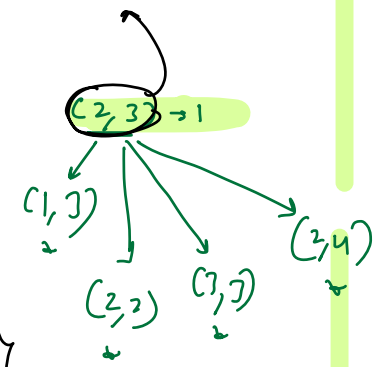
↳ mat[i][j] = 1 → visited

↳ mat[i][j] = 0 → unvisited

// Print the path :

```
void TanVsVir (int mat[][], int N, int M, int i, int 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;
        list.add( make_pair(i,j))
        TanVsVir(mat, N, M, i-1, j, path)
        TanVsVir(mat, N, M, i+1, j, path)
        TanVsVir(mat, N, M, i, j-1, path)
        TanVsVir(mat, N, M, i, j+1, path)
        list.remove()
    }
}
```

Indirectly acting as a check part



→ Recursion one / Back tracking

↓

Spend 1hr/2hr → revise notes

→ Dynamic Programming : 7/8 Sessions

24th July → Dp1 7th Aug → Dp3 → 3 more problem solving on all previous topic

31st July → Dp2 14th Aug → Gp1 → 15th → 17th → 19th