Todays Content:

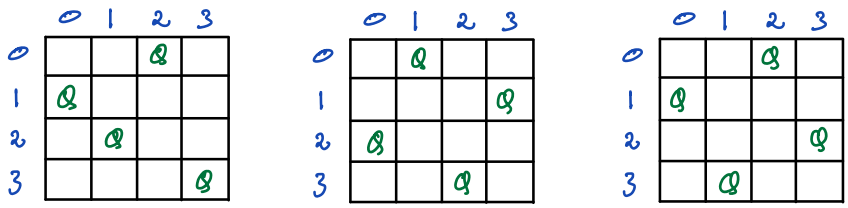
- N-Queens ✓
- Sudoku ✓
- Subset sum 2nd Version ✓

Q) Given N*N mat[], print all valid placement of N Queens such that no queen can kill other queen ↳ Backtracking

Note: If 2 queen belong to same row/column/diagnol they will kill

Ex: N = 4

invalid



parameters: mat[][], N, i → indicates row number

Subproblems: Choice : Columns: 0, 1, 2, 3, .... N-1

Return type: void

valid: mat[i, j]

→col: Iterate on $j^{th}$ col & check

→leftd: (3 2) → (2, 1) → (1 0) → (0, -1)

(i, j) → (i-1, j-1) .....

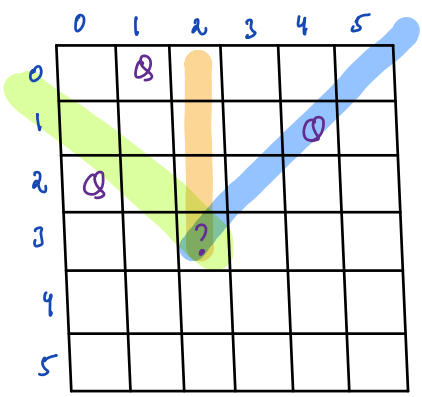
→rightd: (3, 2) → (2, 3) → (1, 4) ....

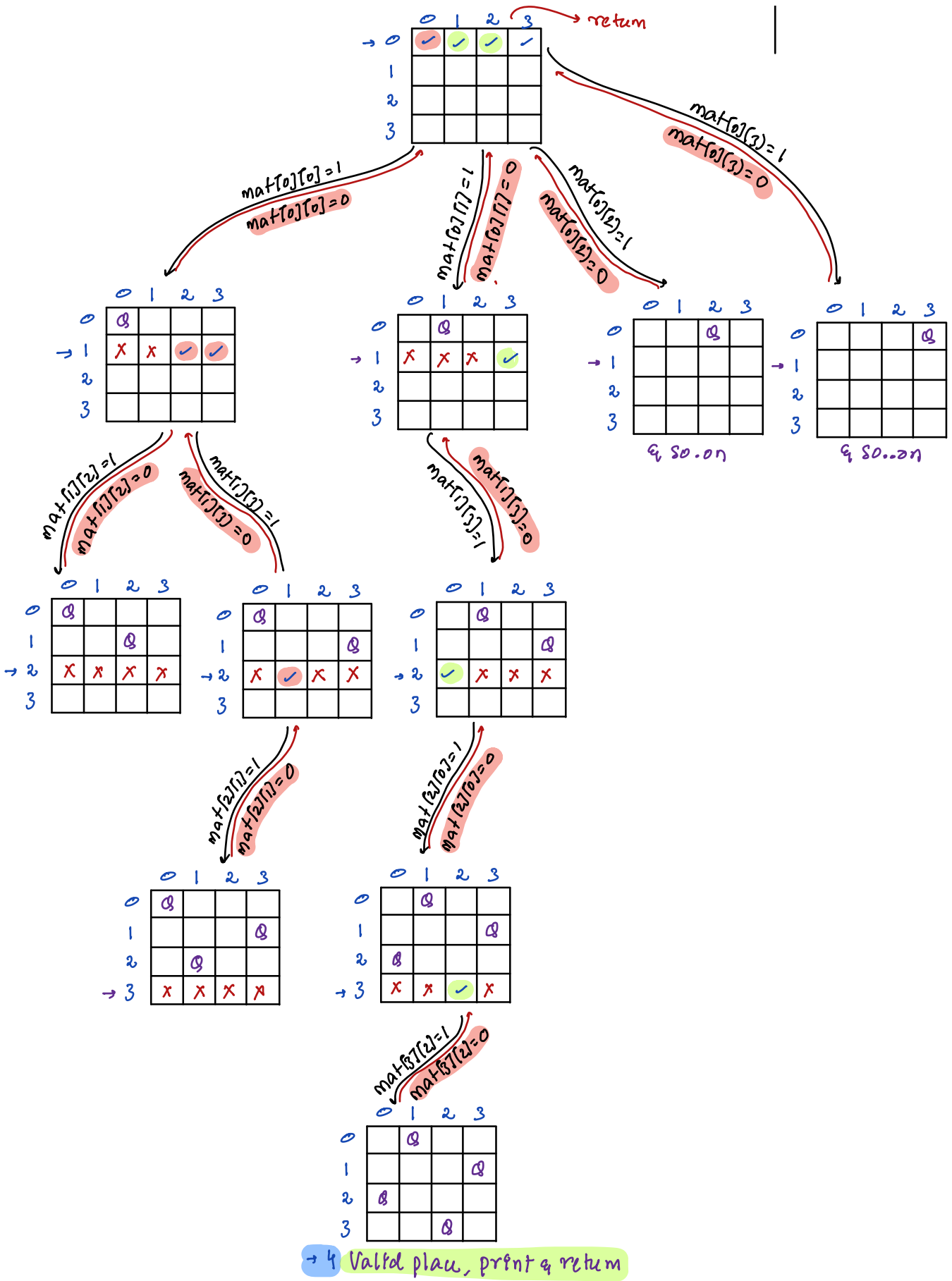
(i, j) → (i-1, j+1) . . . .

```
void NQueens( int mat[][], int N, int i ){
    if( i == N ){                          TC: ≈ O(N!)
        print(mat[][])
        return 1
    }
    // At i^th row choice?
    j = 0; j < N; j++){
        // We want place queen at i^th & j^th col
        // If we can place queen at mat[i,j] such that queen don't kill
        if( valid(mat, i, j) == True){
            mat[i][j] = 1
            NQueens(mat, N, i+1)
            mat[i][j] = 0
        }
    }
}
```

TC: O(N) ? TODO

→ col, leftd, rigd




return
mat[0][0]=1
mat[0][0]=0
mat[0][1]=1
mat[0][1]=0
mat[0][2]=1
mat[0][2]=0
mat[0][3]=1
mat[0][3]=0
mat[1][2]=1
mat[1][2]=0
mat[1][3]=1
mat[1][3]=0
mat[1][1]=1
mat[1][1]=0
mat[2][1]=1
mat[2][1]=0
mat[2][0]=1
mat[2][0]=0
mat[3][2]=1
mat[3][2]=0
& so..on
& so..on
→ 4 Valid place, print & return

20)

mat[9, 9]

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

→ Fill sudoko : We will have Sudoko

: mat[i,j] = 0, Empty slot, needs filled

: mat[i,j] != 0, Non Empty Slot

: Every row, : Every col, : Every cube } Should contain all elements from [1-9]

: Duplicates not allowed

1D ⟶ 2D

i ⟶ (r : i/9  c = i%9)

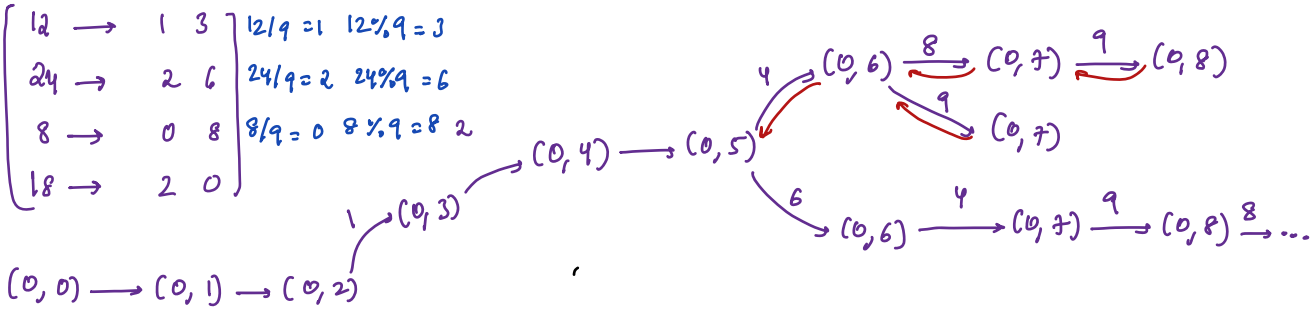
| | | |
|---|---|---|
| 12 → | 1 | 3 |
| 24 → | 2 | 6 |
| 8 → | 0 | 8 |
| 18 → | 2 | 0 |

12/9 = 1  12%9 = 3

24/9 = 2  24%9 = 6

8/9 = 0  8%9 = 8

2




If data already filled on a cell, it's data is not effected.

```
Sudoko(0, 3) {
    if(    ) {
    }
    else {
        i = 1 ... 9) {
            Sudoko(0, 4)
        }
    }
}
```

```
Sudoko(0, 4) {
    if(    ) {
        Sudoko(0, 5)
    }
    else {
        i = 1 ... 9) {
        }
    }
}
```

```
Sudoko(0, 5) {
    if(    ) {
    }
    else {
        i = 1 ... 9) {
            ---- all choices are done
        }
    }
}
```

parameters: mat[][], i → {curr cell pos we are at}

Subproblems: choices: {1, 2, .. 9}

return type: void

```
void Sudoko(int mat[][], int i){   TC: 9^81   SC: O(81)
    if(i==81){                      ↳ upper bound
        print(mat[][])              ↳ approx
        return
    }
    // at pos i: 1D index
    int r= i/9, c= i%9 // cell at mat[r][c]
    if(mat[r][c] != 0){ //already filled
        sudoko(mat, i+1) // goto next fill
    }
    else // not filled
        for(j=1; j<=9; j++){
            // at cell mat[r][c] = j
            //check if we can place j at mat[r][c]
            if(valid(mat, r, c, j) == true){
                mat[r][c] = j
                sudoko(mat, i+1)
                mat[r][c] = 0
            }
        }
    }
}
```

valid: TODO
{ : check in row / col / 9 cube
there should not be any
j }

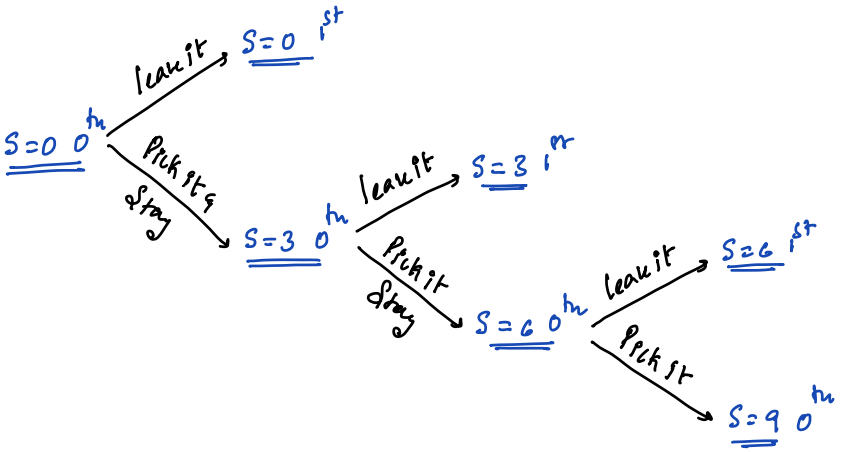
38) Count no: of subsets with sum = k

a) all ele are +ve

b) an ele can be picked as many times as needed

| | 0 | 1 | 2 | 3 | |
|---|---|---|---|---|---|
| ar[4] = { | 3 | 6 | 2 | 5 } | k = 14 |

↳ {3 3 6 2} {2 2 5 5} {3 3 3 3 2}




Note: ele has 2 choices, leave, pick & Stay

```
int countSubSum( int ar[], int N, int k, int i, int sum ){
    if( sum > k ){ return 0; } // To avoid ∞ loops
    if( i == N ){
        if( sum == k ){ return 1; }
        else { return 0; }
    }
    // At i th index choices
    sum = sum + ar[i] // Picking i th ele sum gets updated
    int c1 = countSubSum(ar, N, k, i, sum) // subsets with sum = k
    sum = sum - ar[i] // leave i th ele, sum we are reducing
    int c2 = countSubSum(ar, N, k, i+1, sum) // subsets with sum = k
    return c1 + c2
}
```

undoing updates on sum