Todays Content

a) Intro
b) All subsets
c) All Combinations
d) All Solutions

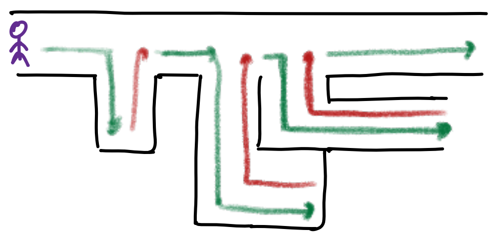
Back Tracking: Generating all solutions using recursion → Backtracking

↳ a. all subsets

b. all combinations

c. all ways

Maze:



Idea: While generating all solutions, choose path but we cannot go any further, backtrack & take another path.

1Q) Given N, print all N digit numbers formed only by 1, 2 in increasing order

N = 2 :

				dec
1	1	0	0	0
1	2	0	1	1
2	1	1	0	2
2	2	1	1	3

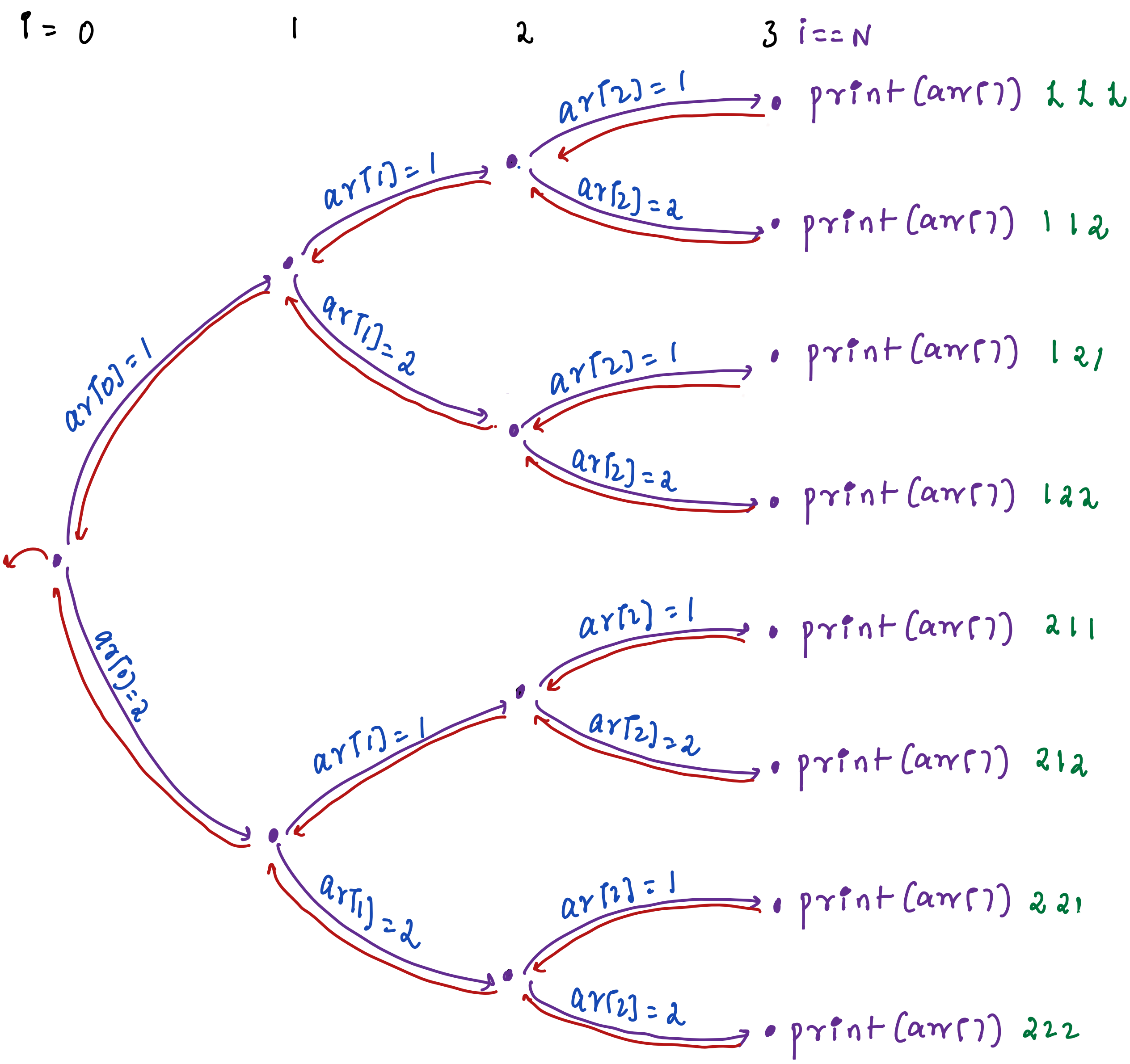
N = 3 :

1	1	1
1	1	2
1	2	1
1	2	2
:		

Idea:

1. Using bitmanipulation : $1 \rightleftarrows 0$ $2 \rightleftarrows 1$
2. Using queues :
3. Using backtracking:

$$\underline{\text{Tracing}}\quad N = 3 : ar[3] = \{\overset{0}{2} \quad \overset{1}{2} \quad \overset{2}{2}\}$$ To store 3 digit number



fun():

parameters: What all parameters we need to pass
ar[N], i: {curr index}, N: {size of arr}

Subproblems: How many choices
2 choices

Returntype: void

```
void printAll(int ar[], int i, int N){
    if( i==N){ // Stop:
        print entire arr[]
        return;
    }
    // At i^th index choices
    ar[i] = 1;
    printAll(ar, i+1, N)
    ar[i] = 2;
    printAll(ar, i+1, N)
    return;
}
main(){
    int ar[N]
    printall(ar, 0, N);
}
```

```
roughcode{
    if(no choice){
        do something
        return
    }
    Choices call
    return
}
```

```
    0  1
ar[2] = { 2  2 }
main(){
    int ar(2)
    printall(ar, 0, N);
}
```

```
void printAll(int arr[], int i=0, int N=2){
1. if(i == N){ print(arr[]) return }
2. ar[i] = 1;
3. printAll(ar, i+1, N)
4. ar[i] = 2;
5. print All (ar, i+1, N)
6. return
}
```

```
void printAll(int arr[], int i=1, int N=2){
1. if(i == N){ print(arr[]) return}
2. ar[i] = 1;
3. printAll(ar, i+1, N)
4. ar[i] = 2;
5. print All (ar, i+1, N)
6. return
}
```

```
void printAll(int arr[], int i=1, int N=2){
1. if(i == N){ print(arr[]) return}
2. ar[i] = 1;
3. printAll(ar, i+1, N)
4. ar[i] = 2;
5. print All (ar, i+1, N)
6. return
}
```

```
void printAll(int arr[], int i=2, int N=2){
1. if(i == N){ print(arr[]) return}
2. ar[i] = 1;
3. printAll(ar, i+1, N)
4. ar[i] = 2;
5. print All (ar, i+1, N)
6. return
}
```

```
void printAll(int arr[], int i=2, int N=2){
1. if(i == N){ print(arr[]) return}
2. ar[i] = 1;
3. printAll(ar, i+1, N)
4. ar[i] = 2;
5. print All (ar, i+1, N)
6. return
}
```

```
void printAll(int arr[], int i=2 int N=2){
1. if(i == N){ print(arr[]) return}
2. ar[i] = 1;
3. printAll(ar, i+1, N)
4. ar[i] = 2;
5. print All (ar, i+1, N)
6. return
}
```

```
void printAll(int arr[], int i=2, int N=2){
1. if(i == N){ print(arr[]) return}
2. ar[i] = 1;
3. printAll(ar, i+1, N)
4. ar[i] = 2;
5. print All (ar, i+1, N)
6. return
}
```

Output:

1 1

1 2

2 1

2 2

Q) Given arr[N] elements, count of subsequence with sum = k

↳ Need not be continous

↳ Order of index maintained

Ex1:

$$arr[3] = \{ \overset{0}{5} \quad \overset{1}{7} \quad \overset{2}{2} \}$$

k = 7, Subsequences = {5, 2} {7} : return 2

Idea1: Generate all subsequence sums & compare == k

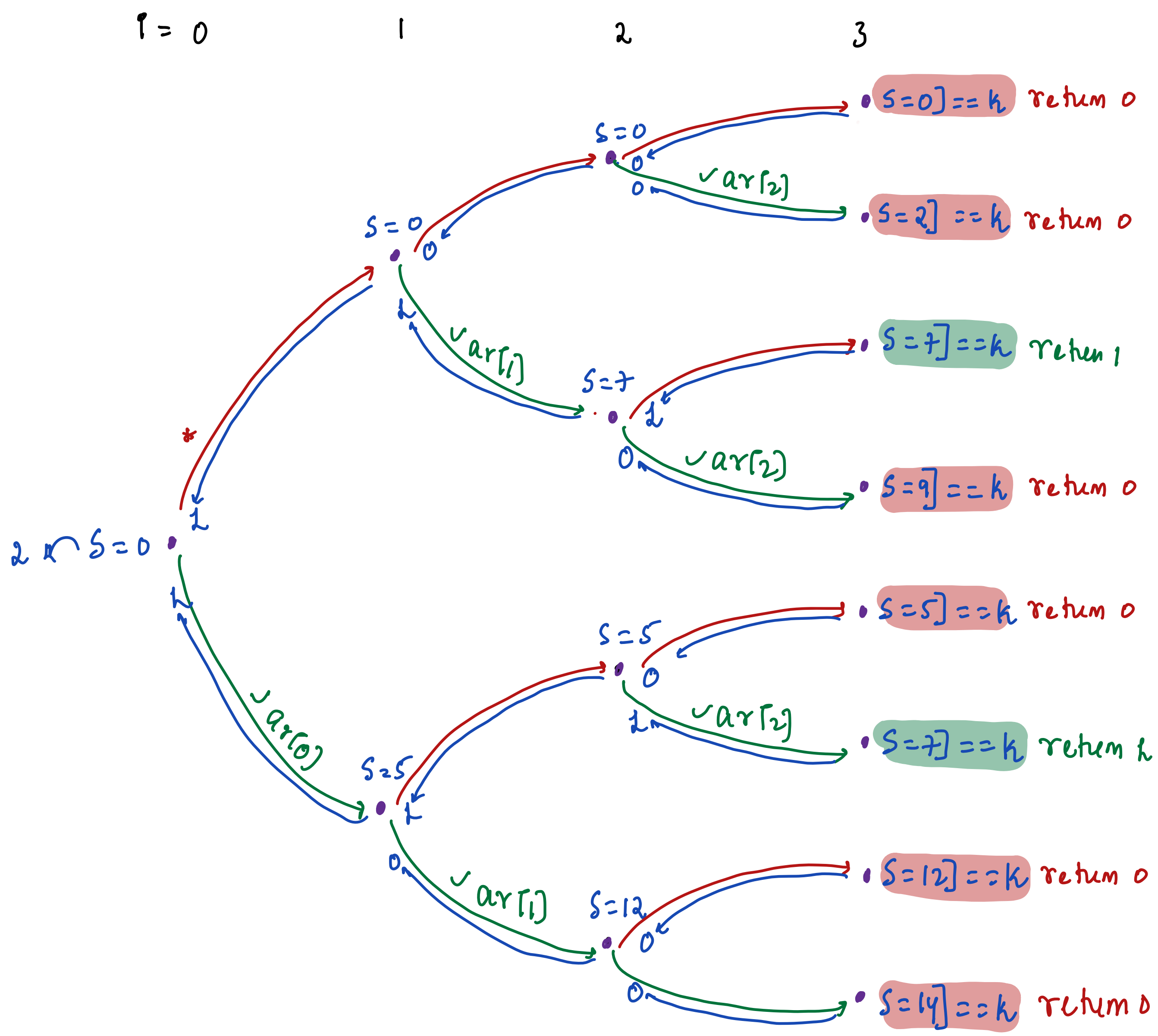
TC: $O(2^N * N)$ SC: $O(N)$

Ideas: 1. Hashmap a+b = k: {subsequence can contain > 2 ele}

2. Using psum[]: Range: {Elements need not be continous}

Idea2: Using backtracking

$$arr[3] = \{ \overset{0}{5} \quad \overset{1}{7} \quad \overset{2}{2} \} \quad k = 7$$



func():

parameters: arr[], i, N, sum, k

Subproblems: 2 choices, pick it, leave it

Returntype: int

```
int CountSubSum(int arr[], int i, int N, int sum, int k){     TC: O(2^N)
                                                              SC: O(N)
    if(i == N){
        if(sum == k){return 1}                               roughcode{
    } else return 0                                              if(no choice){
                                                                     do something
                                                                     return
    // At i^th index choices                                     }
                                                                 Choices call
            // we don't pick i^th sum:                           return
    int l = CountSubSum(arr, i+1, N, sum, k)                  }
            // We pick i^th ele: sum + arr[i]
    int r = CountSubSum(arr, i+1, N, sum+arr[i], k)
    return l+r;
}
```

3Q) Given N*N mat[], print all valid placement of N Queens such that no queen can, kill other queen :

<u>Note</u>: If a queen belong to same row/column/diagnol they will kill

Ex:

N=4

	0	1	2	3
0	Q			
1			Q	
2				Q
3		Q		

Not valid

	0	1	2	3
0		Q		
1				Q
2	Q			
3			Q	

Valid

	0	1	2	3
0			Q	
1	Q			
2				Q
3		Q		

Valid

func():

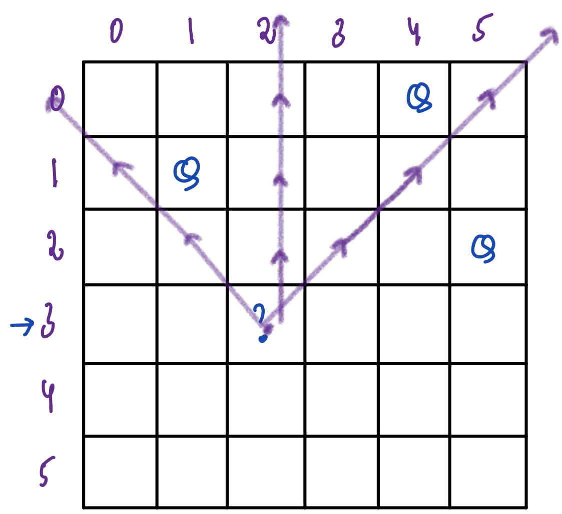
parameters: mat[N][N], i, N

Subproblems: All Columns in i^{th} row are my choices :

	0	1	2	.	.	N-1
i^{th}	Q	Q	.		.	Q

Returntype: void

```
void NQuens(int mat[][], int i, int N){   // Queens to place
    if(i == N){
        print entire mat[][]
        return;
    }
    // At ith row:
    for(int j=0; j<N; j++){
        // Try place at [i, j] = Q;
        if(valid(mat, i, j) == True){
            mat[i][j] = 1;
            NQueens(mat, i+1, N)
            mat[i][j] = 0;
        }
    }
    return;
}
```



valid : mat(i, j) : Check at mat(3, 2)

row : No need, we only place 1, Queen

col : $(3,2) \to (2,2) \to (1,2) \to (0,2) \to (-1,2)$

left : $(3,2) \to (2,1) \to (1,0) \to (0,-1)$

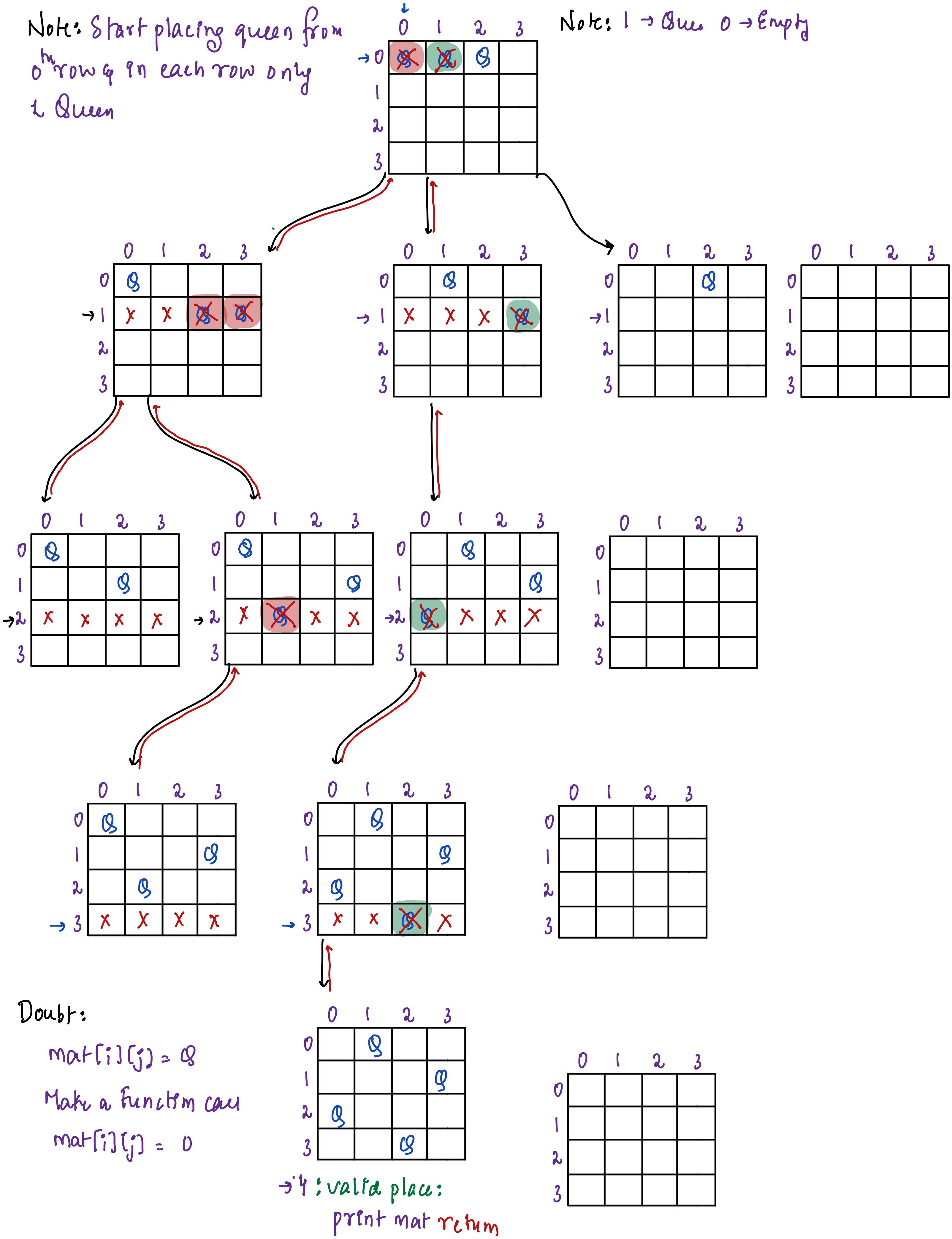
$[x,y] \to [x-1, y-1]$

rigd : $(3,2) \to (2,3) \to (1,4) \to (0,5) \to (-1,6)$

$[x\ y] \to [x-1, y+1]$

Note: Start placing queen from 0^{th} row & in each row only 1 Queen

Note: 1 → Queen 0 → Empty



Doubt:

mat[i][j] = Q

Make a function call

mat[i][j] = 0