backtracking

25 September 2023 22:59

BACKTRACKING

Types of Backtracking

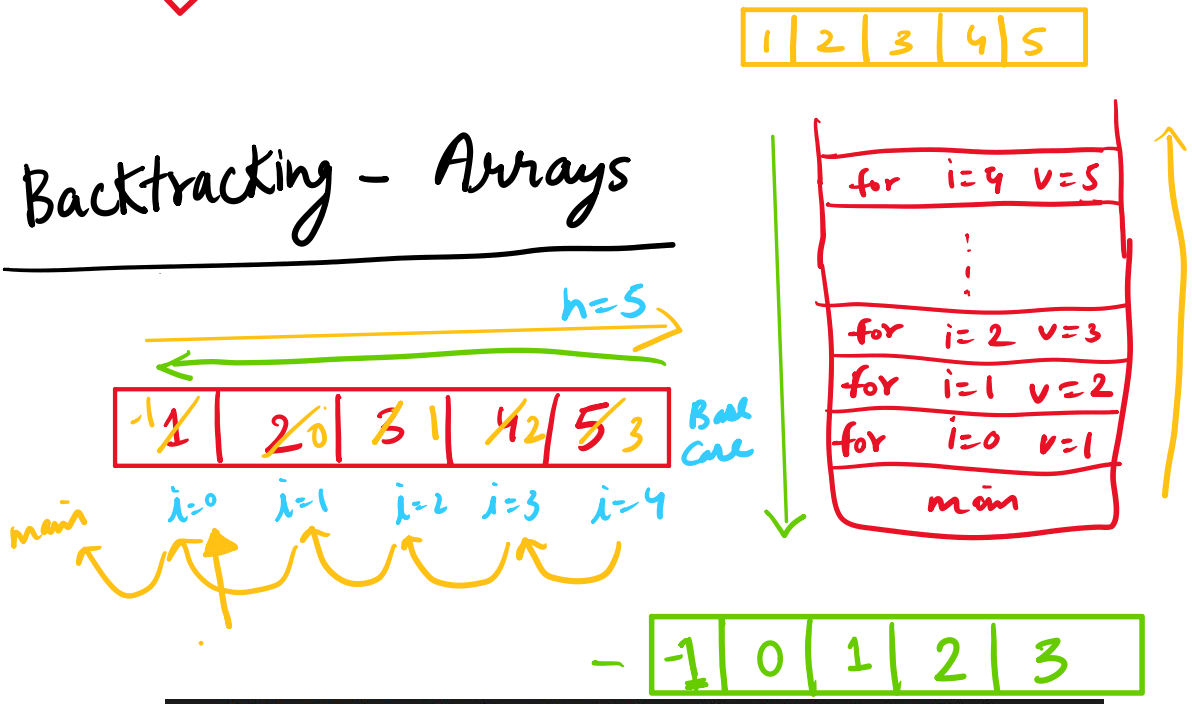
- Decision
- Optimization
- Enumeration

Yes / No Backtracking

From starting to ending how many paths are there.

Shortest Path or decision making

Backtracking - Arrays

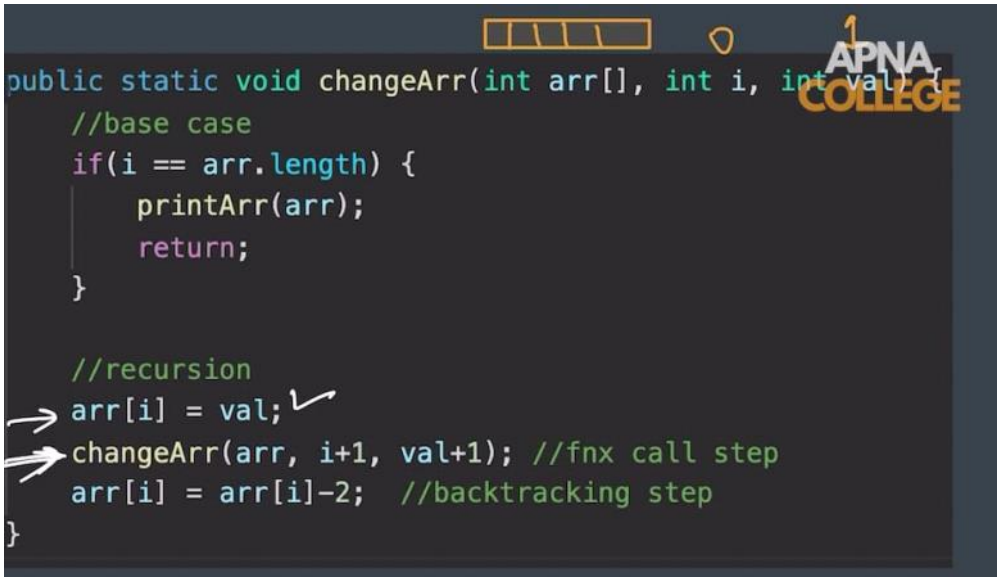


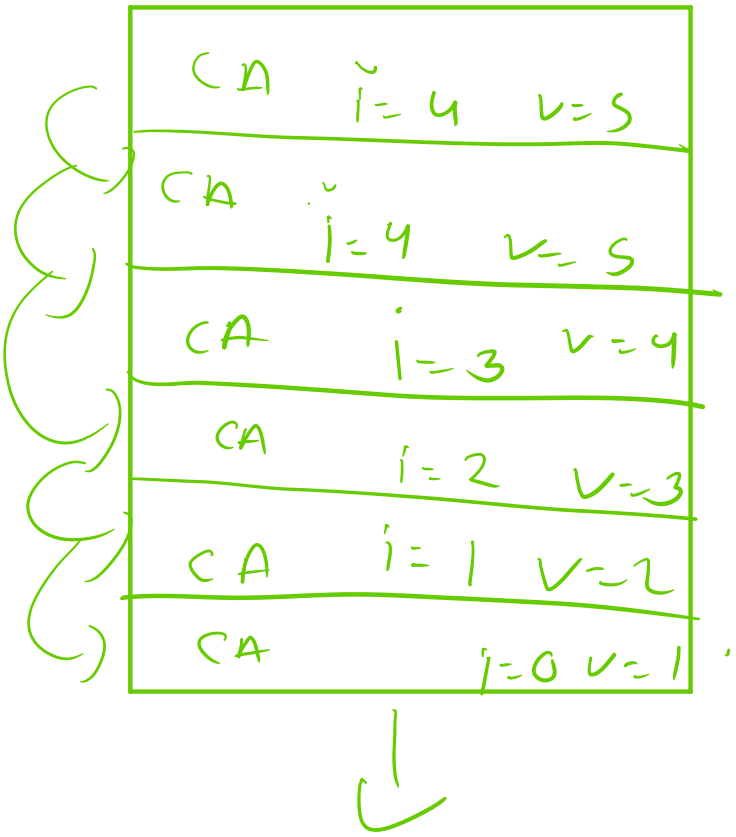

```java
public static void changeArr(int arr[], int i, int val) {
    //base case
    if(i == arr.length) {
        printArr(arr);
        return;
    }
    //recursion
    arr[i] = val;
    changeArr(arr, i+1, val+1);
    arr[i] = arr[i] - 2;
}
```

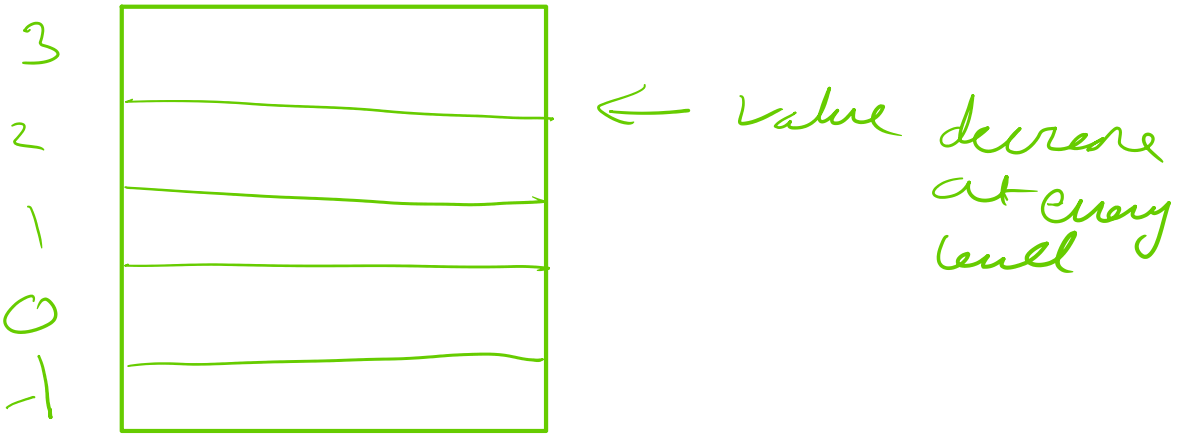
```java
public static void changeArr(int arr[], int i, int val) {
    //base case
    if(i == arr.length) {
        printArr(arr);
        return;
    }
    //recursion
    arr[i] = val;
    changeArr(arr, i+1, val+1);
    arr[i] = arr[i] - 2;
}
```

```java
public static void printArr(int arr[]){
    for(int i=0; i<arr.length; i++) {
        System.out.print(arr[i]+" ");
    }
    System.out.println();
}

Run | Debug
public static void main(String args[]) {
    int arr[] = new int[5];
    changeArr(arr, 0, 1);
    printArr(arr);
}
```







* Find Subsets

Find & print all subsets of a given substring.

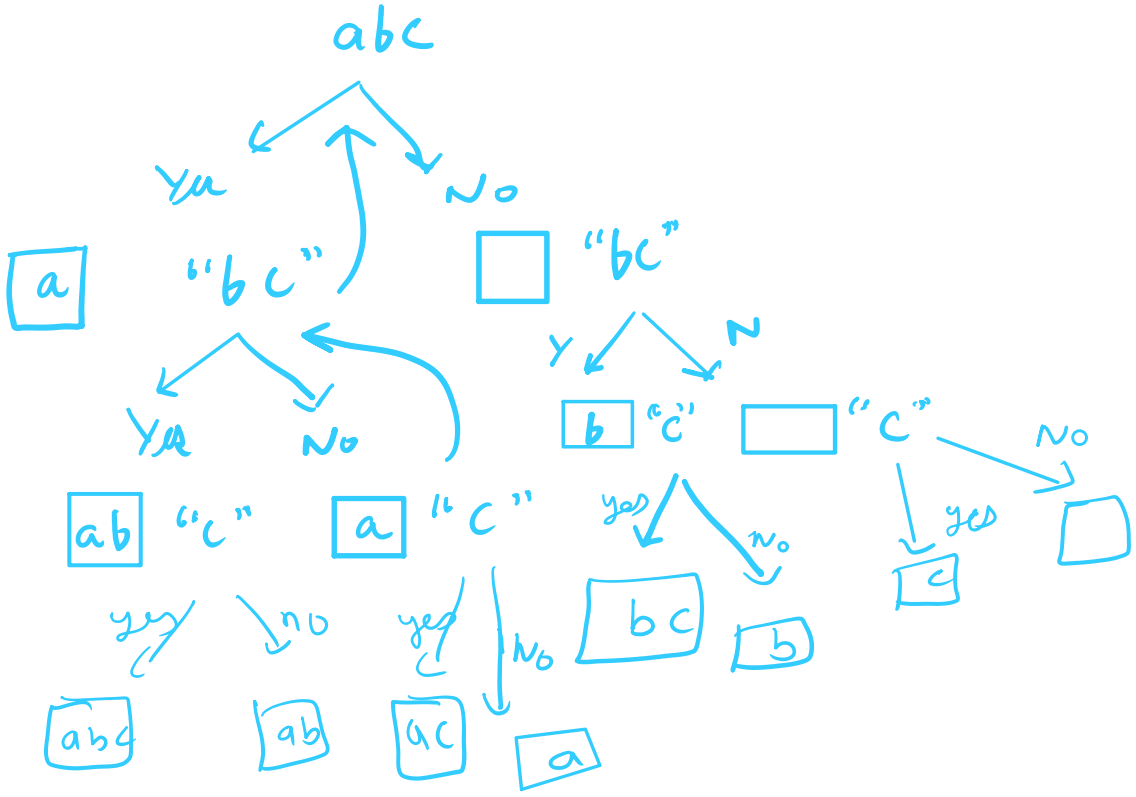
"abc"

a, b, c, ab, bc, ac, abc, " "

a, b, c, ab, bc, ac, abc,

& = subsets

↓
empty set
(null set)
ϕ



find subsets (str, ans){

BC
→ Print ans

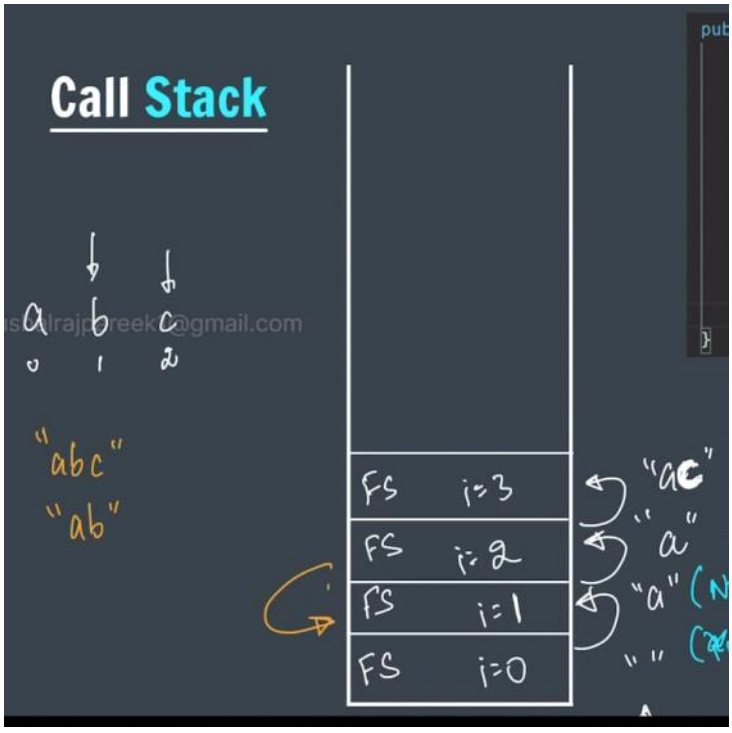
Select one time → ans + str.charAt(i)

don't select → ans

} recursive calls

```java
public static void findSubsets(String str, String ans, int i) {
    //base case
    if(i == str.length()) {
        System.out.println(ans);
        return;
    }

    //Yes choice
    findSubsets(str, ans+str.charAt(i), i+1);
    //No choice
    findSubsets(str, ans, i+1);
}
```



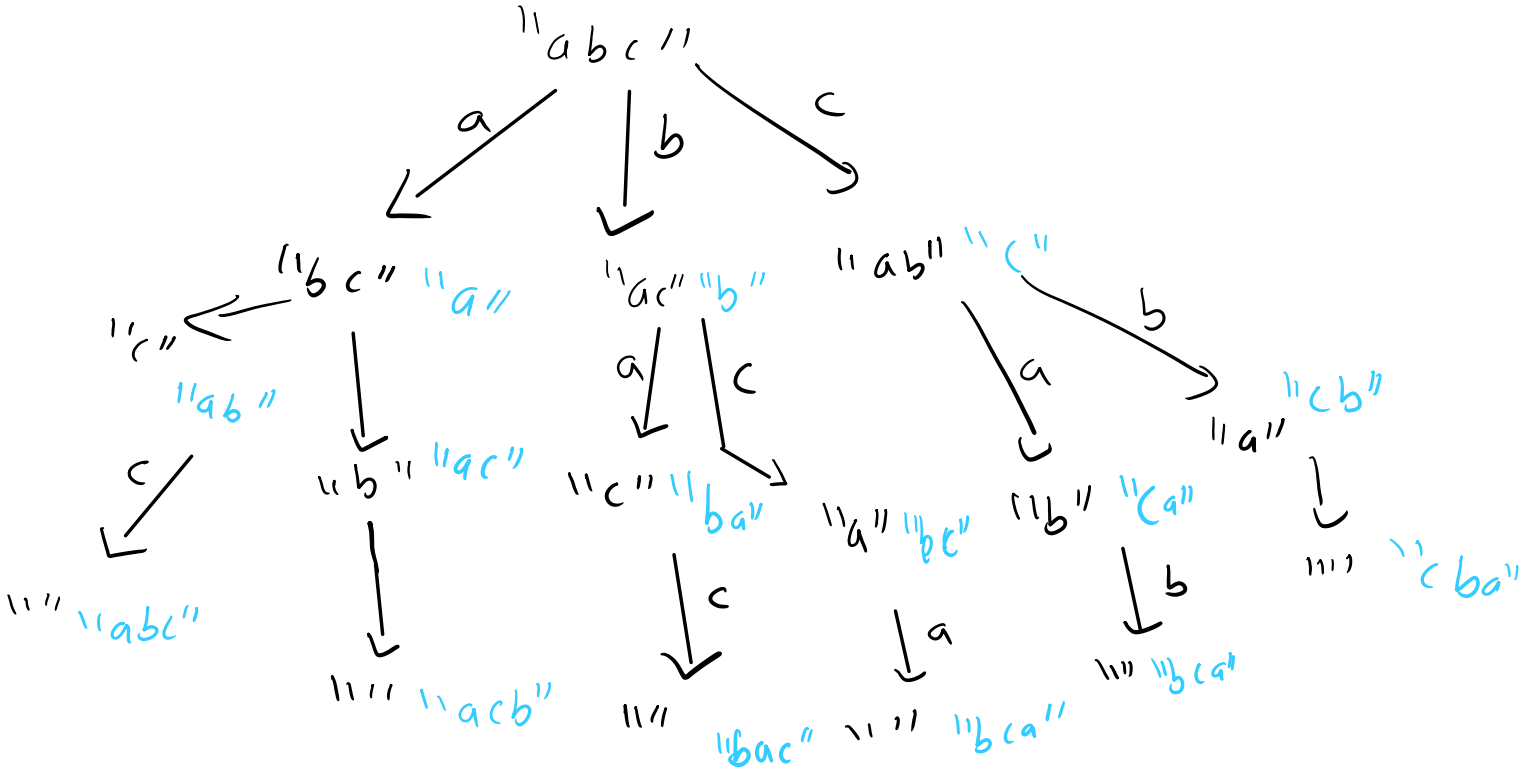
You can draw stack just follow recursion tree.

Find Permutations

Find & print all permutations of a String.

"abc"

abc, acb, bac, bca, cab, cba

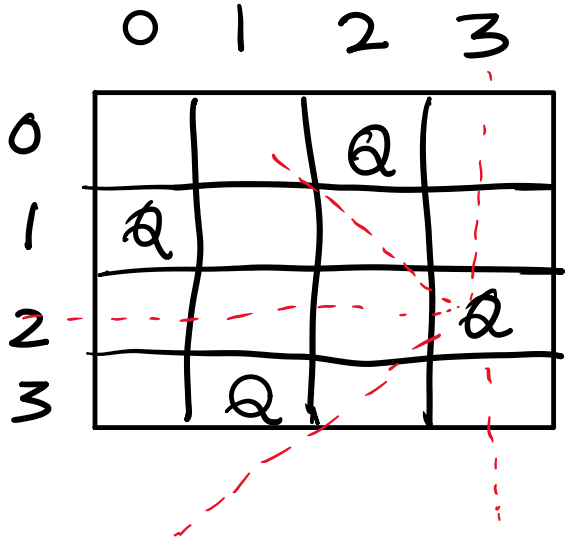


```java
public static void findPermutation(String str, String ans) {
    //base case
    if(str.length() == 0) {
        System.out.println(ans);
        return;
    }
    //recursion
    for(int i=0; i<str.length(); i++) {
        char curr = str.charAt(i);
        //"abcde" => "ab" + "de" = "abde"
        String NewStr = str.substring(0, i) + str.substring(i+1);
        findPermutation(NewStr, ans+curr);
    }
}
```

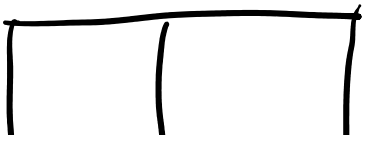
N-Queens

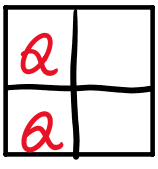
Place N queens on an N×N chessboard such that no 2 queens can attack each other

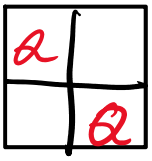
N=4

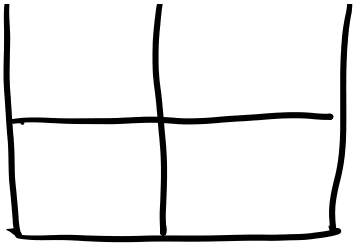


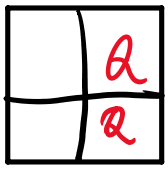
N=2












```java
public static void nQueens(char board[][], int row) {
    //base
    if(row == board.length) {
        printBoard(board);
        return;
    }
    //column loop
    for(int j=0; j<board.length; j++) {
        if(isSafe(board, row, j)) {
            board[row][j] = 'Q';
            nQueens(board, row+1); //function call
            board[row][j] = 'x'; //backtracking step
        }
    }
}
```

```java
public static void printBoard(char board[][]) {
    System.out.println("-------- chess board ---------");
    for(int i=0; i<board.length; i++) {
        for(int j=0; j<board.length; j++) {
            System.out.print(board[i][j] + " ");
        }
        System.out.println();
    }
}
```



```java
//diag right up
for(int i=row-1, j=col+1; i>=0 && j<board.length; i--,j++) {
    if(board[i][j] == 'Q') {
        return false;
    }
}
```

```java
public static boolean isSafe(char board[][], int row, int col) {
    //vertical up
    for(int i=row-1; i>=0; i--) {
        if(board[i][col] == 'Q') {
            return false;
        }
    }
```

```java
public static void main(String args[]) {
    int n = 4;
    char board[][] = new char[n][n];
    //initialize
    for(int i=0; i<n; i++) {
        for(int j=0; j<n; j++) {
            board[i][j] = 'x';
        }
    }
```

```java
//diag left up
for(int i=row-1, j=col-1; i>=0 && j>=0; i--,j--) {
    if(board[i][j] == 'Q') {
        return false;
    }
}
```


$T(n) = 1 \text{ Queen place} * T(n-1) + \text{isSafe}()$

$\downarrow$

$O(n)$

$T(n) = n * T(n-1) + \text{isSafe}()$

Time Complexity [O(n!)]

* N Queens - Time Complexity

N - Queens Total ways count :-

```java
public static void nQueens(char board[][], int row) {
    //base
    if(row == board.length) {
        // printBoard(board);
        count++;
        return;
    }
```

Print Exactly one Pair :-

```java
public static boolean nQueens(char board[][], int row) {
    //base
    if(row == board.length) {
        // printBoard(board);
        count++;
        return true;
    }
    //column loop
    for(int j=0; j<board.length; j++) {
        if(isSafe(board, row, j)) {
            board[row][j] = 'Q';
            if(nQueens(board, row+1)){
                return true;
            }
            board[row][j] = 'x'; //backtracking step
        }
    }

    return false;
}
```

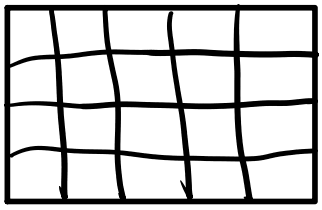
```java
if(nQueens(board, 0)) {
    System.out.println("solution is possible");
    printBoard(board);
} else {
    System.out.println("solution is not possible");
}
// System.out.println("total ways to solve n queens =
```

Grid ways :→

Find number of ways to reach from (0,0) to (N-1, M-1) in a N×M Grid.

Allowed moves - right or down

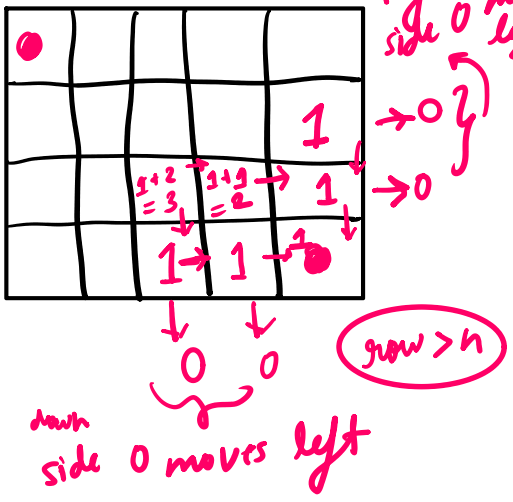
Allowed moves - right, down



call (x, y)

Right
(x, y+1)
w1

Down
(x+1, y)
w2

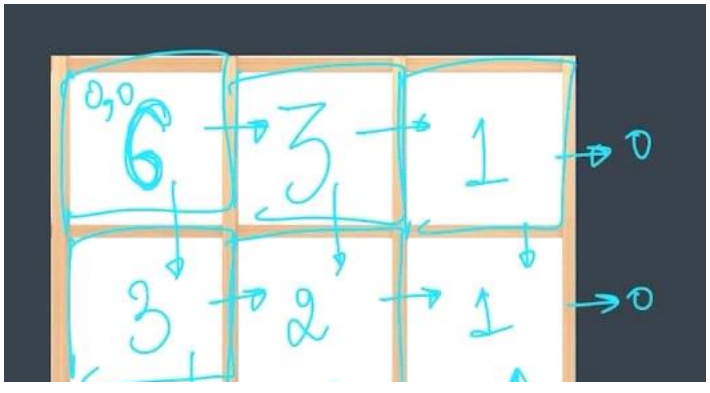


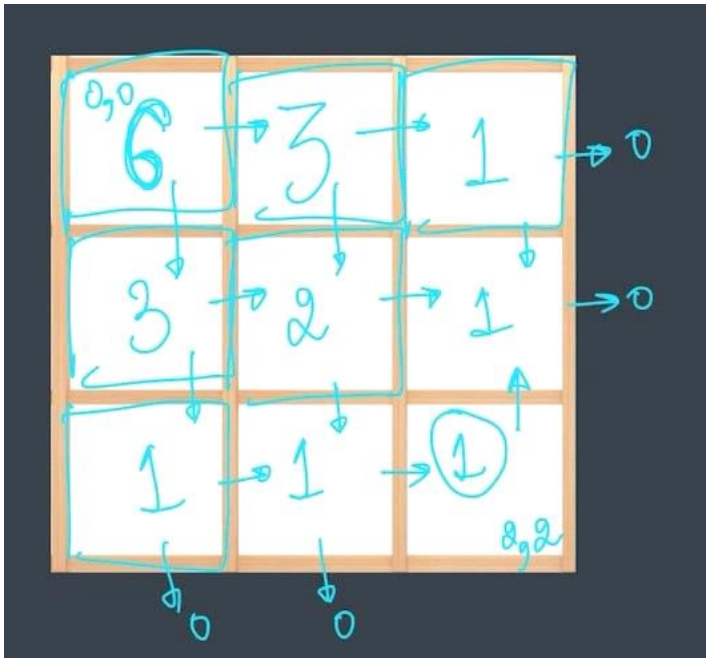
w1 + w2 = total ways

f(x, y) = f(x+1, y) + f(x, y+1)

(Down) (Right)

time complexity Grid ways = 2^{n+m}






```java
public class Classroom {
    public static int gridWays(int i, int j, int n, int m) {
        //base case
        if(i == n-1 && j == m-1) { //condn for last cell
            return 1;
        } else if(i == n || j == n) {
            return 0;
        }

        int w1 = gridWays(i+1, j, n, m);
        int w2 = gridWays(i, j+1, n, m);
        return w1 + w2;
    }
    Run | Debug
    public static void main(String args[]) {
            int n = 3, m = 3;
            System.out.println(gridWays(0, 0, n, m));
    }
}
```

Sudoku :->

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

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

Sudoku

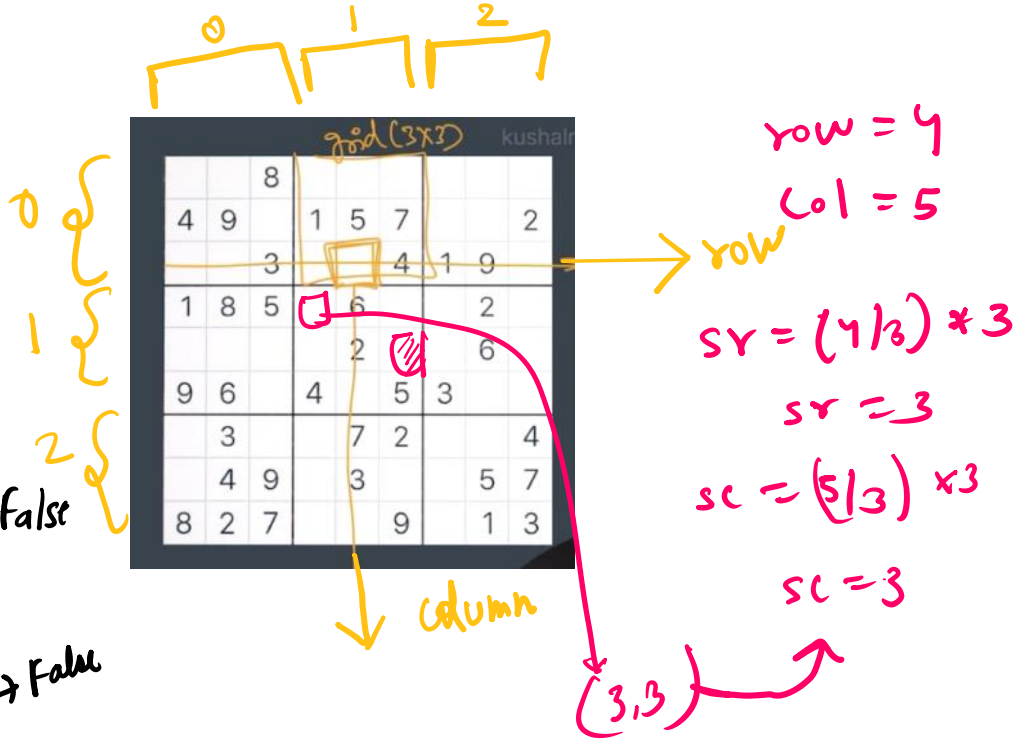
Calculating Grid

column

```
for(int i=0 to 8)
    s[i][col] == digit → false
```

Row

```
for(int j=0 to 8)
    s[row][j] == digit → false
```



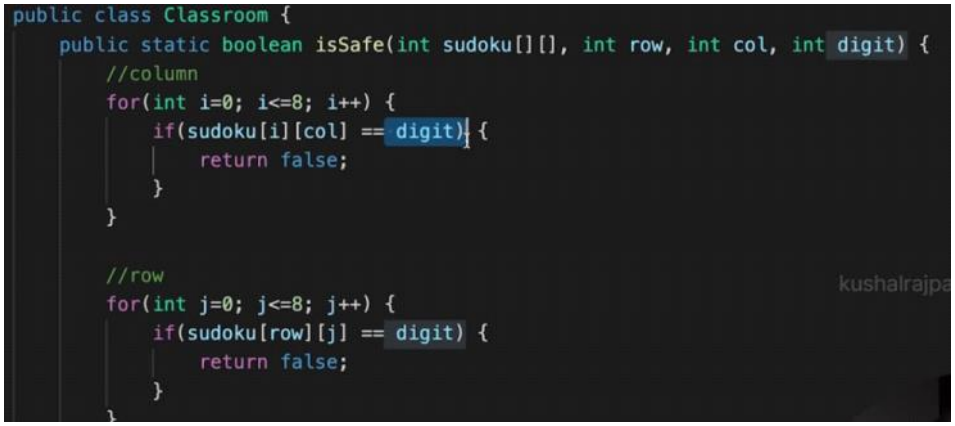
Calculating Grid (row, col) → 3 X 3 Grid (sr, sc)

sr = (row/3) * 3

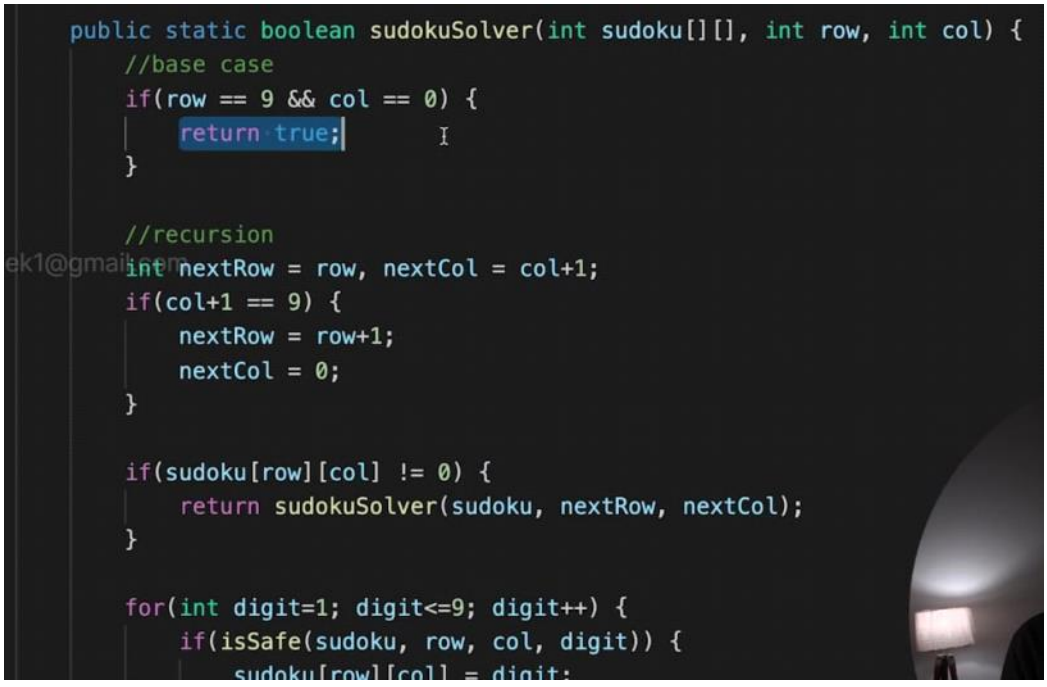
sc = (col/3) * 3

row/3 → 0 1 2

col/3 → 0 1 2


public class Classroom {
public static boolean isSafe(int sudoku[][], int row, int col, int digit) {
//column
for(int i=0; i<=8; i++) {
if(sudoku[i][col] == digit) {
return false;
}
}
//row
for(int j=0; j<=8; j++) {
if(sudoku[row][j] == digit) {
return false;
}
kushalrajpa


//grid
int sr = (row/3) * 3;
int sc = (col/3) * 3;
//3x3 grid
for(int i=sr; i<sr+3; i++) {
for(int j=sc; j<sc+3; j++) {
if(sudoku[i][j] == digit) {
return false;
}
}
}
return true;
}


public static boolean sudokuSolver(int sudoku[][], int row, int col) {
//base case
if(row == 9 && col == 0) {
return true;
}
//recursion
int nextRow = row, nextCol = col+1;
if(col+1 == 9) {
nextRow = row+1;
nextCol = 0;
}
if(sudoku[row][col] != 0) {
return sudokuSolver(sudoku, nextRow, nextCol);
}
for(int digit=1; digit<=9; digit++) {
if(isSafe(sudoku, row, col, digit)) {

```
    if(sudoku[row][col] != 0) {
        return sudokuSolver(sudoku, nextRow, nextCol);
    }

    for(int digit=1; digit<=9; digit++) {
        if(isSafe(sudoku, row, col, digit)) {
            sudoku[row][col] = digit;
            if(sudokuSolver(sudoku, nextRow, nextCol)) {//soln exis
                return true;
            }
            sudoku[row][col] = 0;
        }
    }

    return false;
}
```

```
public static void printSudoku(int sudoku[][]) {
    for(int i=0; i<9; i++) {
        for(int j=0; j<9; j++) {
            System.out.print(sudoku[i][j]+" ");
        }
        System.out.println();
    }
}
```