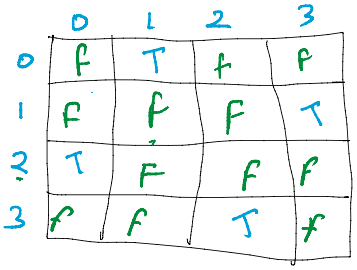
27 Oct

27 October 2022 20:59

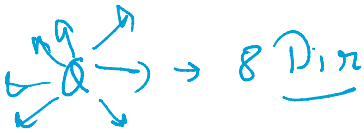
N - Queen :-



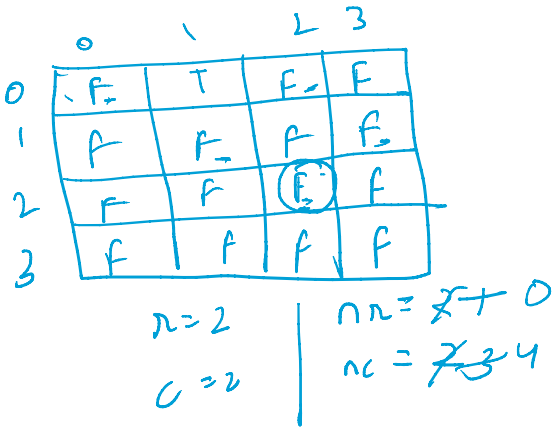
n x n

↳ Place N - queens

N = 4



```java
public static boolean canQueenBePlaced(boolean[][] board, int r, int c) {
    //to move up - r-1
    // to move up-right = r-1,c+1
    //to move up-left = r-1,c-1
    int nr = r;
    int nc = c;
    while (nr >= 0) {
        if (board[nr][nc]) return false;
        nr--;
    }
    nr = r;
    while (nr >= 0 && nc >= 0) {
        if (board[nr][nc]) return false;
        nr--;
        nc--;
    }
    nr = r;
    nc = c;
    while (nr >= 0 && nc < board[0].length) {
        if (board[nr][nc]) return false;
        nr--;
        nc++;
    }
    return true;
}
```



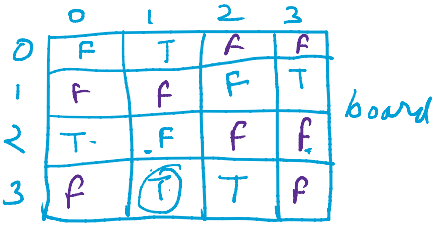
r = 2

c = 2

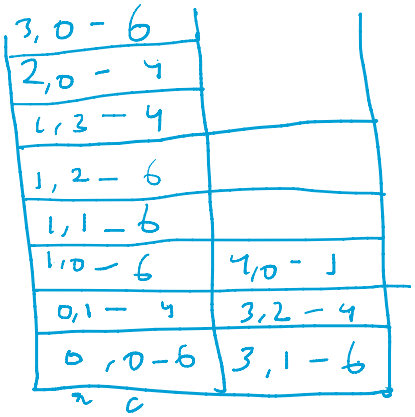
nr = ~~r~~ + 0

nc = ~~2~~ 3 4

```java
public static void nQueenSol(boolean[][] board, int r, int c, List<List<String>> ans) {
    if (r == board.length) {
        ArrayList<String> currList = new ArrayList<>();
        for (int i = 0; i < board.length; i++) {
            String currString = "";
            for (int j = 0; j < board.length; j++) {
                if (board[i][j]) currString += 'Q';
                else currString += '.';
            }
            currList.add(currString);
        }
        ans.add(currList);
        return;
    }
    if (canQueenBePlaced(board, r, c)) {
        board[r][c] = true;
        nQueenSol(board, r + 1, 0, ans);
        board[r][c] = false;
    }
    if (c + 1 < board.length) nQueenSol(board, r, c + 1, ans);
}
```



board



Sudoku - Solver :-

0 1 2 3 4 5 6 7 8

Sudoku - Solver :-

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

r = 2/9 = 0

c = 2%9 = 2

1 - 9

(board, r, c, num)

sr = r - (r%3) = 3

sc = c - (c%3) = 0

num = 11

r = 11/9 = 1

c = 11%9 = 2

58

58/9 = 6

58%9 = 4

```java
public boolean sudokuSolve(char[][] board, int count) {
    if (count == 81) return true;
    int r = count / 9;
    int c = count % 9;
    if (board[r][c] != '.') {
        return sudokuSolve(board, count: count + 1);
    }
    for (int i = 1; i <= 9; i++) {
        char curr = (char) (i + '0');
        if (canNumBePlaced(board, r, c, curr)) {
            board[r][c] = curr;
            boolean recRes = sudokuSolve(board, count: count + 1);
            if (recRes) return true;
            board[r][c] = '.';
        }
    }
    return false;
}
```

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