Input

↳ number, n

Sample Input

4 = n

Sample Output

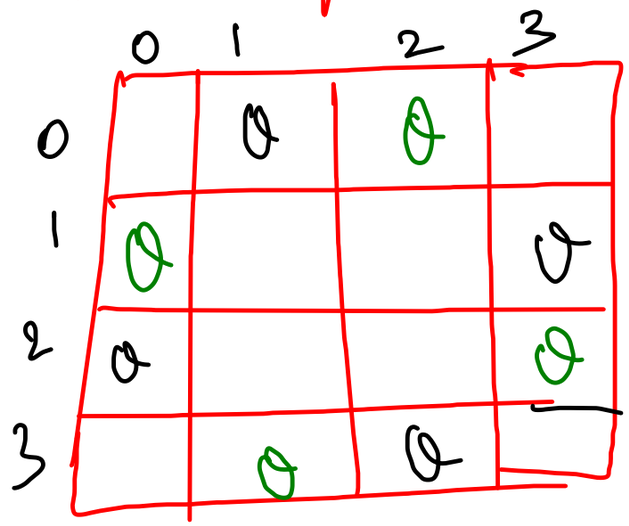
0-1, 1-3, 2-0, 3-2, .

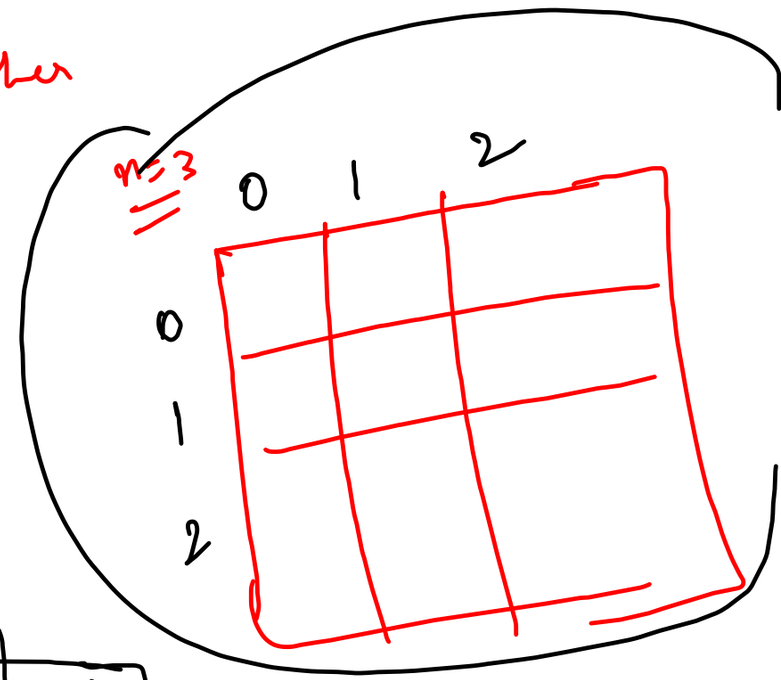
0-2, 1-0, 2-3, 3-1, .

① 1 chess board n×n.

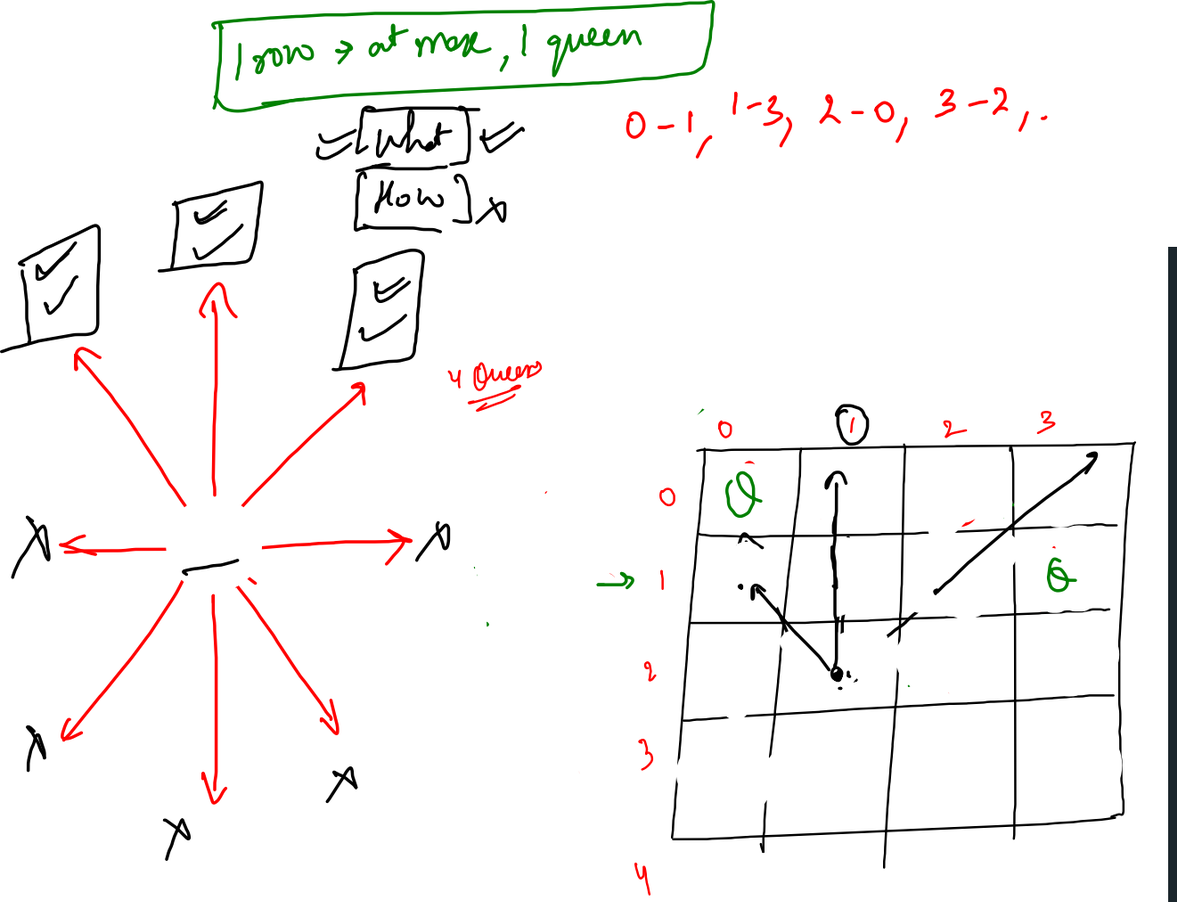
② n queens.

↳ find & print all possible config. in which n queens can be placed on the board such that no two queens attack each other

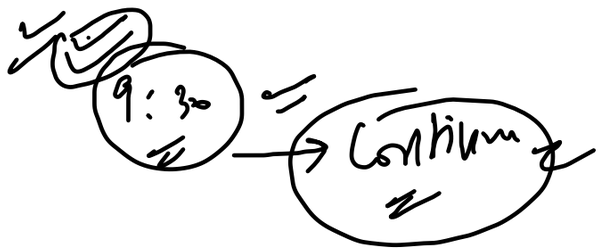


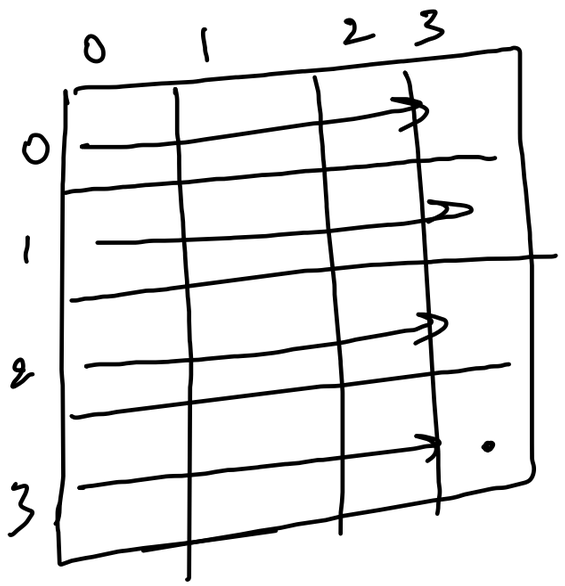


0 Config




```java
// chess[i][j]  == 1 , queen placed
public static void printNQueens(int[][] chess, String csf, int row) {
    if(row == chess.length){
        System.out.println(csf+".");
        return;
    }
    for(int col = 0 ; col < chess[0].length ; col++){
        if(isSafe(chess,row,col)){
            chess[row][col] = 1;
            printNQueens(chess,csf+row+"-"+col+",",row+1);
            chess[row][col] = 0;
        }
    }

}
```






```java
public static void printNQueens(int[][] chess, String csf, int row){
    if(row == chess.length){
        System.out.println(csf+".");
        return;
    }
    for(int col = 0 ; col < chess[0].length ; col++){
        if(isSafe(chess,row,col)){
            chess[row][col] = 1;
            printNQueens(chess,csf+row+"-"+col+", ",row+1);
            chess[row][col] = 0;
        }
    }
}
```

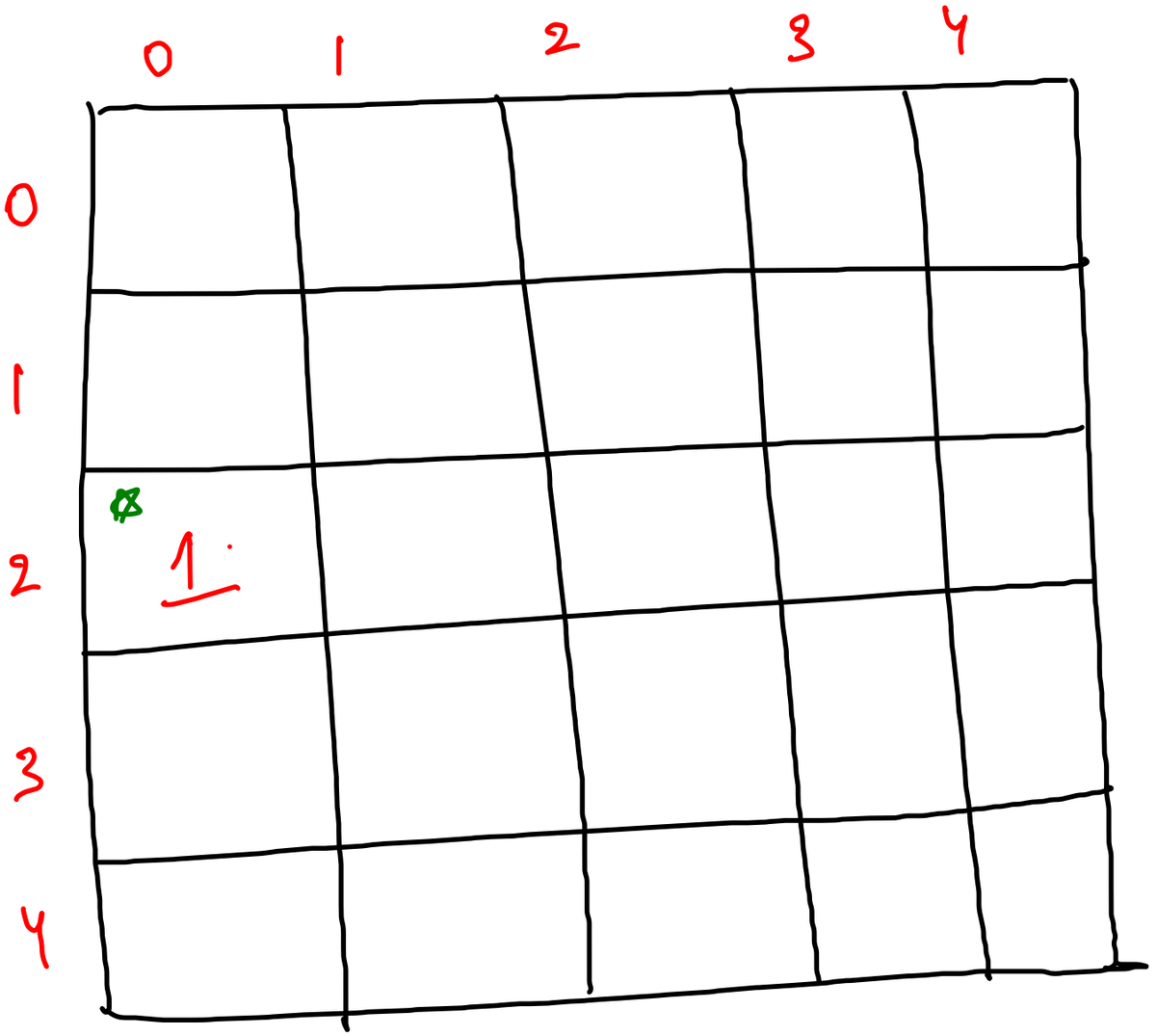
```java
public static boolean isSafe(int chess[][],int row,int col){
    // v. up
    for(int i = row-1, j = col ; i >= 0 ; i--){
        if(chess[i][j] == 1){
            return false;
        }
    }

    // l. dia
    for(int i = row-1 , j = col-1 ; i >= 0 && j >= 0 ; i-- , j--){
        if(chess[i][j] == 1){
            return false;
        }
    }

    // r. dia
    for(int i = row-1 , j = col+1 ; i >= 0 && j < chess[0].length ; i-- , j++){
        if(chess[i][j] == 1){
            return false;
        }
    }

    // implication queen can be placed
    return true;
}
```

Input

```
5
2
0
```

Output

```
25 2 13 8 23
12 7 24 3 14
1 18 15 22 9
6 11 20 17 4
19 16 5 10 21
```

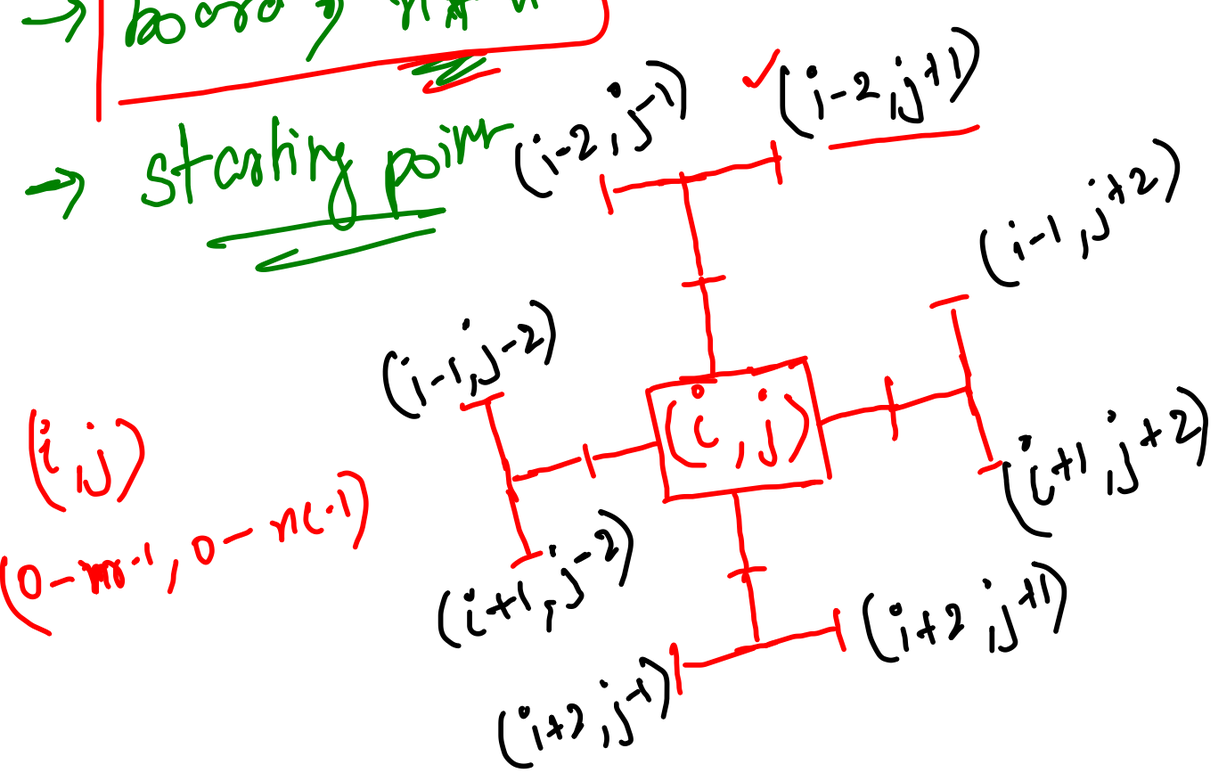


→ board → n * n

→ starting point

(i,j)

(0-n-1, 0-n-1)



(n*n)

bound : 2 more complex

2

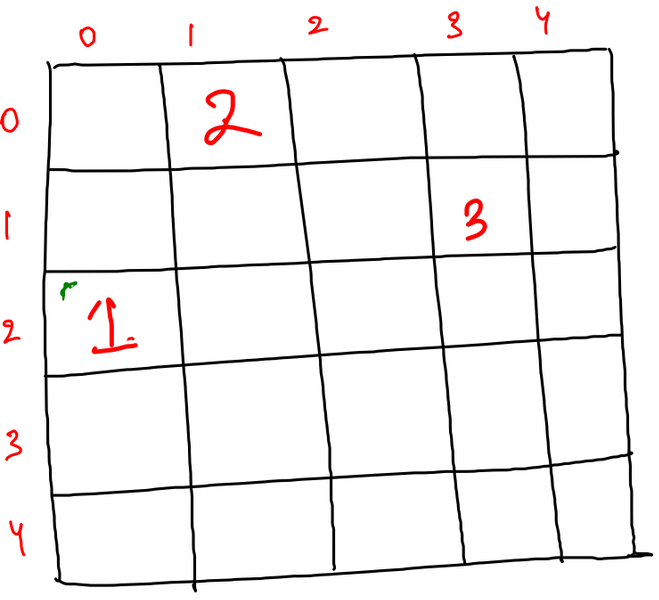
bound : 1 more complex

1

(main)

board is empty

pKT(chess, 2, 0, 1);




```java
public static void printKnightsTour(int[][] chess, int row ,int col , int moveNo) {
    if(row < 0 || col < 0 || row >= chess.length || col >= chess[0].length || chess[row][col] != 0){
        // either invalid pos or block is already visited
        return;
    }
    if(moveNo == chess.length * chess[0].length){
        chess[row][col] = moveNo;
        displayBoard(chess);
        chess[row][col] = 0;
        return;
    }
    chess[row][col] = moveNo; //  mark
    printKnightsTour(chess,row-2,col+1,moveNo+1);// dir1
    printKnightsTour(chess,row-1,col+2,moveNo+1);// dir2
    printKnightsTour(chess,row+1,col+2,moveNo+1);// dir3
    printKnightsTour(chess,row+2,col+1,moveNo+1);// dir4
    printKnightsTour(chess,row+2,col-1,moveNo+1);// dir5
    printKnightsTour(chess,row+1,col-2,moveNo+1);// dir6
    printKnightsTour(chess,row-1,col-2,moveNo+1);// dir7
    printKnightsTour(chess,row-2,col-1,moveNo+1);// dir8
    chess[row][col] = 0; // unmark

}

public static void displayBoard(int[][] chess){
    for(int i = 0; i < chess.length; i++){
        for(int j = 0; j < chess[0].length; j++){
            System.out.print(chess[i][j] + " ");
        }
        System.out.println();
    }

    System.out.println();
}
```

display

main

( )

2

( ) chus ⇒ 1 more

1

(main) chus ⇒ Empty