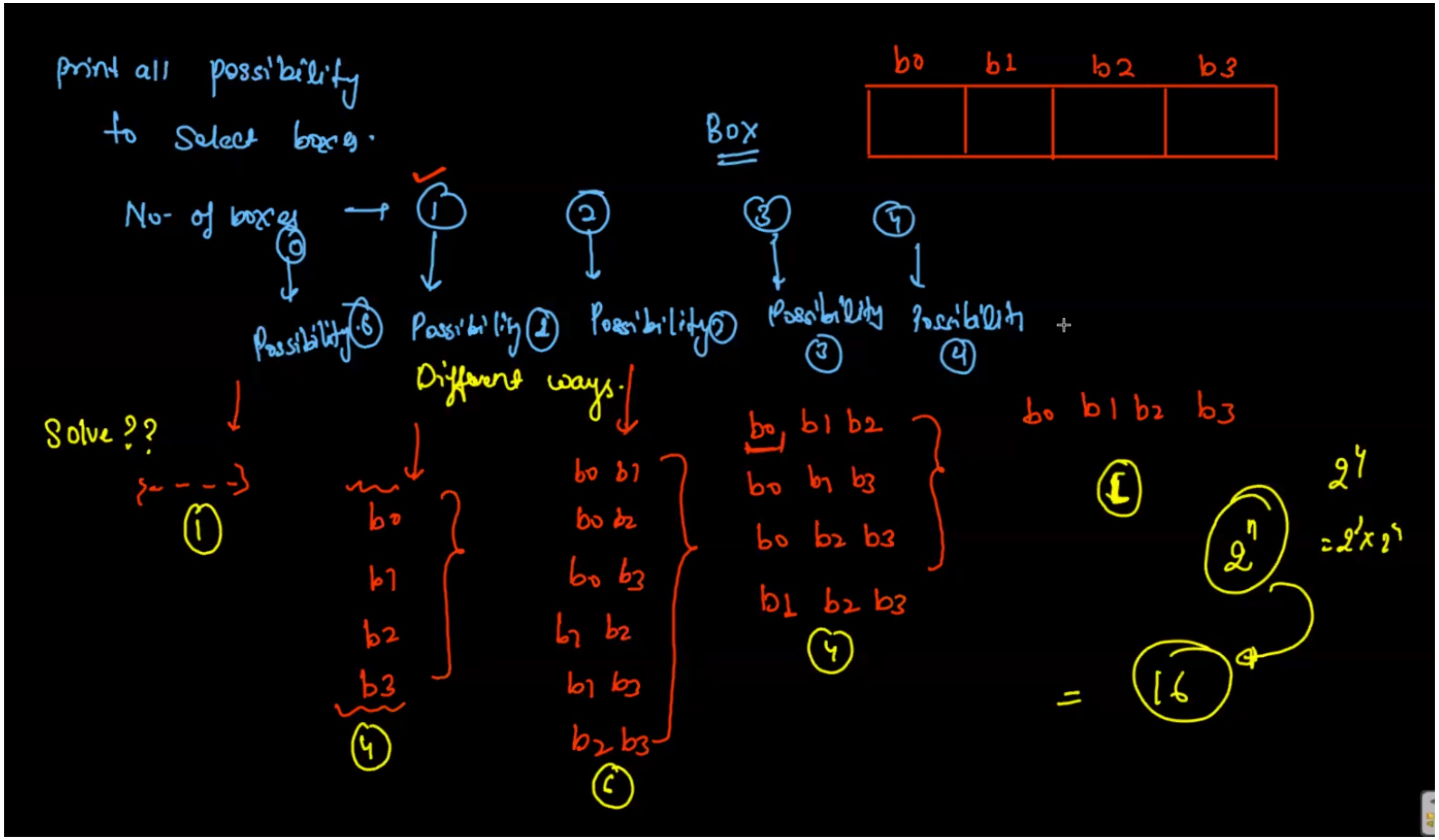


print all possibility
to select boxes.

Box

b0	b1	b2	b3

No. of boxes → 0 ✓1 2 3 4

Possibility 0 Possibility 1 Possibility 2 Possibility 3 Possibility 4

Different ways.

Solve ??

{ - - - }
1

b0
b1
b2
b3
4

b0 b1
b0 b2
b0 b3
b1 b2
b1 b3
b2 b3
6

b0 b1 b2
b0 b1 b3
b0 b2 b3
b1 b2 b3
4

b0 b1 b2 b3
1

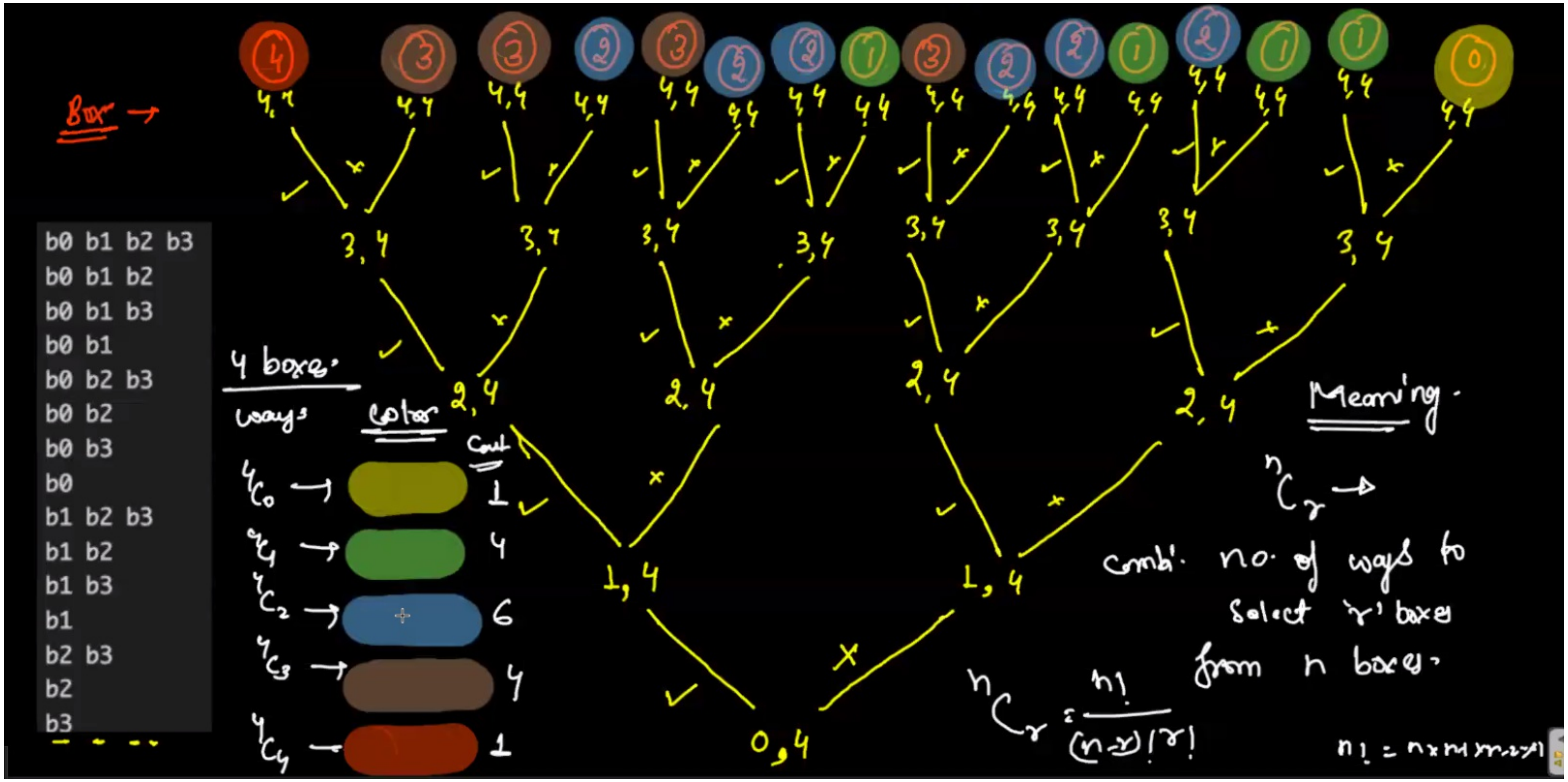
2^4

$2^4 = 2^2 \times 2^2$

$= 16$

$$(x+1)^n = 2^n \Rightarrow \underset{\substack{\uparrow\\0}}{{}^nC_0} + \underset{\substack{\uparrow\\1}}{{}^nC_1} + \underset{\substack{\uparrow\\2}}{{}^nC_2} + \underset{\substack{\uparrow\\3}}{{}^nC_3} + \cdots\cdots + \underset{\substack{\uparrow\\n}}{{}^nC_n} \Big\}$$ No. of Subseq

Select box →



```
b0 b1 b2 b3
b0 b1 b2
b0 b1 b3
b0 b1
b0 b2 b3
b0 b2
b0 b3
b0
b1 b2 b3
b1 b2
b1 b3
b1
b2 b3
b2
b3
- - - -
```

4 boxes

ways	color	Count
${}^4C_0$ →	🟨	1
${}^4C_1$ →	🟩	4
${}^4C_2$ →	🟦	6
${}^4C_3$ →	🟫	4
${}^4C_4$ →	🟥	1

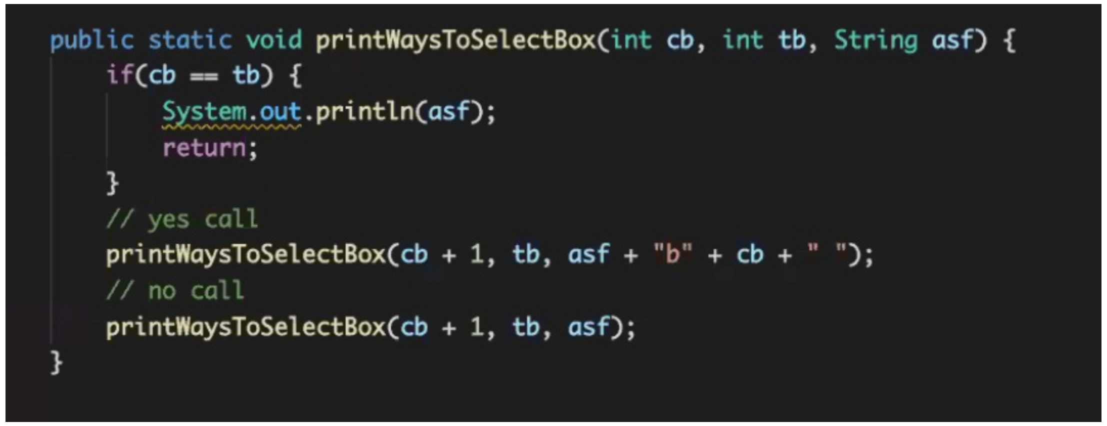
Meaning.

${}^nC_r \Rightarrow$ comb. no. of ways to select 'r' boxes from n boxes.

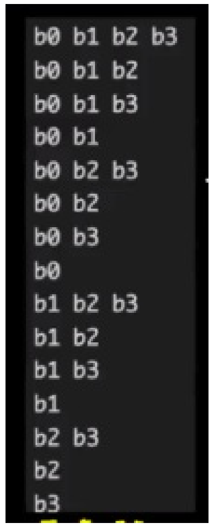
$${}^nC_r = \frac{n!}{(n-r)!\,r!}$$

$n! = n \times n-1 \times n-2 \times \cdots \times 1$

All possiblity of selecting all boxes

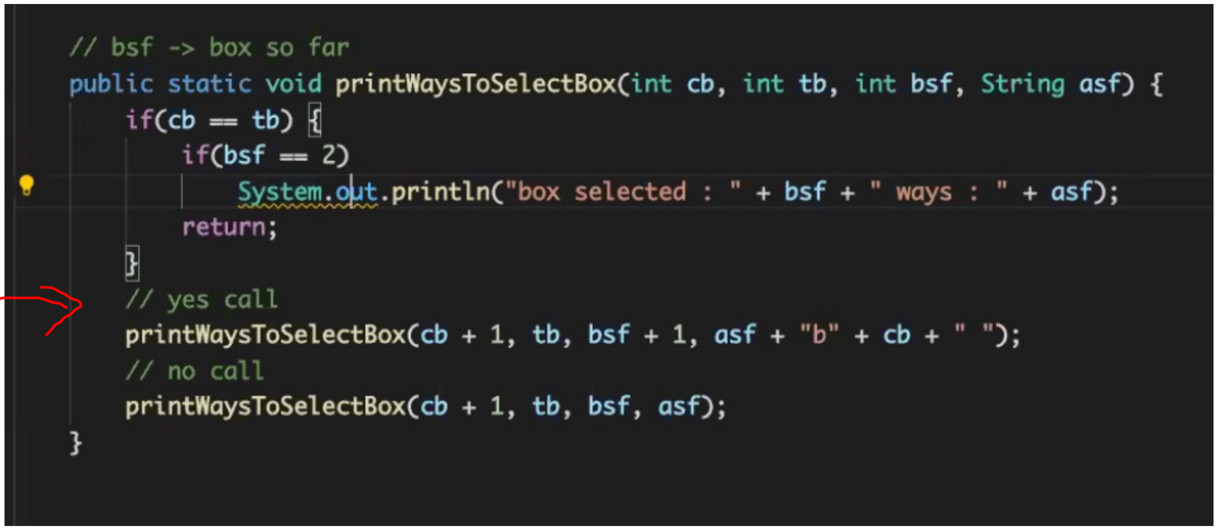


```java
public static void printWaysToSelectBox(int cb, int tb, String asf) {
    if(cb == tb) {
        System.out.println(asf);
        return;
    }
    // yes call
    printWaysToSelectBox(cb + 1, tb, asf + "b" + cb + " ");
    // no call
    printWaysToSelectBox(cb + 1, tb, asf);
}
```

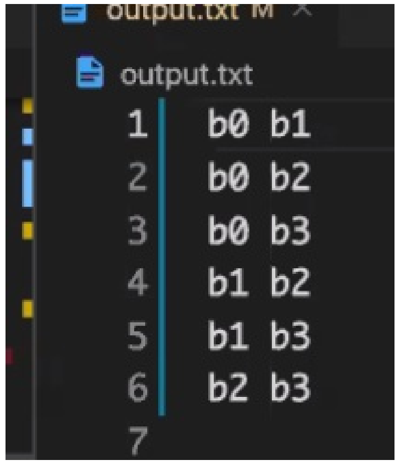


```
b0 b1 b2 b3
b0 b1 b2
b0 b1 b3
b0 b1
b0 b2 b3
b0 b2
b0 b3
b0
b1 b2 b3
b1 b2
b1 b3
b1
b2 b3
b2
b3
```

possibility of selecting r boxes from n boxes



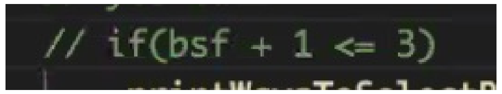
```java
// bsf -> box so far
public static void printWaysToSelectBox(int cb, int tb, int bsf, String asf) {
    if(cb == tb) {
        if(bsf == 2)
            System.out.println("box selected : " + bsf + " ways : " + asf);
        return;
    }
    // yes call
    printWaysToSelectBox(cb + 1, tb, bsf + 1, asf + "b" + cb + " ");
    // no call
    printWaysToSelectBox(cb + 1, tb, bsf, asf);
}
```



output.txt M

output.txt

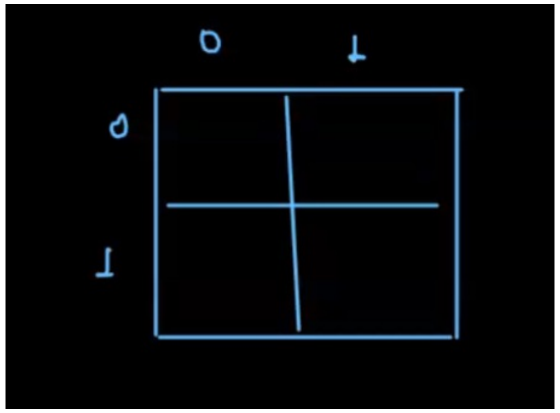
```
b0 b1
b0 b2
b0 b3
b1 b2
b1 b3
b2 b3
```

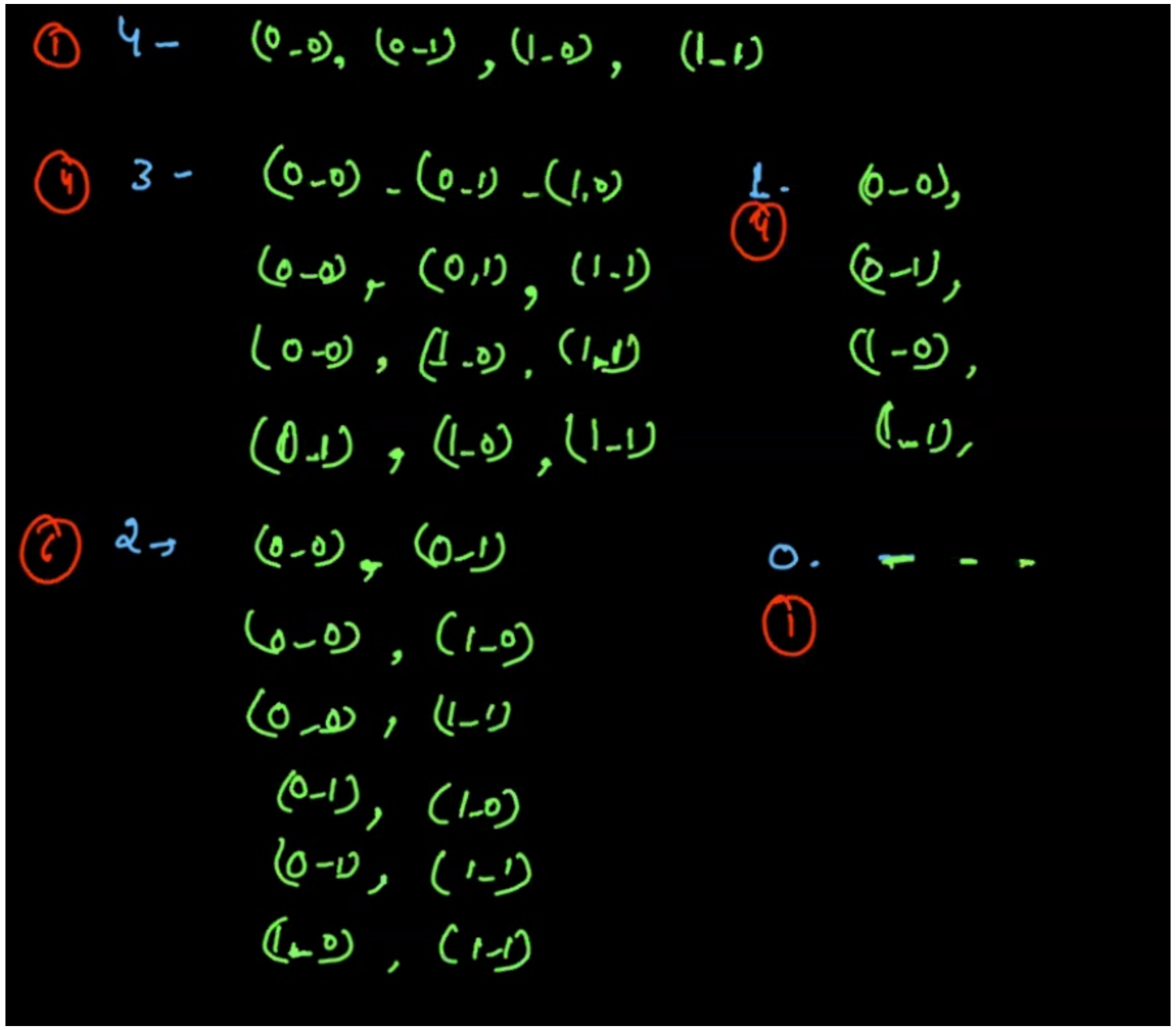


```java
// if(bsf + 1 <= 3)
```

to avoid unnecessary calls

All Possibility of selecting 4 box in 2d?





① 4 - (0-0), (0-1), (1-0), (1-1)

④ 3 - (0-0) - (0-1) - (1,0)

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

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

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

⑥ 2 → (0-0), (0-1)

(0-0), (1-0)

(0-0), (1-1)

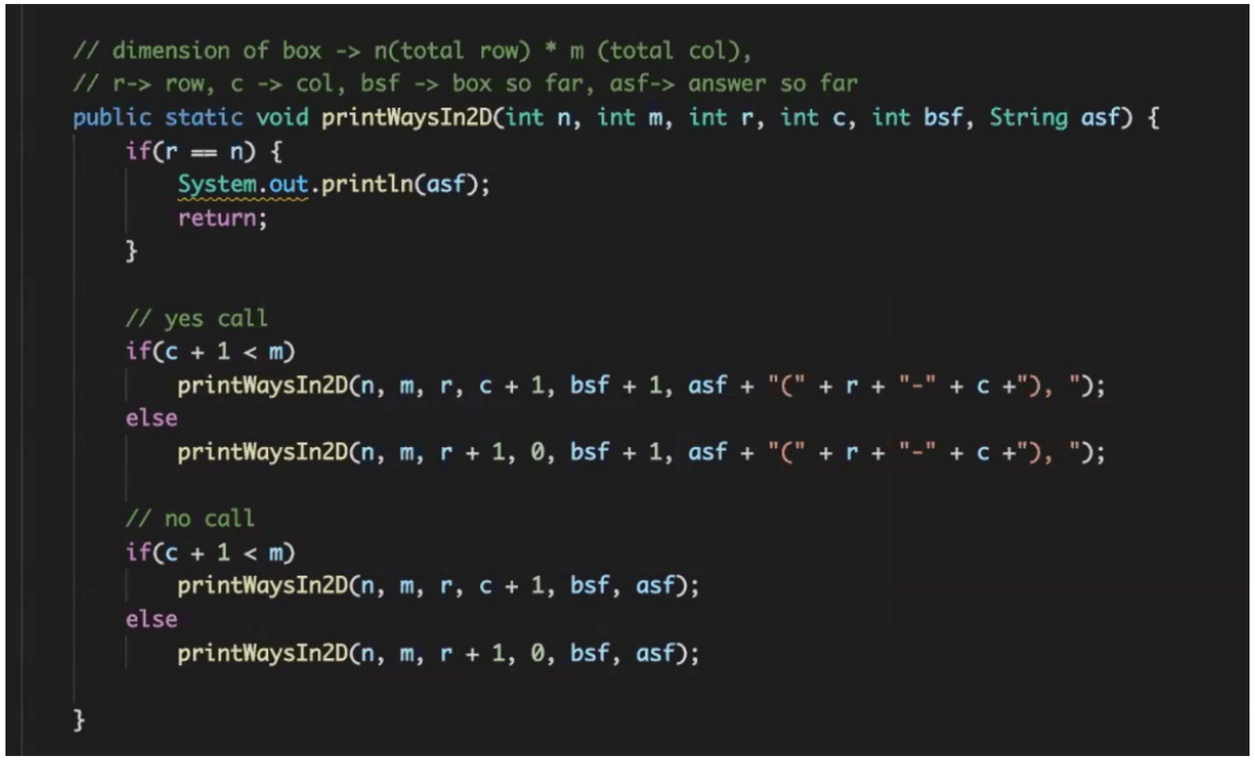
(0-1), (1-0)

(0-1), (1-1)

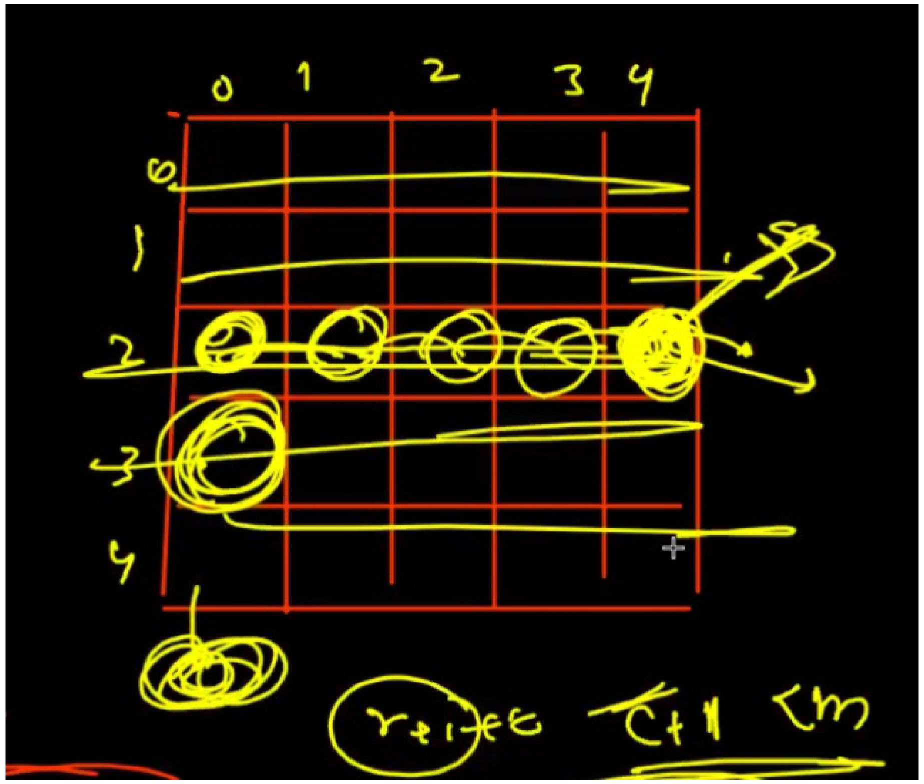
(1-0), (1-1)

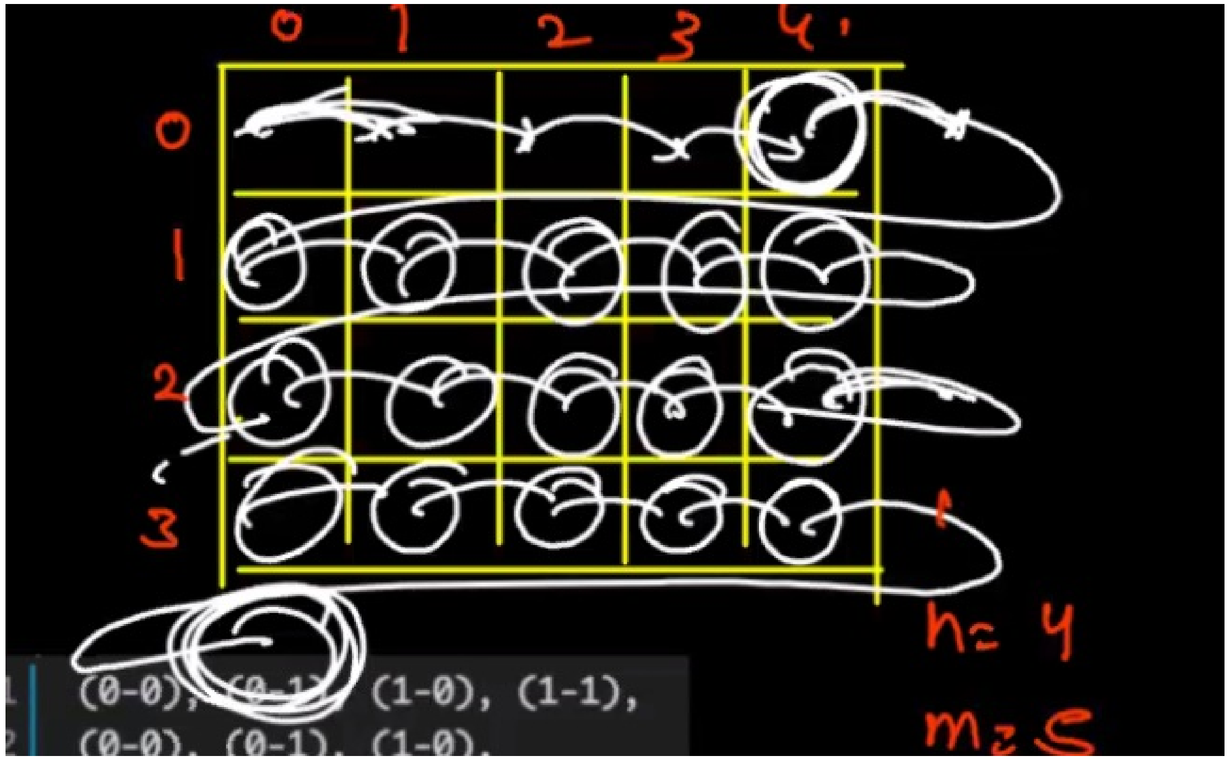
1 - ④ (0-0), (0-1), (1-0), (1-1),

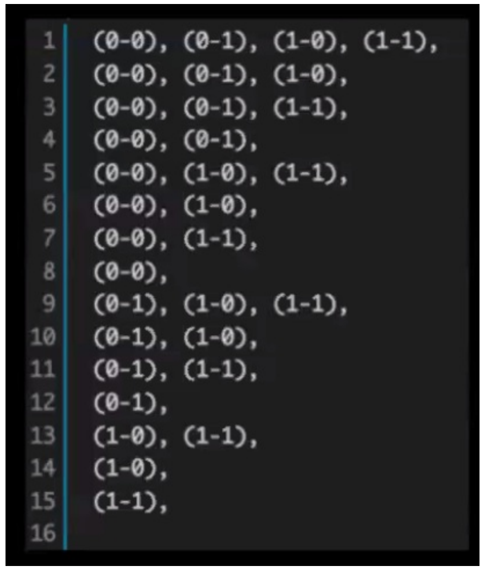
0 - - - - ①



```java
// dimension of box -> n(total row) * m (total col),
// r-> row, c -> col, bsf -> box so far, asf-> answer so far
public static void printWaysIn2D(int n, int m, int r, int c, int bsf, String asf) {
    if(r == n) {
        System.out.println(asf);
        return;
    }

    // yes call
    if(c + 1 < m)
        printWaysIn2D(n, m, r, c + 1, bsf + 1, asf + "(" + r + "-" + c +"), ");
    else
        printWaysIn2D(n, m, r + 1, 0, bsf + 1, asf + "(" + r + "-" + c +"), ");

    // no call
    if(c + 1 < m)
        printWaysIn2D(n, m, r, c + 1, bsf, asf);
    else
        printWaysIn2D(n, m, r + 1, 0, bsf, asf);

}
```



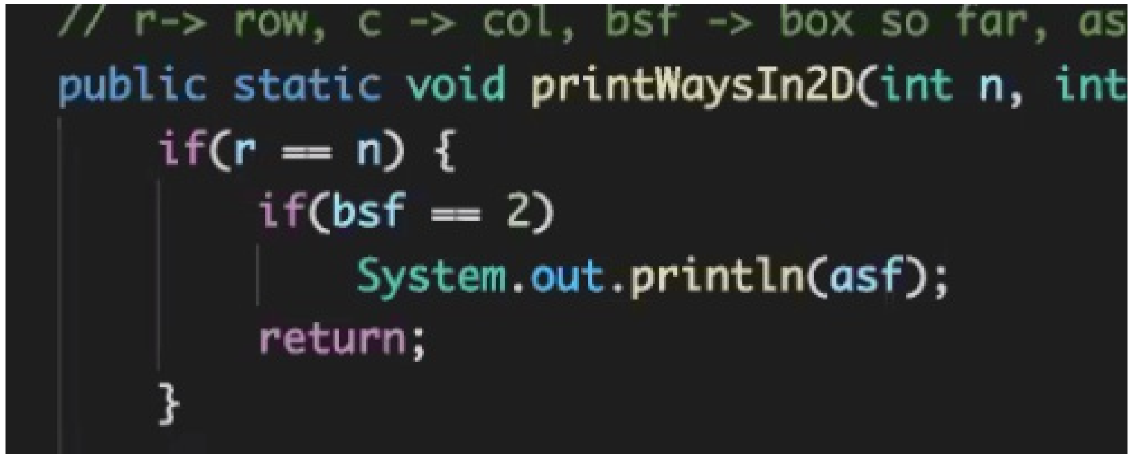




```
(0-0), (0-1), (1-0), (1-1),
(0-0), (0-1), (1-0),
(0-0), (0-1), (1-1),
(0-0), (0-1),
(0-0), (1-0), (1-1),
(0-0), (1-0),
(0-0), (1-1),
(0-0),
(0-1), (1-0), (1-1),
(0-1), (1-0),
(0-1), (1-1),
(0-1),
(1-0), (1-1),
(1-0),
(1-1),

```

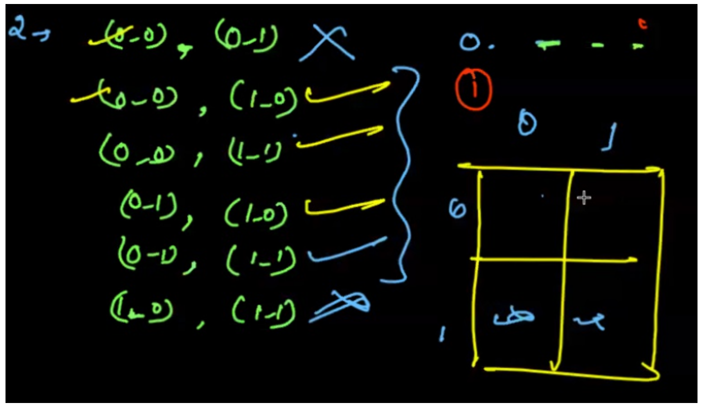
variation :

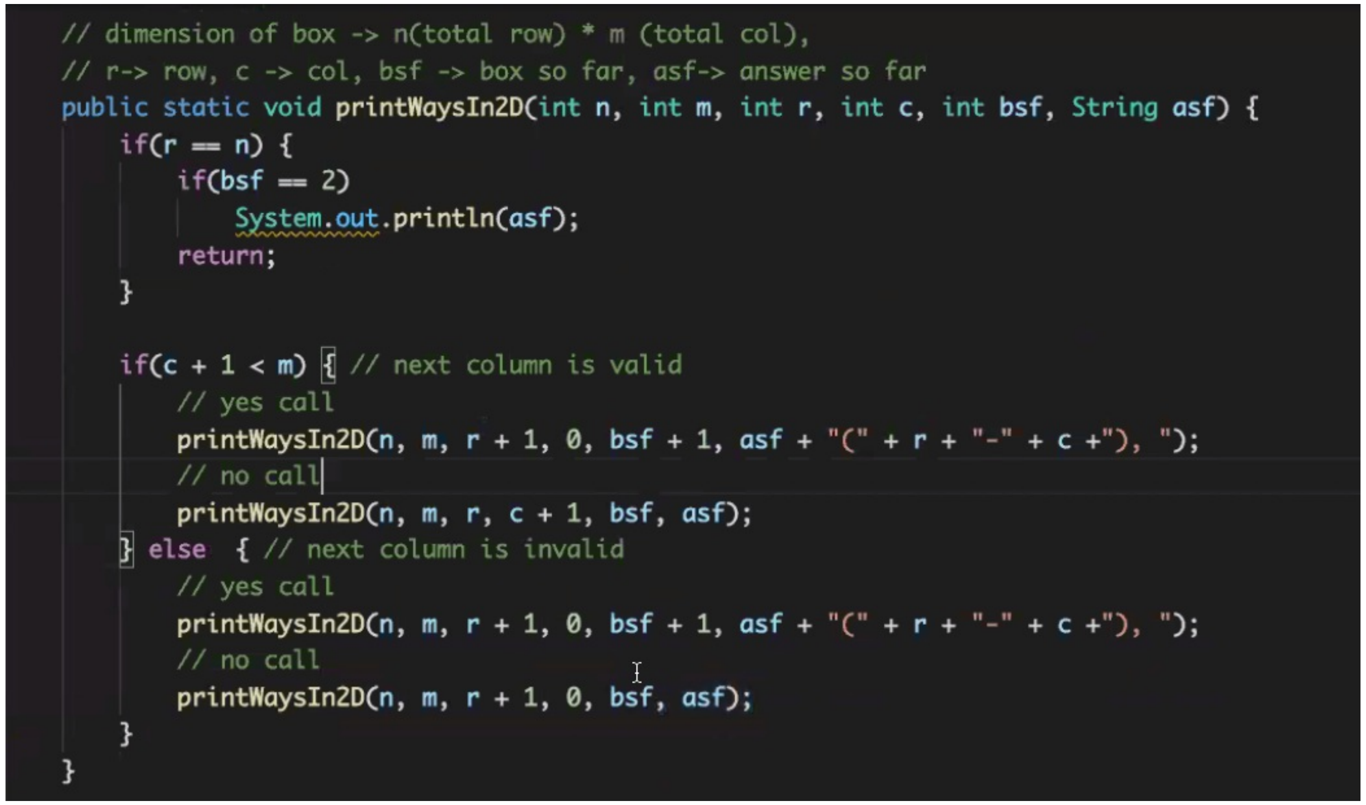
Possibility of selecting only 2 box in 2d?



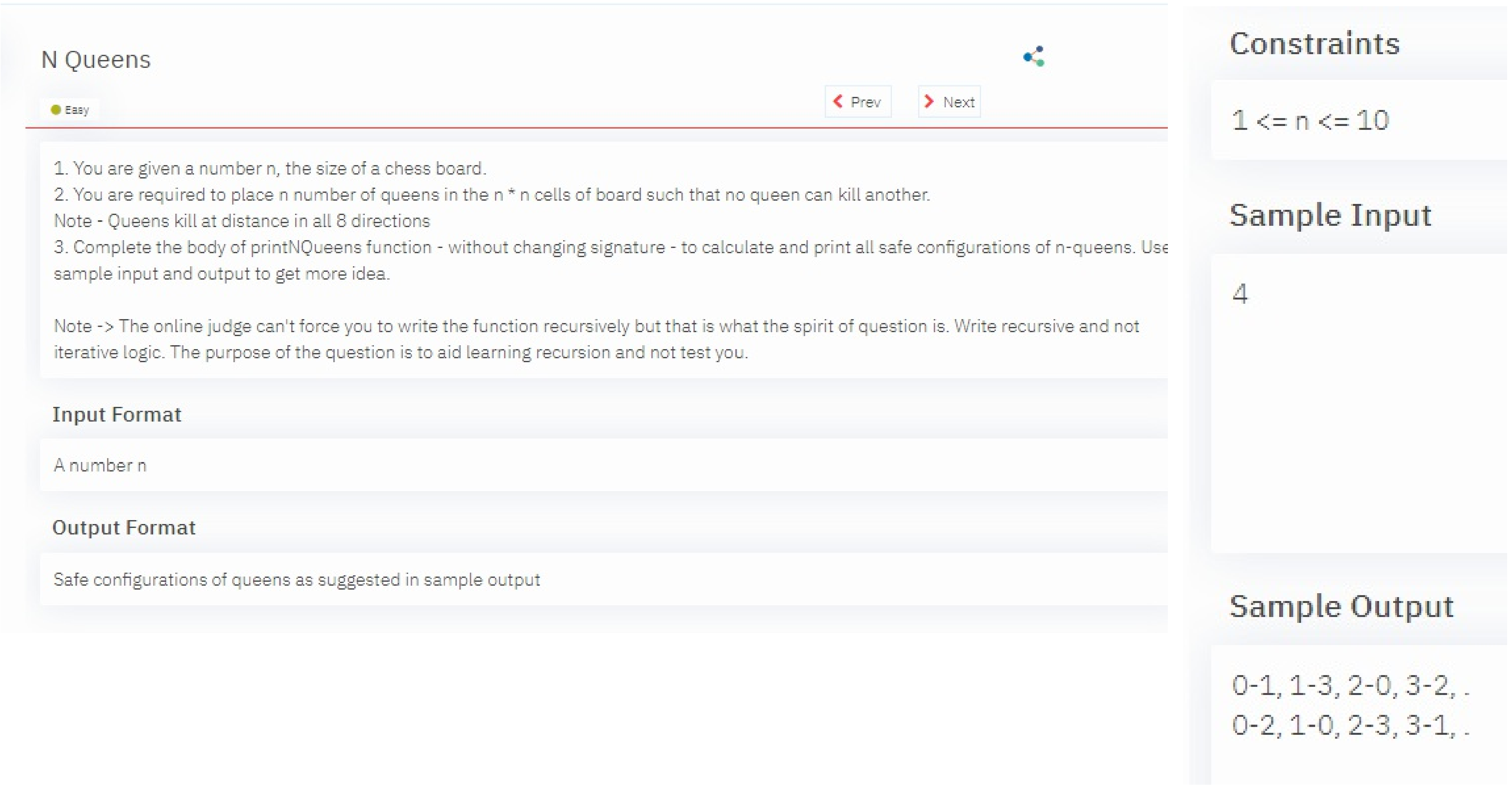
```
// r-> row, c -> col, bsf -> box so far, as
public static void printWaysIn2D(int n, int
    if(r == n) {
        if(bsf == 2)
            System.out.println(asf);
        return;
    }
```

within a row
single box select karna he ?





```
// dimension of box -> n(total row) * m (total col),
// r-> row, c -> col, bsf -> box so far, asf-> answer so far
public static void printWaysIn2D(int n, int m, int r, int c, int bsf, String asf) {
    if(r == n) {
        if(bsf == 2)
            System.out.println(asf);
        return;
    }

    if(c + 1 < m) { // next column is valid
        // yes call
        printWaysIn2D(n, m, r + 1, 0, bsf + 1, asf + "(" + r + "-" + c +"), ");
        // no call
        printWaysIn2D(n, m, r, c + 1, bsf, asf);
    } else  { // next column is invalid
        // yes call
        printWaysIn2D(n, m, r + 1, 0, bsf + 1, asf + "(" + r + "-" + c +"), ");
        // no call
        printWaysIn2D(n, m, r + 1, 0, bsf, asf);
    }
}
```



N Queens

● Easy

Prev Next

1. You are given a number n, the size of a chess board.
2. You are required to place n number of queens in the n * n cells of board such that no queen can kill another.
Note - Queens kill at distance in all 8 directions
3. Complete the body of printNQueens function - without changing signature - to calculate and print all safe configurations of n-queens. Use
sample input and output to get more idea.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Input Format

A number n

Output Format

Safe configurations of queens as suggested in sample output

Constraints

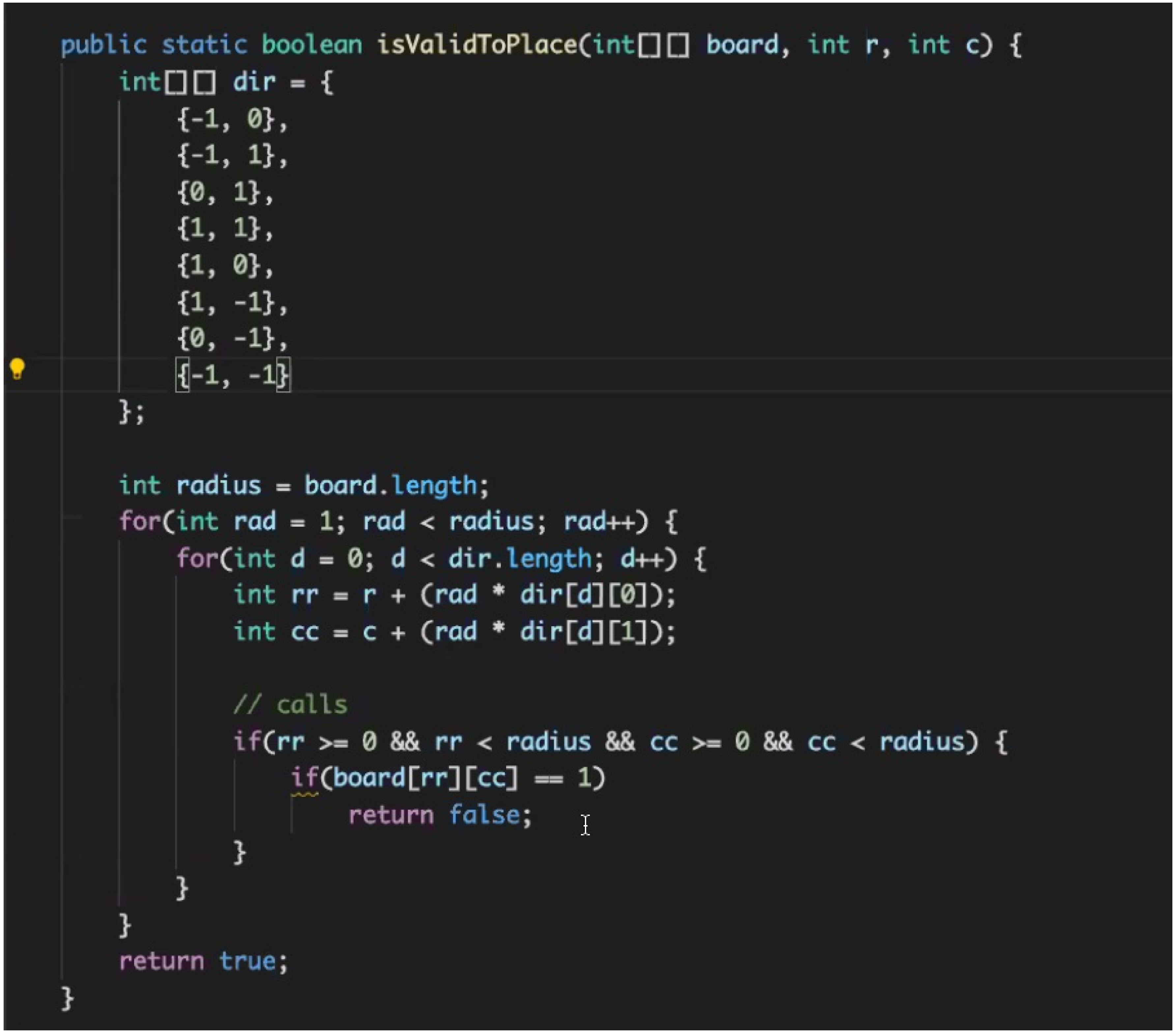
1 <= n <= 10

Sample Input

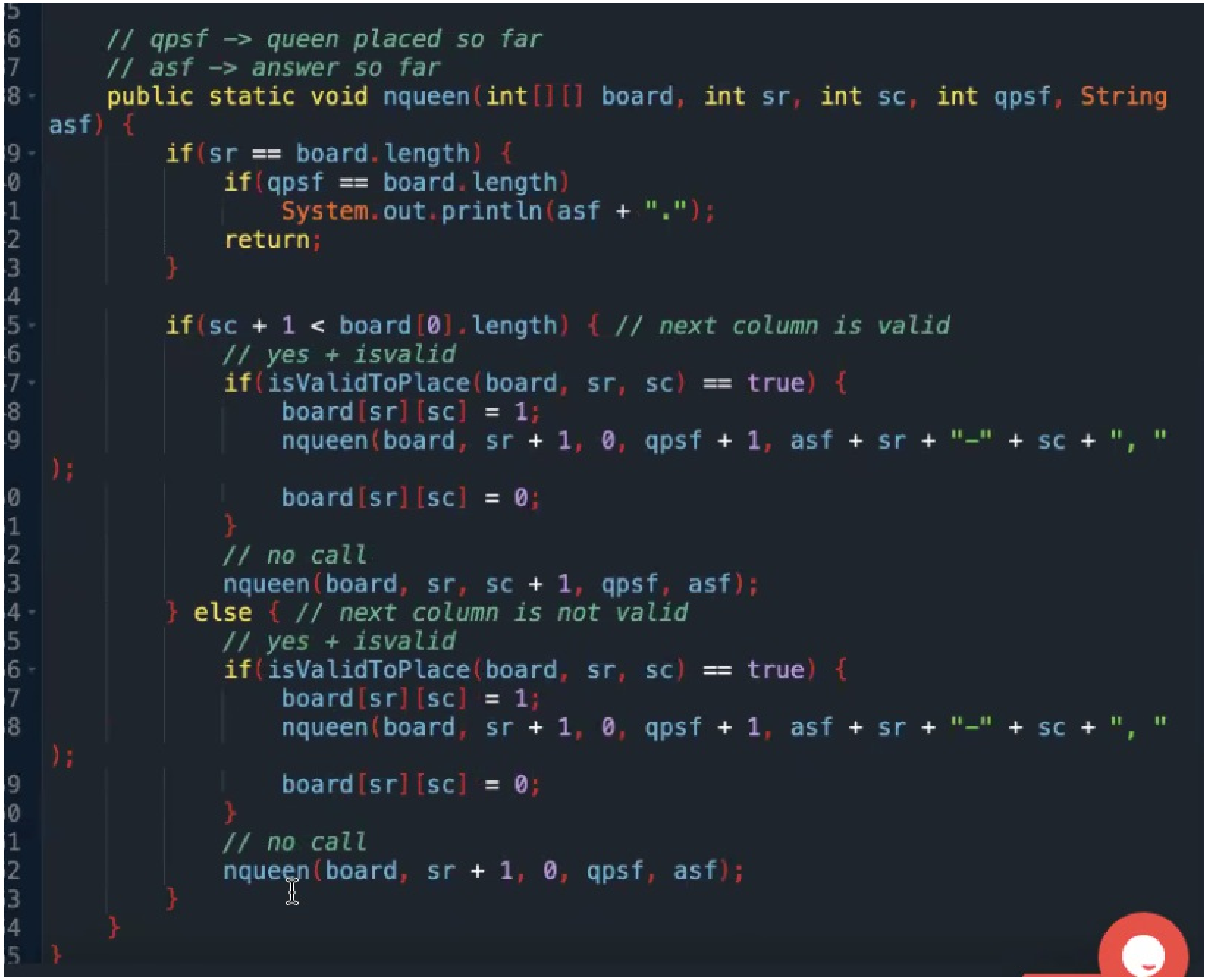
```
4
```

Sample Output

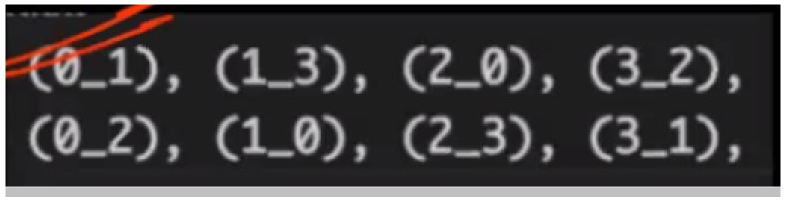
```
0-1, 1-3, 2-0, 3-2, .
0-2, 1-0, 2-3, 3-1, .
```



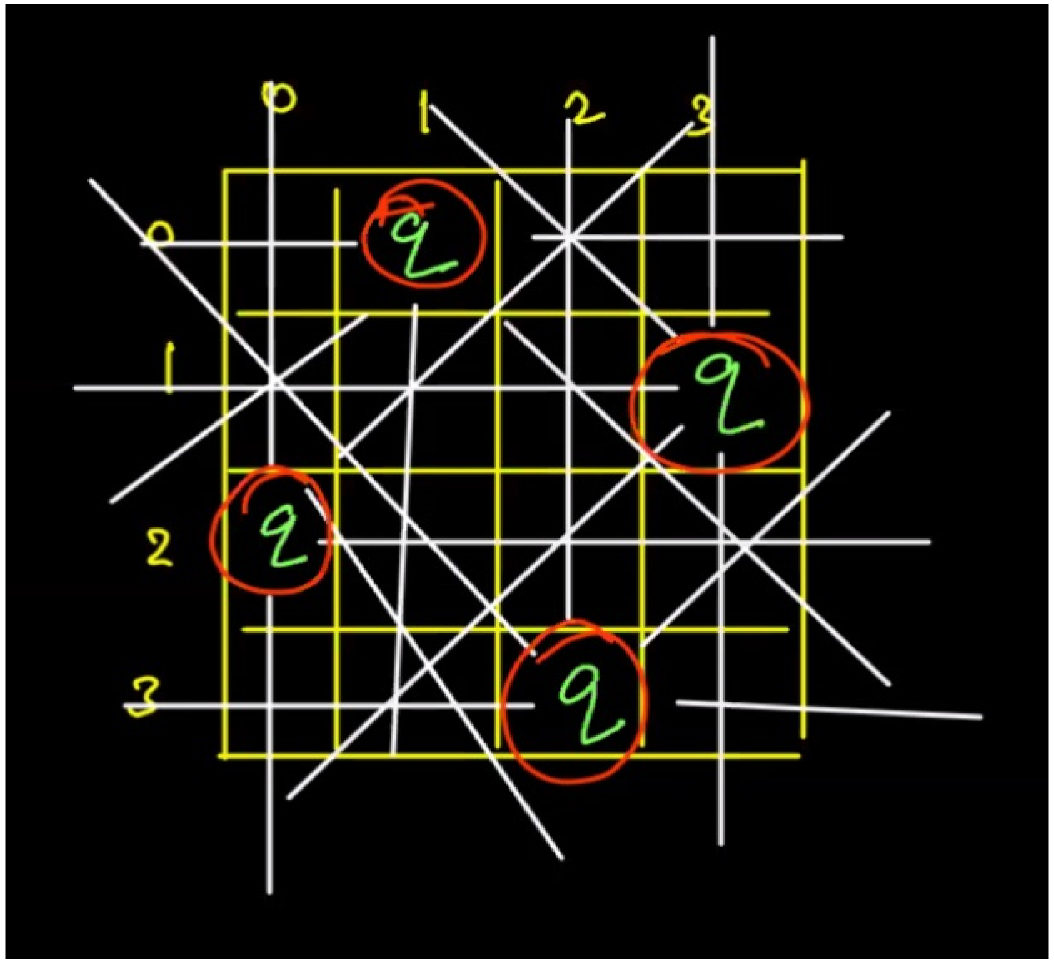
```java
public static boolean isValidToPlace(int[][] board, int r, int c) {
    int[][] dir = {
        {-1, 0},
        {-1, 1},
        {0, 1},
        {1, 1},
        {1, 0},
        {1, -1},
        {0, -1},
        {-1, -1}
    };

    int radius = board.length;
    for(int rad = 1; rad < radius; rad++) {
        for(int d = 0; d < dir.length; d++) {
            int rr = r + (rad * dir[d][0]);
            int cc = c + (rad * dir[d][1]);

            // calls
            if(rr >= 0 && rr < radius && cc >= 0 && cc < radius) {
                if(board[rr][cc] == 1)
                    return false;
            }
        }
    }
    return true;
}
```

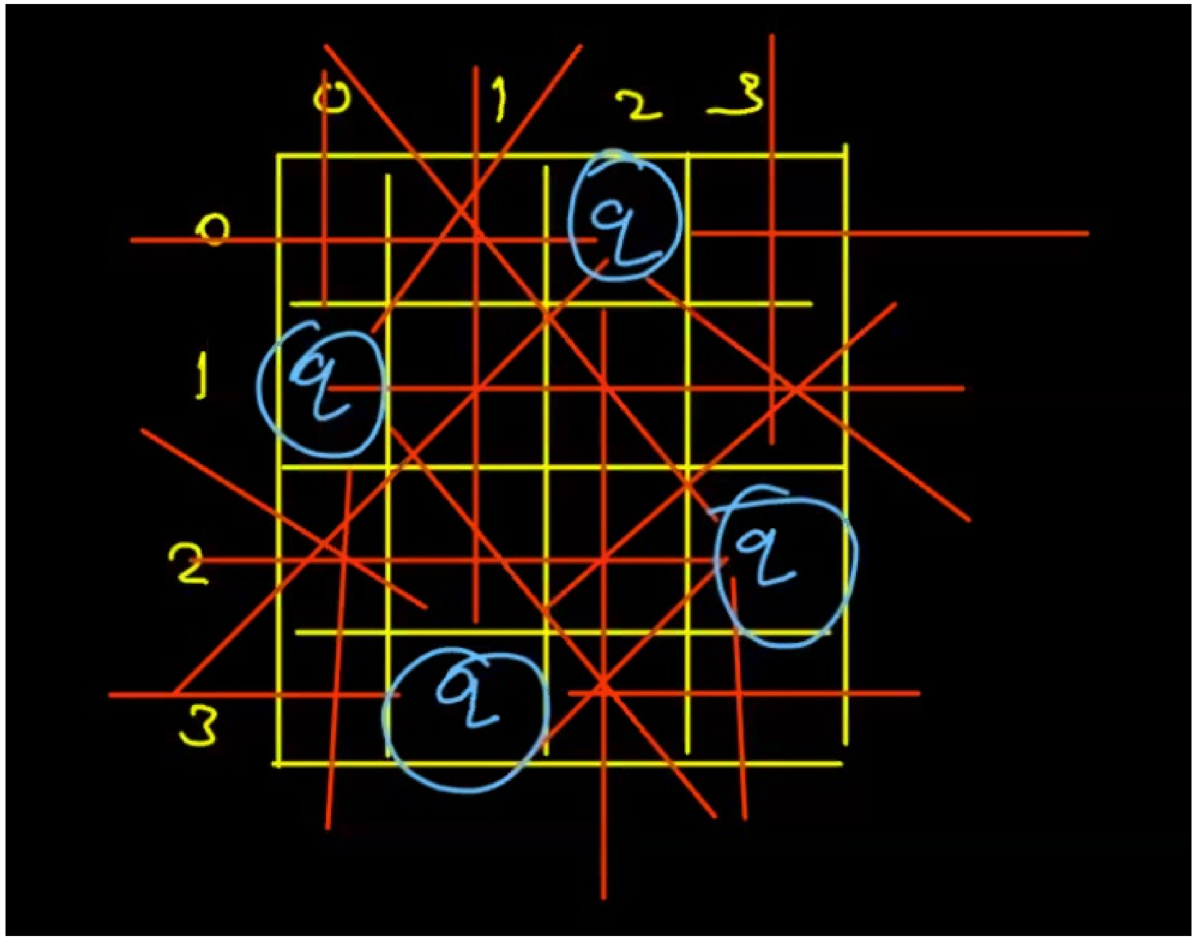


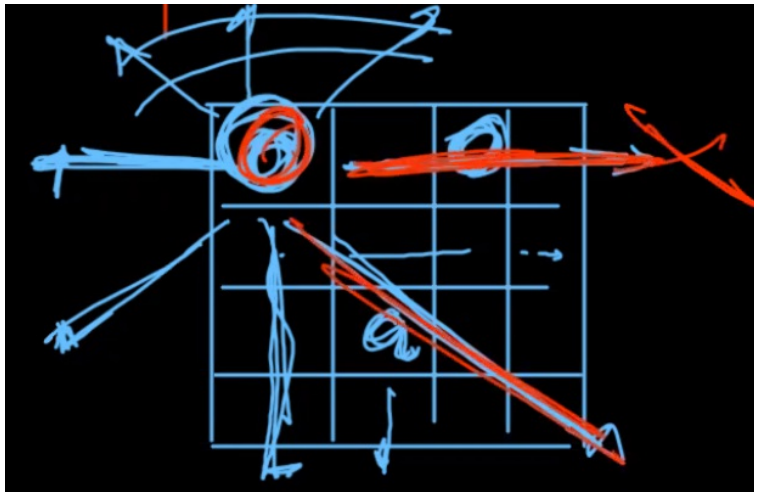
```
    // qpsf -> queen placed so far
    // asf -> answer so far
    public static void nqueen(int[][] board, int sr, int sc, int qpsf, String
asf) {
        if(sr == board.length) {
            if(qpsf == board.length)
                System.out.println(asf + ".");
            return;
        }

        if(sc + 1 < board[0].length) { // next column is valid
            // yes + isvalid
            if(isValidToPlace(board, sr, sc) == true) {
                board[sr][sc] = 1;
                nqueen(board, sr + 1, 0, qpsf + 1, asf + sr + "-" + sc + ", "
);
                board[sr][sc] = 0;
            }
            // no call
            nqueen(board, sr, sc + 1, qpsf, asf);
        } else { // next column is not valid
            // yes + isvalid
            if(isValidToPlace(board, sr, sc) == true) {
                board[sr][sc] = 1;
                nqueen(board, sr + 1, 0, qpsf + 1, asf + sr + "-" + sc + ", "
);
                board[sr][sc] = 0;
            }
            // no call
            nqueen(board, sr + 1, 0, qpsf, asf);
        }
    }
}
```



```
(0_1), (1_3), (2_0), (3_2),
(0_2), (1_0), (2_3), (3_1),
```



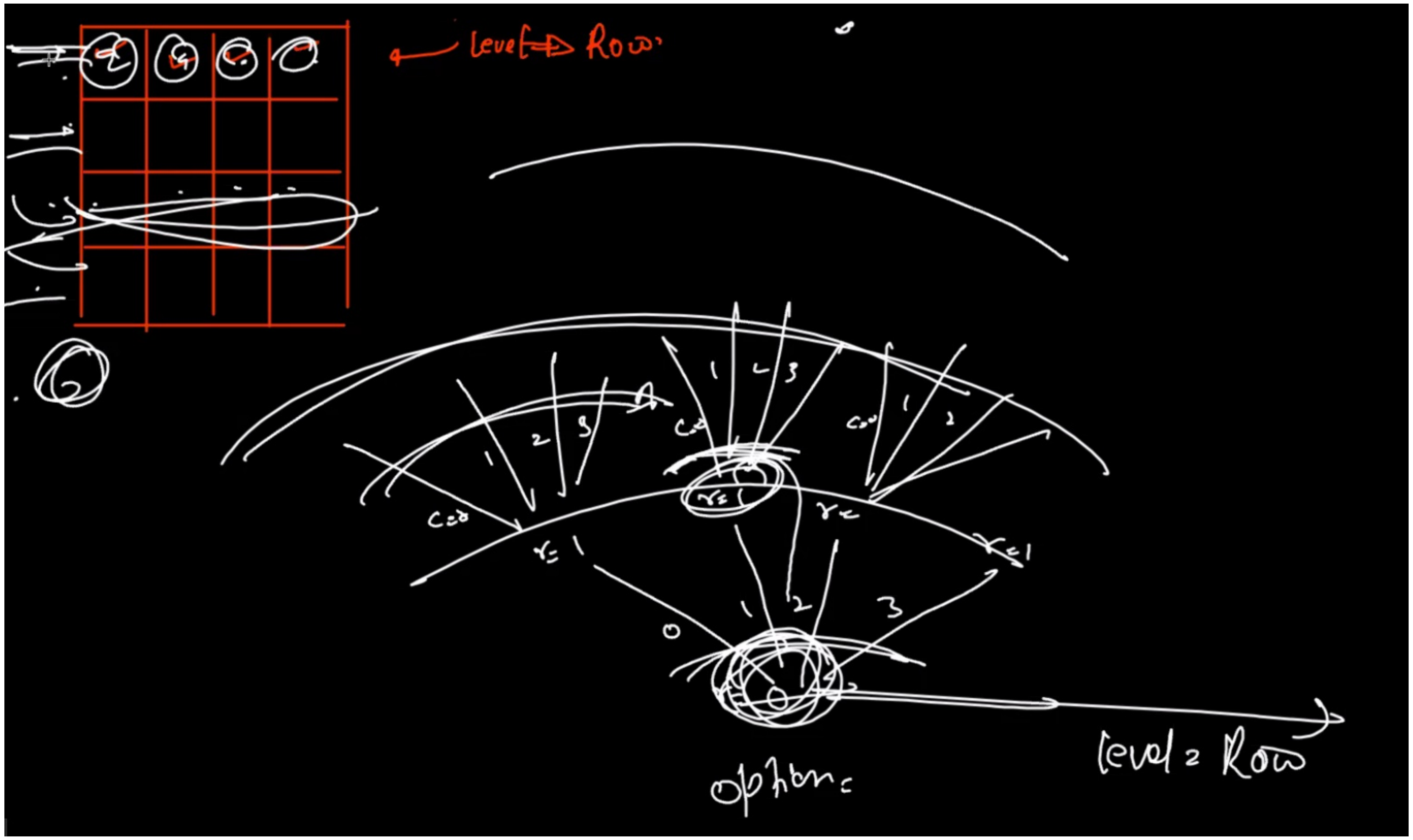


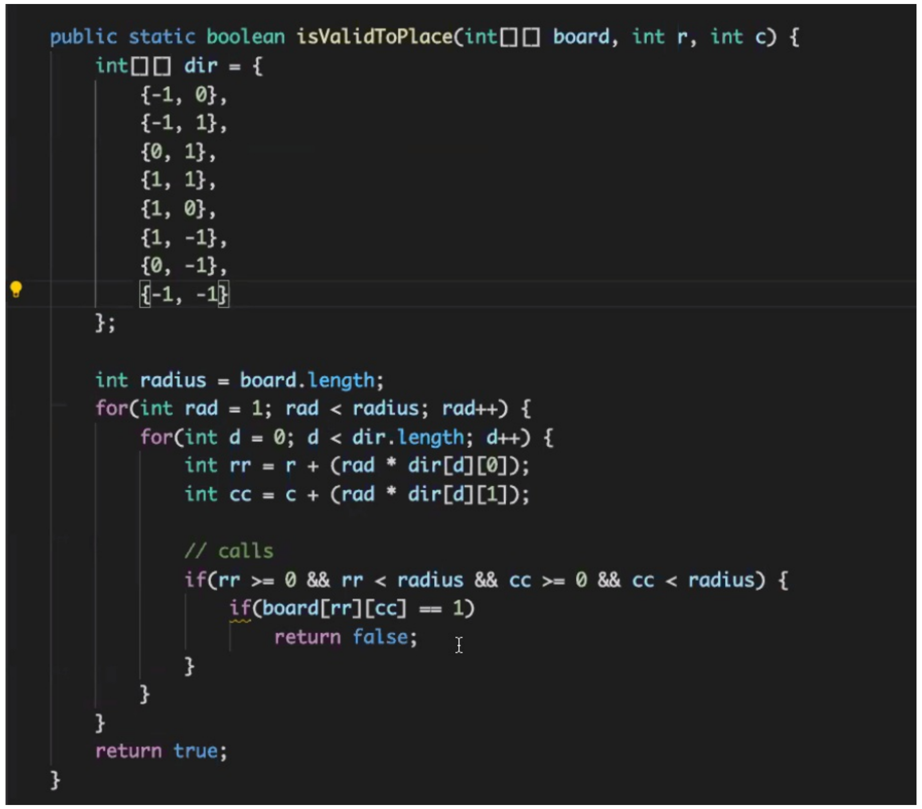


in terms of level option

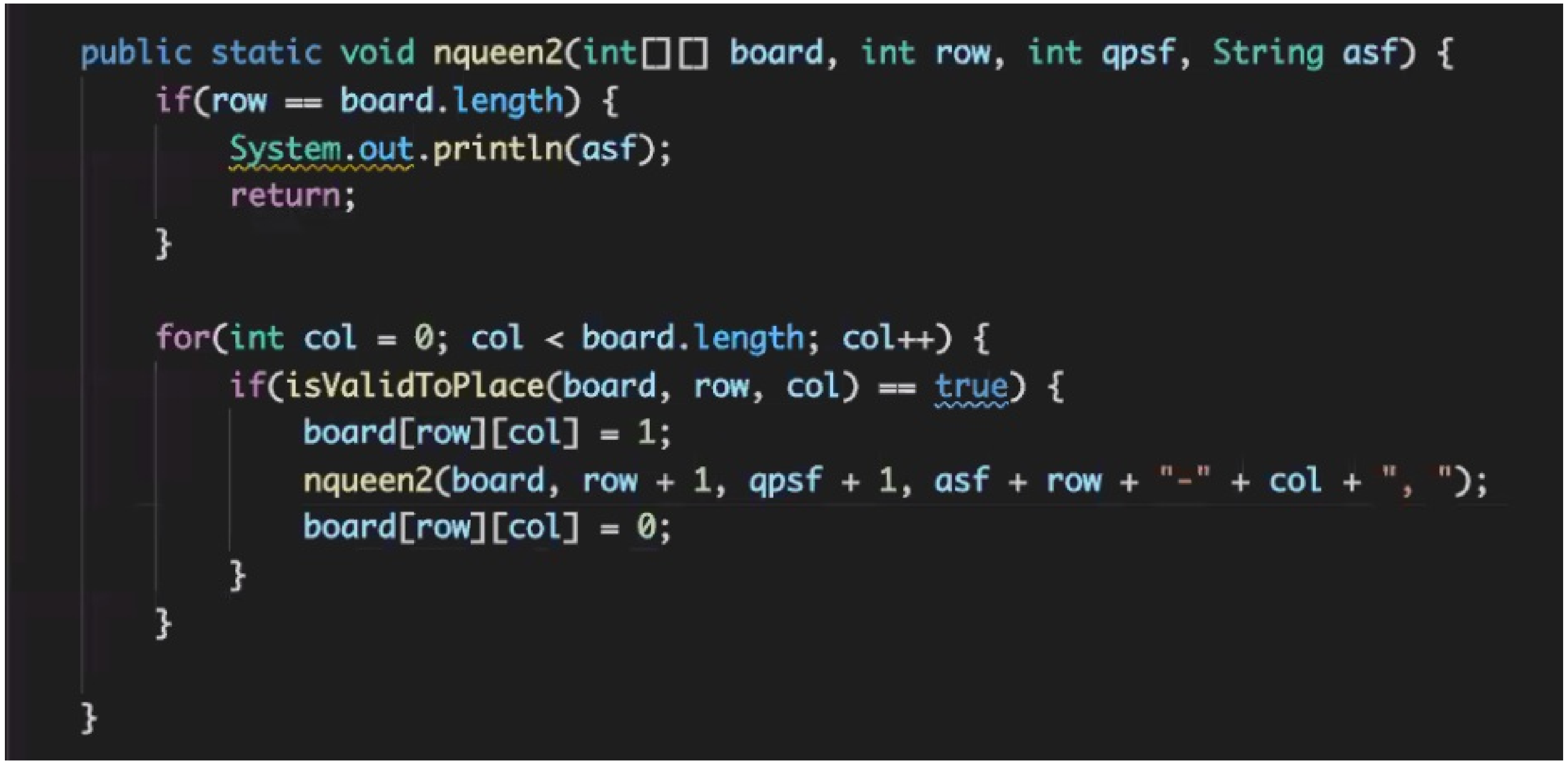
take row as level

take column as option

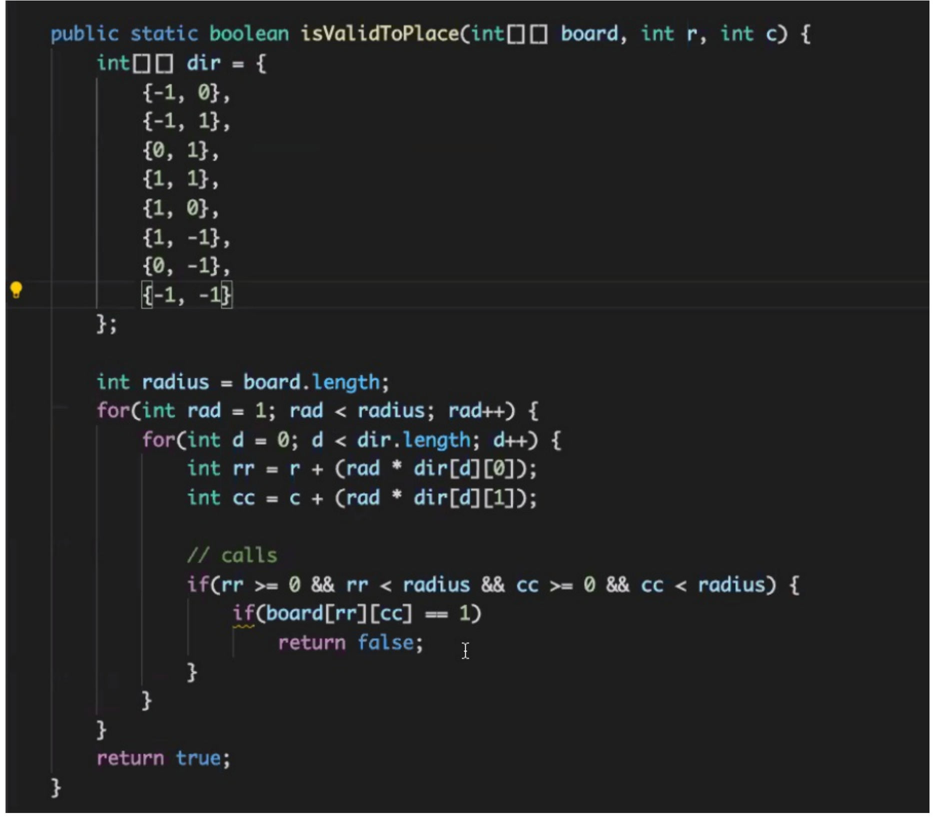




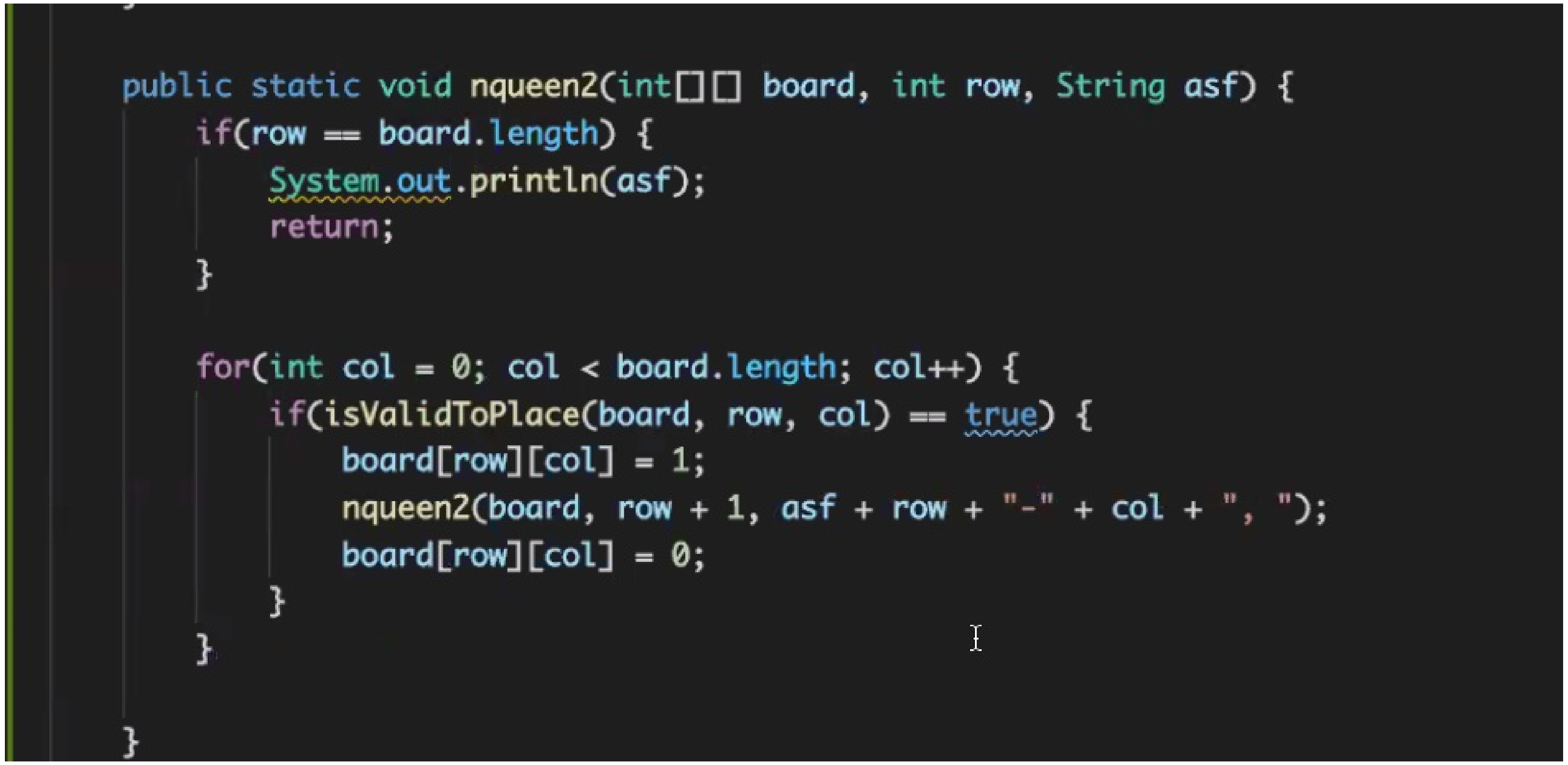
```java
public static boolean isValidToPlace(int[][] board, int r, int c) {
    int[][] dir = {
        {-1, 0},
        {-1, 1},
        {0, 1},
        {1, 1},
        {1, 0},
        {1, -1},
        {0, -1},
        {-1, -1}
    };

    int radius = board.length;
    for(int rad = 1; rad < radius; rad++) {
        for(int d = 0; d < dir.length; d++) {
            int rr = r + (rad * dir[d][0]);
            int cc = c + (rad * dir[d][1]);

            // calls
            if(rr >= 0 && rr < radius && cc >= 0 && cc < radius) {
                if(board[rr][cc] == 1)
                    return false;
            }
        }
    }
    return true;
}
```



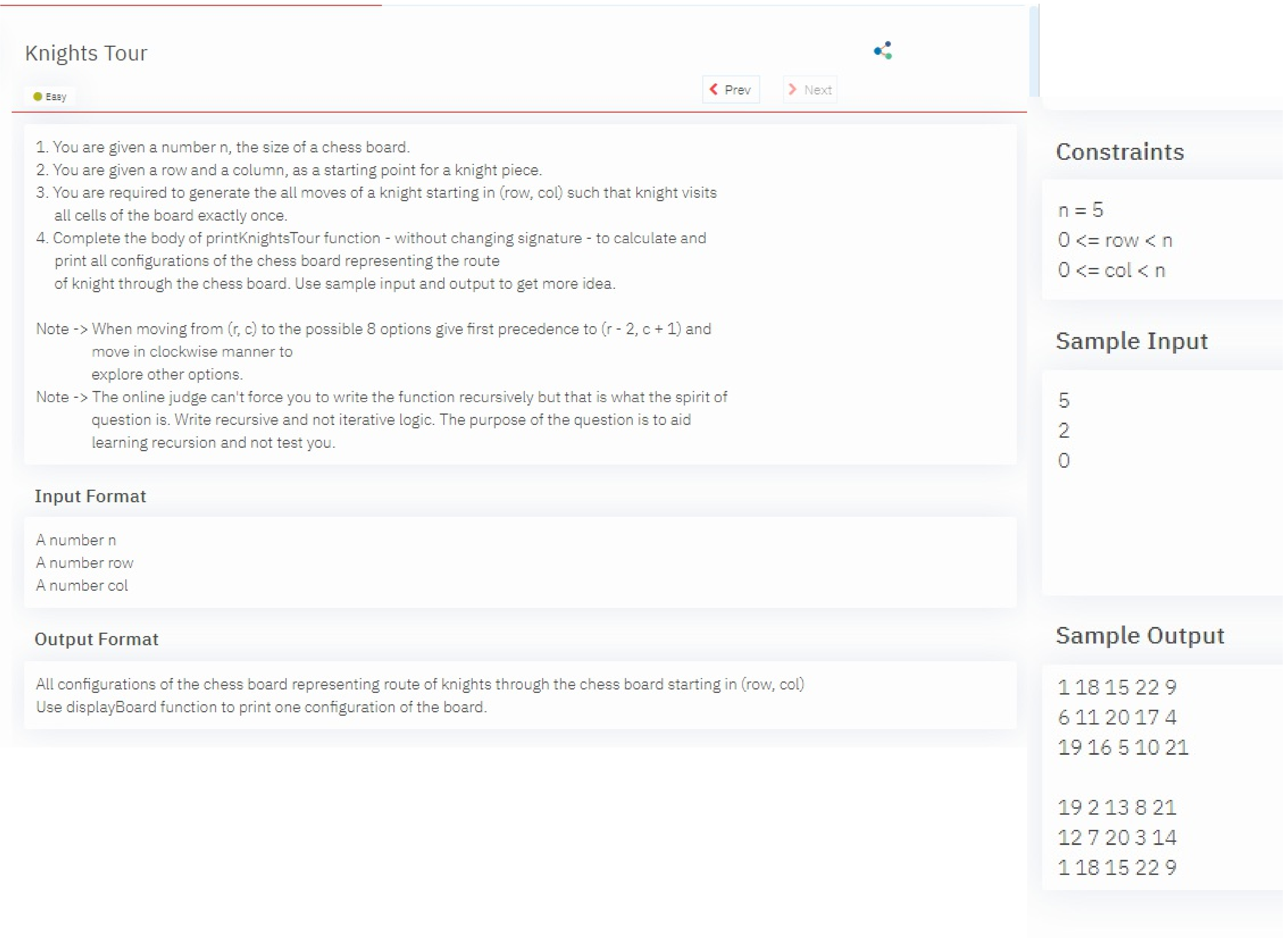
```java
public static void nqueen2(int[][] board, int row, int qpsf, String asf) {
    if(row == board.length) {
        System.out.println(asf);
        return;
    }

    for(int col = 0; col < board.length; col++) {
        if(isValidToPlace(board, row, col) == true) {
            board[row][col] = 1;
            nqueen2(board, row + 1, qpsf + 1, asf + row + "-" + col + ", ");
            board[row][col] = 0;
        }
    }

}
```



```java
public static boolean isValidToPlace(int[][] board, int r, int c) {
    int[][] dir = {
        {-1, 0},
        {-1, 1},
        {0, 1},
        {1, 1},
        {1, 0},
        {1, -1},
        {0, -1},
        {-1, -1}
    };

    int radius = board.length;
    for(int rad = 1; rad < radius; rad++) {
        for(int d = 0; d < dir.length; d++) {
            int rr = r + (rad * dir[d][0]);
            int cc = c + (rad * dir[d][1]);

            // calls
            if(rr >= 0 && rr < radius && cc >= 0 && cc < radius) {
                if(board[rr][cc] == 1)
                    return false;
            }
        }
    }
    return true;
}
```



```java
public static void nqueen2(int[][] board, int row, String asf) {
    if(row == board.length) {
        System.out.println(asf);
        return;
    }

    for(int col = 0; col < board.length; col++) {
        if(isValidToPlace(board, row, col) == true) {
            board[row][col] = 1;
            nqueen2(board, row + 1, asf + row + "-" + col + ", ");
            board[row][col] = 0;
        }
    }

}
```



Knights Tour

Easy

Prev | Next

1. You are given a number n, the size of a chess board.
2. You are given a row and a column, as a starting point for a knight piece.
3. You are required to generate the all moves of a knight starting in (row, col) such that knight visits all cells of the board exactly once.
4. Complete the body of printKnightsTour function - without changing signature - to calculate and print all configurations of the chess board representing the route of knight through the chess board. Use sample input and output to get more idea.

Note -> When moving from (r, c) to the possible 8 options give first precedence to (r - 2, c + 1) and move in clockwise manner to explore other options.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Input Format

A number n
A number row
A number col

Output Format

All configurations of the chess board representing route of knights through the chess board starting in (row, col)
Use displayBoard function to print one configuration of the board.

Constraints

n = 5
0 <= row < n
0 <= col < n

Sample Input

```
5
2
0
```

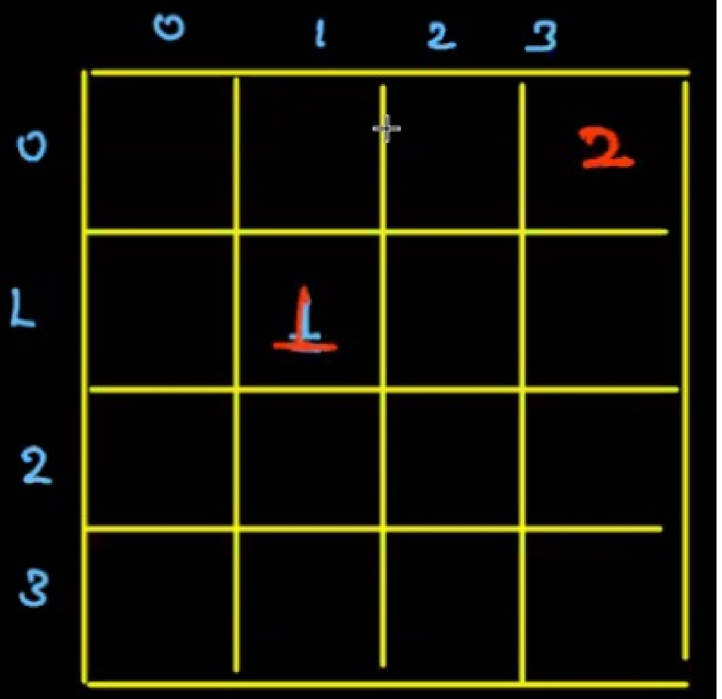
Sample Output

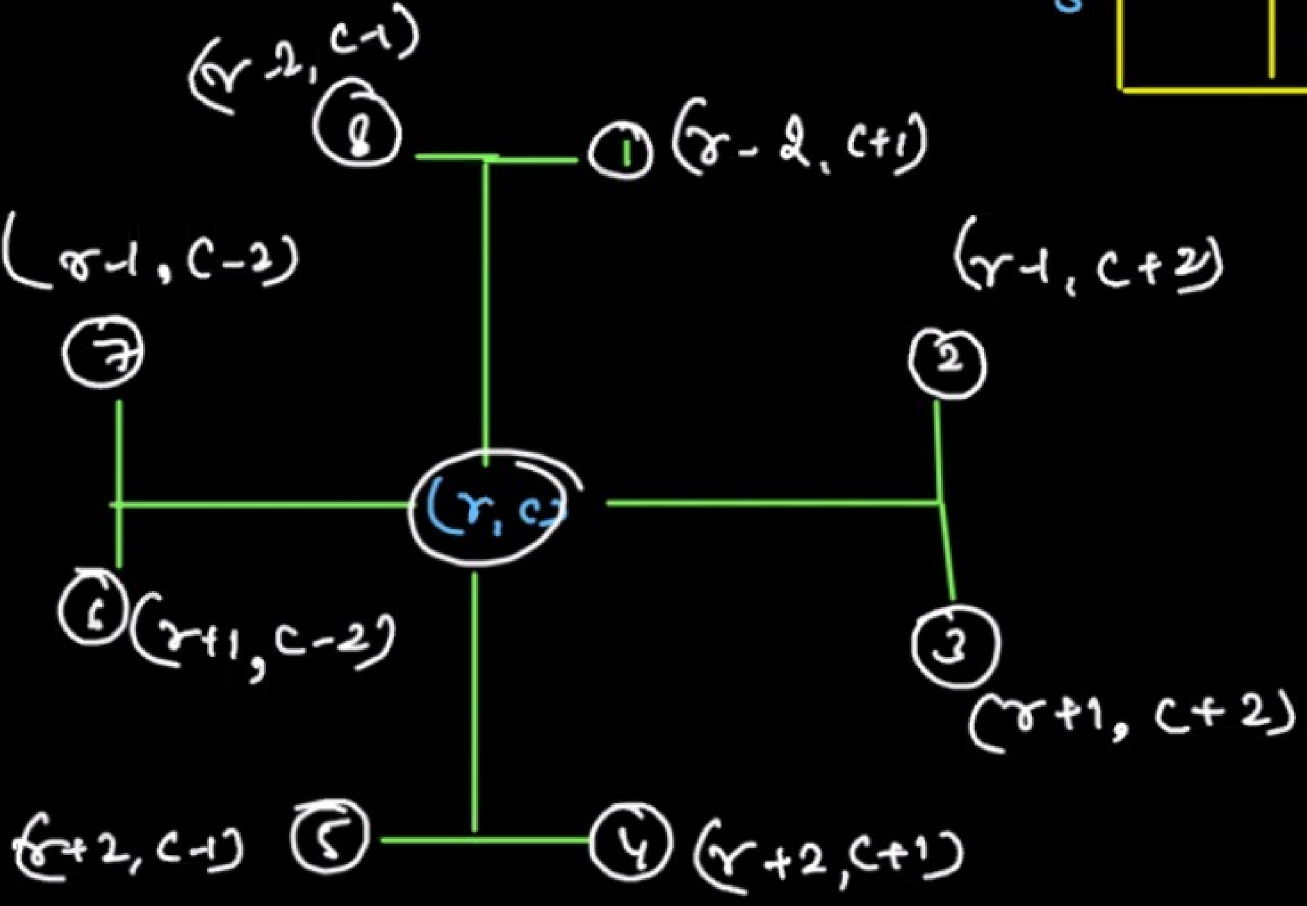
```
1 18 15 22 9
6 11 20 17 4
19 16 5 10 21

19 2 13 8 21
12 7 20 3 14
1 18 15 22 9
```

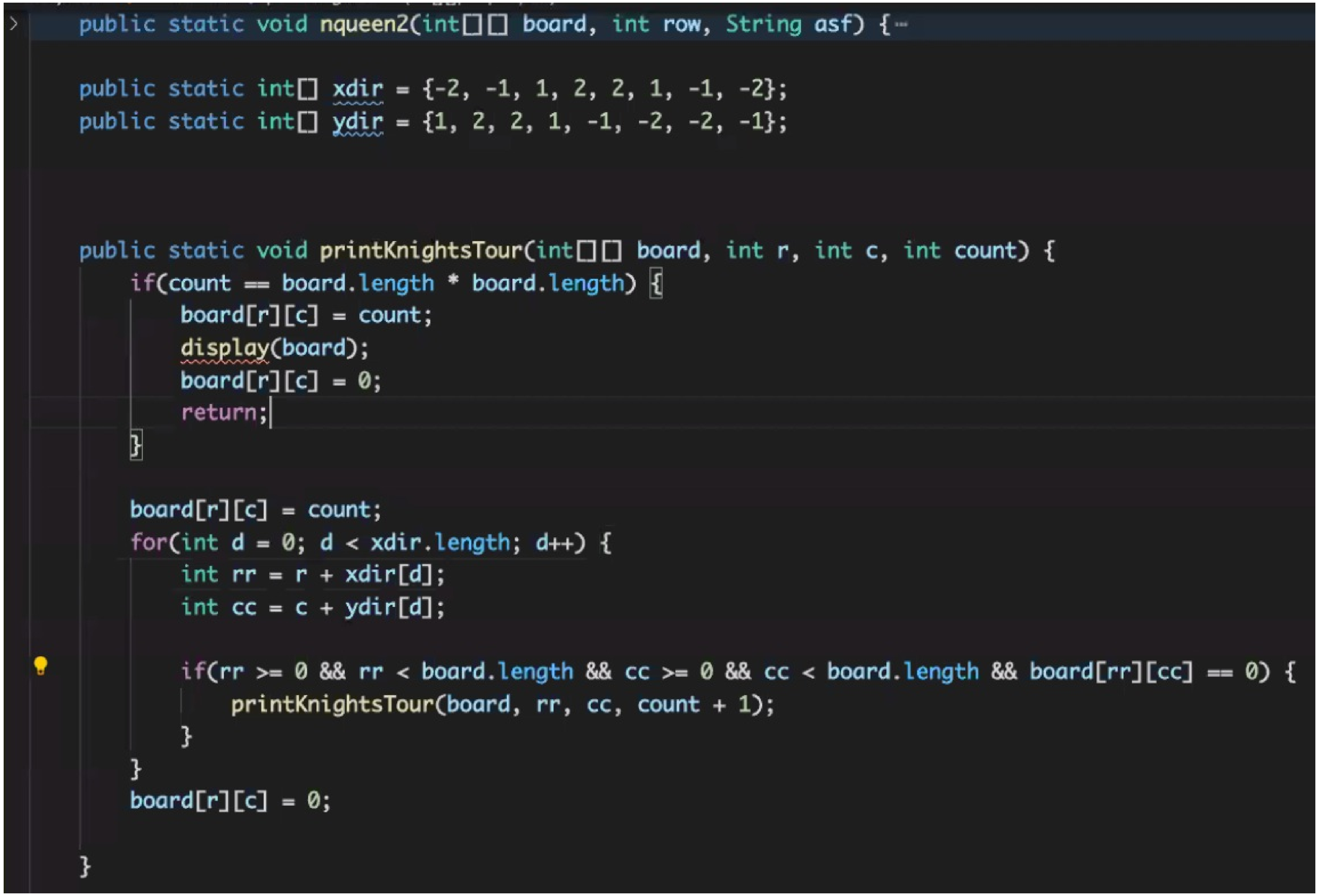
knights Tour

n x n - board

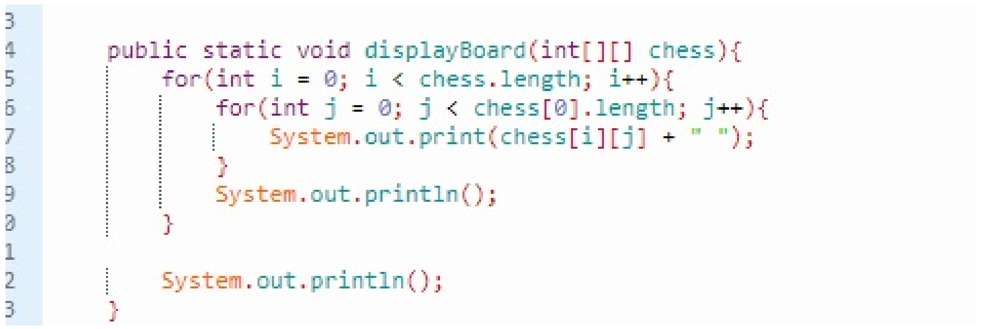
Initial point - (r, c)



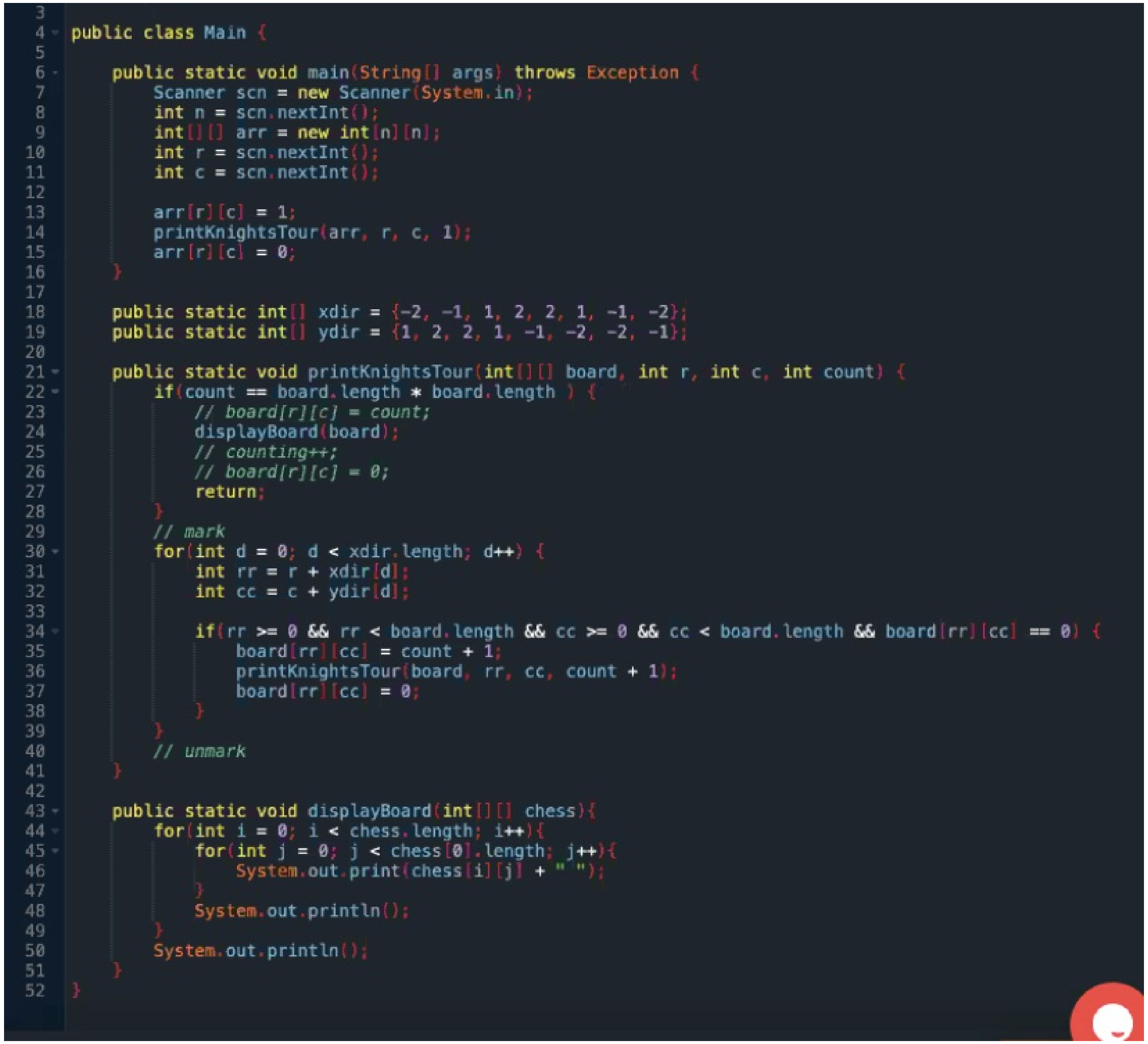




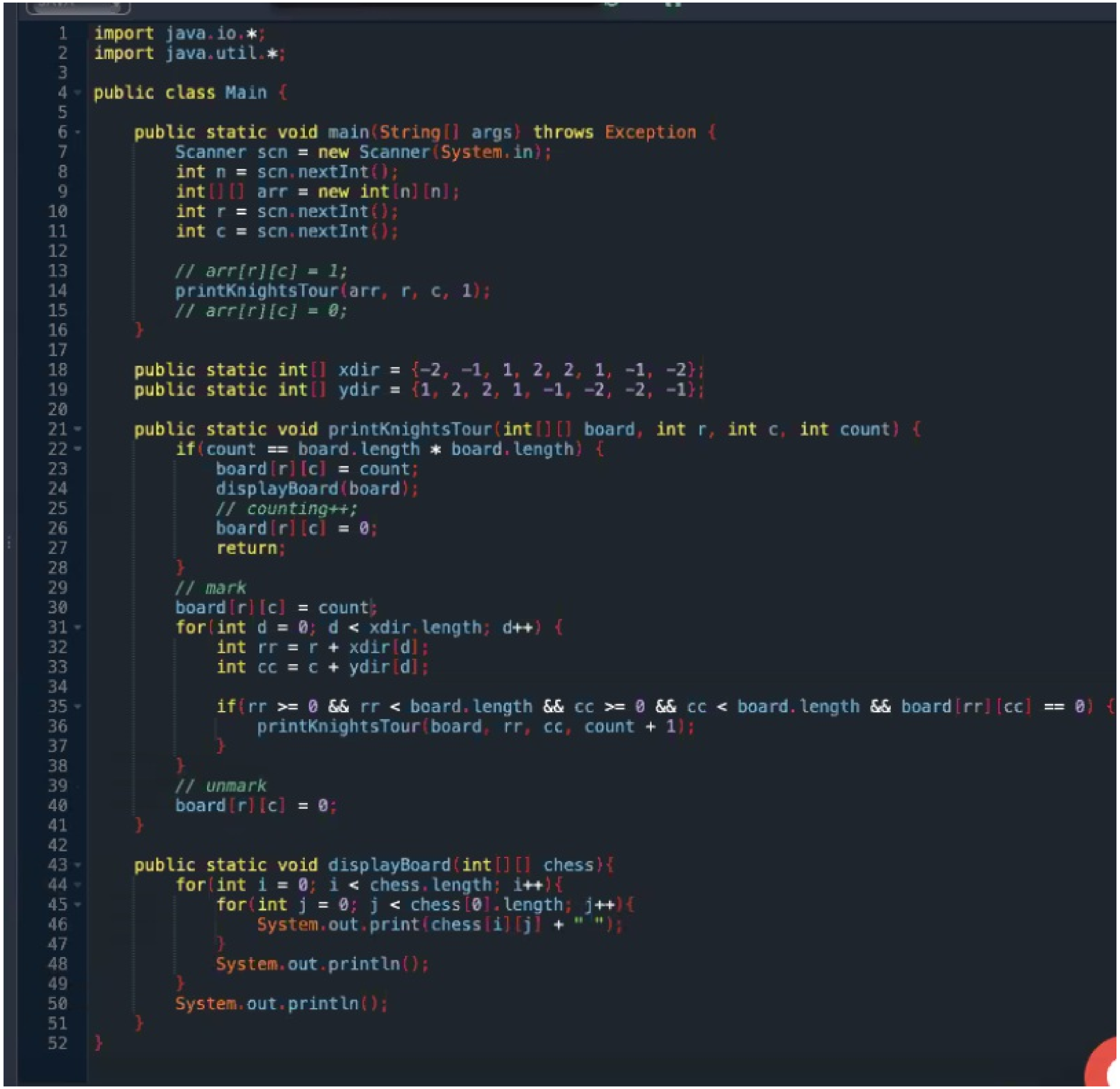
```java
public static void nqueen2(int[][] board, int row, String asf) {⋯

public static int[] xdir = {-2, -1, 1, 2, 2, 1, -1, -2};
public static int[] ydir = {1, 2, 2, 1, -1, -2, -2, -1};



public static void printKnightsTour(int[][] board, int r, int c, int count) {
    if(count == board.length * board.length) {
        board[r][c] = count;
        display(board);
        board[r][c] = 0;
        return;
    }

    board[r][c] = count;
    for(int d = 0; d < xdir.length; d++) {
        int rr = r + xdir[d];
        int cc = c + ydir[d];

        if(rr >= 0 && rr < board.length && cc >= 0 && cc < board.length && board[rr][cc] == 0) {
            printKnightsTour(board, rr, cc, count + 1);
        }
    }
    board[r][c] = 0;

}
```



```java

    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();
    }
```

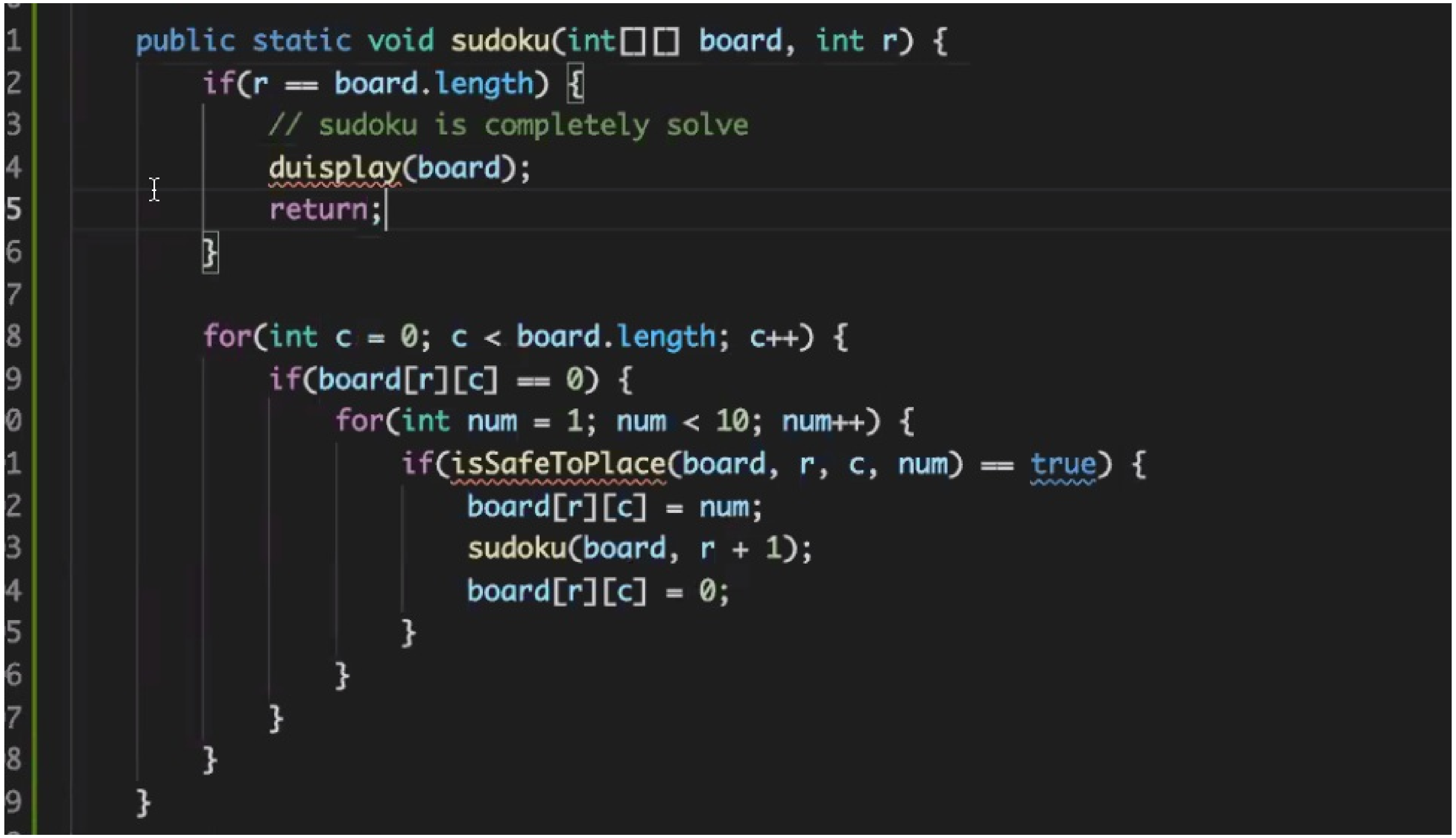


```java

public class Main {

    public static void main(String[] args) throws Exception {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int[][] arr = new int[n][n];
        int r = scn.nextInt();
        int c = scn.nextInt();

        arr[r][c] = 1;
        printKnightsTour(arr, r, c, 1);
        arr[r][c] = 0;
    }

    public static int[] xdir = {-2, -1, 1, 2, 2, 1, -1, -2};
    public static int[] ydir = {1, 2, 2, 1, -1, -2, -2, -1};

    public static void printKnightsTour(int[][] board, int r, int c, int count) {
        if(count == board.length * board.length ) {
            // board[r][c] = count;
            displayBoard(board);
            // counting++;
            // board[r][c] = 0;
            return;
        }
        // mark
        for(int d = 0; d < xdir.length; d++) {
            int rr = r + xdir[d];
            int cc = c + ydir[d];

            if(rr >= 0 && rr < board.length && cc >= 0 && cc < board.length && board[rr][cc] == 0) {
                board[rr][cc] = count + 1;
                printKnightsTour(board, rr, cc, count + 1);
                board[rr][cc] = 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();
    }
}
```

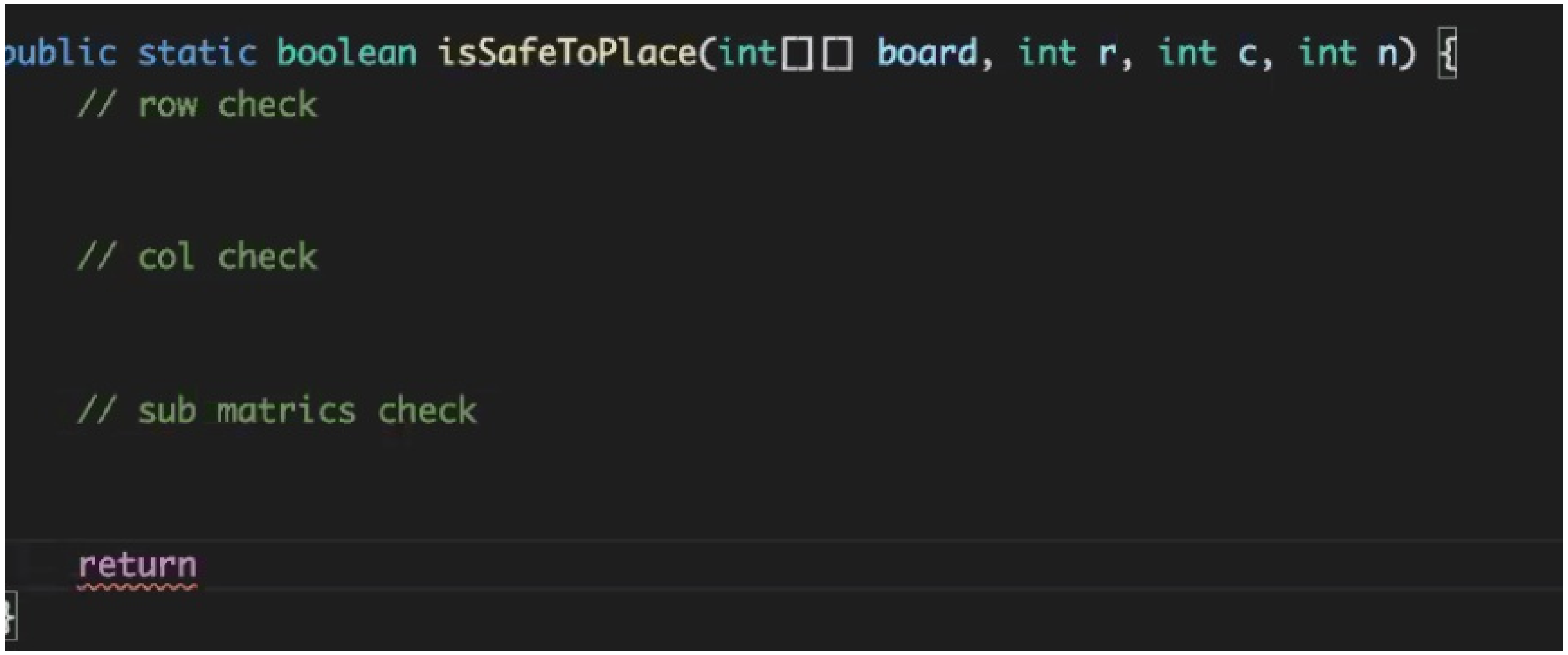


```java
import java.io.*;
import java.util.*;

public class Main {

    public static void main(String[] args) throws Exception {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int[][] arr = new int[n][n];
        int r = scn.nextInt();
        int c = scn.nextInt();

        // arr[r][c] = 1;
        printKnightsTour(arr, r, c, 1);
        // arr[r][c] = 0;
    }

    public static int[] xdir = {-2, -1, 1, 2, 2, 1, -1, -2};
    public static int[] ydir = {1, 2, 2, 1, -1, -2, -2, -1};

    public static void printKnightsTour(int[][] board, int r, int c, int count) {
        if(count == board.length * board.length) {
            board[r][c] = count;
            displayBoard(board);
            // counting++;
            board[r][c] = 0;
            return;
        }
        // mark
        board[r][c] = count;
        for(int d = 0; d < xdir.length; d++) {
            int rr = r + xdir[d];
            int cc = c + ydir[d];

            if(rr >= 0 && rr < board.length && cc >= 0 && cc < board.length && board[rr][cc] == 0) {
                printKnightsTour(board, rr, cc, count + 1);
            }
        }
        // unmark
        board[r][c] = 0;
    }

    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();
    }
}
```

Sudoku



```java
public static void sudoku(int[][] board, int r) {
    if(r == board.length) {
        // sudoku is completely solve
        duisplay(board);
        return;
    }

    for(int c = 0; c < board.length; c++) {
        if(board[r][c] == 0) {
            for(int num = 1; num < 10; num++) {
                if(isSafeToPlace(board, r, c, num) == true) {
                    board[r][c] = num;
                    sudoku(board, r + 1);
                    board[r][c] = 0;
                }
            }
        }
    }
}
```



```
ublic static boolean isSafeToPlace(int[][] board, int r, int c, int n) {
    // row check


    // col check


    // sub matrics check


    return
}
```