Previous.

NQueens.

2. s → "geeks" → geeks ⇒ for( ) s.charAt(i); Sys.o.ln();

3. s → "geeks"
   r → "geeps";

s.length() != r.length()

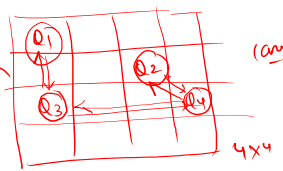
N Queens :- Check:

N = 4

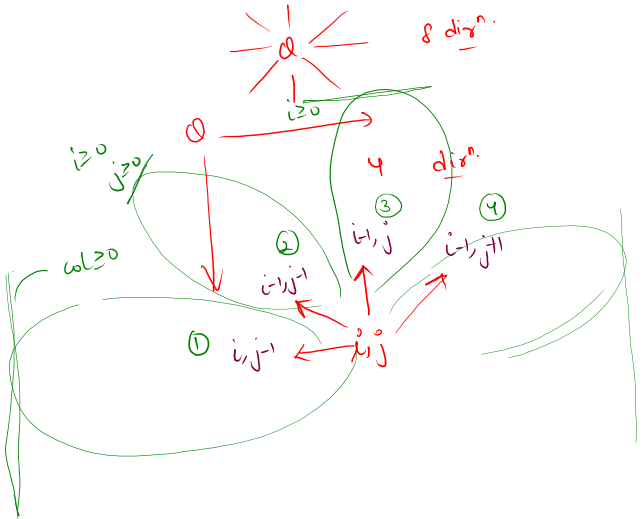
N Queens

N x N Board

"N Queens" → All Queens are safe.




(any) atmax 1 Queen is not safe ("Danger.")




col < n
& i ≥ 0

# Count All Palindromic Rows

Problem | Submissions | Leaderboard | Discussions

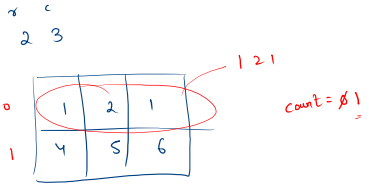
Take a **2-D array** of size m*n as input and **count** the number of **Palindromic Rows** present in the 2-D array.

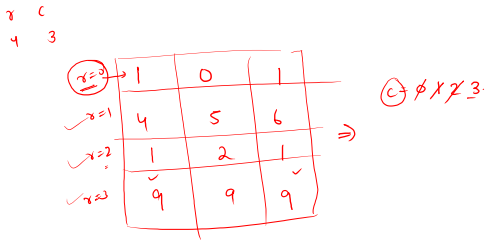
### Sample Input 0

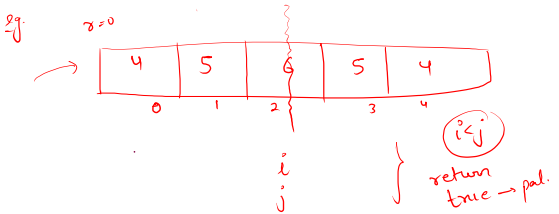
```
2 3
1 2 1
4 5 6
```

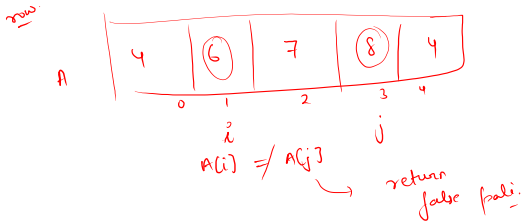
### Sample Output 0

```
1
```




```java
import java.util.*;

public class Solution {
    public static boolean isPalindrome(int [][] A, int row){
        //return true if this row is palindrome else false

        int i = 0;
        int j = A[0].length-1;

        while(i < j){
            if(A[row][i] != A[row][j]){
                return false;
            }
            i++;
            j--;
        }
        return true;

    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int r = scn.nextInt();
        int c = scn.nextInt();
        int [][] A = new int[r][c];
        for(int i = 0; i < r; i++){
            for(int j = 0; j < c; j++){
                A[i][j] = scn.nextInt();
            }
        }

        int count  = 0;
        for(int row = 0; row < r ; row++){
            // in one iteration u hav t0 focus only on individual row
            if(isPalindrome(A, row)){
                count++;
            }

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

# Rotation Check In Matrix

Problem | Submissions | Leaderboard | Discussions

Check whether all rows of a matrix are circular rotations of each other.

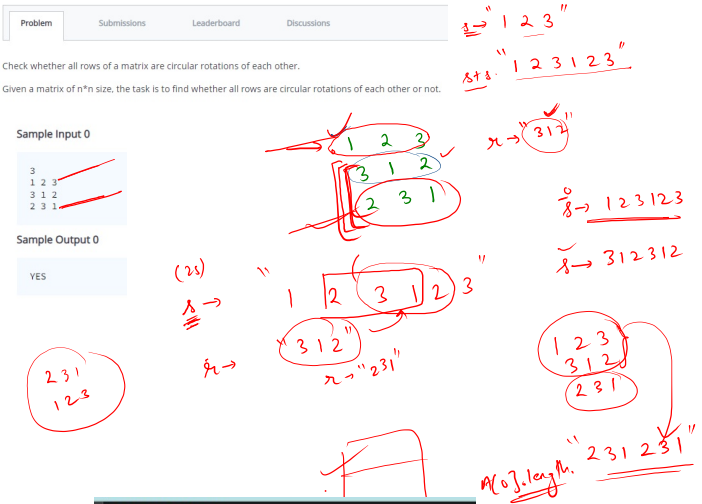
Given a matrix of n*n size, the task is to find whether all rows are circular rotations of each other or not.

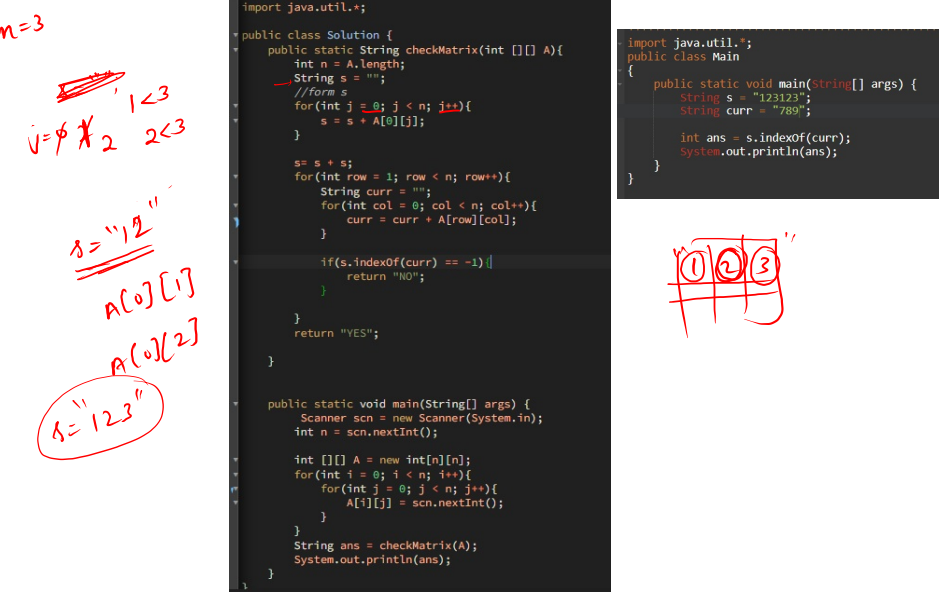
**Sample Input 0**

```
3
1 2 3
3 1 2
2 3 1
```

**Sample Output 0**

```
YES
```





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

public class Solution {
    public static String checkMatrix(int [][] A){
        int n = A.length;
        String s = "";
        //form s
        for(int j = 0; j < n; j++){
            s = s + A[0][j];
        }

        s= s + s;
        for(int row = 1; row < n; row++){
            String curr = "";
            for(int col = 0; col < n; col++){
                curr = curr + A[row][col];
            }

            if(s.indexOf(curr) == -1){
                return "NO";
            }

        }
        return "YES";

    }


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

        int [][] A = new int[n][n];
        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
                A[i][j] = scn.nextInt();
            }
        }
        String ans = checkMatrix(A);
        System.out.println(ans);
    }
}
```

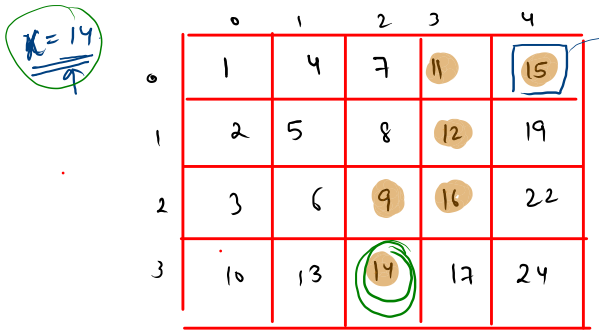
```java
import java.util.*;
public class Main
{
    public static void main(String[] args) {
        String s = "123123";
        String curr = "789";

        int ans = s.indexOf(curr);
        System.out.println(ans);
    }
}
```

## Search in a sorted matrix

Given a m*n matrix and you are also given an integer x. Each row and column of the matrix is sorted in increasing order. You are required to find x in the matrix and print it's location int (row, col) format as discussed in output format below. In case an element is not found, print "Not Found".

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




**Input:** mat[4][4] = { {10, 20, 30, 40}, x = 29
{15, 25, 35, 45},
{27, 29, 37, 48},
{32, 33, 39, 50}}

**Output:** Found at (2, 1)

```
5 4
1 3 5 7
10 11 16 20
23 30 34 60
62 63 65 68
71 72 74 80
65
```