Problem : Find the Peak Element in a 2D Array/Matrix

Problem Statement

Given a 2D matrix mat[][], identify any `peak element` within the matrix.

An element is considered a peak if it is greater than or equal to its four immediate neighbors: `top, bottom, left, and right`. For corner and edge elements, any missing neighbors are treated as having a value of `negative infinity`.

A peak element is not necessarily the global maximum, it only needs to satisfy the condition relative to its adjacent elements. `Multiple peak elements may exist`, and the algorithm may return any one of them.

- Write a function `findPeakElement()` that take the input matrix and returns the first peak element.
- You must also **write a `main` method** that performs the following steps:
    1. **Takes input** for the matrix from the user.
    2. **Validates the input matrix format**, ensuring it contains only numbers separated by commas (with optional spaces).
    3. **Calls the `findPeakElement()` function** with the validated matrix.
    4. **Displays the result** appropriately.

Example 1

```
Input: matrix[][] = [
            [10, 20, 15],
            [21, 30, 14],
            [7,  16, 32]
        ]

Output: [1, 1]

Explanation:
The value at index {1, 1} is 30, which is greater than or equal to all its valid
neighbors: Left = 21, Right = 14, Top = 20, Bottom = 16. So, it satisfies the peak
condition. Alternatively, {2, 2} with value 32 also qualifies as a peak.
```

Example 2

```
Input: mat[][] = [
            [17, 7],
            [11, 10]
        ]

Output: [0, 0]
```

```
Explanation:
The value at index {0 0} is 17. Its neighbors are: Right= 7, Bottom = 11. Since 17
is greater than or equal to both (and top and left are out of bounds), it
qualifies as a peak element.
```

Code

```java
import java.util.*;

class Solution {
    public static int[] findPeakElement(int[][] matrix){
        int n = matrix.length;
        int m = matrix[0].length;

        int ans[]={-1,-1};

        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                int curr = matrix[i][j];
                boolean isPeak = true;

                if (i > 0 && matrix[i - 1][j] > curr)
                    isPeak = false;

                if (i + 1 < n && matrix[i + 1][j] > curr)
                    isPeak = false;

                if (j > 0 && matrix[i][j - 1] > curr)
                    isPeak = false;

                if (j + 1 < m && matrix[i][j + 1] > curr)
                    isPeak = false;

                if (isPeak) {
                    ans[0]=i;
                    ans[1]=j;
                    return ans;
                }
            }
        }

        return ans;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter number of rows:");
        int rows = sc.nextInt();
        System.out.println("Enter number of columns:");
        int cols = sc.nextInt();
```

```
        sc.nextLine(); // newline

        int[][] matrix = new int[rows][cols];
        System.out.println("Enter the matrix row by row (comma-separated
values):");

        for (int i = 0; i < rows; i++) {
            String line = sc.nextLine().trim();
            if (!line.matches("^[0-9,\\s-]+$")) {
                System.out.println("Invalid input format. Only numbers and commas
are allowed.");
                return;
            }

            String[] values = line.split(",");
            if (values.length != cols) {
                System.out.println("Invalid number of columns in row " + (i + 1));
                return;
            }

            for (int j = 0; j < cols; j++) {
                matrix[i][j] = Integer.parseInt(values[j].trim());
            }
        }

        System.out.println();
        System.out.println("Input matrix: ");
        for(int arr[] : matrix){
            System.out.println(Arrays.toString(arr));
        }

        int result[] = findPeakElement(matrix);

        System.out.println();
        System.out.println("Peak Element : "+Arrays.toString(result));
    }
}
```

Terminal Output

```
$ java Solution
Enter number of rows:
3
Enter number of columns:
3
Enter the matrix row by row (comma-separated values):
10, 20, 15
21, 30, 14
7,  16, 32

Input matrix:
```

```
[10, 20, 15]
[21, 30, 14]
[7, 16, 32]

Peak Element : [1, 1]
```

Problem : Flood Fill Algorithm

Problem Statement

You are given a 2D grid image[][], where each `image[i][j]` represents the color of a pixel in the image. Also provided is a coordinate `(sr, sc)` representing the starting pixel (row and column) and a new color value newColor.

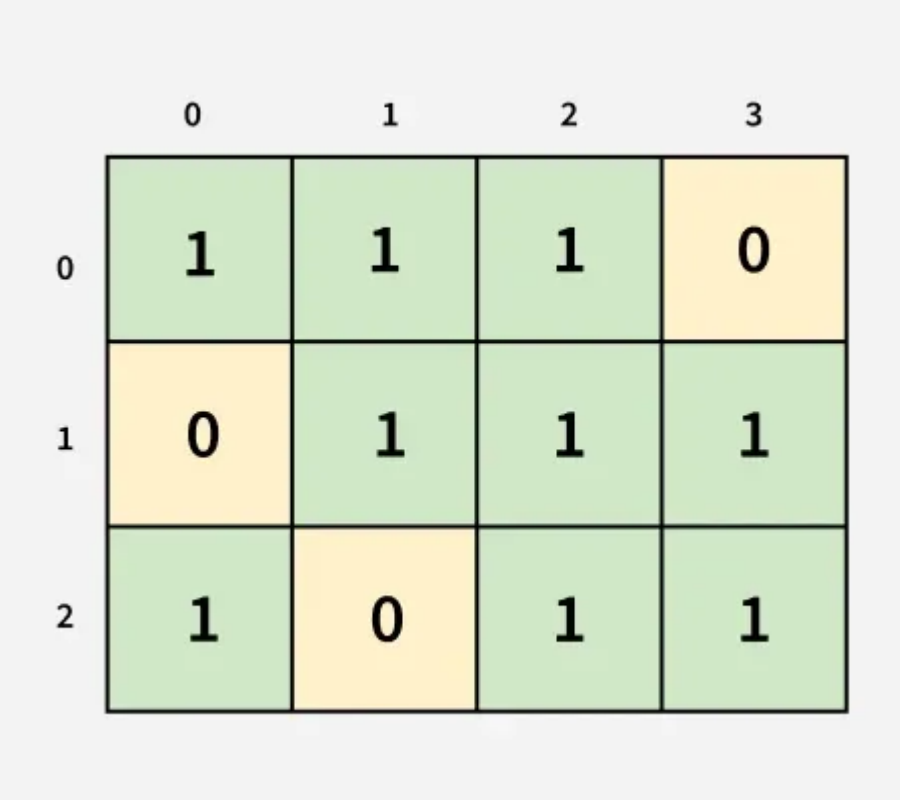
Your task is to perform a flood fill starting from the pixel (sr, sc), changing its color and the color of all connected pixels that have the same original color.

Two pixels are considered connected if they are `adjacent horizontally or vertically (not diagonally)` and have the `same original color`.

- Write a function, `floodFill()`, that Flood-Fills the given matrix.
- You must also **write a `main` method** that performs the following steps:
    1. **Takes input** for the matrix, starting coordinates (sr,sc) and new-color from the user.
    2. **Validates the input matrix format**, ensuring it contains only numbers separated by commas (with optional spaces).
    3. **Calls the `floodFill()` function** with the validated matrix.
    4. **Displays the result** appropriately.

Example:

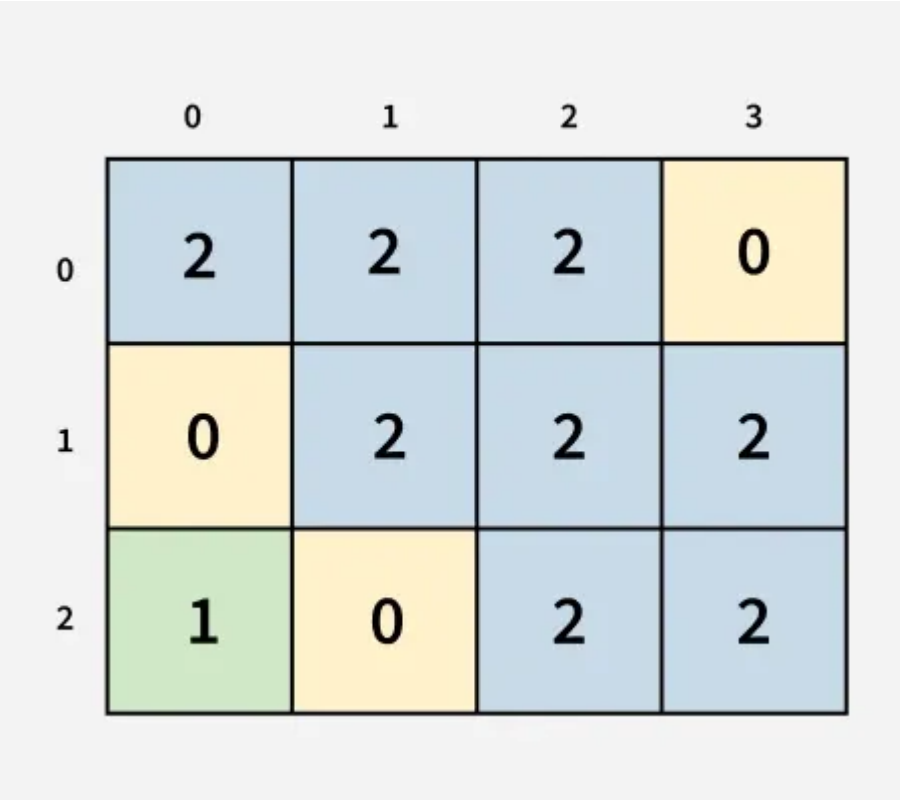
Input



```
image = [
        [1, 1, 1, 0],
        [0, 1, 1, 1],
        [1, 0, 1, 1]
    ],

sr = 1,
sc = 2,
newColor = 2
```

Output




```
image = [
        [2, 2, 2, 0],
        [0, 2, 2, 2],
        [1, 0, 2,2]
    ]
```

Explanation:

```
Starting from pixel (1, 2) with value 1, flood fill updates all connected pixels
(up, down, left, right) with value 1 to 2, resulting in [[2, 2, 2, 0], [0, 2, 2,
2], [1, 0, 2, 2]].
```

Code

```java
import java.util.*;

class Solution {
    private static final int[][] directions = {
        {1, 0}, {-1, 0}, {0, 1}, {0, -1}
    };

    public static int[][] floodFill(int[][] image, int sr, int sc, int newColor) {

        if (image[sr][sc] == newColor) {
            return image;
        }

        int oldColor = image[sr][sc];

        Queue<int[]> q = new LinkedList<>();
        q.offer(new int[]{sr, sc});

        image[sr][sc] = newColor;

        while (!q.isEmpty()) {
            int[] front = q.poll();
            int x = front[0], y = front[1];

            for (int[] direction : directions) {
                int nx = x + direction[0];
                int ny = y + direction[1];

                if (nx >= 0 && nx < image.length &&
                    ny >= 0 && ny < image[0].length &&
                    image[nx][ny] == oldColor) {

                    image[nx][ny] = newColor;
                    q.offer(new int[]{nx, ny});
                }
            }
        }

        return image;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter number of rows:");
        int rows = sc.nextInt();
        System.out.println("Enter number of columns:");
        int cols = sc.nextInt();
        sc.nextLine(); // newline

        int[][] matrix = new int[rows][cols];
        System.out.println("Enter the matrix row by row (comma-separated
```

```
values):");

        for (int i = 0; i < rows; i++) {
            String line = sc.nextLine().trim();
            if (!line.matches("^[0-9,\\s-]+$")) {
                System.out.println("Invalid input format. Only numbers and commas
are allowed.");
                return;
            }

            String[] values = line.split(",");
            if (values.length != cols) {
                System.out.println("Invalid number of columns in row " + (i + 1));
                return;
            }

            for (int j = 0; j < cols; j++) {
                matrix[i][j] = Integer.parseInt(values[j].trim());
            }
        }

        System.out.println("Enter the starting x-coordinate");
        int starting_x = sc.nextInt();
        System.out.println("Enter the starting y-coordinate");
        int starting_y = sc.nextInt();
        System.out.println("Enter the new color value (integer).");
        int new_color = sc.nextInt();

        System.out.println();
        System.out.println("Input image: ");
        for(int arr[] : matrix){
            System.out.println(Arrays.toString(arr));
        }

        int newImage[][] = floodFill(matrix,starting_x,starting_y,new_color);

        System.out.println();
        System.out.println("Output image: ");
        for(int arr[] : newImage){
            System.out.println(Arrays.toString(arr));
        }
    }
}
```

Terminal Output

```
$ java Solution
Enter number of rows:
3
Enter number of columns:
4
```

```
Enter the matrix row by row (comma-separated values):
1, 1, 1, 0
0, 1, 1, 1
1, 0, 1, 1
Enter the starting x-coordinate
1
Enter the starting y-coordinate
2
Enter the new color value (integer).
2

Input image:
[1, 1, 1, 0]
[0, 1, 1, 1]
[1, 0, 1, 1]

Output image:
[2, 2, 2, 0]
[0, 2, 2, 2]
[1, 0, 2, 2]
```

Problem : Rotate Image / Matrix

Problem Statement

You are given an n x n 2D matrix representing an image, rotate the image by 90 degrees (clockwise).

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. DO NOT allocate another 2D matrix and do the rotation.

Write a function, `rotateImage(int[][] matrix)`, that rotates the given matrix.

You must also **write a `main` method** that performs the following steps:

1. **Takes input** for the matrix from the user.
2. **Validates the input matrix format**, ensuring it contains only numbers separated by commas (with optional spaces).
3. **Calls the `rotateImage()` function** with the validated matrix.
4. **Displays the result** appropriately.

Example 1 :

Input:

```
matrix = [
  [1,2,3],
  [4,5,6],
  [7,8,9]
]
```

Output:

```
[
    [7,4,1],
    [8,5,2],
    [9,6,3]
]
```

Example 2 :

Input:

```
matrix = [
  [5,1,9,11],
  [2,4,8,10],
  [13,3,6,7],
```

```
  [15,14,12,16]
]
```

Output:

```
[
   [15,13,2,5],
   [14,3,4,1],
   [12,6,8,9],
   [16,7,10,11]
]
```

Code

```
import java.util.*;

class Solution{

    public static void rotateImage(int matrix[][]){
        //transpose
        for(int i=0; i<matrix.length; i++){
            for(int j=0; j<=i; j++){
                //swap
                int temp = matrix[i][j];
                matrix[i][j] = matrix[j][i];
                matrix[j][i] = temp;
            }
        }

        for(int i=0; i<matrix.length; i++){ // 2-pointer reversal

            int left = 0;
            int right = matrix[i].length-1;

            while(left<right){
                int temp = matrix[i][left];
                matrix[i][left] = matrix[i][right];
                matrix[i][right] = temp;

                left++;
                right--;
            }
        }
    }

    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter number of rows:");
```

```
        int rows = sc.nextInt();
        System.out.println("Enter number of columns:");
        int cols = sc.nextInt();
        sc.nextLine(); // consume newline

        int[][] matrix = new int[rows][cols];
        System.out.println("Enter the matrix row by row (comma-separated
values):");

        for (int i = 0; i < rows; i++) {
            String line = sc.nextLine().trim();
            if (!line.matches("^[0-9,\\s-]+$")) {
                System.out.println("Invalid input format. Only numbers and commas
are allowed.");
                return;
            }

            String[] values = line.split(",");
            if (values.length != cols) {
                System.out.println("Invalid number of columns in row " + (i + 1));
                return;
            }

            for (int j = 0; j < cols; j++) {
                matrix[i][j] = Integer.parseInt(values[j].trim());
            }
        }


        System.out.println();
        System.out.println("Input matrix: ");
        for(int arr[] : matrix){
            System.out.println(Arrays.toString(arr));
        }

        rotateImage(matrix);

        System.out.println();
        System.out.println("Output matrix: ");
        for(int arr[] : matrix){
            System.out.println(Arrays.toString(arr));
        }
    }
}
```

Terminal Output

```
$ java Solution
Enter number of rows:
3
Enter number of columns:
```

```
3
Enter the matrix row by row (comma-separated values):
1,2,3
4,5,6
7,8,9

Input matrix:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

Output matrix:
[7, 4, 1]
[8, 5, 2]
[9, 6, 3]
```

Problem : Self Dividing Numbers

Problem Statement

A *self-dividing number* is a number that is divisible by every digit it contains.

- For example, `128` is a *self-dividing number* because `128 % 1 == 0`, `128 % 2 == 0`, and `128 % 8 == 0`.

A *self-dividing number* is not allowed to contain the digit zero.

Given two integers `left` and `right`, return a list of all the *self-dividing numbers* in the range `[left, right]` (**both inclusive**).

- Write a function `selfDividingNumber(left, right)` that takes two integers `left, right`.
- You must also **write a `main` method** that performs the following steps:

    1. **Takes inputs** `left`, `right` from the user.
    2. **Calls the `selfDividingNumber()` function**.
    3. **Displays the result** appropriately.

Example 1:

```
Input: left = 1, right = 22

Output: [1,2,3,4,5,6,7,8,9,11,12,15,22]
```

Example 1:

```
Input: left = 47, right = 85

Output: [48,55,66,77]
```

Code

```
import java.util.Scanner;
import java.util.*;

class Solution{

    public static List<Integer> selfDividingNumbers(int left, int right){
        List<Integer> arr = new ArrayList<>();

        for(int num = left; num<=right; num++){

            if (num % 10 == 0 || num % 100 == 0 || num % 1000 == 0)
```

```
            continue;

            if(num>0 && num<10){
                arr.add(num);
                continue;
            }

            int temp = num;
            boolean flag = true;
            while(temp!=0){
                int digit = temp%10;

                if(digit==0 || num%digit!=0){
                    flag = false;
                    break;
                }

                temp = temp/10;
            }

            if(flag){
                arr.add(num);
            }
        }

        return arr;
    }

    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter 'left' starting integer.");
        int left = sc.nextInt();

        System.out.println("Enter 'right' ending integer.");
        int right = sc.nextInt();

        List<Integer> result = selfDividingNumbers(left,right);

        System.out.println("result "+ result);
    }
}
```

Terminal Output

```
$ java Solution
Enter 'left' starting integer.
1
Enter 'right' ending integer.
22
result [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 22]
```