[https://leetcode.com/problems/max-area-of-island/](https://leetcode.com/problems/max-area-of-island/" \t "_blank) площадь острова = 1 + площадь его соседей. Подумайте о том, что два раза заходить не надо в клеточку.

class Solution {

    public int maxAreaOfIsland(int[][] grid) {

        int area = 0;

        for (int i = 0; i < grid.length; i++) {

            for (int j = 0; j < grid[0].length; j++) {

                if (grid[i][j] == 1) {

                    area = Math.max(area, dfs(grid, i, j));

                }

            }

        }

        return area;

    }

    public int dfs(int[][] grid, int i, int j) {

        if (i <0 || i >= grid.length || j < 0 || j >= grid[0].length || grid[i][j] == 0) {

            return 0;

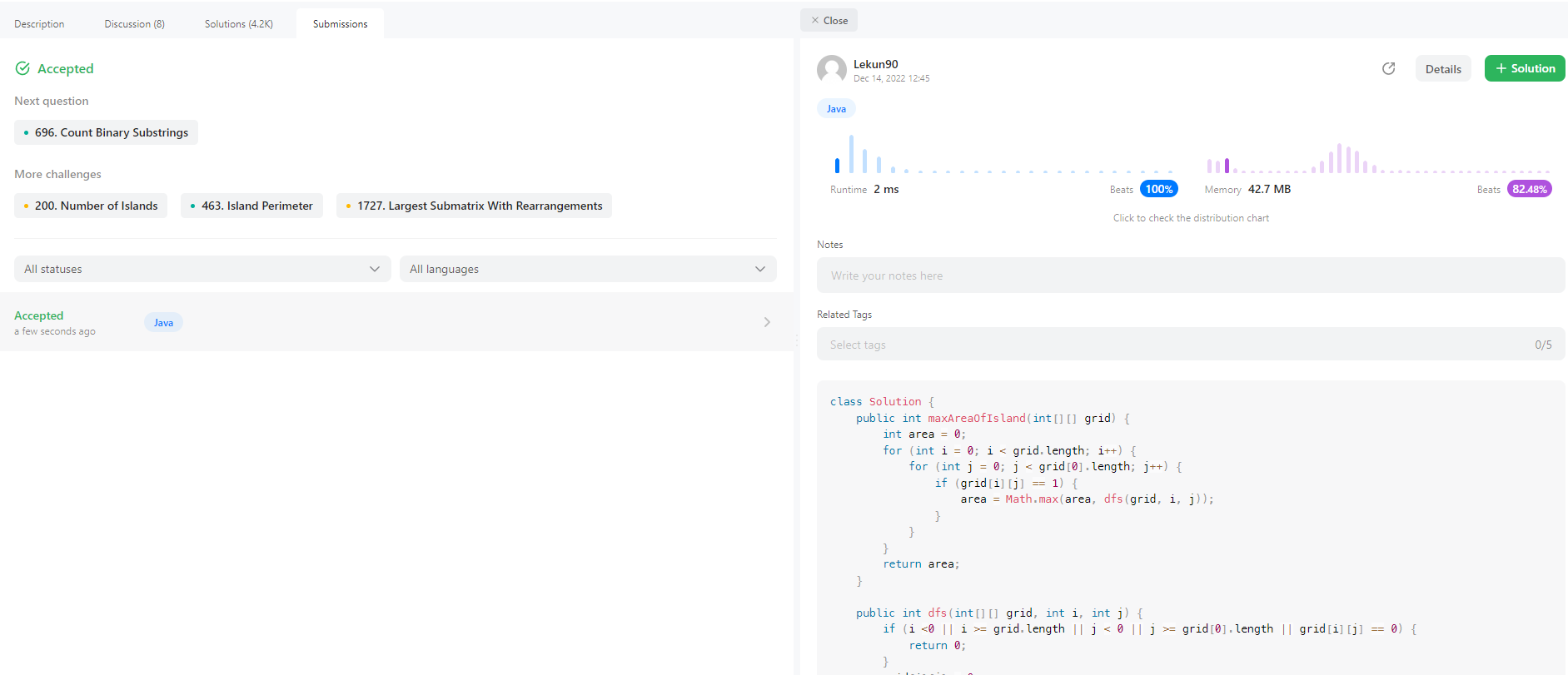
        }

        grid[i][j] = 0;

        return 1 + dfs(grid, i - 1, j) + dfs(grid, i + 1, j) + dfs(grid, i, j - 1) + dfs(grid, i, j + 1);

    }

}



<https://leetcode.com/problems/flood-fill/> - заходим в данную клеточку и помечаем ее и всех соседей такого же цвета, как изначальная клетка, заданным цветом

class Solution {

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

        int mewColor = image[sr][sc];

        if (color != mewColor) {

            dfs(image, sr, sc, mewColor, color);

        }

        return image;

    }

    public void dfs(int[][] image, int sr, int sc, int color, int newColor) {

        if (sr < 0 || sr >= image.length || sc < 0 || sc >= image[0].length || image[sr][sc] != color) {

            return;

        }

        image[sr][sc] = newColor;

        dfs(image, sr + 1, sc, color, newColor);

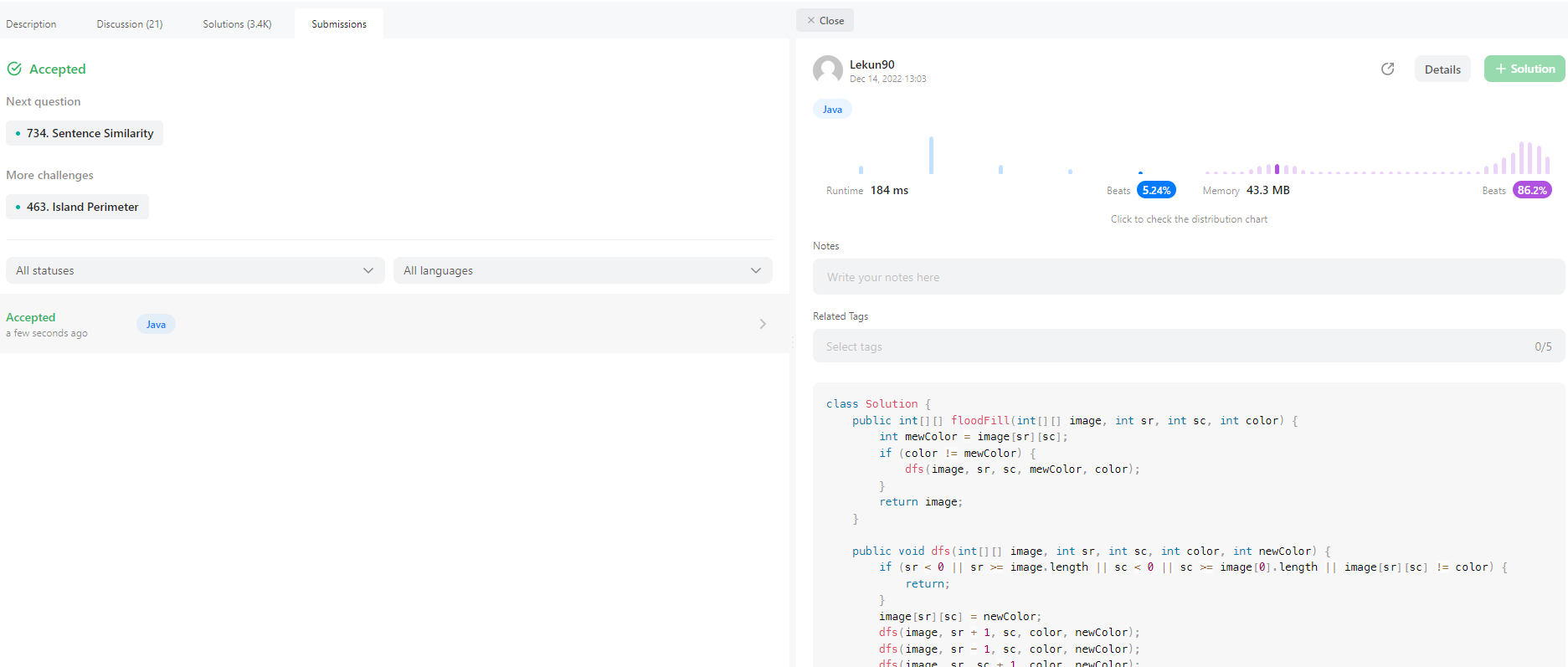
        dfs(image, sr - 1, sc, color, newColor);

        dfs(image, sr, sc + 1, color, newColor);

        dfs(image, sr, sc - 1, color, newColor);

    }

}



<https://leetcode.com/problems/symmetric-tree/>

class Solution {

    public boolean isSymmetric(TreeNode root) {

        if (root == null) {

            return true;

        }

        return mirror(root, root);

    }

    private boolean mirror(TreeNode root1, TreeNode root2) {

        if (root1 == null && root2 == null) {

            return true;

        }

        if (root1 == null || root2 == null || root1.val != root2.val) {

            return false;

        }

        boolean left = mirror(root1.left, root2.right);

        boolean right = mirror(root1.right, root2.left);

        return left && right;

    }

}

