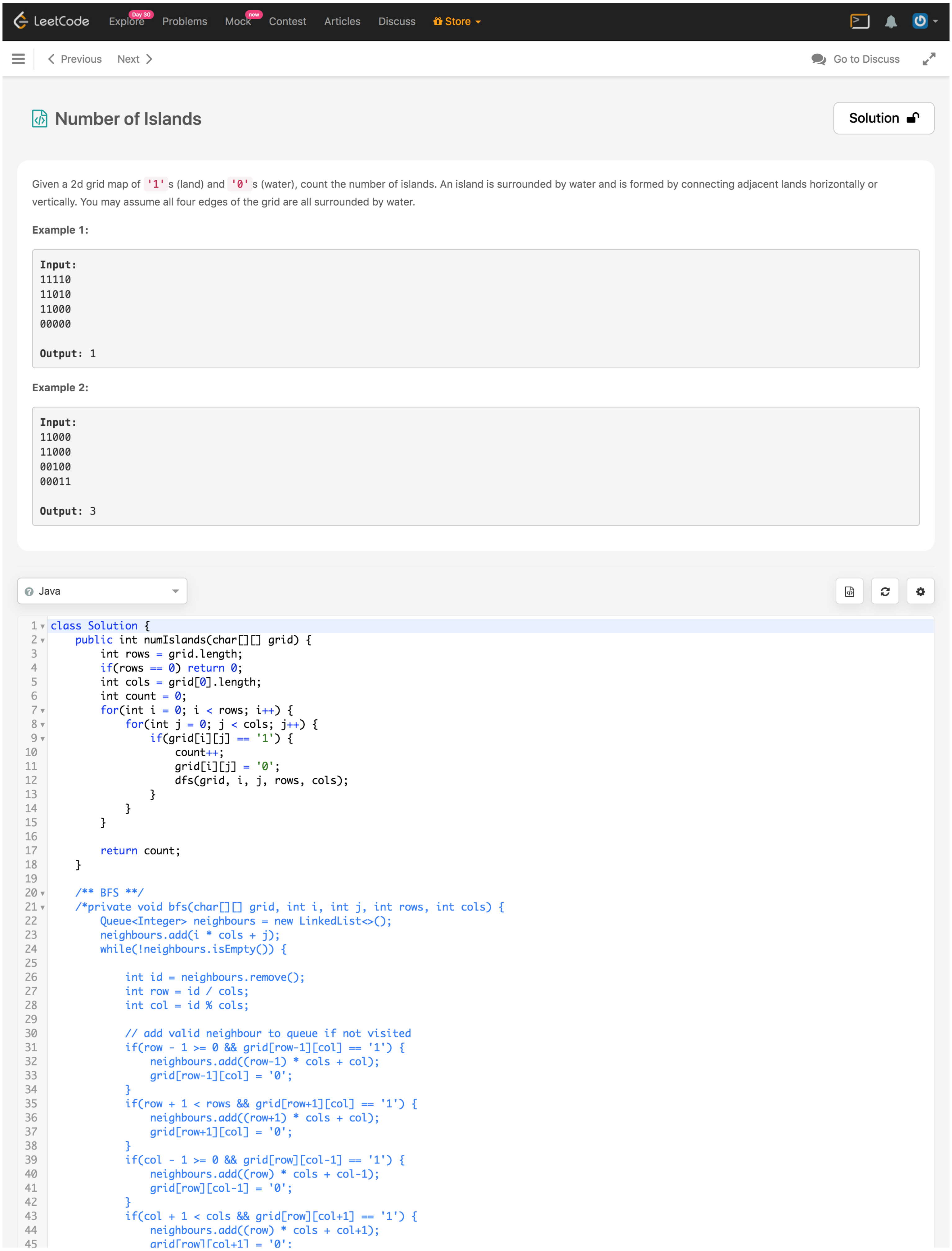

Number of Islands

Given a 2d grid map of `'1'`s (land) and `'0'`s (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
Input:
11110
11010
11000
00000

Output: 1
```

Example 2:

```
Input:
11000
11000
00100
00011

Output: 3
```

```java
class Solution {
    public int numIslands(char[][] grid) {
        int rows = grid.length;
        if(rows == 0) return 0;
        int cols = grid[0].length;
        int count = 0;
        for(int i = 0; i < rows; i++) {
            for(int j = 0; j < cols; j++) {
                if(grid[i][j] == '1') {
                    count++;
                    grid[i][j] = '0';
                    dfs(grid, i, j, rows, cols);
                }
            }
        }

        return count;
    }

    /** BFS **/
    /*private void bfs(char[][] grid, int i, int j, int rows, int cols) {
        Queue<Integer> neighbours = new LinkedList<>();
        neighbours.add(i * cols + j);
        while(!neighbours.isEmpty()) {

            int id = neighbours.remove();
            int row = id / cols;
            int col = id % cols;

            // add valid neighbour to queue if not visited
            if(row - 1 >= 0 && grid[row-1][col] == '1') {
                neighbours.add((row-1) * cols + col);
                grid[row-1][col] = '0';
            }
            if(row + 1 < rows && grid[row+1][col] == '1') {
                neighbours.add((row+1) * cols + col);
                grid[row+1][col] = '0';
            }
            if(col - 1 >= 0 && grid[row][col-1] == '1') {
                neighbours.add((row) * cols + col-1);
                grid[row][col-1] = '0';
            }
            if(col + 1 < cols && grid[row][col+1] == '1') {
                neighbours.add((row) * cols + col+1);
                grid[row][col+1] = '0';
            }
        }
    }*/

    /** DFS Iterative **/
    private void dfs(char[][] grid, int i, int j, int rows, int cols) {
        Stack<Integer> neighbours = new Stack<>();
        neighbours.push(i * cols + j);
        while(!neighbours.isEmpty()) {

            int id = neighbours.pop();
            int row = id / cols;
            int col = id % cols;

            // add valid neighbour to stack if not visited
            if(row - 1 >= 0 && grid[row-1][col] == '1') {
                neighbours.push((row-1) * cols + col);
                grid[row-1][col] = '0';
            }
            if(row + 1 < rows && grid[row+1][col] == '1') {
                neighbours.push((row+1) * cols + col);
                grid[row+1][col] = '0';
            }
            if(col - 1 >= 0 && grid[row][col-1] == '1') {
                neighbours.push((row) * cols + col-1);
                grid[row][col-1] = '0';
            }
            if(col + 1 < cols && grid[row][col+1] == '1') {
                neighbours.push((row) * cols + col+1);
                grid[row][col+1] = '0';
            }
        }
    }
}
```

Custom Testcase (**Contribute**)

Run Code   Submit