1. 题目
2. 题干

给你一个由 '1'（陆地）和 '0'（水）组成的的二维网格，请你计算网格中岛屿的数量。

岛屿总是被水包围，并且每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

1. 示例

示例 1：

输入：grid = [

["1","1","1","1","0"],

["1","1","0","1","0"],

["1","1","0","0","0"],

["0","0","0","0","0"]

]

输出：1

示例 2：

输入：grid = [

["1","1","0","0","0"],

["1","1","0","0","0"],

["0","0","1","0","0"],

["0","0","0","1","1"]

]

输出：3

提示：

m == grid.length

n == grid[i].length

1 <= m, n <= 300

grid[i][j] 的值为 '0' 或 '1'

1. 题解
2. 思路

岛屿问题，用dfs的思路解决。对二维数组进行遍历，寻找等于1的节点，从该节点开始dfs递归，设置已访问过的节点的值为0，防止之后重复访问该节点。然后向四个方向进行尝试访问，如果超出了二维数组的范围或者下一个节点为0，则直接return。

之前，我总是考虑着二维数组边界的问题，不同的边界溢出情况不一样，想着要用不同的if语句来判断，但是这里直接统一成了4个方向进行访问，根据变化后的坐标情况判断坐标是否溢出，就不需要再分边界情况讨论了，非常方便。

1. 代码

class Solution {

    public int numIslands(char[][] grid) {

        //dfs的思路

        //二维数组迭代，判断当前节点是否等于1

        //如果等于1， 则进入递归，往上，下，左，右走，如果下一个点为0或者超过了二维数组的范围则return

        //如果不等于1，则pass

        //用一个num记录岛屿数量

        if(grid == null || grid.length==0){

            return 0;

        }

        int mr = grid.length;

        int nc = grid[0].length;

        int num = 0;

        for(int i = 0; i < mr; ++i){

            for(int j =0; j < nc; ++j){

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

                    //进入递归

                    num++;

                    dfs(grid, i, j);

                }

            }

        }

        return num;

    }

    public void dfs(char[][] grid, int r, int c){

        int mr = grid.length;

        int nc = grid[0].length;

        //递归

        if(r < 0 || c < 0 || r >= mr || c >= nc || grid[r][c] == '0'){

            return;

        }

        //当前节点为 1，访问后设置为0

        grid[r][c] = '0';

        //递归四个方向节点

        dfs(grid, r, c-1);

        dfs(grid, r, c+1);

        dfs(grid, r-1, c);

        dfs(grid, r+1, c);

    }

}