LeetCode 图解 |

> *本文首发于公众号「图解面试算法」，是 图解 LeetCode 系列文章之一。*
>
> *同步博客：https://www.algomooc.com*

本题解作者：nettee

题目描述

给定一个包含了一些 `0` 和 `1` 的非空二维数组 `grid` 。

一个**岛屿**是由一些相邻的 `1` (代表土地) 构成的组合，这里的「相邻」要求两个 `1` 必须在水平或者竖直方向上相邻。你可以假设 `grid` 的四个边缘都被 `0` （代表水）包围着。

找到给定的二维数组中最大的岛屿面积。(如果没有岛屿，则返回面积为 `0` 。)

示例 1:

```
[[0,0,1,0,0,0,0,1,0,0,0,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,1,1,0,1,0,0,0,0,0,0,0,0],
 [0,1,0,0,1,1,0,0,1,0,1,0,0],
 [0,1,0,0,1,1,0,0,1,1,1,0,0],
 [0,0,0,0,0,0,0,0,0,0,1,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,0,0,0,0,0,0,1,1,0,0,0,0]]
```

对于上面这个给定矩阵应返回 6。注意答案不应该是 11 ，因为岛屿只能包含水平或垂直的四个方向的 `1` 。

示例 2:

```
[[0,0,0,0,0,0,0,0]]
```

对于上面这个给定的矩阵, 返回 0。

注意: 给定的矩阵 `grid` 的长度和宽度都不超过 50。

题目解析

这道题的主要思路是深度优先搜索。每次走到一个是 1 的格子，就搜索整个岛屿，并计算当前岛屿的面积。最后返回岛屿面积的最大值。

网格可以看成是一个无向图的结构，每个格子和它上下左右的四个格子相邻。如果四个相邻的格子坐标合法，且是陆地，就可以继续搜索。

在深度优先搜索的时候要注意避免重复遍历。我们可以把已经遍历过的陆地改成 2，这样遇到 2 我们就知道已经遍历过这个格子了，不进行重复遍历。

动画理解

参考代码

C++ 代码：

```
class Solution {
public:
    int maxAreaOfIsland(vector<vector<int>>& grid) {
        int res = 0;
        for (int r = 0; r < grid.size(); r++) {
            for (int c = 0; c < grid[0].size(); c++) {
                if (grid[r][c] == 1) {
                    int a = area(grid, r, c);
                    res = max(res, a);
                }
            }
        }
        return res;
    }

    int area(vector<vector<int>>& grid, int r, int c) {
        if (!(inArea(grid, r, c))) {
            return 0;
        }
        if (grid[r][c] != 1) {
            return 0;
        }
        grid[r][c] = 2;

        return 1
            + area(grid, r - 1, c)
            + area(grid, r + 1, c)
            + area(grid, r, c - 1)
            + area(grid, r, c + 1);
    }

    bool inArea(vector<vector<int>>& grid, int r, int c) {
        return 0 <= r && r < grid.size()
            && 0 <= c && c < grid[0].size();
    }
};
```

Java 代码：

```
class Solution {
    public int maxAreaOfIsland(int[][] grid) {
        int res = 0;
        for (int r = 0; r < grid.length; r++) {
            for (int c = 0; c < grid[0].length; c++) {
                if (grid[r][c] == 1) {
                    int a = area(grid, r, c);
```

```
                    res = Math.max(res, a);
                }
            }
        }
        return res;
    }

    int area(int[][] grid, int r, int c) {
        if (!inArea(grid, r, c)) {
            return 0;
        }
        if (grid[r][c] != 1) {
            return 0;
        }
        grid[r][c] = 2;

        return 1
            + area(grid, r - 1, c)
            + area(grid, r + 1, c)
            + area(grid, r, c - 1)
            + area(grid, r, c + 1);
    }

    boolean inArea(int[][] grid, int r, int c) {
        return 0 <= r && r < grid.length
            && 0 <= c && c < grid[0].length;
    }
}
```

Python 代码：

```
class Solution:
    def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
        res = 0
        for r in range(len(grid)):
            for c in range(len(grid[0])):
                if grid[r][c] == 1:
                    a = self.area(grid, r, c)
                    res = max(res, a)
        return res

    def area(self, grid: List[List[int]], r: int, c: int) -> int:
        if not self.inArea(grid, r, c):
            return 0
        if grid[r][c] != 1:
            return 0
        grid[r][c] = 2

        return 1 \
            + self.area(grid, r - 1, c) \
            + self.area(grid, r + 1, c) \
```

```
            + self.area(grid, r, c - 1) \
            + self.area(grid, r, c + 1)

    def inArea(self, grid: List[List[int]], r: int, c: int) -> bool:
        return 0 <= r < len(grid) and 0 <= c < len(grid[0])
```

复杂度分析

设网格的边长为 n，则时间复杂度为 $O(n^2)$。