# 题目

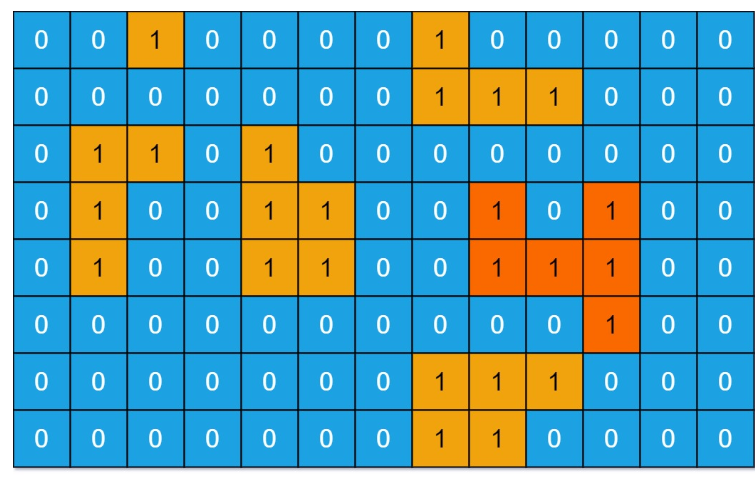
给你一个大小为 m x n 的二进制矩阵 grid 。

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 水平或者竖直的四个方向上 相邻。你可以假设 grid 的四个边缘都被 0（代表水）包围着。

岛屿的面积是岛上值为 1 的单元格的数目。

计算并返回 grid 中最大的岛屿面积。如果没有岛屿，则返回面积为 0 。

示例 1：



输入：grid = [[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]]

输出：6

解释：答案不应该是 11 ，因为岛屿只能包含水平或垂直这四个方向上的 1 。

示例 2：

输入：grid = [[0,0,0,0,0,0,0,0]]

输出：0

提示：

m == grid.length

n == grid[i].length

1 <= m, n <= 50

grid[i][j] 为 0 或 1

# 分析

## 方法一：深度优先遍历

思路：

要计算最大岛屿面积，可以使用深度优先搜索（DFS）来遍历每个岛屿，并计算其面积。具体步骤如下：

1、定义一个变量 max\_area 用于记录最大岛屿面积，初始化为 0。

2、遍历二维矩阵 grid 中的每个元素：

如果当前元素为 1，表示找到岛屿的起始点，通过 DFS 计算该岛屿的面积。

DFS 的过程中，将访问过的岛屿格子标记为 0，防止重复计算。

在 DFS 中，从当前位置向上、下、左、右四个方向递归遍历，累计岛屿面积。

3、在遍历过程中更新 max\_area，保留最大值。

4、最终返回 max\_area。

代码：

class Solution {

public:

// 计算最大岛屿面积的函数

int maxAreaOfIsland(vector<vector<int>>& grid) {

int max\_area = 0; // 最大岛屿面积

int m = grid.size(); // 矩阵的行数

int n = grid[0].size(); // 矩阵的列数

// 遍历矩阵中的每个元素

for (int i = 0; i < m; ++i) {

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

if (grid[i][j] == 1) { // 如果当前元素为 1，表示找到岛屿的起始点

int area = dfs(grid, i, j, m, n); // 计算该岛屿的面积

max\_area = max(max\_area, area); // 更新最大岛屿面积

}

}

}

return max\_area; // 返回最大岛屿面积

}

// 深度优先搜索计算岛屿面积的函数

int dfs(vector<vector<int>>& grid, int i, int j, int m, int n) {

if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] == 0) {

return 0; // 越界或当前格子为水，返回面积 0

}

grid[i][j] = 0; // 将当前格子标记为访问过，防止重复计算

int area = 1; // 当前格子为陆地，面积至少为 1

// 向上、下、左、右四个方向递归遍历，累计岛屿面积

area += dfs(grid, i + 1, j, m, n);

area += dfs(grid, i - 1, j, m, n);

area += dfs(grid, i, j + 1, m, n);

area += dfs(grid, i, j - 1, m, n);

return area; // 返回当前岛屿面积

}

};