题目地址（778. 水位上升的泳池中游泳）

https://leetcode-cn.com/problems/swim-in-rising-water

入选理由

1. 难度是 hard， 适合做压轴。

2. 这是一个新的二分题型 - **DFS + 二分**，类似的题目有很多，比如第 1439 题。这种题套路都很像，难度其实也不算大。

标签

- 二分

难度

- 困难

题目描述

在一个 N x N 的坐标方格 grid 中，每一个方格的值 grid[i][j] 表示在位置 (i,j) 的平台高度。

现在开始下雨了。当时间为 t 时，此时雨水导致水池中任意位置的水位为 t 。你可以从一个平台游向四周相邻的任意一个平台，但是前提是此时水位必须同时淹没这两个平台。假定你可以瞬间移动无限距离，也就是默认在方格内部游动是不耗时的。当然，在你游泳的时候你必须待在坐标方格里面。

你从坐标方格的左上平台 (0，0) 出发。最少耗时多久你才能到达坐标方格的右下平台 (N-1, N-1)?

示例 1:

输入: [[0,2],[1,3]] 输出: 3 解释: 时间为 0 时，你位于坐标方格的位置为 (0, 0)。 此时你不能游向任意方向，因为四个相邻方向平台的高度都大于当前时间为 0 时的水位。

等时间到达 3 时，你才可以游向平台 (1, 1). 因为此时的水位是 3，坐标方格中的平台没有比水位 3 更高的，所以你可以游向坐标方格中的任意位置 示例 2:

输入: [[0,1,2,3,4],[24,23,22,21,5],[12,13,14,15,16],[11,17,18,19,20],[10,9,8,7,6]] 输出: 16 解释: 0 1 2 3 4 24 23 22 21 5 12 13 14 15 16 11 17 18 19 20 10 9 8 7 6

最终的路线用加粗进行了标记。 我们必须等到时间为 16，此时才能保证平台 (0, 0) 和 (4, 4) 是连通的

提示:

2 <= N <= 50. grid[i][j] 位于区间 [0, ..., N*N - 1] 内。

前置知识

- DFS
- 二分

思路

二分查找在 CP 中的一个常见应用是二分答案。在这一类题目中，我们往往需要求出满足条件的最大值或最小值。如果这一取值和条件的成立与否之间满足有序性，我们就可以通过对整个定义域进行二分查找，来找到我们需要的最值。

很明显，这道题的答案是一个连续的空间，从 0 到 max(grid)，其中 max(grid) 表示 grid 中的最大值。

因此一个简单的思路是一个个试。实际上，如果 x 不可以，那么小于 x 的所有值都是不可以的，这正是本题的突破口。基于此，我们可使用讲义中的**最左二分**模板解决。

伪代码:

```
def test(x):
    pass
while l <= r:
    mid = (l + r) // 2
    if test(mid, 0, 0):
        r = mid - 1
    else:
        l = mid + 1
return l
```

这个模板会在很多二分中使用。比如典型的计数型二分，典型的就是计算小于等于 x 的有多少，然后根据答案更新搜索区间。

明确了这点，剩下要做的就是完成 test 了。其实这个就是一个普通的二维网格 dfs，我们从 (0,0) 开始在一个二维网格中搜索，直到无法继续或达到 (N-1,N-1)，如果可以达到 (N-1,N-1)，我们返回 true，否则返回 False 即可。对二维网格的 DFS 不熟悉的同学可以看下我之前写的小岛专题

代码

代码支持：Python，CPP

Python Code:

```python
class Solution:
    def swimInWater(self, grid: List[List[int]]) -> int:
        l, r = 0, max([max(vec) for vec in grid])
        seen = set()

        def test(mid, x, y):
            if x > len(grid) - 1 or x < 0 or y > len(grid[0]) - 1 or y < 0:
                return False
            if grid[x][y] > mid:
                return False
            if (x, y) == (len(grid) - 1, len(grid[0]) - 1):
                return True
            if (x, y) in seen:
                return False
            seen.add((x, y))
            ans = test(mid, x + 1, y) or test(mid, x - 1,
                                              y) or test(mid, x, y + 1) or test(mid, x, y - 1)
            return ans
        while l <= r:
            mid = (l + r) // 2
            if test(mid, 0, 0):
                r = mid - 1
            else:
                l = mid + 1
            seen = set()
        return l
```

CPP Code:

```cpp
class Solution {
public:
    bool dfs(int mid, int x, int y, set<pair<int, int>>& visited, vector<vector<int>>& grid) {
        if (x > grid.size() - 1 || x < 0 || y > grid[0].size() - 1 || y < 0) return false;
        if (grid[x][y] > mid) return false;
        if (x == grid.size() - 1 && y == grid[0].size() - 1) return true;
        if (visited.count({x, y})) return false;
        visited.insert({x, y});
        bool res = dfs(mid, x + 1, y, visited, grid) || dfs(mid, x - 1, y, visited, grid) || dfs(mid, x, y + 1, visited,
        return res;
    }

    int swimInWater(vector<vector<int>>& grid) {
        int l = 0;
        int r = INT_MIN;
        for (int i = 0; i < grid.size(); i++) {
```

```
            for (int j = 0; j < grid[i].size(); j++) r = max(r, grid[i][j]);
        }
        set<pair<int, int>> visited;
        while (l <= r) {
            int mid = l + (r - l) / 2;
            if (dfs(mid, 0, 0, visited, grid)) {
                r = mid - 1;
            } else {
                l = mid + 1;
            }
            visited.clear();
        }
        return l;
    }
};
```

复杂度分析

- 时间复杂度：$O(N\log M)$，其中 M 为 grid 中的最大值， N 为 grid 的总大小。
- 空间复杂度：$O(N)$，其中 N 为 grid 的总大小。

扩展

我们也可以使用二分+BFS 的方式来完成，大家不妨试试看！

上一页 下一页