

# 腐烂的橘子

计算学部十大打卡活动——“龙舞编程新春会”编程打卡（2024-2-4）

力扣994.腐烂的橘子

## 一、题目

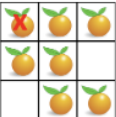
在给定的  $m \times n$  网格 `grid` 中，每个单元格可以有以下三个值之一：

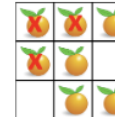
- 值 0 代表空单元格；
  - 值 1 代表新鲜橘子；
  - 值 2 代表腐烂的橘子。
- 每分钟，腐烂的橘子 **周围 4 个方向上相邻** 的新鲜橘子都会腐烂。


返回 *直到单元格中没有新鲜橘子为止所必须经过的最小分钟数*。如果不可能，返回 *-1*。


示例 1：


Minute 0




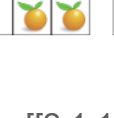




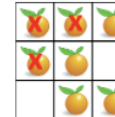











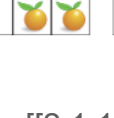
Minute 1

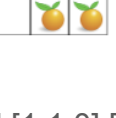
















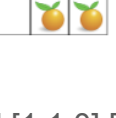
Minute 2

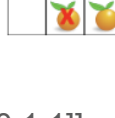
















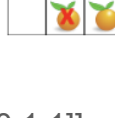
Minute 3






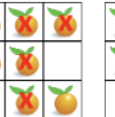


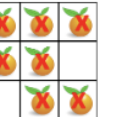









Minute 4

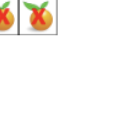












输入： `grid = [[2,1,1],[1,1,0],[0,1,1]]`

输出： 4

示例 2：

输入： `grid = [[2,1,1],[0,1,1],[1,0,1]]`

输出： -1

解释： 左下角的橘子（第 2 行，第 0 列）永远不会腐烂，因为腐烂只会发生在 4 个方向上。

示例 3：

输入： `grid = [[0,2]]`

输出： 0

**解释：** 因为 0 分钟时已经没有新鲜橘子了，所以答案就是 0。

**提示：**

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 10`
- `grid[i][j]` 仅为 0、1 或 2

## 二、题解

### 思路——多源广度优先搜索

由题目可知每分钟每个腐烂的橘子都会使上下左右相邻的新鲜橘子腐烂，这其实是模拟广度优先搜索的过程。上下左右相邻的新鲜橘子就是该腐烂橘子尝试访问的同一层的节点，路径长度就是新鲜橘子被腐烂的时间。

同时，图中腐烂的橘子很可能不止一个，此问题变为 **多源广度优先搜索**，观察到对于所有的腐烂橘子，其实它们在广度优先搜索上是等价于同一层的节点的，所以可以设置一个 **超级源点**。

假设这些腐烂橘子刚开始是新鲜的，而有一个腐烂橘子(**超级源点**)会在下一秒把这些橘子都变腐烂，而这个腐烂橘子刚开始在的时间是  $-1$ ，那么按照广度优先搜索的算法，下一分钟也就是第 0 分钟的时候，这个腐烂橘子会把它们都变成腐烂橘子，然后继续向外拓展，转化为普通的广度优先搜索。

为了确认是否所有新鲜橘子都被腐烂，可以记录一个变量 `cnt` 表示当前网格中的新鲜橘子数，广度优先搜索的时候如果有新鲜橘子被腐烂，则 `cnt--`，最后搜索结束时如果 `cnt` 大于 0，说明有新鲜橘子没被腐烂，返回  $-1$ ，否则返回所有新鲜橘子被腐烂的时间的最大值即可，也可以在广度优先搜索的过程中把已腐烂的新鲜橘子的值由 1 改为 2，最后看网格中是否由值为 1 即新鲜的橘子即可。

# 代码

```
class Solution {
    int cnt=0; //记录新鲜橘子数量
    int dist[10][10]; //记录每个橘子腐烂的时间
    int dir_x[4] = {0, 1, 0, -1}, dir_y[4] = {1, 0, -1, 0}; //上下左右四个方向

public:
    int orangesRotting(vector<vector<int>>& grid) {
        memset(dist, -1, sizeof(dist)); //初始化腐烂时间为-1
        queue<pair<int, int>> q; //广度优先搜索队列，存坐标
        int n = grid.size(), m = grid[0].size();
        int ans = 0;
        int i, j;
        //将所有腐烂橘子入队，模拟超级源点，并记录新鲜橘子数量
        for (i = 0; i < n; i++) {
            for (j = 0; j < m; j++) {
                if (grid[i][j] == 2) {
                    q.push({i, j});
                    dist[i][j] = 0;
                } else if (grid[i][j] == 1) {
                    cnt += 1;
                }
            }
        }
        //广度优先搜索
        while (!q.empty()) {
            pair<int, int> point = q.front(); //取出队首坐标
            q.pop();
            //遍历四个方向，将在区域内的、腐烂时间还为初始化-1的、grid中不为空格的坐标入队
            for (i = 0; i < 4; i++) {
                int tx = point.first + dir_x[i];
                int ty = point.second + dir_y[i];
                if (tx < 0 || tx >= n || ty < 0 || ty >= m || !grid[tx][ty] || ~dist[tx][ty]) {
                    continue;
                }
                dist[tx][ty] = dist[point.first][point.second] + 1; //腐烂时间加一
                q.push({tx, ty}); //入队
                cnt -= 1; //新鲜橘子少一
                ans = dist[tx][ty]; //更新结果
                if (!cnt) {
                    break;
                }
            }
        }
    }
};
```

```
        }  
    }  
    return cnt ? -1 : ans;  
}  
};
```

## 复杂度分析

- 时间复杂度： $O(nm)$ 。即进行一次广度优先搜索的时间，其中  $n$  和  $m$  分别为 *grid* 的行数和列数。
- 空间复杂度： $O(nm)$ 。需要额外的数组记录每个新鲜橘子被腐烂的最短时间，大小为  $O(nm)$ ，且广度优先搜索中队列里存放的状态最多不会超过  $nm$  个，最多需要  $O(nm)$  的空间，所以最后的空间复杂度为  $O(nm)$ 。