0289. 生命游戏

ITCharge 大约 4 分钟

- 标签：数组、矩阵、模拟
- 难度：中等

题目链接

- 0289. 生命游戏 - 力扣

题目大意

描述：给定一个 $m \times n$ 大小的二维数组 $board$，每一个格子都可以看做是一个细胞。每个细胞都有一个初始状态：1 代表活细胞，0 代表死细胞。每个细胞与其相邻的八个位置（水平、垂直、对角线）细胞遵循以下生存规律：

- 如果活细胞周围八个位置的活细胞数少于 2 个，则该位置活细胞死亡；
- 如果活细胞周围八个位置有 2 个或 [illegible] 个活细胞，则该位置活细胞仍然存活；
- 如果活细胞周围八个位置有超过 3 [illegible] 个活细胞，则该位置活细胞死亡；
- 如果死细胞周围正好有 3 个活细胞，则该位置死细胞复活。

二维数组代表的下一个状态是通过将上述规则同时应用于当前状态下的每个细胞所形成的的。其中细胞的出生和死亡是同时发生的。

现在给定 $m \times n$ 的二维数组 $board$ 的当前状态。

要求：返回下一个状态。

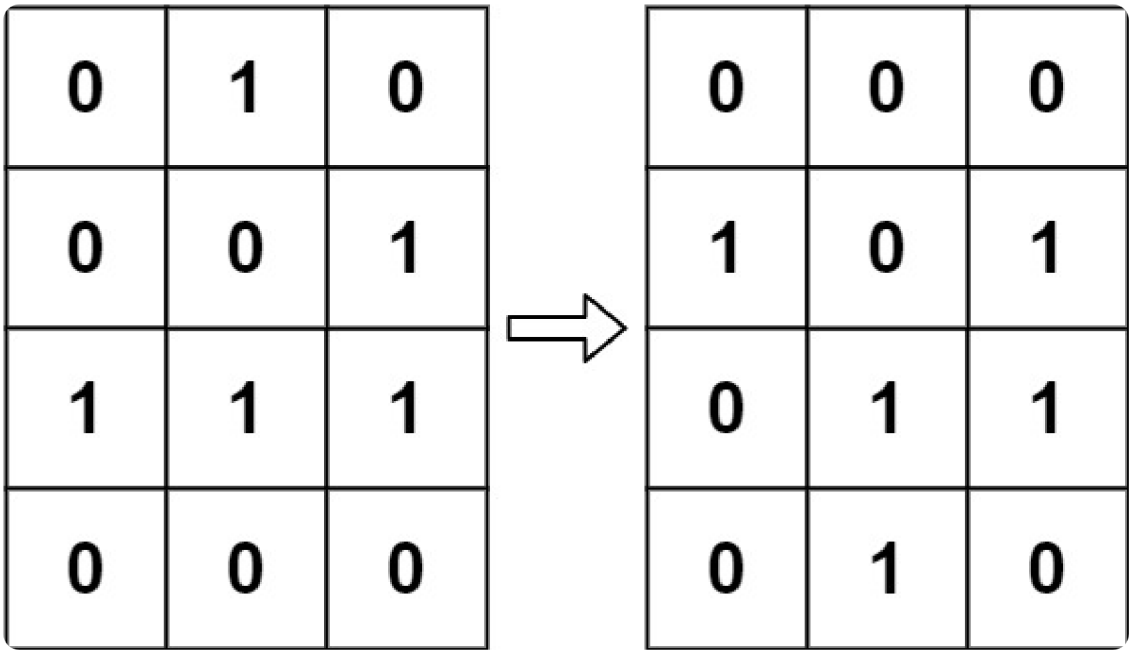
说明：

- $m == board.length$。
- $n == board[i].length$。
- $1 \leq m, n \leq 25$。
- $board[i][j]$ 为 0 或 1。
- **进阶**：
  - 你可以使用原地算法解决本题吗？请注意，面板上所有格子需要同时被更新：你不能先更新某些格子，然后使用它们的更新后的值再更新其他格子。

- 本题中，我们使用二维数组来表示面板。原则上，面板是无限的，但当活细胞侵占了面板边界时会造成问题。你将如何解决这些问题？

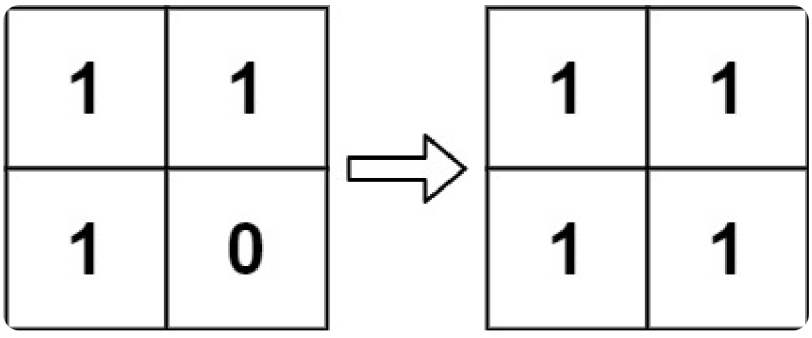
示例：

- 示例 1：




```py
输入：board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]
输出：[[0,0,0],[1,0,1],[0,1,1],[0,1,0]]
```

- 示例 2：




```py
输入：board = [[1,1],[1,0]]
输出：[[1,1],[1,1]]
```

解题思路

思路 1：模拟

因为下一个状态隐含了过去细胞的状态，所以不能直接在原二维数组上直接进行修改。细胞的状态总共有四种情况：

- 死细胞 -> 死细胞，即 $0 \to 0$。
- 死细胞 -> 活细胞，即 $0 \to 1$。
- 活细胞 -> 活细胞，即 $1 \to 1$。
- 活细胞 -> 死细胞，即 $1 \to 0$。

死细胞 -> 死细胞，活细胞 -> 活细胞，不会对前后状态造成影响，所以主要考虑另外两种情况。我们把活细胞 -> 死细胞暂时标记为 -1，并且统计每个细胞周围活细胞数量时，使用绝对值统计，这样 $abs(-1)$ 也可以暂时标记为活细胞。然后把死细胞 -> 活细胞暂时标记为 2，这样判断的时候也不会统计上去。然后开始遍历。

- 遍历二维数组的每一个位置。并对该位置遍历周围八个位置，计算出八个位置上的活细胞数量。
  - 如果此位置是活细胞，并且周围活细胞少于 2 个或超过 3 个，则将其暂时标记为 -1，意为此细胞死亡。
  - 如果此位置是死细胞，并且周围有 3 个活细胞，则将暂时标记为 2，意为此细胞复活。
- 遍历完之后，再次遍历一遍二维数组，如果该位置为 -1，将其赋值为 0，如果该位置为 2，将其赋值为 1。

思路 1：代码

```py
class Solution:
    def gameOfLife(self, board: List[List[int]]) -> None:
        """
        Do not return anything, modify board in-place instead.
        """
        directions = {(1, 0), (1, -1), (0, -1), (-1, -1), (-1, 0), (-1, 1), (0, 1), (1, 1)}

        rows = len(board)
```

```
        cols = len(board[0])

        for row in range(rows):
            for col in range(cols):
                lives = 0
                for direction in directions:
                    new_row = row + direction[0]
                    new_col = col + direction[1]

                    if 0 <= new_row < rows and 0 <= new_col < cols and
abs(board[new_row][new_col]) == 1:
                        lives += 1
                if board[row][col] == 1 and (lives < 2 or lives > 3):
                    board[row][col] = -1
                if board[row][col] == 0 and lives == 3:
                    board[row][col] = 2

        for row in range(rows):
            for col in range(cols):
                if board[row][col] == -1:
                    board[row][col] = 0
                elif board[row][col] == 2:
                    board[row][c^]] = 1
```

思路 1：复杂度分析

- **时间复杂度**：$O(m \times n)$，其中 m、n 分别为 $board$ 的行数和列数。
- **空间复杂度**：$O(m \times n)$。