

力扣+螺旋矩阵+7.5

题目：

← ↻ 🔒 https://leetcode.cn/problems/spiral-matrix/description/

🏠 题库 < > 🔍

📖 题目描述 | ⌚ 通过 × | 📝 笔记 × | 💡 题解 | ⌚ 提交记录

54. 螺旋矩阵

已解答 ✓

中等 🔖 相关标签 🔒 相关企业 💡 提示 Aa

给你一个 m 行 n 列的矩阵 `matrix`，请按照 **顺时针螺旋顺序**，返回矩阵中的所有元素。

示例 1:

1	→	2	→	3
4	→	5		↓
↑				↓
7	←	8	←	9

输入: `matrix = [[1,2,3],[4,5,6],[7,8,9]]`
输出: `[1,2,3,6,9,8,7,4,5]`

示例 2:

1	→	2	→	3	→	4

👍 1.7K | 💬 2.4K | ☆ | 🔗 | ?

解题思路一：

模拟加分类讨论，观察矩阵发现，无论是什么矩阵无非就是在重复上下左右的过程，最后的结果一定是左下右上的循环，那么我们假定一个二维数组，只需要把上下左右四个步骤分离，然后设定后终止条件，那么问题就解决了。假定这个数组是无穷大的，我们可以发现，每次从左移动时，移动的长度永远是right-left,但是这个值是在递减的，换个角度解读就是，每次左移过后，下次一定的起点是在当前列的下一列，列的起点在增加，即每移动一行，++top(top假定为横向移动的列号)。同理，假定我们从上往下看，可以看出每次都是不同的高度而每列的right不变，但是right会递减，即有--right;对于从右到左，它的列是递减的，递减列我们设为bottom,则--bottom，同理最后一项是left++

```
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        vector<int> result;
        if (matrix.empty() || matrix[0].empty()) return result;

        int rows = matrix.size();
        int cols = matrix[0].size();
        int top = 0, bottom = rows - 1, left = 0, right = cols - 1;

        while (top <= bottom && left <= right) {
            // 从左到右
            for (int i = left; i <= right; ++i) {
                result.push_back(matrix[top][i]);
            }
            ++top;

            // 从上到下
            for (int i = top; i <= bottom; ++i) {
                result.push_back(matrix[i][right]);
            }
            --right;

            if (top <= bottom) {
                // 从右到左
                for (int i = right; i >= left; --i) {
                    result.push_back(matrix[bottom][i]);
                }
                --bottom;
            }

            if (left <= right) {
                // 从下到上
                for (int i = bottom; i >= top; --i) {
                    result.push_back(matrix[i][left]);
                }
                ++left;
            }
        }
        return result;
    }
};
```

解题思路二：

控制方向，解题思路一严格来说是按层次模拟，我们仔细观察不难发现，向左遍历时没有改变行坐标，列坐标++，同理，向下遍历时，列坐标不变，行坐标++，再从右到左，列--而行不变，而每次的变化幅度都是1，那么我们抽象出一个控制方向的数组有 $dp[4][2]=\{[0,1],[1,0],[0,-1],[-1,0]\}$ ，再对访问的数据进行标记，我们就可以得到结果了

```
class Solution {
private:
    static constexpr int directions[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        if (matrix.size() == 0 || matrix[0].size() == 0) {
            return {};
        }

        int rows = matrix.size(), columns = matrix[0].size();
        //标记数组避免重复访问和方便退出
        vector<vector<bool>> visited(rows, vector<bool>(columns));
        int total = rows * columns;
        vector<int> order(total);

        int row = 0, column = 0;
        int directionIndex = 0;
        for (int i = 0; i < total; i++) {
            order[i] = matrix[row][column];
            visited[row][column] = true;
            int nextRow = row + directions[directionIndex][0], nextColumn =
            column + directions[directionIndex][1];
            //四种条件对应四个方向
            if (nextRow < 0 || nextRow >= rows || nextColumn < 0 || nextColumn
            >= columns || visited[nextRow][nextColumn]) {
                // %4是方便循环取
                directionIndex = (directionIndex + 1) % 4;
            }
            //更新下一次的坐标
            row += directions[directionIndex][0];
            column += directions[directionIndex][1];
        }
        return order;
    }
};
```