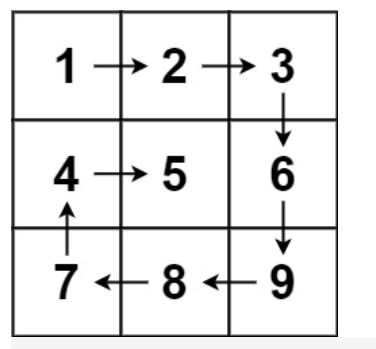
# 题目

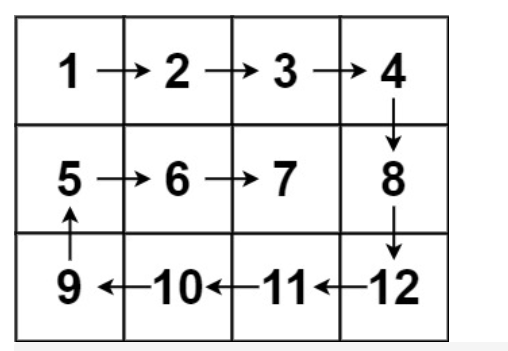
给你一个m行n列的矩阵matrix，请按照顺时针螺旋顺序，返回矩阵中的所有元素。

示例 1：



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：



输入：matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]

输出：[1,2,3,4,8,12,11,10,9,5,6,7]

提示：

m == matrix.length

n == matrix[i].length

1 <= m, n <= 10

-100 <= matrix[i][j] <= 100

# 分析

## 方法一：模拟（推荐）

**解题思路：**

1、初始化边界：

上边界：top = 0

下边界：bottom = m - 1

左边界：left = 0

右边界：right = n - 1

2、顺时针遍历：

从左到右遍历上边界。

从上到下遍历右边界。

从右到左遍历下边界（如果下边界仍在范围内）。

从下到上遍历左边界（如果左边界仍在范围内）。

3、收缩边界：

每次完成一个方向的遍历后，收缩相应的边界（如 top++、right-- 等）。

4、终止条件：

当 top > bottom 或 left > right 时，停止遍历。

**代码：**

class Solution {

public:

vector<int> spiralOrder(vector<vector<int>>& matrix) {

vector<int> result;

if (matrix.empty() || matrix[0].empty()) return result;

int m = matrix.size(); // 行数

int n = matrix[0].size(); // 列数

int top = 0, bottom = m - 1, left = 0, right = n - 1;

while (top <= bottom && left <= right) {

// 1. 从左到右遍历上边界

for (int j = left; j <= right; ++j) {

result.push\_back(matrix[top][j]);

}

++top; // 收缩上边界

// 2. 从上到下遍历右边界

for (int i = top; i <= bottom; ++i) {

result.push\_back(matrix[i][right]);

}

--right; // 收缩右边界

// 3. 从右到左遍历下边界

if (top <= bottom) {

for (int j = right; j >= left; --j) {

result.push\_back(matrix[bottom][j]);

}

--bottom; // 收缩下边界

}

// 4. 从下到上遍历左边界

if (left <= right) {

for (int i = bottom; i >= top; --i) {

result.push\_back(matrix[i][left]);

}

++left; // 收缩左边界

}

}

return result;

}

};

## 方法一：模拟

可以模拟打印矩阵的路径。初始位置是矩阵的左上角，初始方向是向右，当路径超出界限或者进入之前访问过的位置时，顺时针旋转，进入下一个方向。

判断路径是否进入之前访问过的位置需要使用一个与输入矩阵大小相同的辅助矩阵visited，其中的每个元素表示该位置是否被访问过。当一个元素被访问时，将visited 中的对应位置的元素设为已访问。

如何判断路径是否结束？由于矩阵中的每个元素都被访问一次，因此路径的长度即为矩阵中的元素数量，当路径的长度达到矩阵中的元素数量时即为完整路径，将该路径返回。

**代码：**

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();

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];

if (nextRow < 0 || nextRow >= rows || nextColumn < 0 || nextColumn >= columns || visited[nextRow][nextColumn]) {

directionIndex = (directionIndex + 1) % 4;

}

row += directions[directionIndex][0];

column += directions[directionIndex][1];

}

return order;

}

};

**复杂度分析：**

时间复杂度：(mn)，其中m和n分别是输入矩阵的行数和列数。矩阵中的每个元素都要被访问一次。

空间复杂度：O(mn)。需要创建一个大小为m×n 的矩阵visited 记录每个位置是否被访问过。

## 方法二：按层模拟

可以将矩阵看成若干层，首先打印最外层的元素，其次打印次外层的元素，直到打印最内层的元素。

定义矩阵的第k层是到最近边界距离为k的所有顶点。例如，下图矩阵最外层元素都是第1层，次外层元素都是第2层，剩下的元素都是第3层。

[[1, 1, 1, 1, 1, 1, 1],

[1, 2, 2, 2, 2, 2, 1],

[1, 2, 3, 3, 3, 2, 1],

[1, 2, 2, 2, 2, 2, 1],

[1, 1, 1, 1, 1, 1, 1]]

对于每层，从左上方开始以顺时针的顺序遍历所有元素。假设当前层的左上角位于(top,left)，右下角位于(bottom,right)，按照如下顺序遍历当前层的元素。

从左到右遍历上侧元素，依次为(top,left) 到(top,right)。

从上到下遍历右侧元素，依次为(top+1,right) 到(bottom,right)。

如果left<right且top<bottom，则从右到左遍历下侧元素，依次为 (bottom,right−1)到(bottom,left+1)，以及从下到上遍历左侧元素，依次为 (bottom,left)到(top+1,left)。

遍历完当前层的元素之后，将left和top分别增加1，将right和bottom分别减少1，进入下一层继续遍历，直到遍历完所有元素为止。

class Solution {

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();

vector<int> order;

int left = 0, right = columns - 1, top = 0, bottom = rows - 1;

while (left <= right && top <= bottom) {

for (int column = left; column <= right; column++) {

order.push\_back(matrix[top][column]);

}

for (int row = top + 1; row <= bottom; row++) {

order.push\_back(matrix[row][right]);

}

if (left < right && top < bottom) {

for (int column = right - 1; column > left; column--) {

order.push\_back(matrix[bottom][column]);

}

for (int row = bottom; row > top; row--) {

order.push\_back(matrix[row][left]);

}

}

left++;

right--;

top++;

bottom--;

}

return order;

}

};

**复杂度分析：**

时间复杂度：O(mn)，其中m和n分别是输入矩阵的行数和列数。矩阵中的每个元素都要被访问一次。

空间复杂度：O(1)。除了输出数组以外，空间复杂度是常数。

**另一种写法（推荐）：**

class Solution

{

public:

vector<int> spiralOrder(vector<vector<int>>& matrix)

{

if (matrix.empty()) return {};

vector<int> res;

int l = 0; //左边界

int r = matrix[0].size() - 1; //右边界

int t = 0; //上边界

int b = matrix.size() - 1; //下边界

while (true)

{

//left -> right

for (int i = l; i <= r; i++) res.push\_back(matrix[t][i]);

if (++t > b) break;

//top -> bottom

for (int i = t; i <= b; i++) res.push\_back(matrix[i][r]);

if (--r < l) break;

//right -> left

for (int i = r; i >= l; i--) res.push\_back(matrix[b][i]);

if (--b < t) break;

//bottom -> top

for (int i = b; i >= t; i--) res.push\_back(matrix[i][l]);

if (++l > r) break;

}

return res;

}

};