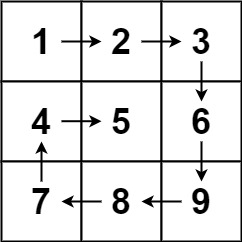
<https://leetcode.com/problems/spiral-matrix/>

Given an m x n matrix, return *all elements of the* matrix *in spiral order*.

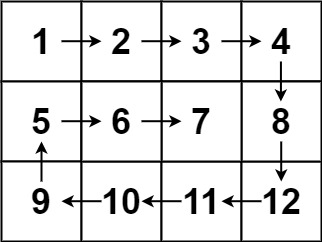
**Example 1:**



Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [1,2,3,6,9,8,7,4,5]

**Example 2:**



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

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

**Constraints:**

m == matrix.length

n == matrix[i].length

1 <= m, n <= 10

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

**Attempt 1: 2023-08-17**

**Solution 1: Intuitive switch cases (120 min)**

**Style 1:**

class Solution {

    public List<Integer> spiralOrder(int[][] matrix) {

        List<Integer> result = new ArrayList<>();

        int cur\_row = 0;

        int cur\_col = 0;

        int rows = matrix.length;

        int cols = matrix[0].length;

        int right\_board = cols;

        int bottom\_board = rows;

        int top\_board = -1;

        int left\_board = -1;

        // direction 0 代表向右, 1 代表向下, 2 代表向左, 3 代表向上

        int direction = 0;

        while(true) {

            if(result.size() == rows \* cols) {

                return result;

            }

            result.add(matrix[cur\_row][cur\_col]);

            switch(direction) {

                // 向右

                case 0:

                    if(cur\_col == right\_board - 1) {

                        direction = 1;

                        top\_board += 1;

                        cur\_row += 1;

                    } else {

                        cur\_col += 1;

                    }

                    break;

                // 向下

                case 1:

                    if(cur\_row == bottom\_board - 1) {

                        direction = 2;

                        right\_board -= 1;

                        cur\_col -= 1;

                    } else {

                        cur\_row += 1;

                    }

                    break;

                // 向左

                case 2:

                    if(cur\_col == left\_board + 1) {

                        direction = 3;

                        bottom\_board -= 1;

                        cur\_row -= 1;

                    } else {

                        cur\_col -= 1;

                    }

                    break;

                // 向上

                case 3:

                    if(cur\_row == top\_board + 1) {

                        direction = 0;

                        left\_board += 1;

                        cur\_col += 1;

                    } else {

                        cur\_row -= 1;

                    }

                    break;

            }

        }

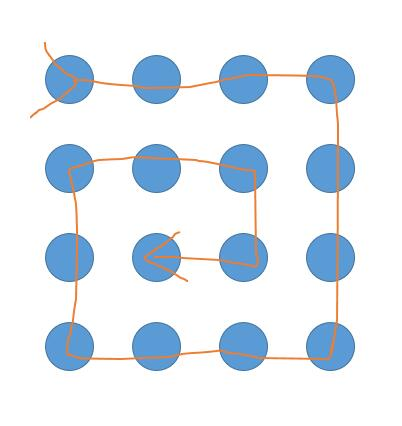
    }

}

**Refer to**

<https://leetcode.wang/leetCode-54-Spiral-Matrix.html>

从第一个位置开始，螺旋状遍历二维矩阵。



**解法一**

可以理解成贪吃蛇，从第一个位置开始沿着边界走，遇到边界就转换方向接着走，直到走完所有位置。

/\*

\* direction 0 代表向右, 1 代表向下, 2 代表向左, 3 代表向上

\*/

public List<Integer> spiralOrder(int[][] matrix) {

    List<Integer> ans = new ArrayList<>();

    if(matrix.length == 0){

        return ans;

    }

    int start\_x = 0,

    start\_y = 0,

    direction = 0,

    top\_border = -1,  //上边界

    right\_border = matrix[0].length,  //右边界

    bottom\_border = matrix.length, //下边界

    left\_border = -1; //左边界

    while(true){

        //全部遍历完结束

        if (ans.size() == matrix.length \* matrix[0].length) {

            return ans;

        }

        //注意 y 方向写在前边，x 方向写在后边

        ans.add(matrix[start\_y][start\_x]);

        switch (direction) {

            //当前向右

            case 0:

                //继续向右是否到达边界

                //到达边界就改变方向，并且更新上边界

                if (start\_x + 1 == right\_border) {

                    direction = 1;

                    start\_y += 1;

                    top\_border += 1;

                } else {

                    start\_x += 1;

                }

                break;

            //当前向下

            case 1:

                //继续向下是否到达边界

                //到达边界就改变方向，并且更新右边界

                if (start\_y + 1 == bottom\_border) {

                    direction = 2;

                    start\_x -= 1;

                    right\_border -= 1;

                } else {

                    start\_y += 1;

                }

                break;

            case 2:

                if (start\_x - 1 == left\_border) {

                    direction = 3;

                    start\_y -= 1;

                    bottom\_border -= 1;

                } else {

                    start\_x -= 1;

                }

                break;

            case 3:

                if (start\_y - 1 == top\_border) {

                    direction = 0;

                    start\_x += 1;

                    left\_border += 1;

                } else {

                    start\_y -= 1;

                }

                break;

        }

    }

}

时间复杂度：O（m \* n），m 和 n 是数组的长宽。

空间复杂度：O（1）。

**总**

在 leetcode 的 solution 和 discuss 看了下，基本就是这个思路了，只是实现上有些不同，怎么用来标记是否走过，当前方向，怎么遍历，实现有些不同，但本质上是一样的。就是充分理解题意，然后模仿遍历的过程。

**Style 2:**

**Wrong Solution:**

**If we remove condition in each for loop as "result.size() < m \* n" it will error out as below:**

**Input**

**matrix = [[1,2,3],[4,5,6],[7,8,9]]**

**Output**

**[1,2,3,6,9,8,7,4,5,5]**

**Expected**

**[1,2,3,6,9,8,7,4,5]**

class Solution {

    public List<Integer> spiralOrder(int[][] matrix) {

        List<Integer> result = new LinkedList<>();

        if (matrix == null || matrix.length == 0) return result;

        int m = matrix.length;

        int n = matrix[0].length;

        int up = 0,  down = m - 1;

        int left = 0, right = n - 1;

        while (result.size() < m \* n) {

            // Go right

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

                result.add(matrix[up][j]);

            }

            // Go down

            for (int i = up + 1; i <= down - 1; i++) {

                result.add(matrix[i][right]);

            }

            // Go left

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

                result.add(matrix[down][j]);

            }

            // Go Up

            for (int i = down - 1; i >= up + 1; i--) {

                result.add(matrix[i][left]);

            }

            // Update board

            left++; right--; up++; down--;

        }

        return result;

    }

}

**Correct Solution**

class Solution {

    public List<Integer> spiralOrder(int[][] matrix) {

        List<Integer> result = new LinkedList<>();

        if (matrix == null || matrix.length == 0) return result;

        int m = matrix.length;

        int n = matrix[0].length;

        int up = 0,  down = m - 1;

        int left = 0, right = n - 1;

        while (result.size() < m \* n) {

            // Go right

            for (int j = left; j <= right && result.size() < m \* n; j++) {

                result.add(matrix[up][j]);

            }

            // Go down

            for (int i = up + 1; i <= down - 1 && result.size() < m \* n; i++) {

                result.add(matrix[i][right]);

            }

            // Go left

            for (int j = right; j >= left && result.size() < m \* n; j--) {

                result.add(matrix[down][j]);

            }

            // Go Up

            for (int i = down - 1; i >= up + 1 && result.size() < m \* n; i--) {

                result.add(matrix[i][left]);

            }

            // Update board

            left++; right--; up++; down--;

        }

        return result;

    }

}

**Refer to**

<https://leetcode.com/problems/spiral-matrix/solutions/20599/super-simple-and-easy-to-understand-solution/comments/185257>

AN INTERVIEW FRIENDLY SOLUTION, the conditions to check borders are all the same.

res.size() < n \* m

class Solution {

    public List<Integer> spiralOrder(int[][] matrix) {

        List<Integer> res = new LinkedList<>();

        if (matrix == null || matrix.length == 0) return res;

        int n = matrix.length, m = matrix[0].length;

        int up = 0,  down = n - 1;

        int left = 0, right = m - 1;

        while (res.size() < n \* m) {

            for (int j = left; j <= right && res.size() < n \* m; j++)

                res.add(matrix[up][j]);

            for (int i = up + 1; i <= down - 1 && res.size() < n \* m; i++)

                res.add(matrix[i][right]);

            for (int j = right; j >= left && res.size() < n \* m; j--)

                res.add(matrix[down][j]);

            for (int i = down - 1; i >= up + 1 && res.size() < n \* m; i--)

                res.add(matrix[i][left]);

            left++; right--; up++; down--;

        }

        return res;

    }

}

**Style 3:**

class Solution {

    public List<Integer> spiralOrder(int[][] matrix) {

        List<Integer> result = new LinkedList<>();

        if (matrix == null || matrix.length == 0) return result;

        int m = matrix.length;

        int n = matrix[0].length;

        int up = 0,  down = m - 1;

        int left = 0, right = n - 1;

        // Equal way

        //while (result.size() < m \* n) {

        while (up <= down && left <= right) {

            // Go right

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

                result.add(matrix[up][j]);

            }

            up++;

            // Go down

            for (int i = up; i <= down; i++) {

                result.add(matrix[i][right]);

            }

            right--;

            // Go left

            if(up <= down) {

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

                    result.add(matrix[down][j]);

                }

            }

            down--;

            // Go Up

            if(left <= right) {

                for (int i = down; i >= up; i--) {

                    result.add(matrix[i][left]);

                }

            }

            left++;

        }

        return result;

    }

}

**Refer to**

<https://leetcode.com/problems/spiral-matrix/solutions/20599/super-simple-and-easy-to-understand-solution/>

This is a very simple and easy to understand solution. I traverse right and increment rowBegin, then traverse down and decrement colEnd, then I traverse left and decrement rowEnd, and finally I traverse up and increment colBegin.

The only tricky part is that when I traverse left or up I have to check whether the row or col still exists to prevent duplicates. If anyone can do the same thing without that check, please let me know!

Any comments greatly appreciated.

public class Solution {

    public List<Integer> spiralOrder(int[][] matrix) {

        List<Integer> res = new ArrayList<Integer>();

        if (matrix.length == 0) {

            return res;

        }

        int rowBegin = 0;

        int rowEnd = matrix.length-1;

        int colBegin = 0;

        int colEnd = matrix[0].length - 1;

        while (rowBegin <= rowEnd && colBegin <= colEnd) {

            // Traverse Right

            for (int j = colBegin; j <= colEnd; j ++) {

                res.add(matrix[rowBegin][j]);

            }

            rowBegin++;

            // Traverse Down

            for (int j = rowBegin; j <= rowEnd; j ++) {

                res.add(matrix[j][colEnd]);

            }

            colEnd--;

            if (rowBegin <= rowEnd) {

                // Traverse Left

                for (int j = colEnd; j >= colBegin; j --) {

                    res.add(matrix[rowEnd][j]);

                }

            }

            rowEnd--;

            if (colBegin <= colEnd) {

                // Traver Up

                for (int j = rowEnd; j >= rowBegin; j --) {

                    res.add(matrix[j][colBegin]);

                }

            }

            colBegin ++;

        }

        return res;

    }

}