王争的算法训练营

习题课：找规律题

配套习题（10）：

面试题 01.08. 零矩阵 （简单）

剑指 Offer 61. 扑克牌中的顺子 （中等）

面试题 16.11. 跳水板（简单）

面试题 01.05. 一次编辑（中等）

面试题 16.15. 珠玑妙算 （中等）

面试题 16.04. 井字游戏（中等）

55. 跳跃游戏 （中等）

48. 旋转图像 （中等）

54. 螺旋矩阵（中等，字节电商 22 届暑期实习，出处）

240. 搜索二维矩阵 II （中等）

题型说明：

较常考，但不要花太多时间

跟纯编程题相反，难在找规律，编程实现简单

找规律题也叫做逻辑题，主要考察脑力，有点类似智力问题，但比智力题要简单。只要找到了规律，编程实现一般比较简单。比如90度翻转二维矩阵，顺时针循环打印二维矩阵。

如何准备这类题型?

没有固定套路和细分题型。整体上面试中考到的题目，规律都不会很难想，考察的是脑力，需要长期锻炼，很难突击，建议大家不要浪费时间在这个上面，碰到题目就做一下，碰不到就算了。

解题技巧：

先抛开它是一道算法题，抛开要用计算机解决，就用你人脑去解决，你觉得应该怎么做。举一些具体的例子，看每个具体的例子如何来求解。从中得到启发，总结出规律。

找规律题

面试题 01.08. 零矩阵（简单）

编写一种算法，若M × N矩阵中某个元素为0，则将其所在的行与列清零。

示例 1：

```
输入：
[
  [1,1,1],
  [1,0,1],
  [1,1,1]
]
输出：
[
  [1,0,1],
  [0,0,0],
  [1,0,1]
]
```

示例 2：

```
输入：
[
  [0,1,2,0],
  [3,4,5,2],
  [1,3,1,5]
]
输出：
[
  [0,0,0,0],
  [0,4,5,0],
  [0,3,1,0]
]
```

```java
class Solution {
    public void setZeroes(int[][] matrix) {
        int n = matrix.length;
        if (n == 0) return;
        int m = matrix[0].length;
        // 哪一行，哪一列要清空为0
        boolean[] zeroRows = new boolean[n];
        boolean[] zeroColumns = new boolean[m];
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (matrix[i][j] == 0) {
                    zeroRows[i] = true;
                    zeroColumns[j] = true;
                }
            }
        }
        // 设置为0
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                if (zeroRows[i] || zeroColumns[j]) {
                    matrix[i][j] = 0;
                }
            }
        }
    }
}
```

时间复杂度：O(n*m)，空间复杂度：O(n+m)

剑指 Offer 61. 扑克牌中的顺子

剑指 Offer 61. 扑克牌中的顺子

难度 简单　110　收藏　分享　切换为英文　接收动态　反馈

从扑克牌中随机抽5张牌，判断是不是一个顺子，即这5张牌是不是连续的。2～10为数字本身，A为1，J为11，Q为12，K为13，而大、小王为 0 ，可以看成任意数字。A 不能视为 14。

示例 1:

```
输入: [1,2,3,4,5]
输出: True
```

举例->总结规律，猜想->举例验证

找规律题

例题讲解： 例题1：剑指 Offer 61. 扑克牌中的顺子

```
class Solution {
    public boolean isStraight(int[] nums) {
        boolean[] dup = new boolean[14];
        int min = 100;
        int max = -1;
        for (int i = 0; i < 5; i++) {
            if (nums[i] != 0) {
                if (dup[nums[i]]) return false;
                else dup[nums[i]] = true;

                if (nums[i] < min) min = nums[i];
                if (nums[i] > max) max = nums[i];
            }
        }
        return (max-min) < 5;
    }
}
```

时间复杂度：O(1)，空间复杂度：O(1)

找规律题

面试题 16.11. 跳水板（简单）

你正在使用一堆木板建造跳水板。有两种类型的木板，其中长度较短的木板长度为 `shorter`，长度较长的木板长度为 `longer`。你必须正好使用 `k` 块木板。编写一个方法，生成跳水板所有可能的长度。

返回的长度需要从小到大排列。

示例 1

```
输入：
shorter = 1
longer = 2
k = 3
输出：  [3,4,5,6]
解释：
可以使用 3 次 shorter，得到结果 3；使用 2 次 shorter 和 1 次 longer，得到结果 4 。
以此类推，得到最终结果。
```

提示：

- 0 < shorter <= longer
- 0 <= k <= 100000

找规律题

面试题 16.11. 跳水板（简单）

时间复杂度：O(k)，空间复杂度：O(1)（不把result算进去的话）

```
class Solution {
    public int[] divingBoard(int shorter, int longer, int k) {
        // 特殊情况处理
        if (k == 0) return new int[0];
        if (shorter == longer) return new int[] {k*shorter};

        int[] result = new int[k+1];
        // 长板子个数：0、1、2...k
        for (int i = 0; i <= k; ++i) {
            result[i] = i*longer + (k-i)*shorter;
        }
        return result;
    }
}
```

面试题 01.05. 一次编辑（中等）

字符串有三种编辑操作:插入一个字符、删除一个字符或者替换一个字符。 给定两个字符串，编写一个函数判定它们是否只需要一次(或者零次)编辑。

示例 1:

```
输入:
first = "pale"
second = "ple"
输出: True
```

示例 2:

```
输入:
first = "pales"
second = "pal"
输出: False
```

```java
class Solution {
    public boolean oneEditAway(String first, String second) {
        int n  = first.length();
        int m = second.length();
        // 长度相差大于1，无法通过一次编辑匹配
        if (Math.abs(n-m)>1) return false;

        // 长度相等，要么完全相同，要么只有一个不同
        int diffCount = 0;
        if (n == m) {
            for (int i = 0; i < first.length(); ++i) {
                if (first.charAt(i) != second.charAt(i)) {
                    diffCount++;
                }
            }
            return diffCount <= 1;
        }

        // 长度相差1，插入或者删除
        diffCount = 0;
        int i = 0;
        int j = 0;
        while (i < n && j < m) {
            if (first.charAt(i) == second.charAt(j)) {
                i++;
                j++;
            } else {
                diffCount++;
                if (n > m) {
                    i++;
                } else {
                    j++;
                }
            }
        }
        return diffCount <= 1;
    }
}
```

时间复杂度：O(n)，空间复杂度：O(1)

找规律题

面试题 16.15. 珠玑妙算 （中等）

珠玑妙算游戏（the game of master mind）的玩法如下。

计算机有4个槽，每个槽放一个球，颜色可能是红色（R）、黄色（Y）、绿色（G）或蓝色（B）。例如，计算机可能有RGGB 4种（槽1为红色，槽2、3为绿色，槽4为蓝色）。作为用户，你试图猜出颜色组合。打个比方，你可能会猜YRGB。要是猜对某个槽的颜色，则算一次“猜中”；要是只猜对颜色但槽位猜错了，则算一次“伪猜中”。注意，“猜中”不能算入“伪猜中”。

给定一种颜色组合 `solution` 和一个猜测 `guess`，编写一个方法，返回猜中和伪猜中的次数 `answer`，其中 `answer[0]` 为猜中的次数，`answer[1]` 为伪猜中的次数。

示例：

```
输入： solution="RGBY",guess="GGRR"
输出： [1,1]
解释： 猜中1次，伪猜中1次。
```

提示：

- `len(solution) = len(guess) = 4`
- `solution` 和 `guess` 仅包含 `"R"`, `"G"`, `"B"`, `"Y"` 这4种字符

```java
class Solution {
    public int[] masterMind(String solution, String guess) {
        int n = solution.length();
        boolean[] hited = new boolean[n];//guess中哪些字符已经猜中了
        boolean[] used = new boolean[n];//solution中哪些字符已经被匹配用掉了
        // 先计算猜中的
        int hitCount = 0;
        for (int i = 0; i < n; ++i) {
            if (solution.charAt(i) == guess.charAt(i)) {
                hited[i] = true;
                used[i] = true;
                hitCount++;
            }
        }
        // 再计算伪猜中的
        int fakeHitCount = 0;
        for (int i = 0; i < n; ++i) {
            if (hited[i]) continue;
            // 拿每个guess中的字符到solution中查找
            for (int j = 0; j < n; ++j) {
                if (solution.charAt(j) == guess.charAt(i) && !used[j]) {
                    used[j] = true;
                    fakeHitCount++;
                    break;
                }
            }
        }
        return new int[] {hitCount, fakeHitCount};
    }
}
```

可以优化，去掉hited数组

时间复杂度：O(n)，空间复杂度：O(n)

找规律题

面试题 16.04. 井字游戏（中等）

设计一个算法，判断玩家是否赢了井字游戏。输入是一个 N x N 的数组棋盘，由字符" "，"X"和"O"组成，其中字符" "代表一个空位。

以下是井字游戏的规则：

- 玩家轮流将字符放入空位（" "）中。
- 第一个玩家总是放字符"O"，且第二个玩家总是放字符"X"。
- "X"和"O"只允许放置在空位中，不允许对已放有字符的位置进行填充。
- 当有N个相同（且非空）的字符填充任何行、列或对角线时，游戏结束，对应该字符的玩家获胜。
- 当所有位置非空时，也算为游戏结束。
- 如果游戏结束，玩家不允许再放置字符。

如果游戏存在获胜者，就返回该游戏的获胜者使用的字符（"X"或"O"）；如果游戏以平局结束，则返回"Draw"；如果仍会有行动（游戏未结束），则返回 "Pending"。

示例 1：

```
输入： board = ["O X"," XO","X O"]
输出： "X"
```

示例 2：

```
输入： board = ["OOX","XXO","OXO"]
输出： "Draw"
解释： 没有玩家获胜且不存在空位
```

示例 3：

```
输入： board = ["OOX","XXO","OX "]
输出： "Pending"
解释： 没有玩家获胜且仍存在空位
```

```java
class Solution {
    public String tictactoe(String[] board) {
        int n = board.length;
        char[][] boards = new char[n][n];
        for (int i = 0; i < n; ++i) {
            boards[i] = board[i].toCharArray();
        }

        boolean determined = false; //表示是否已经发现有人赢了
        // 检查行
        for (int i = 0; i < n; ++i) {
            if (boards[i][0] == ' ') continue;
            determined = true;
            for (int j = 1; j < n; ++j) {
                if (boards[i][j] != boards[i][0]) {
                    determined = false;
                    break;
                }
            }
            if (determined) return "" + boards[i][0];
        }
        // 检查列
        for (int j = 0; j < n; ++j) {
            if (boards[0][j] == ' ') continue;
            determined = true;
            for (int i = 1; i < n; ++i) {
                if (boards[i][j] != boards[0][j]) {
                    determined = false;
                }
            }
            if (determined) return "" + boards[0][j];
        }
```

可以换一种判断方法

时间复杂度：O(n^2)，空间复杂度：O(1)

```
// 检查对角线:左上->右下
if (boards[0][0] != ' ') {
    int i = 1;
    int j = 1;
    determined = true;
    while (i < n && j < n) {
        if (boards[i][j] != boards[0][0]) {
            determined = false;
            break;
        }
        i++;
        j++;
    }
    if (determined) return boards[0][0] + "";
}
// 检查对角线:左下->右上
if (boards[n-1][0] != ' ') {
    int i = n-2;
    int j = 1;
    determined = true;
    while (i >= 0 && j < n) {
        if (boards[i][j] != boards[n-1][0]) {
            determined = false;
            break;
        }
        i--;
        j++;
    }
    if (determined) return "" + boards[n-1][0];
}
```

```
        // 上面没有找到哪方赢，判定游戏是否还能继续玩
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (boards[i][j] == ' ') return "Pending";
            }
        }
        // 游戏结束了，平局
        return "Draw";
    }
}
```

找规律题

55. 跳跃游戏 （中等）

给定一个非负整数数组 `nums` ，你最初位于数组的 **第一个下标** 。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

判断你是否能够到达最后一个下标。

示例 1：

```
输入：nums = [2,3,1,1,4]
输出：true
解释：可以先跳 1 步，从下标 0 到达下标 1，然后再从下标 1 跳 3 步到达最后一个下标。
```

示例 2：

```
输入：nums = [3,2,1,0,4]
输出：false
解释：无论怎样，总会到达下标为 3 的位置。但该下标的最大跳跃长度是 0 ， 所以永远不可能到达最后一个下标。
```

提示：

- `1 <= nums.length <= 3 * 10^4`
- `0 <= nums[i] <= 10^5`

55. 跳跃游戏 （中等）

时间复杂度：O(n)，空间复杂度：O(1)

```
class Solution {
    public boolean canJump(int[] nums) {
      int reachedMax = 0;
      for (int i = 0; i < nums.length; ++i) {
        if (i > reachedMax) return false;
        if (i+nums[i] > reachedMax) {
          reachedMax = i + nums[i];
        }
        if (reachedMax >= nums.length-1) return true;
      }
      return false;
    }
}
```

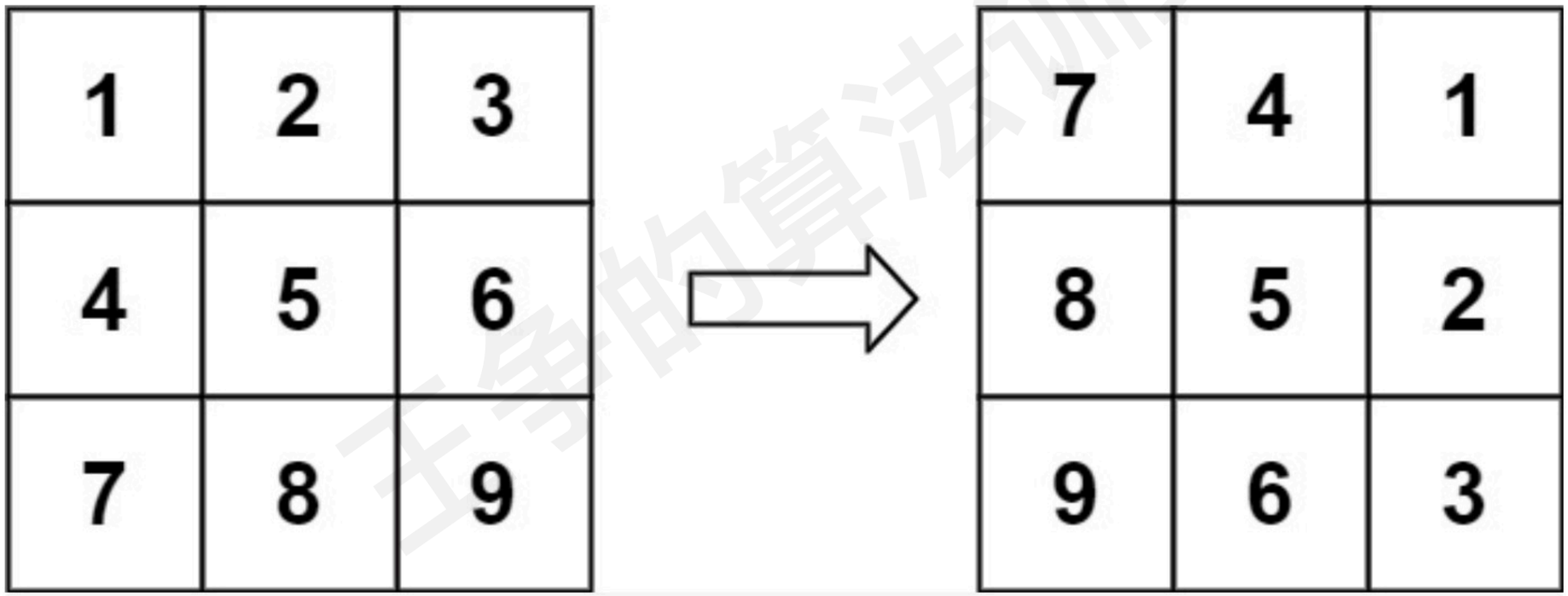
找规律题

48. 旋转图像 （中等）

给定一个 *n* × *n* 的二维矩阵 `matrix` 表示一个图像。请你将图像顺时针旋转 90 度。

你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。**请不要** 使用另一个矩阵来旋转图像。

示例 1：




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

48. 旋转图像 （中等）

三种解法：

1）借助辅助数组（非原地）

2）用翻转代替旋转（原地）

3）标准原地旋转（原地）

1）借助辅助数组

时间复杂度：O(n^2)，空间复杂度：O(n^2)

```
class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        int[][] tmp = new int[n][n];
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                tmp[j][n-i-1] = matrix[i][j];
            }
        }
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                matrix[i][j] = tmp[i][j];
            }
        }
    }
}
```

2）用翻转代替旋转

时间复杂度：O(n^2)，空间复杂度：O(1)

```
class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        // 先上下翻转
        for (int i = 0; i < n/2; ++i) {
            for (int j = 0; j < n; ++j) {
                swap(matrix, i, j, n-i-1, j);
            }
        }
        // 再对角翻转（左上-右下）
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < i; ++j) {
                swap(matrix, i, j, j, i);
            }
        }
    }

    private void swap(int[][] matrix, int i, int j, int p, int q) {
        int tmp = matrix[i][j];
        matrix[i][j] = matrix[p][q];
        matrix[p][q] = tmp;
    }
}
```

3）标准原地旋转

```
private void swap4(int[][] a, int i1, int j1, int i2, int j2, int i3, int j3, int i4, int j4) {
    int tmp = a[i1][j1];
    a[i1][j1] = a[i4][j4];
    a[i4][j4] = a[i3][j3];
    a[i3][j3] = a[i2][j2];
    a[i2][j2] = tmp;
}
```

3）标准原地旋转

时间复杂度：O(n^2)，空间复杂度：O(1)

```
class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        int s1_i = 0;
        int s1_j = 0;
        while (n > 1) {
            int s2_i = s1_i;
            int s2_j = s1_j+n-1;
            int s3_i = s1_i+n-1;
            int s3_j = s1_j+n-1;
            int s4_i = s1_i+n-1;
            int s4_j = s1_j;

            for (int move = 0; move<=n-2; ++move) {
                int p1_i = s1_i + 0;
                int p1_j = s1_j + move;
                int p2_i = s2_i + move;
                int p2_j = s2_j + 0;
                int p3_i = s3_i + 0;
                int p3_j = s3_j - move;
                int p4_i = s4_i - move;
                int p4_j = s4_j + 0;
                swap4(matrix, p1_i, p1_j, p2_i, p2_j, p3_i, p3_j, p4_i, p4_j);
            }
            s1_i++;
            s1_j++;
            n-=2;
        }
    }
}
```

拓展题型：

1. n*n的二维矩阵，沿上下中线翻转、沿左右中线翻转

2. n*n的二维矩阵，沿左上-右下对角线翻转、沿左下-右上对角线翻转

3. n*n的二维矩阵，旋转90度、180度、270度

4. n*m的二维矩阵，沿上下中线翻转、沿左右中线翻转

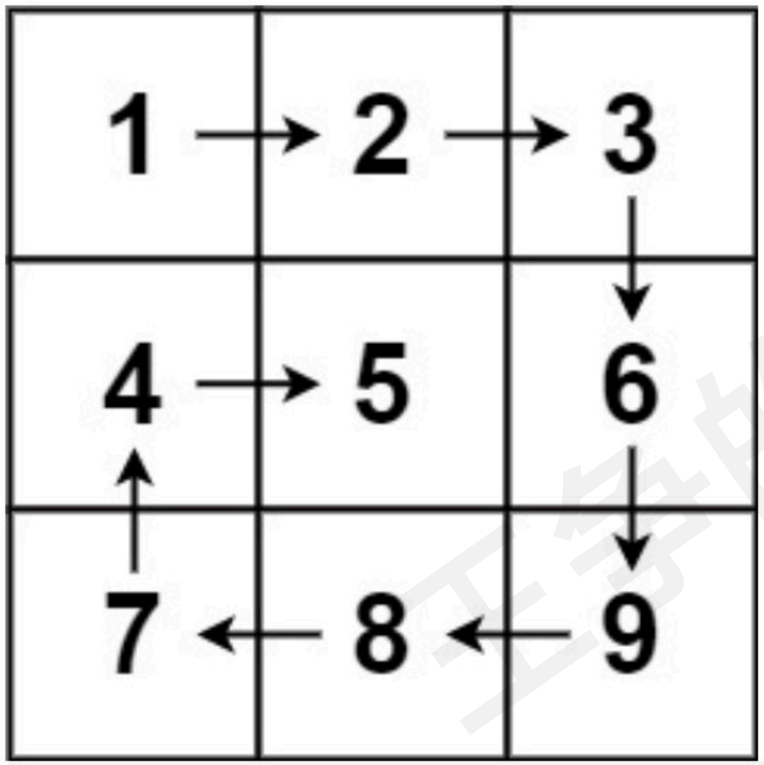
5. n*m的二维矩阵，沿左上-右下对角线翻转、沿左下-右上对角线翻转（无法实现）

6. n*m的二维矩阵，旋转90度、180度、270度 （需要重新申请新的存储数组）

54. 螺旋矩阵（中等）

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。

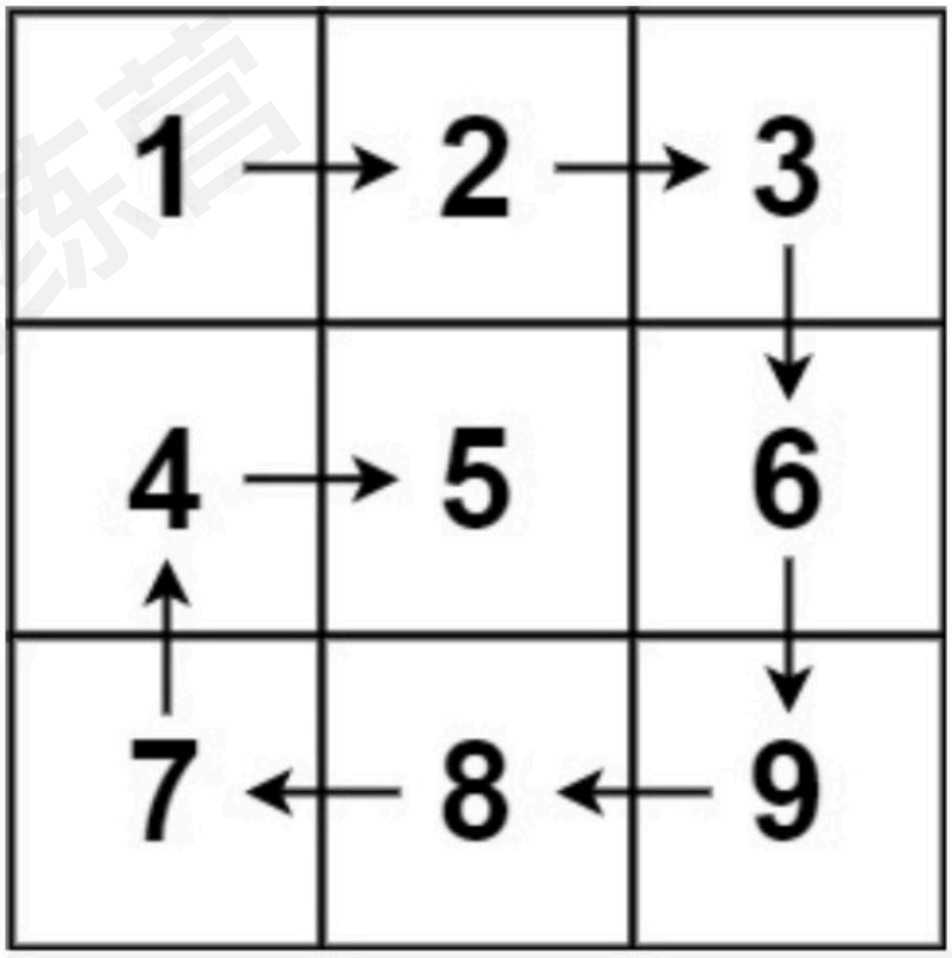
示例 1：




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

```java
public List<Integer> spiralOrder(int[][] matrix) {
    int m = matrix.length;
    int n = matrix[0].length;
    List<Integer> result = new ArrayList<>();
    int left = 0;
    int right = n-1;
    int top = 0;
    int bottom = m-1;
    while (left<=right && top <= bottom) {
        for (int j = left; j <= right; ++j) {
            result.add(matrix[top][j]);
        }
        for (int i = top+1; i <= bottom; ++i) {
            result.add(matrix[i][right]);
        }
        if (top != bottom) {
            for (int j = right-1; j >= left; --j) {
                result.add(matrix[bottom][j]);
            }
        }
        if (left != right) {
            for (int i = bottom-1; i > top; --i) {
                result.add(matrix[i][left]);
            }
        }
        left++;
        right--;
        top++;
        bottom--;
    }
    return result;
}
```

时间复杂度：O(m*n)，空间复杂度：O(1) 不把result算进去的话



找规律题

240. 搜索二维矩阵 II （中等）

编写一个高效的算法来搜索 `m x n` 矩阵 `matrix` 中的一个目标值 `target` 。该矩阵具有以下特性：

- 每行的元素从左到右升序排列。
- 每列的元素从上到下升序排列。

示例 1：

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

```java
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int h = matrix.length;
        int w = matrix[0].length;
        int i = 0;
        int j = w-1;
        // 根据matrix[i][j]跟target的大小关系，一层一层的剥离
        while (i <= h-1 && j>= 0) {
            if (matrix[i][j] == target) {
                return true;
            }

            if (matrix[i][j] > target) {
                j--;
                continue;
            }

            if (matrix[i][j] < target) {
                i++;
                continue;
            }
        }
        return false;
    }
}
```

时间复杂度：O(h+w)，空间复杂度：O(1)

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

提问环节