# Leetcode 题解 - 数组与矩阵

* [Leetcode 题解 - 数组与矩阵](#leetcode-题解---数组与矩阵)
	+ [1. 把数组中的 0 移到末尾](#1-把数组中的-0-移到末尾)
	+ [2. 改变矩阵维度](#2-改变矩阵维度)
	+ [3. 找出数组中最长的连续 1](#3-找出数组中最长的连续-1)
	+ [4. 有序矩阵查找](#4-有序矩阵查找)
	+ [5. 有序矩阵的 Kth Element](#5-有序矩阵的-kth-element)
	+ [6. 一个数组元素在 [1, n] 之间，其中一个数被替换为另一个数，找出重复的数和丢失的数](#6-一个数组元素在-%5B1-n%5D-之间，其中一个数被替换为另一个数，找出重复的数和丢失的数)
	+ [7. 找出数组中重复的数，数组值在 [1, n] 之间](#7-找出数组中重复的数，数组值在-%5B1-n%5D-之间)
	+ [8. 数组相邻差值的个数](#8-数组相邻差值的个数)
	+ [9. 数组的度](#9-数组的度)
	+ [10. 对角元素相等的矩阵](#10-对角元素相等的矩阵)
	+ [11. 嵌套数组](#11-嵌套数组)
	+ [12. 分隔数组](#12-分隔数组)

## 1. 把数组中的 0 移到末尾

283. Move Zeroes (Easy)

[Leetcode](https://leetcode.com/problems/move-zeroes/description/) / [力扣](https://leetcode-cn.com/problems/move-zeroes/description/)

For example, given nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].

public void moveZeroes(int[] nums) {
 int idx = 0;
 for (int num : nums) {
 if (num != 0) {
 nums[idx++] = num;
 }
 }
 while (idx < nums.length) {
 nums[idx++] = 0;
 }
}

## 2. 改变矩阵维度

566. Reshape the Matrix (Easy)

[Leetcode](https://leetcode.com/problems/reshape-the-matrix/description/) / [力扣](https://leetcode-cn.com/problems/reshape-the-matrix/description/)

Input:
nums =
[[1,2],
 [3,4]]
r = 1, c = 4

Output:
[[1,2,3,4]]

Explanation:
The row-traversing of nums is [1,2,3,4]. The new reshaped matrix is a 1 \* 4 matrix, fill it row by row by using the previous list.

public int[][] matrixReshape(int[][] nums, int r, int c) {
 int m = nums.length, n = nums[0].length;
 if (m \* n != r \* c) {
 return nums;
 }
 int[][] reshapedNums = new int[r][c];
 int index = 0;
 for (int i = 0; i < r; i++) {
 for (int j = 0; j < c; j++) {
 reshapedNums[i][j] = nums[index / n][index % n];
 index++;
 }
 }
 return reshapedNums;
}

## 3. 找出数组中最长的连续 1

485. Max Consecutive Ones (Easy)

[Leetcode](https://leetcode.com/problems/max-consecutive-ones/description/) / [力扣](https://leetcode-cn.com/problems/max-consecutive-ones/description/)

public int findMaxConsecutiveOnes(int[] nums) {
 int max = 0, cur = 0;
 for (int x : nums) {
 cur = x == 0 ? 0 : cur + 1;
 max = Math.max(max, cur);
 }
 return max;
}

## 4. 有序矩阵查找

240. Search a 2D Matrix II (Medium)

[Leetcode](https://leetcode.com/problems/search-a-2d-matrix-ii/description/) / [力扣](https://leetcode-cn.com/problems/search-a-2d-matrix-ii/description/)

[
 [ 1, 5, 9],
 [10, 11, 13],
 [12, 13, 15]
]

public boolean searchMatrix(int[][] matrix, int target) {
 if (matrix == null || matrix.length == 0 || matrix[0].length == 0) return false;
 int m = matrix.length, n = matrix[0].length;
 int row = 0, col = n - 1;
 while (row < m && col >= 0) {
 if (target == matrix[row][col]) return true;
 else if (target < matrix[row][col]) col--;
 else row++;
 }
 return false;
}

## 5. 有序矩阵的 Kth Element

378. Kth Smallest Element in a Sorted Matrix ((Medium))

[Leetcode](https://leetcode.com/problems/kth-smallest-element-in-a-sorted-matrix/description/) / [力扣](https://leetcode-cn.com/problems/kth-smallest-element-in-a-sorted-matrix/description/)

matrix = [
 [ 1, 5, 9],
 [10, 11, 13],
 [12, 13, 15]
],
k = 8,

return 13.

解题参考：[Share my thoughts and Clean Java Code](https://leetcode-cn.com/problems/kth-smallest-element-in-a-sorted-matrix/discuss/85173)

二分查找解法：

public int kthSmallest(int[][] matrix, int k) {
 int m = matrix.length, n = matrix[0].length;
 int lo = matrix[0][0], hi = matrix[m - 1][n - 1];
 while (lo <= hi) {
 int mid = lo + (hi - lo) / 2;
 int cnt = 0;
 for (int i = 0; i < m; i++) {
 for (int j = 0; j < n && matrix[i][j] <= mid; j++) {
 cnt++;
 }
 }
 if (cnt < k) lo = mid + 1;
 else hi = mid - 1;
 }
 return lo;
}

堆解法：

public int kthSmallest(int[][] matrix, int k) {
 int m = matrix.length, n = matrix[0].length;
 PriorityQueue<Tuple> pq = new PriorityQueue<Tuple>();
 for(int j = 0; j < n; j++) pq.offer(new Tuple(0, j, matrix[0][j]));
 for(int i = 0; i < k - 1; i++) { // 小根堆，去掉 k - 1 个堆顶元素，此时堆顶元素就是第 k 的数
 Tuple t = pq.poll();
 if(t.x == m - 1) continue;
 pq.offer(new Tuple(t.x + 1, t.y, matrix[t.x + 1][t.y]));
 }
 return pq.poll().val;
}

class Tuple implements Comparable<Tuple> {
 int x, y, val;
 public Tuple(int x, int y, int val) {
 this.x = x; this.y = y; this.val = val;
 }

 @Override
 public int compareTo(Tuple that) {
 return this.val - that.val;
 }
}

## 6. 一个数组元素在 [1, n] 之间，其中一个数被替换为另一个数，找出重复的数和丢失的数

645. Set Mismatch (Easy)

[Leetcode](https://leetcode.com/problems/set-mismatch/description/) / [力扣](https://leetcode-cn.com/problems/set-mismatch/description/)

Input: nums = [1,2,2,4]
Output: [2,3]

Input: nums = [1,2,2,4]
Output: [2,3]

最直接的方法是先对数组进行排序，这种方法时间复杂度为 O(NlogN)。本题可以以 O(N) 的时间复杂度、O(1) 空间复杂度来求解。

主要思想是通过交换数组元素，使得数组上的元素在正确的位置上。

public int[] findErrorNums(int[] nums) {
 for (int i = 0; i < nums.length; i++) {
 while (nums[i] != i + 1 && nums[nums[i] - 1] != nums[i]) {
 swap(nums, i, nums[i] - 1);
 }
 }
 for (int i = 0; i < nums.length; i++) {
 if (nums[i] != i + 1) {
 return new int[]{nums[i], i + 1};
 }
 }
 return null;
}

private void swap(int[] nums, int i, int j) {
 int tmp = nums[i];
 nums[i] = nums[j];
 nums[j] = tmp;
}

## 7. 找出数组中重复的数，数组值在 [1, n] 之间

287. Find the Duplicate Number (Medium)

[Leetcode](https://leetcode.com/problems/find-the-duplicate-number/description/) / [力扣](https://leetcode-cn.com/problems/find-the-duplicate-number/description/)

要求不能修改数组，也不能使用额外的空间。

二分查找解法：

public int findDuplicate(int[] nums) {
 int l = 1, h = nums.length - 1;
 while (l <= h) {
 int mid = l + (h - l) / 2;
 int cnt = 0;
 for (int i = 0; i < nums.length; i++) {
 if (nums[i] <= mid) cnt++;
 }
 if (cnt > mid) h = mid - 1;
 else l = mid + 1;
 }
 return l;
}

双指针解法，类似于有环链表中找出环的入口：

public int findDuplicate(int[] nums) {
 int slow = nums[0], fast = nums[nums[0]];
 while (slow != fast) {
 slow = nums[slow];
 fast = nums[nums[fast]];
 }
 fast = 0;
 while (slow != fast) {
 slow = nums[slow];
 fast = nums[fast];
 }
 return slow;
}

## 8. 数组相邻差值的个数

667. Beautiful Arrangement II (Medium)

[Leetcode](https://leetcode.com/problems/beautiful-arrangement-ii/description/) / [力扣](https://leetcode-cn.com/problems/beautiful-arrangement-ii/description/)

Input: n = 3, k = 2
Output: [1, 3, 2]
Explanation: The [1, 3, 2] has three different positive integers ranging from 1 to 3, and the [2, 1] has exactly 2 distinct integers: 1 and 2.

题目描述：数组元素为 1~n 的整数，要求构建数组，使得相邻元素的差值不相同的个数为 k。

让前 k+1 个元素构建出 k 个不相同的差值，序列为：1 k+1 2 k 3 k-1 ... k/2 k/2+1.

public int[] constructArray(int n, int k) {
 int[] ret = new int[n];
 ret[0] = 1;
 for (int i = 1, interval = k; i <= k; i++, interval--) {
 ret[i] = i % 2 == 1 ? ret[i - 1] + interval : ret[i - 1] - interval;
 }
 for (int i = k + 1; i < n; i++) {
 ret[i] = i + 1;
 }
 return ret;
}

## 9. 数组的度

697. Degree of an Array (Easy)

[Leetcode](https://leetcode.com/problems/degree-of-an-array/description/) / [力扣](https://leetcode-cn.com/problems/degree-of-an-array/description/)

Input: [1,2,2,3,1,4,2]
Output: 6

题目描述：数组的度定义为元素出现的最高频率，例如上面的数组度为 3。要求找到一个最小的子数组，这个子数组的度和原数组一样。

public int findShortestSubArray(int[] nums) {
 Map<Integer, Integer> numsCnt = new HashMap<>();
 Map<Integer, Integer> numsLastIndex = new HashMap<>();
 Map<Integer, Integer> numsFirstIndex = new HashMap<>();
 for (int i = 0; i < nums.length; i++) {
 int num = nums[i];
 numsCnt.put(num, numsCnt.getOrDefault(num, 0) + 1);
 numsLastIndex.put(num, i);
 if (!numsFirstIndex.containsKey(num)) {
 numsFirstIndex.put(num, i);
 }
 }
 int maxCnt = 0;
 for (int num : nums) {
 maxCnt = Math.max(maxCnt, numsCnt.get(num));
 }
 int ret = nums.length;
 for (int i = 0; i < nums.length; i++) {
 int num = nums[i];
 int cnt = numsCnt.get(num);
 if (cnt != maxCnt) continue;
 ret = Math.min(ret, numsLastIndex.get(num) - numsFirstIndex.get(num) + 1);
 }
 return ret;
}

## 10. 对角元素相等的矩阵

766. Toeplitz Matrix (Easy)

[Leetcode](https://leetcode.com/problems/toeplitz-matrix/description/) / [力扣](https://leetcode-cn.com/problems/toeplitz-matrix/description/)

1234
5123
9512

In the above grid, the diagonals are "[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]", and in each diagonal all elements are the same, so the answer is True.

public boolean isToeplitzMatrix(int[][] matrix) {
 for (int i = 0; i < matrix[0].length; i++) {
 if (!check(matrix, matrix[0][i], 0, i)) {
 return false;
 }
 }
 for (int i = 0; i < matrix.length; i++) {
 if (!check(matrix, matrix[i][0], i, 0)) {
 return false;
 }
 }
 return true;
}

private boolean check(int[][] matrix, int expectValue, int row, int col) {
 if (row >= matrix.length || col >= matrix[0].length) {
 return true;
 }
 if (matrix[row][col] != expectValue) {
 return false;
 }
 return check(matrix, expectValue, row + 1, col + 1);
}

## 11. 嵌套数组

565. Array Nesting (Medium)

[Leetcode](https://leetcode.com/problems/array-nesting/description/) / [力扣](https://leetcode-cn.com/problems/array-nesting/description/)

Input: A = [5,4,0,3,1,6,2]
Output: 4
Explanation:
A[0] = 5, A[1] = 4, A[2] = 0, A[3] = 3, A[4] = 1, A[5] = 6, A[6] = 2.

One of the longest S[K]:
S[0] = {A[0], A[5], A[6], A[2]} = {5, 6, 2, 0}

题目描述：S[i] 表示一个集合，集合的第一个元素是 A[i]，第二个元素是 A[A[i]]，如此嵌套下去。求最大的 S[i]。

public int arrayNesting(int[] nums) {
 int max = 0;
 for (int i = 0; i < nums.length; i++) {
 int cnt = 0;
 for (int j = i; nums[j] != -1; ) {
 cnt++;
 int t = nums[j];
 nums[j] = -1; // 标记该位置已经被访问
 j = t;

 }
 max = Math.max(max, cnt);
 }
 return max;
}

## 12. 分隔数组

769. Max Chunks To Make Sorted (Medium)

[Leetcode](https://leetcode.com/problems/max-chunks-to-make-sorted/description/) / [力扣](https://leetcode-cn.com/problems/max-chunks-to-make-sorted/description/)

Input: arr = [1,0,2,3,4]
Output: 4
Explanation:
We can split into two chunks, such as [1, 0], [2, 3, 4].
However, splitting into [1, 0], [2], [3], [4] is the highest number of chunks possible.

题目描述：分隔数组，使得对每部分排序后数组就为有序。

public int maxChunksToSorted(int[] arr) {
 if (arr == null) return 0;
 int ret = 0;
 int right = arr[0];
 for (int i = 0; i < arr.length; i++) {
 right = Math.max(right, arr[i]);
 if (right == i) ret++;
 }
 return ret;
}