189. 旋转数组

地址

旋转数组

题目

- 中文

189. 旋转数组

难度 中等 1020 收藏 分享 切换为英文 接收动态 反馈

给定一个数组，将数组中的元素向右移动 `k` 个位置，其中 `k` 是非负数。

进阶：

- 尽可能想出更多的解决方案，至少有三种不同的方法可以解决这个问题。
- 你可以使用空间复杂度为 O(1) 的 **原地** 算法解决这个问题吗？

示例 1：

```
输入：nums = [1,2,3,4,5,6,7], k = 3
输出：[5,6,7,1,2,3,4]
解释：
向右旋转 1 步：[7,1,2,3,4,5,6]
向右旋转 2 步：[6,7,1,2,3,4,5]
向右旋转 3 步：[5,6,7,1,2,3,4]
```

示例 2：

```
输入：nums = [-1,-100,3,99], k = 2
输出：[3,99,-1,-100]
解释：
向右旋转 1 步：[99,-1,-100,3]
向右旋转 2 步：[3,99,-1,-100]
```

提示：

- $1 <= nums.length <= 2 * 10^4$
- $-2^{31} <= nums[i] <= 2^{31} - 1$
- $0 <= k <= 10^5$

通过次数 285,847 提交次数 629,033

- English

189. Rotate Array

难度 中等 1020 收藏 分享 切换为中文 接收动态 反馈

Given an array, rotate the array to the right by `k` steps, where `k` is non-negative.

Example 1:

```
Input: nums = [1,2,3,4,5,6,7], k = 3
Output: [5,6,7,1,2,3,4]
Explanation:
rotate 1 steps to the right: [7,1,2,3,4,5,6]
rotate 2 steps to the right: [6,7,1,2,3,4,5]
rotate 3 steps to the right: [5,6,7,1,2,3,4]
```

Example 2:

```
Input: nums = [-1,-100,3,99], k = 2
Output: [3,99,-1,-100]
Explanation:
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]
```

Constraints:

- `1 <= nums.length <= 10^5`
- `-2^31 <= nums[i] <= 2^31 - 1`
- `0 <= k <= 10^5`

Follow up:

- Try to come up with as many solutions as you can. There are at least **three** different ways to solve this problem.
- Could you do it in-place with `O(1)` extra space?

思路 1 ：辅助数组

1. 使用两个辅助数组:
    a. 一个存放原数组前部分的元素;
    b. 一个存放原数组后部分的元素.
2. 长度为 1 的情况，数组不需要处理，直接返回；
3. 长度为 2 的情况：
    a. 选取点为偶数，不需要处理，直接返回；

b. 选取点为奇数时，将两个元素对调.

4. 长度为 其他的情况：

a. 移动次数为 k % nums.length;

b. 分别用循环，将数组中的选取点前后部分的元素分别装给前后辅助数组；

c. 辅助数组元素对调，赋值给原数组.

复杂度分析

- 时间复杂度 : O(n).
- 空间复杂度 : O(n).

代码

```
public void rotate(int[] nums, int k) {

        // 数组长度为 1，不需要处理，直接返回
        if (nums.length == 1)
            return;

        // 数组长度为2
        if (nums.length == 2) {
            if (k % 2 == 0) // 当选取点为偶数时，不需处理，直接返回
                return;
            else {          // 当选取点为奇数时，将两个元素位置对调
                int temp = nums[0];
                nums[0] = nums[1];
                nums[1] = temp;
                return;
            }
        }

        // 将要移动的次数
        k = k % nums.length;

        // 辅助数组
        int[] frontElementsTemporary = new int[nums.length - k];
        int[] backElementsTemporary = new int[k];
```

```
        // get the front elements to front temporary array
        for (int i = 0, j = 0; i < nums.length - k; i++, j++) {
            frontElementsTemporary[j] = nums[i];
        }

        int start = nums.length - k;

        // get the back elemnts to back temporary array
        for (int i = start, j = 0; i < nums.length; i++, j++) {
            backElementsTemporary[j] = nums[i];
        }

        // repalce elements
        // back to front
        for (int i = 0; i < k; i++) {
            nums[i] = backElementsTemporary[i];
        }

        // front to back
        for (int i = k, j = 0; i < nums.length; i++, j++) {
            nums[i] = frontElementsTemporary[j];
        }
```

复杂度分析：

- 时间复杂度 ：O(n)
- 空间复杂度 ：O(n)

#Leetcode/Array#