88.合并两个有序数组

地址

合并两个有序数组

题目

88. 合并两个有序数组

难度 简单 1011 收藏 分享 切换为英文 接收动态 反馈

给你两个有序整数数组 `nums1` 和 `nums2` ，请你将 `nums2` 合并到 `nums1` 中，使 `nums1` 成为一个有序数组。

初始化 `nums1` 和 `nums2` 的元素数量分别为 `m` 和 `n` 。你可以假设 `nums1` 的空间大小等于 `m + n` ，这样它就有足够的空间保存来自 `nums2` 的元素。

示例 1：

```
输入：nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
输出：[1,2,2,3,5,6]
```

示例 2：

```
输入：nums1 = [1], m = 1, nums2 = [], n = 0
输出：[1]
```

提示：

- `nums1.length == m + n`
- `nums2.length == n`
- `0 <= m, n <= 200`
- `1 <= m + n <= 200`
- `-10^9 <= nums1[i], nums2[i] <= 10^9`

通过次数 397,984 | 提交次数 778,019

88. Merge Sorted Array

难度 简单 1011 收藏 分享 切换为中文 接收动态 反馈

You are given two integer arrays `nums1` and `nums2`, sorted in **non-decreasing order**, and two integers `m` and `n`, representing the number of elements in `nums1` and `nums2` respectively.

Merge `nums1` and `nums2` into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* `nums1`. To accommodate this, `nums1` has a length of `m + n`, where the first `m` elements denote the elements that should be merged, and the last `n` elements are set to `0` and should be ignored. `nums2` has a length of `n`.

Example 1:

```
Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]
Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from
nums1.
```

Example 2:

```
Input: nums1 = [1], m = 1, nums2 = [], n = 0
Output: [1]
Explanation: The arrays we are merging are [1] and [].
The result of the merge is [1].
```

Example 3:

```
Input: nums1 = [0], m = 0, nums2 = [1], n = 1
Output: [1]
Explanation: The arrays we are merging are [] and [1].
The result of the merge is [1].
Note that because m = 0, there are no elements in nums1. The 0 is only there to
ensure the merge result can fit in nums1.
```

Constraints:

- `nums1.length == m + n`
- `nums2.length == n`
- `0 <= m, n <= 200`
- `1 <= m + n <= 200`
- -10^9 `<= nums1[i], nums2[j] <=` 10^9

Follow up: Can you come up with an algorithm that runs in `O(m + n)` time?

思路 1 ：双指针逆向尾部遍历

1. nums1.length >= m+n 意味着考察 **原地遍历**，将空间复杂度降到 **O（1）**；
2. 所以，可以将 nums2 放入 nums1 中操作，不用再申请额外的内存空间;

3. 原地修改时，为了避免从前往后遍历导致原有数组元素被破坏掉，要选择从后往前遍历;
4. 创建三个指针:
   - p1 : 指向 nums1 初始化元素数量的末尾；
   - p2 : 指向 nums2 初始化元素数量的末尾；
   - tail : 指向 nums1 的末尾，用来存放待排元素.
5. 遍历循环结束条件： 当 p1 和 p2 都小于 0，即 nums1 和 nums2 的初始化元素数量都被遍历完全.

复杂度分分析

- 时间复杂度：O(n)
- 空间复杂度：O(1)

代码

```
// Java
// Time : 2021 - 07 -18

public void merge(int[] nums1, int m, int[] nums2, int n) {

        int p1 = m - 1, p2 = n - 1;     // p1 - nums1 尾指针, p2 - nums2 尾指针
        int tail = m + n - 1;           // 存储尾指针
        int cur;                        // 获取存储值
        while (p1 >= 0 || p2 >= 0) {    // p1 && p2 == -1 时，遍历比较结束
            if (p1 == -1) {             // nums1 待排元素为空
                cur = nums2[p2--];      // 获取 nums2 的待排元素
            } else if (p2 == -1) {      // nums2 待排元素为空
                cur = nums1[p1--];      // 获取 nums1 的待排元素
            } else if (nums1[p1] > nums2[p2]) { // nums1 尾部元素 > nums2 尾部元素
                cur = nums1[p1--];      // 获取 nums1 的待排元素
            } else {
                cur = nums2[p2--];      // 获取 nums2 的待排元素
            }
            nums1[tail--] = cur;        // 待排元素插入到存储尾部，并将存储为指针递减1
        }
    }
```

#Leetcode/Linked List#