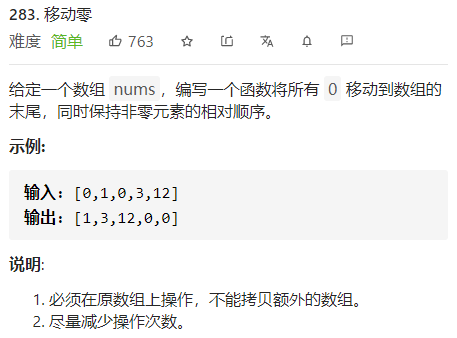
# 283. Move Zeroes 移动零



将数组中所有的 0 挪到数组的末尾，同时要保证其他非 0 元素的相对位置。

举例：

给定数组 nums = [0, 1, 0, 3, 12]

函数运行后结果为 [1, 3, 12, 0, 0]

数组中本来有的两个 0 都被挪到了数组末尾。

与此同时，其他的非 0 元素保持它们原有的位置不动。

解法一：使用辅助空间

扫描一遍整个数组，把所有的非 0 元素全都拿出来，把它们放到原先的数组中，之后将没填补的位置全都赋成 0 。

首先，声明新的vector容器，把它叫做 nonZeroElements，用于存储所有的非 0 元素。

vector<int> nonZeroElements;

之后扫描一遍整个数组，把nums数组中所有的非 0 元素全都 push\_back进我们新声明的 nonZeroElements 这个数组中

        for (int i = 0; i < nums.size(); i++)

            if (nums[i]) // nums[i] != 0

                nonZeroElements.push\_back(nums[i]);

接下来遍历一遍整个 nonZeroElements 容器，将其中的所有元素全都放回给 nums 中。nums[i] 从 0 开始便依次放进 nonZeroElements[i]。

        for (int i = 0; i < nonZeroElements.size(); i++)

            nums[i] = nonZeroElements[i];

起始是在 nonZeroElements.size() 的这个索引的 位置，而终止是在 i < nums.size()，

        for (int i = nonZeroElements.size(); i < nums.size(); i++)

            nums[i] = 0;

**测试用例**

比如说我想测试这个数组

    int arr[] = {0, 1, 0, 3, 12};

由于我们写的函数要传入一个 vector，所以我们可以创建一个 vector，管它叫 vec。

创建的方法是：传入这个数组的头指针arr，尾指针 arr + sizeof(arr)/sizeof(int)) 这个数组arr的size 再除以这个数组中每一个元素对应的 size。

|  |
| --- |
| arr 是个数组  sizeof (arr)就是这个数组所占的内存总量（字节数),  sizeof(int)就是单个元素所占的内存.  所以呢，sizeof (arr) / sizeof (int) 就是arr数组的元素的个数. |

    vector<int> vec(arr, arr + sizeof(arr)/sizeof(int));

下面调用写的这个逻辑，实例化一个Solution，直接用 .moveZeroes() 来调用这个方法，传入我们刚刚创建的 vec。

为了可视化看到函数运行的结果，调用完vec，我们再用循环遍历 vec 的所有元素，将 vec 的所有元素打印出来。

int main() {

    int arr[] = {0, 1, 0, 3, 12};

    vector<int> vec(arr, arr + sizeof(arr)/sizeof(int));

    Solution().moveZeroes(vec);

    for (int i = 0; i < vec.size(); i++)

        cout << vec[i] << " ";

    cout << endl;

    return 0;

}

解法二：不使用辅助空间

之前解法一的空间复杂度是 O(n)，因为我们新创建了一个数组 nonZeroElements来存放所有的非0元素。我们现在的算法不是原地的算法，我们利用了额外的存储空间/辅助空间。

如何优化呢？如何在原地完成对非零元素的移动呢？

我们可以在遍历数组的过程中，一旦遇到非0的元素，就把这个元素直接放到前面的位置。

由于非0元素的个数一定小于等于整个数组中元素的个数，所以我们不用担心这个赋值过程覆盖掉有用的信息。

我们可以设置另外一个索引，只需要在 [0…k ) 前闭后开，在这个区间中保存所有当前遍历过的非0 元素。

初始化时 k为 0，所以前闭后开的 [0,0) 这个位置其实是没有元素的。

动画演示这个算法是怎么走的：

* 一开始我们遍历第一个元素是 0，是0元素，我们不用管它；
* 第二个元素是1，是非0元素，我们把它放到k这个索引指向的位置；
* 之后为了准备下一个非0的元素，相应的k 这个指针向前加1；
* 继续下一个元素是0，我们不用管它；
* 再一个元素是3，由于3是非0元素，就把它移到k所在的索引位置，同时为了准备下一个非0元素，k再加1；
* 之后继续遍历这个数组12，是一个非0元素，我们把12放到当前k所指向的位置，之后k又加1；
* 最后我么遍历下一个元素，发现已经到了数组的结尾，这个数组遍历结束

此时，从0到k，前闭后开的区间，就存放了整个数组中所有的非0元素；

接下来我们要做的事情就是从当前的k开始，将剩余的所有元素都赋值为0。

首先设置int型 k，初始为0，nums中，[0...k)的元素均为非0元素；

遍历到第i个元素后，保证[0...i]中所有非0元素都按照顺序排列在[0...k)中

对nums数组进行遍历:

一旦发现 nums[i] 是非0元素的话，首先我们要做的是把当前的这个元素nums[i]放在 num[k]的位置，之后k要++。

        for (int i = 0; i < nums.size(); i++)

            if (nums[i])

                nums[k++] = nums[i];

也可以这样写：

        for (auto x : nums)

            if (x)

                nums[k++] = x;

最后，我们要将 nums 中的剩余位置都设置为0

        // 将nums剩余的位置放置为0

        for (int i = k; i < nums.size(); i++)

            nums[i] = 0;

上面的写法太啰嗦，可以直接写成：

        while (k < nums.size())

            nums[k++] = 0;

完成了算法，时间复杂度依然是 O(n)，但是空间复杂度变为 O(1)。我们没有使用任何的辅助空间，原地完成了moveZeroes的操作。

解法三：swap()

解法2中我们可以在遍历数组的过程中，一旦遇到非0的元素，就把这个元素直接放到前面的位置。

有没有可能将非0元素放在前面的同时，也把0元素放在了后面呢？

我们可以将非0元素和0元素进行交换位置来达到这样的效果。

遍历这个数组，

* 第一个元素是0元素，我们遍历下一个元素；
* 第二个元素是非0元素，此时将这个元素与k这个索引所指向的元素交换位置，同时k++；
* 下一个元素是0元素，不管它；
* 再下一个元素是3，将3这个非0元素与k所指的位置交换元素，同时k++；
* 遍历下一个元素12，又是一个非0元素，将这个元素与k所指的当前这个元素交换位置，同时k++；
* 再往下遍历已经没有元素继续能遍历了，整个数组遍历完成；

此时，非0元素都在数组的前面，而从k所指的索引开始向后就全都是0元素。这样我们就不需要再从k开始向后为每一个元素赋值了。

代码：

还是需要索引k，它的意义也没有变

之后除了保证遍历到第i个元素后，保证[0…i)中所有非0元素都按照顺序排列在[0…k)中，

同时还要保证[k…in )这个区间都为0元素；

之前只是简单地赋值，现在只是将 nums[k] 和 nums[i] 元素进行位置的交换，交换完这个位置后 k++。之后不需要为k以后的元素赋0值。

        // 遍历到第i个元素后，保证[0...i]中所有非0元素

        // 都按照顺序排列在[0...k)中

        // 同时，[k...i]为0

        for (int i = 0; i < nums.size(); i++)

            if (nums[i])

                swap(nums[k++], nums[i]);

然而，在特殊情况下，如果整个数组都是非0元素，就会出现每一个非0元素都自己和自己交换位置。

改进：加一个特殊的判断，if (i != k)如果索引i和k不等的时候，才真正地执行这个操作；

否则这个元素不需要自己和自己进行交换，也就是在i == k的时候，k直接++就行了。

        // 遍历到第i个元素后，保证[0...i]中所有非0元素

        // 都按照顺序排列在[0...k)中

        // 同时，[k...i]为0

        for (int i = 0; i < nums.size(); i++)

            if (nums[i])

                if (i != k)

                    swap(nums[k++], nums[i]);

                else // i == k

                    k++;

Time: O(n)

Space: O(1)