Previous Next

27. Remove Element

March 13, 2016 | 229.8K views

Average Rating: 4.55 (142 votes)

Given an array *nums* and a value *val*, remove all instances of that value **in-place** and return the new length.

Do not allocate extra space for another array, you must do this by **modifying the input array in-place** with O(1) extra memory.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

Example 1:

```
Given nums = [3,2,2,3], val = 3,

Your function should return length = 2, with the first two elements of nums being 2.

It doesn't matter what you leave beyond the returned length.
```

Example 2:

```
Given nums = [0,1,2,2,3,0,4,2], val = 2,

Your function should return length = 5, with the first five elements of nums
 containing 0, 1, 3, 0, and 4.

Note that the order of those five elements can be arbitrary.

It doesn't matter what values are set beyond the returned length.
```

Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by **reference**, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without making a copy)
int len = removeElement(nums, val);

// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

Summary

This is a pretty easy problem, but one may get confused by the term "in-place" and think it is impossible to remove an element from the array without making a copy of the array.

Hints

1. Try two pointers.
2. Did you use the fact that the order of elements can be changed?
3. What happens when the elements to remove are rare?

Solution

Approach 1: Two Pointers

Intuition

Since this question is asking us to remove all elements of the given value in-place, we have to handle it with $O(1)$ extra space. How to solve it? We can keep two pointers i and j, where i is the slow-runner while j is the fast-runner.

Algorithm

When $nums[j]$ equals to the given value, skip this element by incrementing j. As long as $nums[j] \neq val$, we copy $nums[j]$ to $nums[i]$ and increment both indexes at the same time. Repeat the process until j reaches the end of the array and the new length is i.

This solution is very similar to the solution to Remove Duplicates from Sorted Array.

Java Copy

```java
public int removeElement(int[] nums, int val) {
    int i = 0;
    for (int j = 0; j < nums.length; j++) {
        if (nums[j] != val) {
            nums[i] = nums[j];
            i++;
        }
    }
    return i;
}
```

Complexity analysis

- Time complexity : $O(n)$. Assume the array has a total of n elements, both i and j traverse at most $2n$ steps.
- Space complexity : $O(1)$.

Approach 2: Two Pointers - when elements to remove are rare

Intuition

Now consider cases where the array contains few elements to remove. For example, $nums = [1, 2, 3, 5, 4], val = 4$. The previous algorithm will do unnecessary copy operation of the first four elements. Another example is $nums = [4, 1, 2, 3, 5], val = 4$. It seems unnecessary to move elements $[1, 2, 3, 5]$ one step left as the problem description mentions that the order of elements could be changed.

Algorithm

When we encounter $nums[i] = val$, we can swap the current element out with the last element and dispose the last one. This essentially reduces the array's size by 1.

Note that the last element that was swapped in could be the value you want to remove itself. But don't worry, in the next iteration we will still check this element.

Java Copy

```java
public int removeElement(int[] nums, int val) {
    int i = 0;
    int n = nums.length;
    while (i < n) {
        if (nums[i] == val) {
            nums[i] = nums[n - 1];
            // reduce array size by one
            n--;
        } else {
            i++;
        }
    }
    return n;
}
```

Complexity analysis

- Time complexity : $O(n)$. Both i and n traverse at most n steps. In this approach, the number of assignment operations is equal to the number of elements to remove. So it is more efficient if elements to remove are rare.
- Space complexity : $O(1)$.

Rate this article:

Previous Next

Comments: 249 Sort By

Type comment here... (Markdown is supported)

Preview Post

zhengzhicong 294 November 12, 2018 5:04 PM

python3:

```
class Solution:
    def removeElement(self, nums, val):
```

Read More

38 Share Reply

SHOW 11 REPLIES

ray-kim-12 32 December 11, 2018 5:22 AM

Simple Java Solution:
Runtime: 4ms, faster than 98.69%

```
class Solution {
```

Read More

29 Share Reply

Dreyri 42 March 3, 2019 3:00 AM Report

C++, 4ms, faster than 100%

```
int removeElement(vector<int>& nums, int val) {
    return std::distance(nums.begin(), std::remove(nums.begin(), nums.end(), va
```

Read More

42 Share Reply

SHOW 3 REPLIES

terrible_whiteboard 626 May 19, 2020 6:08 PM

I made a video if anyone is having trouble understanding the solution (clickable link)
https://youtu.be/hCVTvyO6WUk

Read More

19 Share Reply

fibrahimb 28 March 30, 2019 4:13 AM Report

This problem is poorly worded. int arrays are IMMUTABLE.

17 Share Reply

SHOW 5 REPLIES

dreambo8563 15 July 24, 2018 10:36 AM

I can get the correct result of [2,2] locally. but get [3,2] in the live editor.... . anythings wrong??

```
func removeElement(nums []int, val int) int {
        count := 0
```

Read More

15 Share Reply

SHOW 4 REPLIES

Abdulhamed 12 September 22, 2018 2:54 PM Report

Ideas how to make it better ?

```
    public int removeElement(int[] nums, int val) {
        int other =0;
```

Read More

9 Share Reply

a89839909 19 July 5, 2018 1:56 PM

I have a question. Does the solution really remove the value from nums? I think it just throws those values to the end and returns n, but internally the list still contains the values. For Python, we can use pop() or remove() to implement an in-place algorithm.

Read More

5 Share Reply

SHOW 1 REPLY

prmichaelsen 17 January 27, 2019 5:08 AM

I'm proud I ended up with the same exact solution as approach 2. Yay :)

7 Share Reply

comeCU 58 August 24, 2018 3:29 PM Report

the order of those elements can be arbitrary

6 Share Reply

SHOW 1 REPLY

‹ 1 2 3 4 5 6 ... 24 25 ›