448 Find All Numbers Disappeared in an Array

2018年4月11日 17:35

Given an array of integers where 1 ≤ a[i] ≤ *n* (*n* = size of array), some elements appear twice and others appear once.
Find all the elements of [1, *n*] inclusive that do not appear in this array.
Could you do it without extra space and in O(*n*) runtime? You may assume the returned list does not count as extra space.
Example:
Input:
[4,3,2,7,8,2,3,1]
Output:
[5,6]

来自 <https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/description/>

给定一个范围在 1 ≤ a[i] ≤ *n* (*n* = 数组大小) 的 整型数组，数组中的元素一些出现了两次，另一些只出现一次。
找到所有在 [1, *n*] 范围之间没有出现在数组中的数字。
您能在不使用额外空间且时间复杂度为*O(n)*的情况下完成这个任务吗? 你可以假定返回的数组不算在额外空间内。
示例:
输入:
[4,3,2,7,8,2,3,1]
输出:
[5,6]

Solution for Python3:

```python
class Solution:
    def findDisappearedNumbers(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        for i in range(len(nums)):
           m = abs(nums[i]) - 1;
           nums[m] = -nums[m] if nums[m] > 0 else nums[m]
        return [i + 1 for i in range(len(nums)) if nums[i] > 0]
```

Solution for C++:

```cpp
class Solution {
public:
    vector<int> findDisappearedNumbers(vector<int>& nums) {
        int n = nums.size(), m;
        for (int i = 0; i < n; i++) {
            m = abs(nums[i]) - 1; //abs是为了防止相同元素之前已经取负
```

```
                nums[m] = nums[m] > 0 ? -nums[m] : nums[m]
            }
            vector<int> res;
            for (int i = 0; i < n; i++) {
                if (nums[i] > 0)
                    res.push_back(i + 1);
            }
            return res;
        }
    };
```