268 Missing Number

2018年4月8日 20:00

Question:

Given an array containing *n* distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.
Example 1
Input: [3,0,1]
Output: 2
Example 2
Input: [9,6,4,2,3,5,7,0,1]
Output: 8

Note:
Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

来自 <https://leetcode.com/problems/missing-number/description/>

给出一个包含 0, 1, 2, ..., n 中 *n* 个数的序列，
找出 0 .. *n* 中没有出现在序列中的那个数。

Solution for Python3:

```python
class Solution1:
    def missingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        for i in range(1, len(nums)):
           nums[0] += nums[i] - i
        return -(nums[0] - len(nums))


class Solution2:
    def missingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        missing = len(nums)
        for i, num in enumerate(nums):
           missing ^= i ^ num
        return missing

class Solution3:
    def missingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        expected_sum = len(nums) * (len(nums) + 1) // 2
        actual_sum = sum(nums)
```

```
        return expected_sum - actual_sum
```

Solution for C++:

```cpp
class Solution1 {
public:
    int missingNumber(vector<int>& nums) {
        for (int i = 1; i < nums.size(); i++) {
            nums[0] += nums[i] - i;
        }
        return -(nums[0] - nums.size());
    }
};

class Solution2 {
public:
    int missingNumber(vector<int>& nums) {
        int missing = nums.size();
        for (int i = 0; i < nums.size(); i++) {
            missing ^= i ^ nums[i];
        }
        return missing;
    }
};

class Solution3 {
public:
    int missingNumber(vector<int>& nums) {
        int expectedSum = nums.size() * (nums.size() + 1) / 2;
        int actualSum = 0;
        for (int num : nums)
            actualSum += num;
        return expectedSum - actualSum;
    }
};
```

Appendix:

异或算法分析:

1) 列：index: [0,1,2,3] -> [0,1,2,3,4] 完整index
2) nums:[0,1,3,4] -> [0,1,2,3,4] 完整nums
3) 完整index^完整nums = (0^0)^(1^1)^(2^2)^(3^3)^(4^4)=0。
4) 完整index^不完整nums=(0^0)^(1^1)^(2^3)^(3^4)^4（4是少的index)
5) =0^0^1^1^2^3^3^4^4=2
6) 异或运算满足交换律