

260. Single Number III

Aug. 25, 2019 | 9.3K views

Previous Next

★★★★★
Average Rating: 4.53 (19 votes)

Given an array of numbers `nums`, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once.

Example:

Input: [1,2,1,3,2,5]
Output: [3,5]

Note:

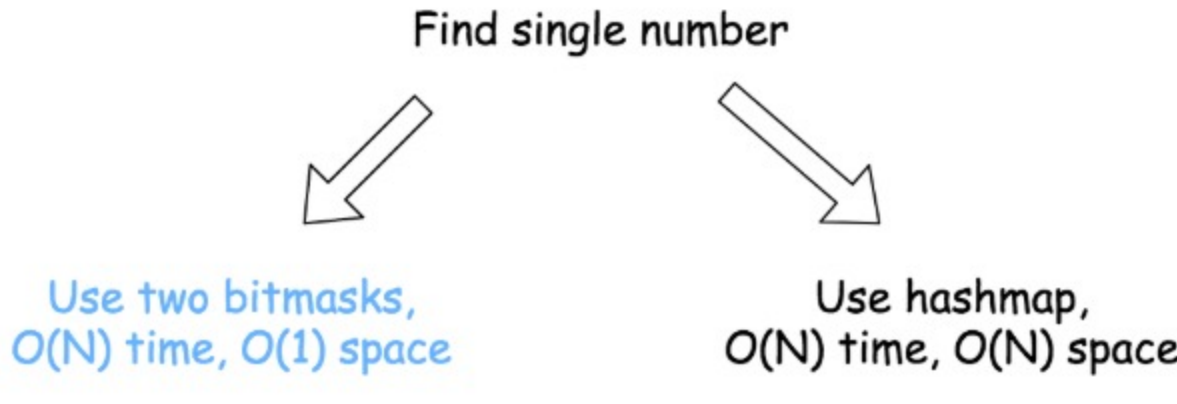
1. The order of the result is not important. So in the above example, [5, 3] is also correct.
2. Your algorithm should run in linear runtime complexity. Could you implement it using only constant space complexity?

Solution

Overview

The problem could be solved in $\mathcal{O}(N)$ time and $\mathcal{O}(N)$ space by using hashmap.

To solve the problem in a constant space is a bit tricky but could be done with the help of two bitmasks.



Approach 1: Hashmap

Build a hashmap : element -> its frequency. Return only the elements with the frequency equal to 1.

Implementation

```
Java Python Copy
1 from collections import Counter
2 class Solution:
3     def singleNumber(self, nums: int) -> List[int]:
4         hashmap = Counter(nums)
5         return [x for x in hashmap if hashmap[x] == 1]
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ to iterate over the input array.
- Space complexity : $\mathcal{O}(N)$ to keep the hashmap of N elements.

Approach 2: Two bitmasks

Prerequisites

This article will use two bitwise tricks, discussed in details last week :

- If one builds an array bitmask with the help of XOR operator, following `bitmask ^= x` strategy, the bitmask would keep only the bits which appear odd number of times. That was discussed in details in the article [Single Number II](#).

bitmask

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

x = 2

0	0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---

bitmask^x,
the first appearance of x

0	0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---

bitmask^x^x,
the second appearance of x

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

- `x & (-x)` is a way to isolate the rightmost 1-bit, i.e. to keep the rightmost 1-bit and to set all the others bits to zero. Please refer to the article [Power of Two](#) for the detailed explanation.

x & (-x)
keeps the rightmost 1-bit
and sets all the other bits to 0

x = 7

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

-x = ~x + 1

1	1	1	1	1	1	0	0	1
---	---	---	---	---	---	---	---	---

x & (-x)

0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---

x = 6

0	0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---

-x = ~x + 1

1	1	1	1	1	0	1	1	0
---	---	---	---	---	---	---	---	---

x & (-x)

0	0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---

Intuition

An interview tip. Imagine, you have a problem to identify an array element (or elements), which appears exactly given number of times. Probably, the key is to build first an array bitmask using XOR operator. Examples: [In-Place Swap](#), [Single Number](#), [Single Number II](#).

So let's create an array bitmask : `bitmask ^= x`. This bitmask will not keep any number which appears twice because XOR of two equal bits results in a zero bit `a^a = 0`.

Instead, the bitmask would keep only the difference between two numbers (let's call them x and y) which appear just once. The difference here it's the bits which are different for x and y.

bitmask^x^y^a^a
= difference
between x and y

bitmask

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

x = 1

0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---

y = 2

0	0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---

a = 3

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

a = 3

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

bitmask^x^y^a^a

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

Could we extract x and y directly from this bitmask? No. Though we could use this bitmask as a marker to separate x and y.

Let's do `bitmask & (-bitmask)` to isolate the rightmost 1-bit, which is different between x and y. Let's say this is 1-bit for x, and 0-bit for y.

isolate rightmost 1-bit
which is different for
x and y

bitmask

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

x = 1

0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---

y = 2

0	0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---

a = 3

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

a = 3

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

bitmask =
bitmask^x^y^a^a

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

diff =
bitmask & (-bitmask)

0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---

Now let's use XOR as before, but for the new bitmask `x_bitmask`, which will contain only the numbers which have 1-bit in the position of `bitmask & (-bitmask)`. This way, this new bitmask will contain only number x `x_bitmask = x`, because of two reasons:

- y has 0-bit in the position `bitmask & (-bitmask)` and hence will not enter this new bitmask.
- All numbers but x will not be visible in this new bitmask because they appear two times.

x_bitmask

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

x = 1

0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---

y = 2

0	0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---

a = 3

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

a = 3

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

x_bitmask =
x_bitmask^x^a^a

0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---

diff =
bitmask & (-bitmask)

0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---

Voila, x is identified. Now to identify y is simple: `y = bitmask^x`.

Implementation

```
Java Python Copy
1 class Solution:
2     def singleNumber(self, nums: int) -> List[int]:
3         # difference between two numbers (x and y) which were seen only once
4         bitmask = 0
5         for num in nums:
6             bitmask ^= num
7
8         # rightmost 1-bit diff between x and y
9         diff = bitmask & (-bitmask)
10
11         x = 0
12         for num in nums:
13             # bitmask which will contain only x
14             if num & diff:
15                 x ^= num
16
17         return [x, bitmask^x]
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ to iterate over the input array.
- Space complexity : $\mathcal{O}(1)$, it's a constant space solution.

Rate this article: ★★★★★

Previous

Next

Comments: 5

Sort By



Type comment here... (Markdown is supported)

Preview

Post



pixelbookcoder ★ 5 August 29, 2019 7:28 AM

```
nums = [1,2,1,3,2,5]
seen = []
for num in nums:
    if num in seen:
```

Read More

5 | Share | Reply

SHOW 2 REPLIES



Fnaf ★ 8 August 30, 2019 10:17 AM

@andvay can you do a series on segment tree?

2 | Share | Reply

SHOW 2 REPLIES



lenchen1112 ★ 1005 December 5, 2019 4:30 PM

clean version of bitwise operation in Python3:

```
from functools import reduce, partial
from operator import xor, and_
```

Read More

1 | Share | Reply

SHOW 1 REPLY



meowlicious99 ★ 512 April 29, 2020 9:34 PM

second algorithm is only technically $\mathcal{O}(1)$ the size of bitmask increases linearly with each addition. Sure it fits in this case in one int. But that would be equivalent of preallocating a huge Set in solution 1 and calling it $\mathcal{O}(1)$.

if something increases linearly then its not $\mathcal{O}(1)$.

Read More

0 | Share | Reply



Zizhen-Huang ★ 92 December 30, 2019 8:12 AM

// difference between two numbers (x and y) which were seen only once ?

How come line 5 will produce the difference? Sometimes it could produce the sum or multiplication

0 | Share | Reply

SHOW 1 REPLY