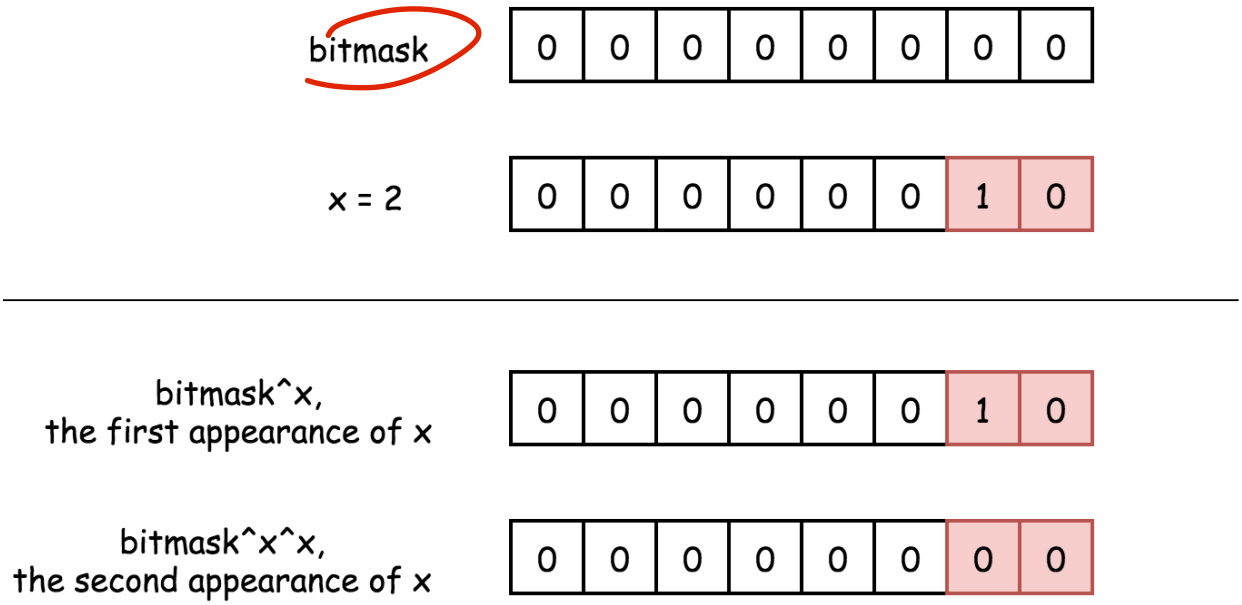
Approach 2: Two bitmasks

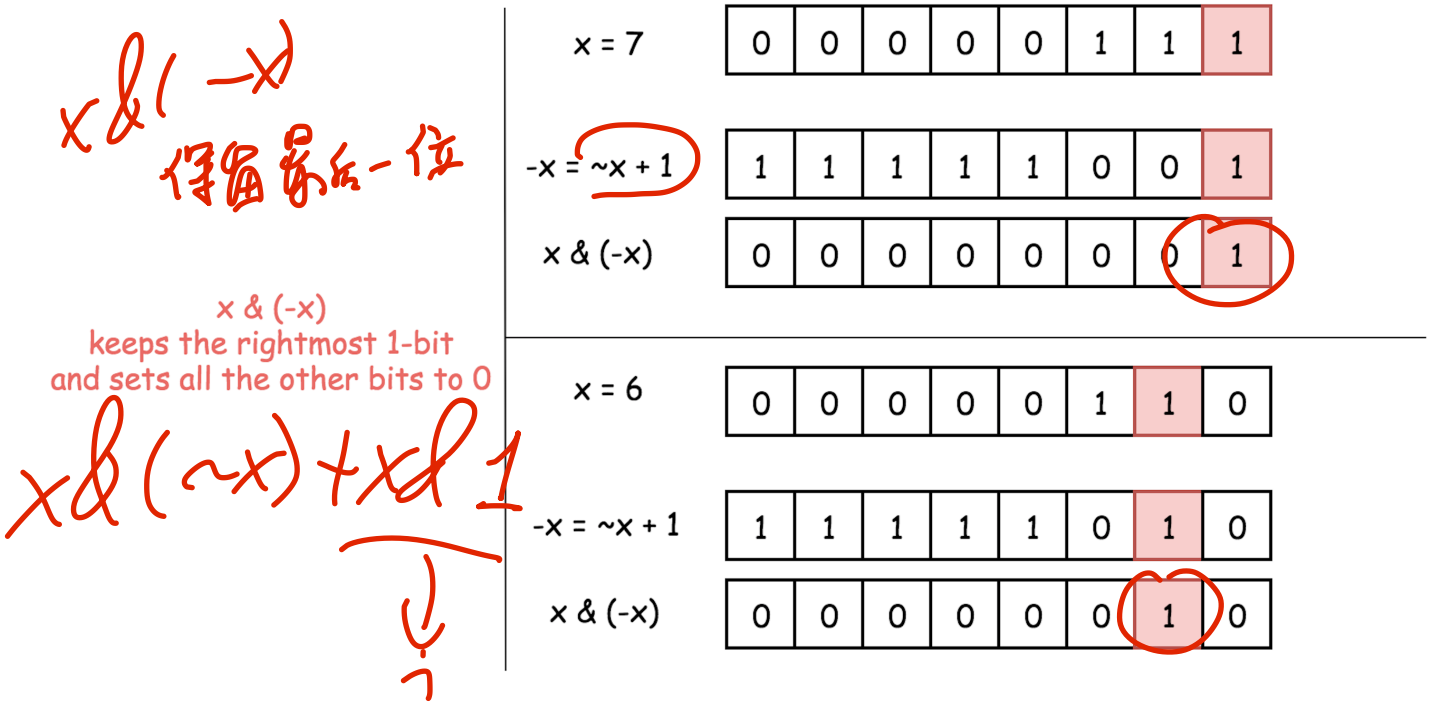
Prerequisites

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 (https://leetcode.com/articles/single-number-ii/).



- `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 (https://leetcode.com/articles/power-of-two/) for the detailed explanation.

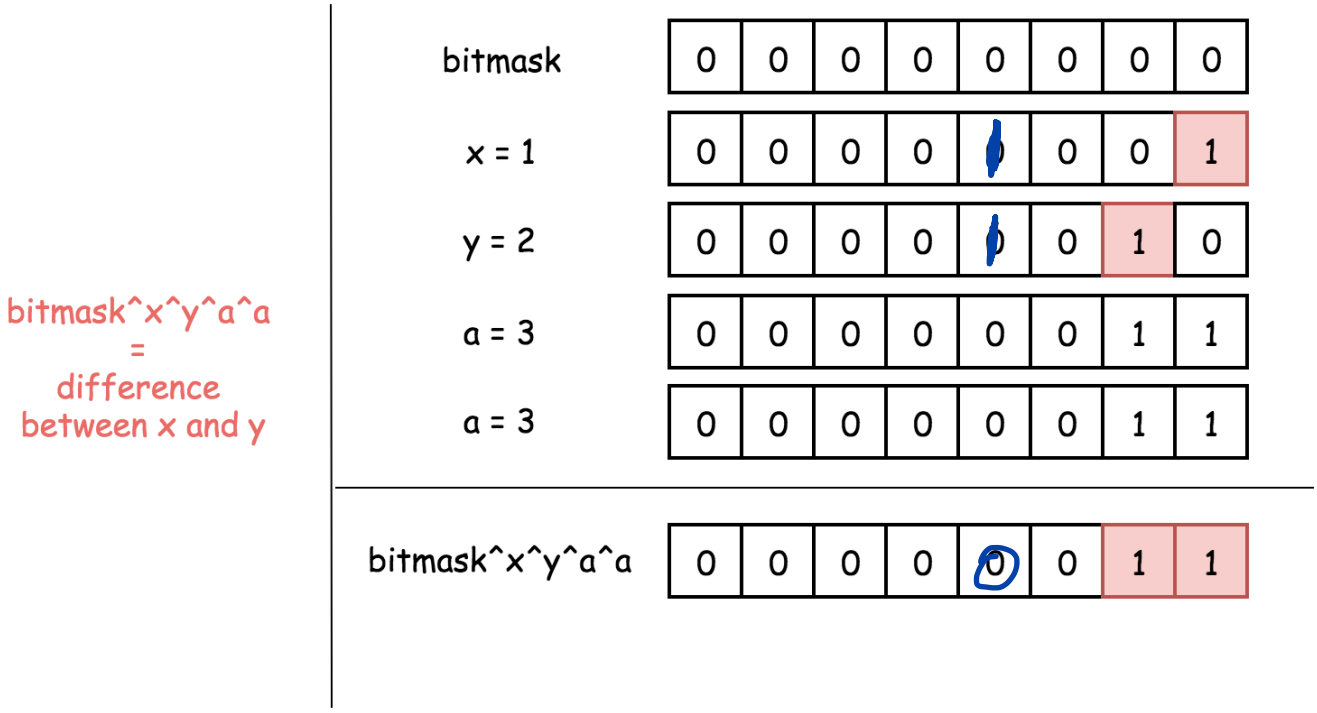


Intuition

> An interview tip. Imagine, you have a problem to indentify 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 (leetcode.com/articles/single-number-ii/356460/Single-Number-II/324042), Single Number (https://leetcode.com/articles/single-number/), Single Number II (leetcode.com/articles/single-number-ii/356460/Single-Number-II/324042).

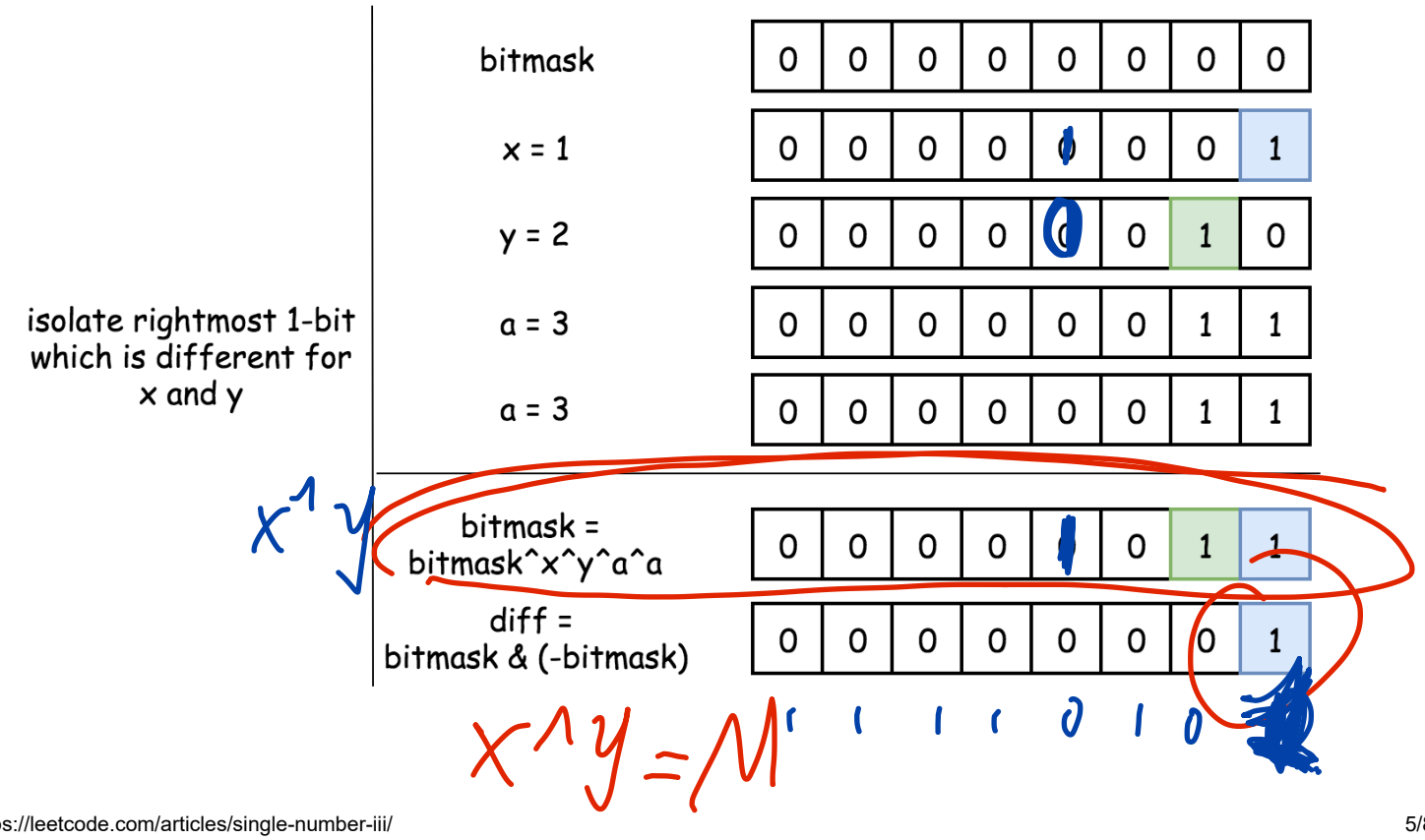
So let's create an array bitmask : `bitmask ^= x` . This bitmask will *not* keep any number which appears twice because XOR of two equal bits results in a zero bit `a^a = 0` .

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.



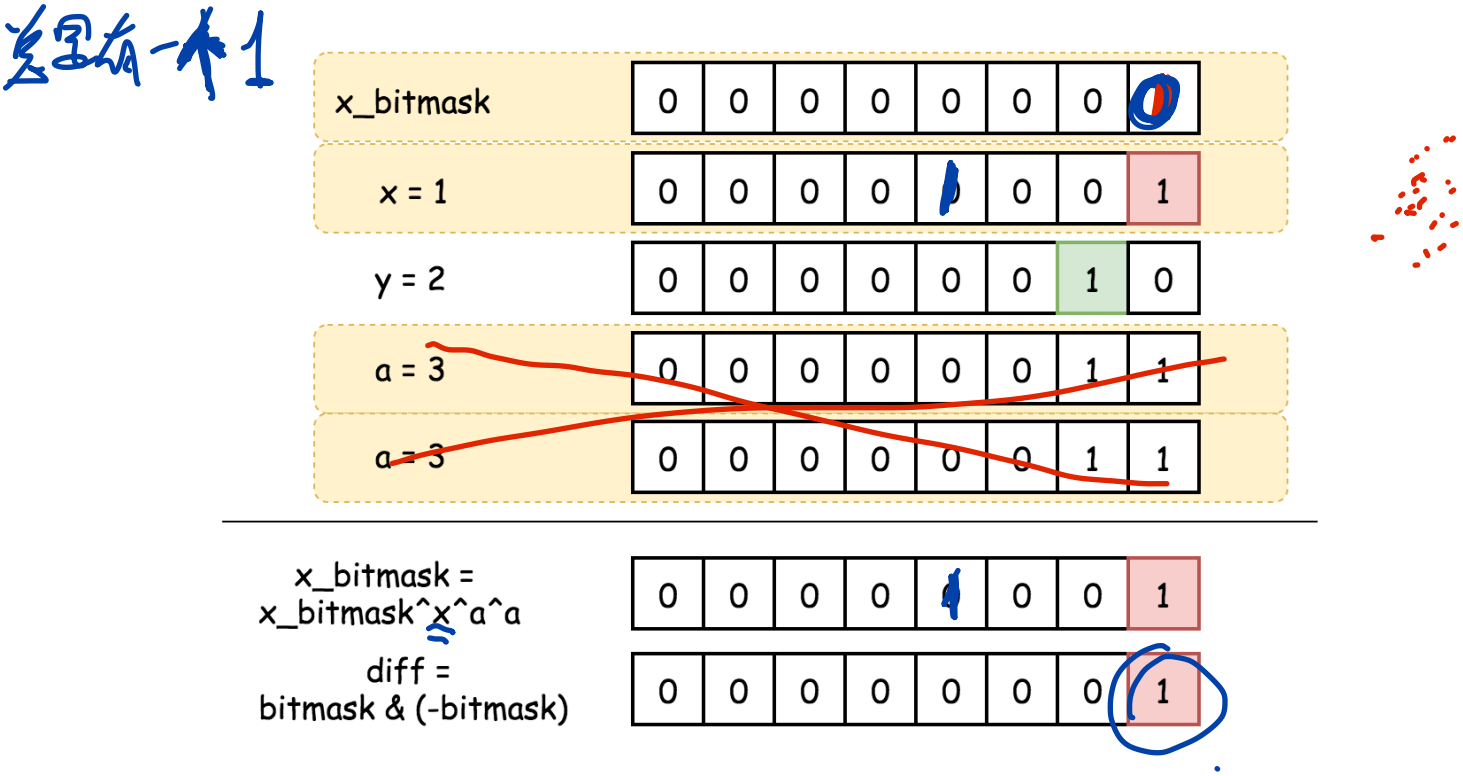
> Could we extract x and y directly from this bitmask? No. Though we could use this bitmask as a marker to separate x and y.

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.



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.



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

Implementation

Java | Python

Copy

```java
class Solution {
  public int[] singleNumber(int[] nums) {
    // difference between two numbers (x and y) which were seen only once
    int bitmask = 0;
    for (int num : nums) bitmask ^= num;

    // rightmost 1-bit diff between x and y
    int diff = bitmask & (-bitmask);

    int x = 0;
    // bitmask which will contain only x
    for (int num : nums) if ((num & diff) != 0) x ^= num;

    return new int[]{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.

Analysis written by @liaison (https://leetcode.com/liaison/) and @andvary (https://leetcode.com/andvary/)

Rate this article:

Previous (/articles/maximum-product-of-word-lengths/)

Next (/articles/maximum-level-sum-of-a-binary-tree/)

Comments: 4

Sort By

Type comment here... (Markdown is supported)

Preview | Post

pixelbookcoder (/pixelbookcoder) ★ 4 August 28, 2019 8:58 PM