[C++]| Explained Everything w/ WHY XOR WORKS| BRUTE FORCE TO OPTIMIZED| STEP BY STEP DRY RUN

[aryanttripathi](https://leetcode.com/aryanttripathi/)

5684

23316

Feb 15, 2022

C

***Brief note about Question-***

* We have to return the number which is *appears single time* in the array.

Let's take an example not given in question -

Suppose our array is arr[]: [5, 1, 3, 1, 3, 4, 5, 7, 4]

So, our answer should be '7' as it appears only single time in the array.

***Solution - I (using unorderd map, Accepted)-***

* Since we are very obedient person and don't want to do anything extra from our side.
* So, on seeing the question, the first idea that clicks to our mind is to ***store frequency*** of all over the elements.
* After storing frequency, we will simply ***find the element whose frequency is 1***.
* The element whose frequency is 1, that is our answer.

Time Complexity --> O(n) // where n is the size of the array

Space Complexity --> O(n) // we will store the frequency of every element

It paases [ 61 / 61 ] in built test cases

***So, How we will implement that?***

* Whenever we heard the name *frequency*, we will call our one and only superhero i.e unordered\_map.
* We will store frequency of each element in our unorderd map.
* After that, we **traverse from the map**, and for every element we check whether it's **frequency is 1** or not.
* If it's frequency is 1, then we will simply store the answer and break the loop.
* See, an example.

Suppose our array is arr[]: [5, 1, 3, 1, 3, 4, 5, 7, 4]

so, after storing frequency of each element in our map, our map looks like -

5 -> 2

1 -> 2

3 -> 2

4 -> 2

7 -> 1 On traversing, we will see that it's frequency is 1, so this should be our answer.

**Code (C++)**

class Solution {

public:

int singleNumber(vector<int>& arr) {

int n = arr.size(); // taking the size of the array

unordered\_map<int, int> mp; // unordered map to store the frequency

// storing frequency in the map

for(int i = 0; i < n; i++)

{

mp[arr[i]]++;

}

int ans; // variable to store our answer

for(auto x: mp) // traverse from the map

{

if(x.second == 1) //if frequency of any elemennt is 1

{

ans = x.first; // store in our answer

break; // break the loop, as we got our answer now

}

}

return ans; // return ans

}

};

***Solution - II (using property of xor, Accepted)-***

* Now, the question arises, is their any way so that we do not need to store frequency, i.e  
  **can we space optimized our answer**?
* Since, we want not to use map, so we will see our map that where is the problem?
* We observe a very basic thing that other than our answer, **frequency of each element is 2**, that means our question wants to say that, ***every element appears twice except the single one and we have to find that particular single element.***
* So, if every element appears twice, *do we really need to store the frequency of each element*?
* Answer is **No**, we do not need to store the frequency of each element because we already know that, for each element their is only two ways, **either it's frequency is 1 or it's frequency is 2**. Their is not any third option other than that.
* In this way, we can think of ***the property of XOR-***

0 ^ 0 --> 0

0 ^ 1 --> 1

1 ^ 0 --> 1

1 ^ 1 --> 0

* If we see, we will find out that ***xor of two same number is zero***.
* Can we use this property here?
* Answer is ***Yes***. But how?
* We will run a loop from all over the array, and simply start taking xor of two.
* We do xor between prev answer of xor and current element.
* See dry run for more clarification.
* But, Question is **WHY**? why this works?

We know every number is appears twice except a single number which appears only single time.

See, we already discuss a thing a that xor of a same number with itself is zero, i.e A ^ A = 0

Now, we will look some more property of xor-

1) xor of a same number with itself is zero, i.e A ^ A = 0

2) xor is commutative that means a ^ b = b ^ a.

3) xor of any number with zero is the number itself i.e A ^ 0 = A.

Suppose our array is arr[]: [5, 1, 3, 1, 3, 4, 5, 7, 4]

we will rearrange the array, and take all the numbers together, then our array looks like

arr[]: [1, 1, 3, 3, 4, 4, 5, 5, 7]

now, take xor of all numbers -

1 ^ 1 ^ 3 ^ 3 ^ 4 ^ 4 ^ 5 ^ 5 ^ 7 (rearrange the array)

0 ^ 0 ^ 0 ^ 0 ^ 7 (see point number 1)

7 (see point number 3)

* Now, The question again arises, **do we really need to rearrange the array ?**
* Answer is ***NO***.
* We already know a property of xor, that says we don't need to rearrange the array. But wait, where?
* xor is commutative that means a ^ b = b ^ a. (see above point number 2)
* Okay, if this is ? **Can you do this without rearaanging the array?**
* Answer is **Yes**, see below-

So, our array is arr[]: [5, 1, 3, 1, 3, 4, 5, 7, 4]

[5, 1, 3, 1, 3, 4, 5, 7, 4]

↑ ↑

5 ^ 1 = 4

[5, 1, 3, 1, 3, 4, 5, 7, 4]

↑

4 ^ 3 = 7 (prev answer xor current index)

[5, 1, 3, 1, 3, 4, 5, 7, 4]

↑

7 ^ 1 = 6 (prev answer xor current index)

[5, 1, 3, 1, 3, 4, 5, 7, 4]

↑

6 ^ 3 = 5 (prev answer xor current index)

[5, 1, 3, 1, 3, 4, 5, 7, 4]

↑

5 ^ 4 = 1 (prev answer xor current index)

[5, 1, 3, 1, 3, 4, 5, 7, 4]

↑

1 ^ 5 = 4 (prev answer xor current index)

[5, 1, 3, 1, 3, 4, 5, 7, 4]

↑

4 ^ 7 = 3 (prev answer xor current index)

[5, 1, 3, 1, 3, 4, 5, 7, 4]

↑

3 ^ 4 = 7 (prev answer xor current index)

So, the element which we got as left should be our answer, so the answer is '7'

Time Complexity --> O(n) // where n is the size of the array

Space Complexity --> O(1) // we are not using anything extra from our side

It paases [ 61 / 61 ] in built test cases

**Code (C++)**

class Solution {

public:

int singleNumber(vector<int>& arr) {

int n = arr.size(); // extracting the size of the array

// traverse from the array

for(int i = 0; i < n - 1; i++)

{

arr[i + 1] = arr[i] ^ arr[i + 1]; // (prev answer xor current index)

}

return arr[n- 1]; // return left over element

}

};

***If u find this useful , please consider to give a upvote!!***