Find the element that appears once

Given an array where every element occurs three times, except one element which occurs only once. Find the element that occurs once. Expected time complexity is O(n) and O(1) extra space.
Examples:

```
Input: arr[] = {12, 1, 12, 3, 12, 1, 1, 2, 3, 3}
Output: 2
```

We can use sorting to do it in O(nLogn) time. We can also use hashing, but the worst case time complexity of hashing may be more than O(n) and hashing requires extra space.

The idea is to use bitwise operators for a solution that is O(n) time and uses O(1) extra space. The solution is not easy like other XOR based solutions, because all elements appear odd number of times here. The idea is taken from here.

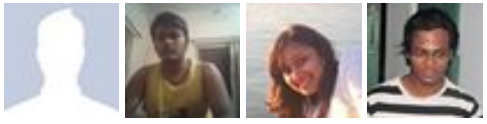
Run a loop for all elements in array. At the end of every iteration, maintain following two values.

ones: The bits that have appeared 1st time or 4th time or 7th time .. etc.

twos: The bits that have appeared 2nd time or 5th time or 8th time .. etc.

Finally, we return the value of 'ones'

How to maintain the values of 'ones' and 'twos'?
'ones' and 'twos' are initialized as 0. For every new element in array, find out the common set bits in the new element and previous value of 'ones'. These common set bits are actually the bits that should be added to 'twos'. So do bitwise OR of the common set bits with 'twos'. 'twos' also gets some extra bits that appear third time. These extra bits are removed later.



Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion
Geometric Algorithms

Update ‘ones’ by doing XOR of new element with previous value of ‘ones’. There may be some bits which appear 3rd time. These extra bits are also removed later.

Both ‘ones’ and ‘twos’ contain those extra bits which appear 3rd time. Remove these extra bits by finding out common set bits in ‘ones’ and ‘twos’.

```
#include <stdio.h>

int getSingle(int arr[], int n)
{
    int ones = 0, twos = 0 ;

    int common_bit_mask;

    // Let us take the example of {3, 3, 2, 3} to understand this
    for( int i=0; i< n; i++ )
    {
        /* The expression "one & arr[i]" gives the bits that are
           there in both 'ones' and new element from arr[].  We
           add these bits to 'twos' using bitwise OR

           Value of 'twos' will be set as 0, 3, 3 and 1 after 1st,
           2nd, 3rd and 4th iterations respectively */
        twos  = twos | (ones & arr[i]);


        /* XOR the new bits with previous 'ones' to get all bits
           appearing odd number of times

           Value of 'ones' will be set as 3, 0, 2 and 3 after 1st,
           2nd, 3rd and 4th iterations respectively */
        ones  = ones ^ arr[i];


        /* The common bits are those bits which appear third time
           So these bits should not be there in both 'ones' and 'twos'
           common_bit_mask contains all these bits as 0, so that the b
           be removed from 'ones' and 'twos'

           Value of 'common_bit_mask' will be set as 00, 00, 01 and 10
           after 1st, 2nd, 3rd and 4th iterations respectively */
        common_bit_mask = ~(ones & twos);


        /* Remove common bits (the bits that appear third time) from '
```


Popular Posts

All permutations of a given string

Memory Layout of C Programs

Understanding “extern” keyword in C

Median of two sorted arrays

Tree traversal without recursion and without stack!

Structure Member Alignment, Padding and Data Packing

Intersection point of two Linked Lists

Lowest Common Ancestor in a BST.

Check if a binary tree is BST or not

Sorted Linked List to Balanced BST

```
            Value of 'ones' will be set as 3, 0, 0 and 2 after 1st,
            2nd, 3rd and 4th iterations respectively */
        ones &= common_bit_mask;


        /* Remove common bits (the bits that appear third time) from '

            Value of 'twos' will be set as 0, 3, 1 and 0 after 1st,
            2nd, 3rd and 4th itearations respectively */
        twos &= common_bit_mask;

        // uncomment this code to see intermediate values
        //printf (" %d %d \n", ones, twos);
    }

    return ones;
}

int main()
{
    int arr[] = {3, 3, 2, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("The element with single occurrence is %d ",
            getSingle(arr, n));
    return 0;
}
```

Output:

```
2
```

Time Complexity: O(n)
Auxiliary Space: O(1)

Following is another O(n) time complexity and O(1) extra space method suggested by *aj*. We can sum the bits in same positions for all the numbers and take modulo with 3. The bits for which sum is not multiple of 3, are the bits of number with single occurrence.
Let us consider the example array {5, 5, 5, 8}. The 101, 101, 101, 1000
Sum of first bits%3 = (1 + 1 + 1 + 0)%3 = 0;
Sum of second bits%3 = (0 + 0 + 0 + 0)%0 = 0;
Sum of third bits%3 = (1 + 1 + 1 + 0)%3 = 0;
Sum of fourth bits%3 = (1)%3 = 1;

Hence number which appears once is 1000

```
#include <stdio.h>
#define INT_SIZE 32

int getSingle(int arr[], int n)
{
    // Initialize result
    int result = 0;

    int x, sum;

    // Iterate through every bit
    for (int i = 0; i < INT_SIZE; i++)
    {
      // Find sum of set bits at ith position in all
      // array elements
      sum = 0;
      x = (1 << i);
      for (int j=0; j< n; j++ )
      {
          if (arr[j] & x)
            sum++;
      }

      // The bits with sum not multiple of 3, are the
      // bits of element with single occurrence.
      if (sum % 3)
        result |= x;
    }

    return result;
}

// Driver program to test above function
int main()
{
    int arr[] = {12, 1, 12, 3, 12, 1, 1, 2, 3, 2, 2, 3, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("The element with single occurrence is %d ",
            getSingle(arr, n));
    return 0;
}
```

```
7
```


Recent Comments

This article is compiled by **Sumit Jain** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Tpoics:

- Check if a number is multiple of 9 using bitwise operators
- How to swap two numbers without using a temporary variable?
- Divide and Conquer | Set 4 (Karatsuba algorithm for fast multiplication)
- Find position of the only set bit
- Swap all odd and even bits
- Add two bit strings
- Write your own strcmp that ignores cases
- Binary representation of a given number

Writing code in comment? Please use **ideone.com** and share the link here.