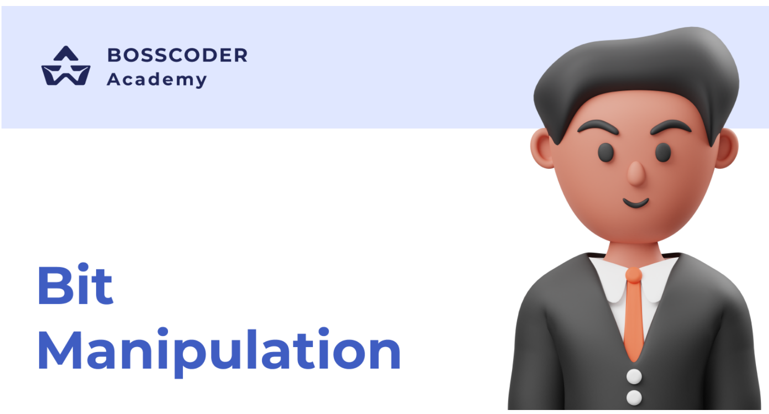
Bit Manipulation

Gyanu Mayank

17th February, 2023



Agenda

1. Bit Manipulation Basics and different type of Bit Operators
2. Problems

Bit Manipulation

Bit manipulation is the process of applying logical operations on a sequence of bits to achieve a required result.

List of Bitwise operators

&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Bitwise left shift
>>	Bitwise right shift

Bitwise AND -A & B

Values for bit combinations

a b a & b

----

0 0 0
0 1 0
1 0 0
1 1 1

Example:
A = 21 (10101) and B = 6 (110)
A & B =1 0 1 0 1 & 0 0 1 1 0= 4.

Bitwise OR-A|B

Values for bit combinations

a b a | b

----

0 0 0

1 1 1

Example:

A = 21 (10101) OR B = 6 (110)

A & B =1 0 1 0 1 | 0 0 1 1 0= 23.

Bitwise XOR-A^B

Values for bit combinations

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Example:

A = 21 (10101) XOR B = 6 (110)

A ^ B =1 0 1 0 1 ^ 0 0 1 1 0= 19.

Right Shift Operators- A>>B

`A >> x` implies shifting the bits of A to the right by x positions. The last x bits are lost this way.

Example

A = 29 (11101) and x = 2,

so A >> 2 means

0 0 1 1 1 **0 1** >> 2

01 is lost and the rest of the digit shift to the right

0 0 0 0 1 1 1 = 7

`A >> x` is equal to division by pow(2, x)

Left Shift Operators- A<<B

`A << x` implies shifting the bits of A to the left by x positions. The first x bits are lost this way.

Example

0 0 1 1 1 0 1 >> 2

00 is lost and the rest of the digit shifts to the left

1 1 1 0 1 0 0=116

`A << x` is equal to multiplication by pow(2, x)

Property

1. A& B & C=A&(B&C)=(A&B)&C
2. A |B | C=A|(B|C)==(A|B)|C
3. A ^B ^ C=A^(B ^ C)==(A ^ B) ^ C

Problems

1. **Kth Bit is set or not**

   Given a number N and a bit number K, check if the K^{th} bit of N is set or not. A bit is called set if it is 1.

   Example

   Input: n = 5, k = 1

   Output: SET

   Explanation: 5 is represented as 101 in binary and has its first bit set.

   Approach 1

   Check whether the K-th bit is set or not Using Left Shift Operator

   Left shift given number 1 by k to create a number that has only set a bit as kth bit.temp = 1 << k If **bitwise AND of n and temp** is non-zero, then result is SET else result is NOT SET.

   Implementation

```
#include <bits/stdc++.h>
using namespace std;
```

```
    if (n & (1 << k))
        cout << "SET";
    else
        cout << "NOT SET";
}

int main()
{
    int n, k;
    cin >> n >> k;
    isKthBitSet(n, k);
    return 0;
}
```

Approach 2
Check whether the K-th bit is set or not Using Right Shift Operator:

If we right shift n by k, we get the last bit as 1 if the Kth bit is set else 0

Implementation

```
#include <bits/stdc++.h>
using namespace std;

void isKthBitSet(int n, int k)
{
    if ((n>>k) & (1))
        cout << "SET";
    else
        cout << "NOT SET";
}

int main()
{
```

```
    isKthBitSet(n, k);
    return 0;
}
```

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

2. **Least Significant Bit which is set**
Find the position of the first 1 from right to left, in the binary representation of an Integer.
Examples:
Input: n = 18
Output: 2
Explanation: Binary Representation of 18 is 010010, hence the position of the first set bit from the right is 2.
Approach
Start from 0 positions and iterate till the 32nd-bit position if any bit is 1 break the loop and print the index else move left till 32.
Implementation

```
#include <bits/stdc++.h>
using namespace std;

int PositionRightmostSetbit(int n)
{
     if (n == 0)
    {
        return 0;
    }
    else {
        int pos = 1;
        for (int i = 0; i < 32; i++) {
```

```
            else
                break;
        }
        return pos;
    }
}

int main()
{
    int n;
    cin>>n;
    cout << PositionRightmostSetbit(n);
    return 0;
}
```

Time Complexity: O($\log_2$n), Traversing through all the bits of N, where at max there are logN bits.
Space Complexity: O(1)

3. [Find the element that appears once rest of the element appears twice]()

Given an array of integers. All numbers occur twice except one number which occurs once. Find the number.
Example :
Input: n=7, arr[] = {2, 3, 5, 4, 5, 3, 4}
Output: 2

Brute Force Method

Check every element if it appears once or not. Once an element with a single occurrence is found, return it.

Implementation

```
int SingleOccuringELement(int a[], int n)
{
    for (int i = 0; i < n; i++)
    {
        int count = 0;
        for (int j = 0; j < n; j++)
        {

            if (a[i] == a[j])
            {
                count++;
            }
        }
        if (count == 1)
        {
            return a[i];
        }
    }

    return -1;
}

int main()
{
    int n;
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];
    cout << SingleOccuringELement(a, n);
    return 0;
}
```

frequency of the elements and after that, we can iterate the hashmap to find the element with frequency 1.

Implementation

```
#include <bits/stdc++.h>
using namespace std;

int SingleOccuringELement(int a[],int n)
{
   unordered_map<int,int> mm;
      for(int i=0;i<n;i++)
     {
           mm[a[i]]++;
     }
      for(auto x:mm)
     {
           if(x.second==1) return x.first;
     }
}

int main()
{
    int n;
    cin>>n;
    int a[n];
    for(int i=0;i<n;i++) cin>>a[i];
    cout << SingleOccuringELement(a,n);
    return 0;
}
```

Bitwise Based Approach

The best solution is to use XOR. XOR of all array elements gives us the number with a

1. XOR of a number with itself is 0.
2. XOR of a number with 0 is the number itself.

Implementation

```
#include <bits/stdc++.h>
using namespace std;

int SingleOccuringELement(int a[],int n)
{
    int res = a[0];
    for (int i = 1; i < n; i++)
        res = res ^ a[i];

    return res;
}

int main()
{
    int n;
    cin>>n;
    int a[n];
    for(int i=0;i<n;i++) cin>>a[i];
    cout << SingleOccuringELement(a,n);
    return 0;
}
```

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

4. [Find element occuring once when all others are present thrice]()

Given an array of integers **arr[]** of length N, every element appears thrice except for one which occurs once.

BOSSCODER ACADEMY UPSKILL WITH US

Input: N = 4 arr[] = {1, 10, 1, 1}

Output: 10

Approach

We will traverse for every bit position in all the numbers if the bit is 1 for any number that appears thrice then it will be divisible by 3 if it is not then the number that appears once has it's a bit as 1 there so we can get the number that appears once using this concept.

Implementation

```
#include <bits/stdc++.h>
using namespace std;

int SingleOccuringELement(int a[],int n)
{
   int ans=0;
        for(int i=0;i<32;i++)
        {
            int bit=1<<i;
            int c=0;
            for(int j=0;j<n;j++)
            {
                if(a[j]&bit)
                c++;
            }
            if(c%3)
            ans|=bit;
        }
        return ans;
}

int main()
{
```

BOSSCODER ACADEMY UPSKILL WITH US

```
    int a[n];
    for(int i=0;i<n;i++) cin>>a[i];
    cout << SingleOccuringELement(a,n);
    return 0;
}
```

Time Complexity: O(N)
Space Complexity: O(1)

5. **Find the two non-repeating elements in an array of repeating elements**
Given an array in which all numbers except two are repeated once. Find those two numbers.
Example
Input: n=6 ,a[]={1,1,2,2,3,4}
Output:3,4
Method 1(Use Sorting)
First, sort all the elements. In the sorted array, by comparing adjacent elements we can easily get the non-repeating elements.
Implementation

```
#include <bits/stdc++.h>
using namespace std;

vector<int> SingleOccuringELementTwo(int a[
{
   sort(a, a + n);

    vector<int> ans;
    for (int i = 0; i < n - 1; i = i + 2) {
        if (a[i] != a[i + 1]) {
            ans.push_back(a[i]);
```

```
    }

    if (ans.size() == 1)
        ans.push_back(a[n - 1]);

    return ans;
}

int main()
{
    int n;
    cin>>n;
    int a[n];
    for(int i=0;i<n;i++) cin>>a[i];
      vector<int>ans=SingleOccuringELement1
   cout<<ans[0]<<" "<<ans[1]<<endl;
    return 0;
}
```

Method 2(Use XOR)

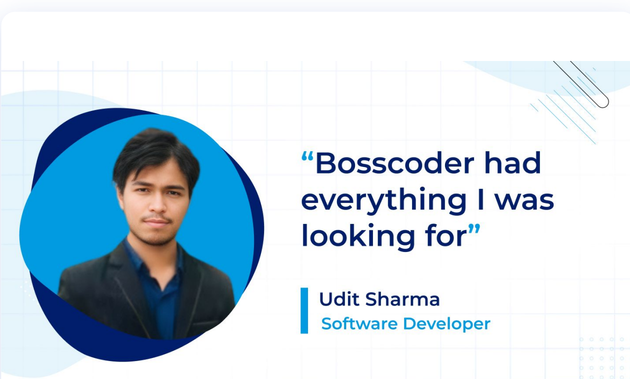
Let x and y be the non-repeating elements we are looking for and arr[] be the input array. First, calculate the XOR of all the array elements. All the bits that are set in xor will be set in one non-repeating element (x or y) and not in others. So if we take any set bit of xor and divide the elements of the array into two sets – one set of elements with the same bit set and another set with the same bit not set. By doing so, we will get x in one set and y in another set. Now if we do XOR of all the elements in the first set, we will get the first non-repeating element, and by doing the same in other sets we will get the second non-

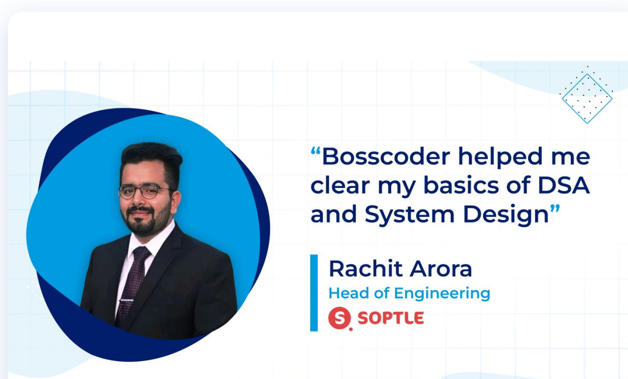
BOSSCODER ACADEMY UPSKILL WITH US

```
#include <bits/stdc++.h>
using namespace std;

void SingleOccuringELementTwo(int a[], int
{
    int Xor = a[0];

    int set_bit_no;
    int i;
    int x = 0;
    int y = 0;
    for (i = 1; i < n; i++)
        Xor ^= a[i];
    set_bit_no = Xor & ~(Xor - 1);
    for (i = 0; i < n; i++)
    {
        if (a[i] & set_bit_no)
            x = x ^ a[i];
        else
        {
            y = y ^ a[i];
        }
    }
    cout << x << " " << y;
}

int main()
{
    int n;
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];
```

```
    return 0;
}
```

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





BOSSCODER ACADEMY

UPSKILL WITH US

for

Read

and System Design

Read

View All blogs

BOSSCODER ACADEMY

Helping ambitious learners upskill themselves & shift to top product based companies.

Lets hear all about it.

Who are we

- About us
- Blog
- Attend a FREE Event
- Privacy Policy
- Terms & Condition
- Pricing and Refund Policy

Contact Us

Email: ask@bosscoderacademy.com

Follow us on

- LinkedIn
- Youtube
- Instagram
- Telegram
- Reviews on Quora