Maaz Follow
Programmer at Ubisoft, fascinated about Artificial Intelligence. Follow me & we will learn together.
Jul 9, 2017 · 4 min read

XOR - The magical bitwise operator

Understanding bit manipulation provide new approaches you never knew existed to solve a particular problem. Let us do what's necessary to start developing this bit-wise approach.

In this article, we will discuss about the magical powers of XOR bitwise operator.

XOR is a really surprising operator. You can never imagine the things it makes possible for us to do. Before seeing what it can do, lets us revise what we may already know about the operator.

Bitwise **XOR (^)** like the other operators (except ~) also take two equal-length bit patterns. If both bits in the compared position of the bit patterns are 0 or 1, the bit in the resulting bit pattern is 0, otherwise 1.

In short, it means that it returns 1 only if exactly one bit is set to 1 out of the two bits in comparison (Exclusive OR).
A = 5 = 0101, B = 3 = 0011
A ^ B = 0101 ^ 0011 = 0110 = 6

That was the basic stuff about XOR. Now let's see what all magical powers does the XOR operator possess! I would like to explain them by giving some problems before, which will help you understand the properties clearly.

Return the rightmost 1 in the binary representation of a number.

Example: For 1010, you should perform some operations to give 0010 as the output. For 1100, you should give 0100. Similarly for 0001, you should return 0001.

Try finding the solution yourself. The biggest hint you have is you can do it using (^) operator. After you're done, scroll down.

Solution:

For this problem, you need to know a property of binary subtraction. Check if you can find out the property in the examples below,

1000 – 0001 = 0111

0100 – 0001 = 0011

1100 – 0001 = 1011

The property is, the difference between a binary number n and n-1 is all the bits on the right of the rightmost 1 are flipped including the rightmost 1. Using this amazing property, we can get our solution as

x ^ (x & (x - 1))

The only way you can totally understand how the above solution is working is by trying it out for different binary numbers on a piece of paper.

(If you have a CS background and understood whatever i have said up till now, then congrats! You now already know 80% about a powerful data structure called **Fenwick Tree** or **Binary Indexed Tree.** You can look up on it to learn the 20% or let me know if you want my next article to be about it.)

Interesting! isn’t it? Let’s see some more features.

For a given array of repeated elements, exactly one element is not repeated. You need to return the non-repeated element.

[1, 2, 5, 4, 6, 8, 9, 2, 1, 4, 5, 8, 9]

You can refer the example above. You will need to return 6.

There exists a solution of linear complexity.

Solution:

This one is kinda straightforward. You’ll need to know the following properties

n ^ n = 0

n ^ 0 = n

Algorithm:

1. Create a variable v with value 0.
2. Iterate over array from i = 0 to i = n-1
3. Perform v ^ arr[i] and store the result in v for every iteration.
4. Return v.

```
arr = [1, 2, 5, 4, 6, 8, 9, 2, 1, 4, 5, 8, 9]
v = 0
for i in range(len(arr)):
  v = v ^ arr[i]
print(value)
```

So, for my final question, i would like to give you a problem that is quite challenging. This one do not require the use of any new property of XOR other than the ones mentioned above.

Write a function to determine the number of bits required to convert integer A to integer B.

Input: 31, 14

Output: 2

Explanation: You have 31 (11111) and 14 (01110). To convert 31 to 14 we would have to flip the leftmost bit and the rightmost bit of 31.

Number of bits needed to flip is 2, so we return 2 as the answer.

Input: 12, 7

Output: 3

I suggest that you try to implement it before looking at the solution below. Good luck!

If you feel that this has helped you in any way, then please like and follow me for more fascinating insights in Computer Science.

Solution:

```python
def bitswaprequired(a, b):
  count = 0
  c = a^b
  while(c != 0):
    count += c & 1
    c = c >> 1
  return count
```