



← Notes

▲ Fun With Bits : Print all subsets of a set

12

Bit

Bit-manipulation

Bit-shift

Set

Subset

Subset-sum

2

At times, bit-manipulations come as a very handy tool for simplifying the problem, improve efficiency. Bit-manipulations is a kind of low level optimization tool.

Today, lets see some bitwise operations and cool some cool snippets which involves bits-manipulations. Are you a lover of short codes? You will like this. And to end our discussion, we will see how to print all possible subsets of a set. Also from this we can solve subset sum problem too.

Ready?

To start our discussion, lets focus on the basic operations. If you already know this, you can skip this part.

Basic Operations

1. Bitwise AND

$$0 \& 0 = 0$$

$$0 \& 1 = 0$$

$$1 \& 0 = 0$$

$$1 \& 1 = 1$$

i.e. as long as both bits are not SET, the outcome will always be zero (false).

So lets say, we want: $5 \& 6$ $5 \rightarrow 101$ and $6 \rightarrow 110$

$$101$$

$$\&$$

$$110$$

$$===$$

$$100 \rightarrow 4$$

Hence the answer 4. You got it, right? Easy one.

Similarly,

2. Bitwise OR

$$0 \& 0 = 0$$

$$0 \& 1 = 1$$

$$1 \& 0 = 1$$

$$1 \& 1 = 1$$

i.e. as long as any one of the bits is SET, the outcome will be one (true).

3. Bitwise XOR

$$0 \& 0 = 0$$

$$0 \& 1 = 1$$

$$1 \& 0 = 1$$

$$1 \& 1 = 0$$

i.e. whenever we see the same bits, answer goes zero (false), one (true) otherwise.

The basic operations other than AND, OR, XOR are Left shift and Right shift.

1. Left Shifting This shifts the bit-pattern of any number to left by specified positions. E.g. $5 \ll 1$ i.e. $(101) \ll 1$ and after shifting the pattern, we get 1010 i.e. 10, you see the last zero? Yes zeros get appended in left shift.

Similarly, 2. Right Shifting: This shifts the bit-pattern of number to right by specified positions. E.g. $5 \gg 1$ i.e. $(101) \gg 1$ and after shifting the pattern, we get 10 i.e. 2, you see that the rightmost '1' got disappeared?

Having done the base work, lets move ahead to see some interesting things.

Bit Masking

Masking : Hiding

Yes, we can hide the value of bit. In simple words, using bit masking we can set, reset any bit from a number. There are 2 methods to hide the data.

1. OR with 1

No matter what the given bit is, once you OR it with 1, it will always be 1. So, this way, you can hide i.e. mask the actual data of given bit.

2. AND with 0

Similarly, No matter what the given bit is, once you AND it with 0, it will always be 0. So, this way also, you can hide i.e. mask the actual data of given bit.

Ok, you must have got what the Bit-Masking is I think.

Now we will see some really useful operations using bits.

1. Multiplying a number by 2 ---> Left Shift the number by 1.

We know that numbers are stored in binary inside computer. Also we know left shifting appends zeros at the end. Can we not use these two facts to multiply a number by 2. Figure out how? Yes, just left shift the number by 1 and we get the answer.

Here is an exercise for you. Can you generalized this to multiply a number by any power of 2? i.e. 2, 4, 8 ... I think you do this within few seconds, right?

2. Division by 2 ---> Right Shift the number by 1

Similar to the multiplication. For odd numbers, it gives Floor value of the answer.

3. Finding powers of 2 --> Left shift 1 by (exponent)

Powers of 2 are 1, 2, 4, 8, ... and in binary if we see, 1, 10, 100, 1000, ... Also, if see it in different perspective, can we not use what we have learnt already. See how. $1 = 1$ $2 = 1 * 2$ $4 = 1 * 2 * 2$ $8 = 1 * 2 * 2 * 2$. . . It means this is just multiplying 2, for given number of times with 1. To find 4th power of 2, multiply 2, 4 times. i.e. $(1 << 4)$.

So, for nth power of 2, the answer is (1 left shift by n).

4. Check if number is power of 2 ---> $ans = (n \& (n-1) == 0)$.

If we observe, all powers of 2 have only one bit SET (right-most one) and all other are RESET. And all numbers which are one less than any power of 2 ($power \geq 1$), have all bits SET. E.g. 8 and 7, 1000 and 111 E.g, 16 and 15, 10000 and 1111 No other numbers have such property. So, Now can you come of with some idea to check if that number n is a power of 2? Ok, hint is, make use of (n-1) too.

Oh, yes! We need to just and them, and if we get the result to be 0, then that is a power of 2, not otherwise.

5. Convert the decimal number to binary ---> Loop and print number&1.

We know that numbers are stored in binary. Why convert them by our own method? Instead make use of that thing. We have a number say n. Now if we AND that number with 1, what will we get? The status of right-most bit, right? Then we will right shift the number by 1, again AND it with 1, we will again get right-most bit which will actually be second last in original number. Then right shift original number by 2, AND with 1,

get the status. Right shift number by 3, AND with 1, get the status.... and so on.

We just need to print this in reversed order now and this is our binary representation, is this not?

I think now you have got the gist of bit-wise operations. So, I think it is enough explanation. Now I will give you the direct solutions of problems, fine?

6. Check if number is even or odd ---> AND with 1

We just need to check the status of last bit.

```
if(n&1==0):  
    EVEN  
else:  
    ODD
```

7. Get the status of xth (1-based indexing) bit from right in number n. ---> AND with 1 shifted to xth bit.

```
if(n&(1<<x-1)==0):  
    RESET  
else:  
    SET
```

Arithmetic operators have higher priority than bit-wise shifting. So, (x-1) will be evaluated first and then 1<<(that_ans).

8. SET the xth bit in number ---> XOR with 1 shifted to xth bit

(Assuming it is RESET already)

```
n = n ^ (1<<x-1)
```

9. Toggle the xth bit in number ---> XOR with 1 shifted to xth bit

```
n = n ^ (1<<x-1)
```

10. Check if two numbers are equal ---> XOR them

```
if(a^b==0):  
    EQUAL  
else:  
    NOTEQUAL
```

I have a problem in my mind. Want to hear it? Here it is -- Given array of n elements where every number occurs in pair i.e. even number of times except one. Find that number. Let me know once you solve it.

11. Did you know this:

If we have 3 number a , b and c where $c = a \oplus b$, then XORing any 2 numbers gives the 3rd.

This is the trick used in swapping two numbers. Have a look at it [here](#) and try to map these two points.

12. Do you know the concept of XOR Doubly Linked List?

Let me tell you in short: XOR Doubly Linked List is a variation of Doubly Linked List which uses a lesser memory per node than original version. Doubly Linked List stores addresses of previous and next nodes. But XOR doubly linked list uses the property of XOR discussed above and hence needs only one address field per node. Google it for more.

13. RESET the right-most SET bit in a number n

```
n = n & (n - 1)
```

14. Check how many bit the number is! Bit-length

```
len = 0
while(n):
    len += 1
    n >>= 1
```

15. Counting number of set bits

```
cnt = 0
while(n):
    cnt += 1
    n &= (n - 1)
```

16. A little difficult problem of *finding all subsets of a set* can be solved very easily with bit-wise operations.

Lets say, Set has n elements. We will use the bits of numbers to show the presence of element in set. The key is - If x th bit in number is SET i.e. 1, then x th element in given

set is present in current subset. Got it? We will loop from 0 to $(2^n) - 1$, and for each number num, we will check the SET bits in it and take corresponding elements. In each iteration, we will have one subset.

At the end we will have **all 2^n subsets of set**. Once you have a subset, you can find the sum of that subset if asked.

Implement it now for fun and practice.

This is it. I think I should stop here. I will keep adding more bit-tricks to this post whenever I come to know. Comment them if you know any.

Let me know if you liked this post.

Have look at other posts by me [here](#).

Thank you!

Like 0 Tweet 1 G+1 0


AUTHOR



Bhavesh Munot

 System Software Engineer ...

 Pune, Maharashtra, India

 9 notes

Write Note

My Notes

Drafts

TRENDING NOTES

[Strings And String Functions](#)

written by Vinay Singh

[Segment Tree and Lazy Propagation](#)

written by Akash Sharma

[Number Theory - II](#)

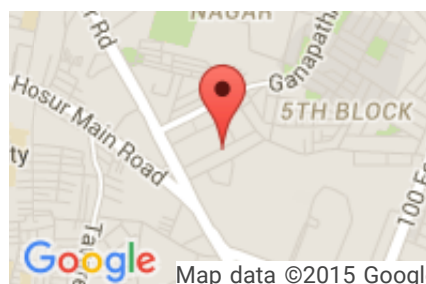
written by Tanmay Chaudhari

[Matrix exponentiation](#)

written by Mike Koltsov

[Graph Theory - Part II](#)

written by Pawel Kacprzak

[more ...](#)**ABOUT US**[Blog](#)[Engineering Blog](#)[Updates & Releases](#)[Team](#)[Careers](#)[In the Press](#)**HACKEREARTH**[API](#)[Chrome Extension](#)[CodeTable](#)[HackerEarth Academy](#)[Developer Profile](#)[Resume](#)[Campus Ambassadors](#)[Get Me Hired](#)[Privacy](#)[Terms of Service](#)**DEVELOPERS**[AMA](#)[Code Monk](#)[Judge Environment](#)[Solution Guide](#)[Problem Setter Guide](#)[Practice Problems](#)[HackerEarth Challenges](#)[College Challenges](#)**RECRUIT**[Developer Sourcing](#)[Lateral Hiring](#)[Campus Hiring](#)[FAQs](#)[Customers](#)[Annual Report](#)**REACH US**

IIIrd Floor, Salarpuria Business Center,
4th B Cross Road, 5th A Block,
Koramangala Industrial Layout,
Bangalore, Karnataka 560095, India.

✉ contact@hackerearth.com

☎ +91-80-4155-4695

☎ +1-650-461-4192



© 2015 HackerEarth