

[← Notes](#)

1

**Bit Manipulation**

164

Code Monk

Bit

Bit-fields

Bit-shift-operators

Bit-manipulation

Bitset

LIVE EVENTS

Working on bytes, or data types comprising of bytes like ints, floats, doubles or even data structures which stores large amount of bytes is normal for a programmer. In some cases , a programmer needs to go beyond this - that is to say that in a deeper level where the importance of bits is realized.

Operations with bits are used in **Data compression** (data is compressed by converting it from one representation to another, to reduce the space) ,**Exclusive-Or Encryption** (an algorithm to encrypt the data for safety issues). In order to encode, decode or compress files we have to extract the data at bit level. Bitwise Operations are faster and closer to the system and sometimes optimize the program to a good level.

We all know that 1 byte comprises of 8 bits and any integer or character can be represented using bits in computers, which we call its binary form(contains only 1 or 0) or in its base 2 form.

Example:

$$\begin{aligned} 1) 14 &= \{11110\}_2 \\ &= 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0 \\ &= 14. \end{aligned}$$

$$\begin{aligned} 2) 20 &= \{10100\}_2 \\ &= 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0 \\ &= 20. \end{aligned}$$

For characters, we use ASCII representation, which are in the form of integers which again can be represented using bits as explained above.

Bitwise Operators:

There are different bitwise operations used in the bit manipulation. These bit operations operate on the individual bits of the bit patterns. Bit operations are fast and can be used in optimizing time complexity. Some common bit operators are:

NOT (~): Bitwise NOT is an unary operator that flips the bits of the number i.e., if the *i*th bit is 0, it will change it to 1 and vice versa. Bitwise NOT is nothing but simply the one's complement of a number. Lets take an example.

$$\begin{aligned} N &= 5 = (101)_2 \\ \sim N &= \sim 5 = \sim(101)_2 = (010)_2 = 2 \end{aligned}$$

?

AND (&): Bitwise AND is a binary operator that operates on two equal-length bit patterns. If both bits in the compared position of the bit patterns are 1, the bit in the resulting bit pattern is 1, otherwise 0.

$$A = 5 = (101)_2, B = 3 = (011)_2 \quad A \& B = (101)_2 \& (011)_2 = (001)_2 = 1$$

OR (|): Bitwise OR is also a binary operator that operates on two equal-length bit patterns, similar to bitwise AND. If both bits in the compared position of the bit patterns are 0, the bit in the resulting bit pattern is 0, otherwise 1.

$$A = 5 = (101)_2, B = 3 = (011)_2$$

$$A | B = (101)_2 | (011)_2 = (111)_2 = 7$$

XOR (^): Bitwise XOR also takes 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.

$$A = 5 = (101)_2, B = 3 = (011)_2$$

$$A \wedge B = (101)_2 \wedge (011)_2 = (110)_2 = 6$$

Left Shift (<<): Left shift operator is a binary operator which shifts the some number of bits, in the given bit pattern, to the left and appends 0 at the end. Left shift is equivalent to multiplying the bit pattern with 2^k (if we are shifting k bits).

$$1 \ll 1 = 2 = 2^1$$

$$1 \ll 2 = 4 = 2^2 \quad 1 \ll 3 = 8 = 2^3$$

$$1 \ll 4 = 16 = 2^4$$

...

$$1 \ll n = 2^n$$

Right Shift (>>): Right shift operator is a binary operator which shifts the some number of bits, in the given bit pattern, to the right and appends 1 at the end. Right shift is equivalent to dividing the bit pattern with 2^k (if we are shifting k bits).

$$4 \gg 1 = 2$$

$$6 \gg 1 = 3$$

$$5 \gg 1 = 2$$

$$16 \gg 4 = 1$$

X	Y	X&Y	X Y	X^Y	~(X)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Bitwise operators are good for saving space and sometimes to cleverly remove dependencies.

Note: All left and right side taken in this article, are taken with reference to the reader.

Lets discuss some algorithms based on bitwise operations:

1) How to check if a given number is a power of 2 ?

Consider a number N and you need to find if N is a power of 2. Simple solution to this problem is to repeatedly divide N by 2 if N is even. If we end up with a 1 then N is power of 2, otherwise not. There are a special case also. If N = 0 then it is not a power of 2. Let's code it.

Implementation:

```
bool isPowerOfTwo(int x)
{
    if(x == 0)
        return false;
    else
    {
        while(x % 2 == 0) x /= 2;
        return (x == 1);
    }
}
```

Above function will return true if x is a power of 2, otherwise false.

Time complexity of the above code is $O(\log N)$.

The same problem can be solved using bit manipulation. Consider a number x that we need to check for being a power for 2. Now think about the binary representation of (x-1). (x-1) will have all the bits same as x, except for the rightmost 1 in x and all the bits to the right of the rightmost 1.

Let, $x = 4 = (100)_2$

$x - 1 = 3 = (011)_2$

Let, $x = 6 = (110)_2$

$x - 1 = 5 = (101)_2$

It might not seem obvious with these examples, but binary representation of (x-1) can be obtained by simply flipping all the bits to the right of rightmost 1 in x and also including the rightmost 1.

Now think about $x \& (x-1)$. $x \& (x-1)$ will have all the bits equal to the x except for the rightmost 1 in x.

Let, $x = 4 = (100)_2$

1

LIVE EVENTS

?

$$x - 1 = 3 = (011)_2$$

$$x \& (x-1) = 4 \& 3 = (100)_2 \& (011)_2 = (000)_2$$

$$\text{Let, } x = 6 = (110)_2$$

$$x - 1 = 5 = (101)_2$$

$$x \& (x-1) = 6 \& 5 = (110)_2 \& (101)_2 = (100)_2$$

Properties for numbers which are powers of 2, is that they have one and only one bit set in their binary representation. If the number is neither zero nor a power of two, it will have 1 in more than one place. So if x is a power of 2 then $x \& (x-1)$ will be 0.

Implementation:

```
bool isPowerOfTwo(int x)
{
    // x will check if x == 0 and !(x & (x - 1)) will check if x is a power of
    2 or not
    return (x && !(x & (x - 1)));
}
```

2) Count the number of ones in the binary representation of the given number.

The basic approach to evaluate the binary form of a number is to traverse on it and count the number of ones. But this approach takes $\log_2 N$ of time in every case.

Why $\log_2 N$?

As to get a number in its binary form, we have to divide it by 2, until it gets 0, which will take $\log_2 N$ of time.

With bitwise operations, we can use an algorithm whose running time depends on the number of ones present in the binary form of the given number. This algorithm is much better, as it will reach to $\log N$, only in its worst case.

```
int count_one (int n)
{
    while( n )
    {
        n = n&(n-1);
        count++;
    }
    return count;
}
```

Why this algorithm works ?

As explained in the previous algorithm, the relationship between the bits of x and $x-1$. So

?

as in $x-1$, the rightmost 1 and bits right to it are flipped, then by performing $x \& (x-1)$, and storing it in x , will reduce x to a number containing number of ones (in its binary form) less than the previous state of x , thus increasing the value of count in each iteration.

Example:

$n = 23 = \{10111\}_2$.

- Initially, count = 0.
- Now, n will change to $n \& (n-1)$. As $n-1 = 22 = \{10110\}_2$, then $n \& (n-1)$ will be $\{10111\}_2 \& \{10110\}_2$, which will be $\{10110\}_2$ which is equal to 22. Therefore n will change to 22 and count to 1.
- As $n-1 = 21 = \{10101\}_2$, then $n \& (n-1)$ will be $\{10110\}_2 \& \{10101\}_2$, which will be $\{10100\}_2$ which is equal to 20. Therefore n will change to 20 and count to 2.
- As $n-1 = 19 = \{10011\}_2$, then $n \& (n-1)$ will be $\{10100\}_2 \& \{10011\}_2$, which will be $\{10000\}_2$ which is equal to 16. Therefore n will change to 16 and count to 3.
- As $n-1 = 15 = \{01111\}_2$, then $n \& (n-1)$ will be $\{10000\}_2 \& \{01111\}_2$, which will be $\{00000\}_2$ which is equal to 0. Therefore n will change to 0 and count to 4.
- As $n = 0$, the the loop will terminate and gives the result as 4.

Complexity: $O(K)$, where K is the number of ones present in the binary form of the given number.

3) Check if the i^{th} bit is set in the binary form of the given number.

To check if the i^{th} bit is set or not (1 or not), we can use AND operator. How?

Let's say we have a number N , and to check whether its i^{th} bit is set or not, we can AND it with the number 2^i . The binary form of 2^i contains only i^{th} bit as set (or 1), else every bit is 0 there. When we will AND it with N , and if the i^{th} bit of N is set, then it will return a non zero number (2^i to be specific), else 0 will be returned.

Using Left shift operator, we can write 2^i as $1 \ll i$. Therefore:

```
bool check (int N)
{
    if( N & (1 << i) )
        return true;
    else
        return false;
}
```

Example:

Let's say $N = 20 = \{10100\}_2$. Now let's check if its 2nd bit is set or not (starting from 0). For that, we have to AND it with $2^2 = 1 \ll 2 = \{100\}_2$.

$\{10100\} \& \{100\} = \{100\} = 2^2 = 4$ (non-zero number), which means its 2nd bit is set.

4) How to generate all the possible subsets of a set ?

A big advantage of bit manipulation is that it can help to iterate over all the subsets of an N-element set. As we all know there are 2^N possible subsets of any given set with N elements. What if we represent each element in a subset with a bit. A bit can be either 0 or 1, thus we can use this to denote whether the corresponding element belongs to this given subset or not. So each bit pattern will represent a subset.

Consider a set A of 3 elements.

$A = \{a, b, c\}$

Now, we need 3 bits, one bit for each element. 1 represent that the corresponding element is present in the subset, whereas 0 represent the corresponding element is not in the subset. Let's write all the possible combination of these 3 bits.

$0 = (000)_2 = \{\}$

$1 = (001)_2 = \{c\}$

$2 = (010)_2 = \{b\}$

$3 = (011)_2 = \{b, c\}$

$4 = (100)_2 = \{a\}$

$5 = (101)_2 = \{a, c\}$

$6 = (110)_2 = \{a, b\}$

$7 = (111)_2 = \{a, b, c\}$

Pseudo Code:

```
possibleSubsets(A, N):
    for i = 0 to  $2^N$ :
        for j = 0 to N:
            if jth bit is set in i:
                print A[j]
        print '\n'
```

Implementation:

```
void possibleSubsets(char A[], int N)
{
    for(int i = 0; i < (1 << N); ++i)
    {
        for(int j = 0; j < N; ++j)
            if(i & (1 << j))
                cout << A[j] << ' ';
        cout << endl;
    }
}
```

}

5) Find the largest power of 2 (**most significant bit** in binary form), which is less than or equal to the given number N.

Idea: Change all the bits which are at the right side of the most significant digit, to 1.

Property: As we know that when all the bits of a number N are 1, then N must be equal to $2^i - 1$, where i is the number of bits in N.

Example:

Let's say binary form of a N is $\{1111\}_2$ which is equal to 15.

$15 = 2^4 - 1$, where 4 is the number of bits in N.

This property can be used to find the largest power of 2 less than or equal to N. How?

If we somehow, change all the bits which are at right side of the most significant bit of N to 1, then the number will become $x + (x-1) = 2 * x - 1$, where x is the required answer.

Example:

Let's say $N = 21 = \{10101\}$, here most significant bit is the 4th one. (counting from 0th digit) and so the answer should be 16.

So let's change all the right side bits of the most significant bit to 1. Now the number changes to

$\{11111\} = 31 = 2 * 16 - 1 = Y$ (let's say).

Now the required answer is $(Y+1) \gg 1$ or $(Y+1)/2$.

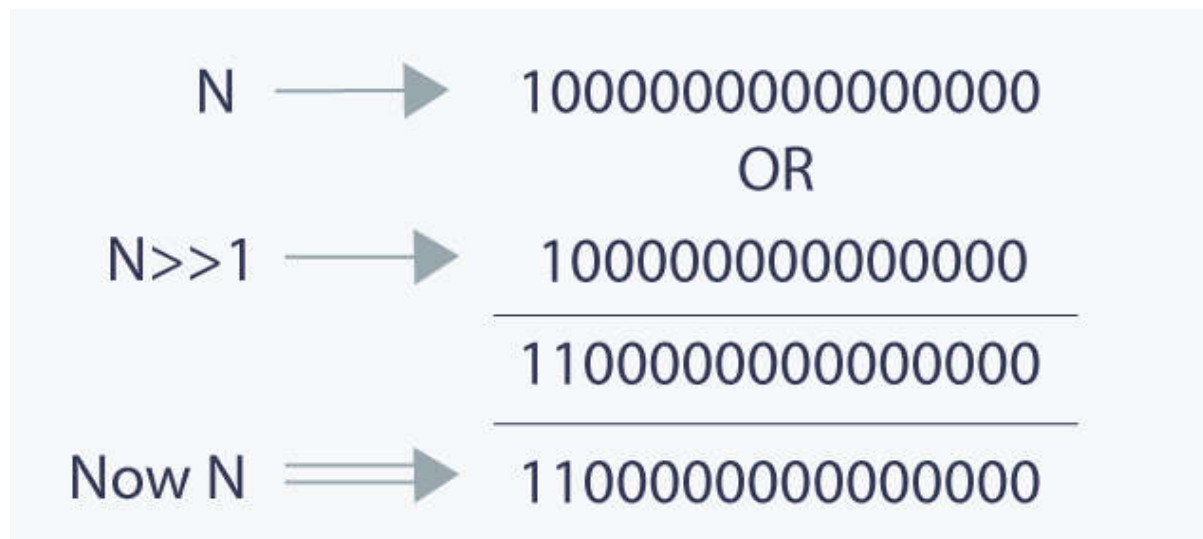
Now the question arises here is how can we change all right side bits of most significant bit to 1?

Let's take the N as 16 bit integer and binary form of N is $\{1000000000000000\}$.

Here we have to change all the right side bits to 1.

Initially we will copy that most significant bit to its adjacent right side by:

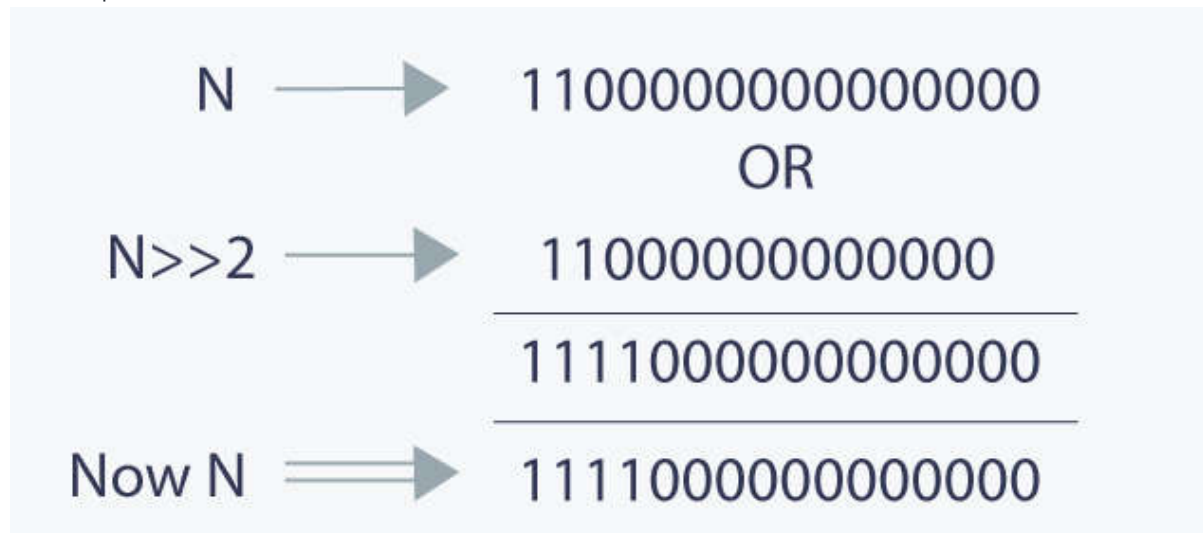
$N = N | (N \gg 1)$.


 1
LIVE EVENTS

As you can see, in above diagram, after performing the operation, rightmost bit has been copied to its adjacent place.

Now we will copy the 2 rightmost set bits to their adjacent right side.

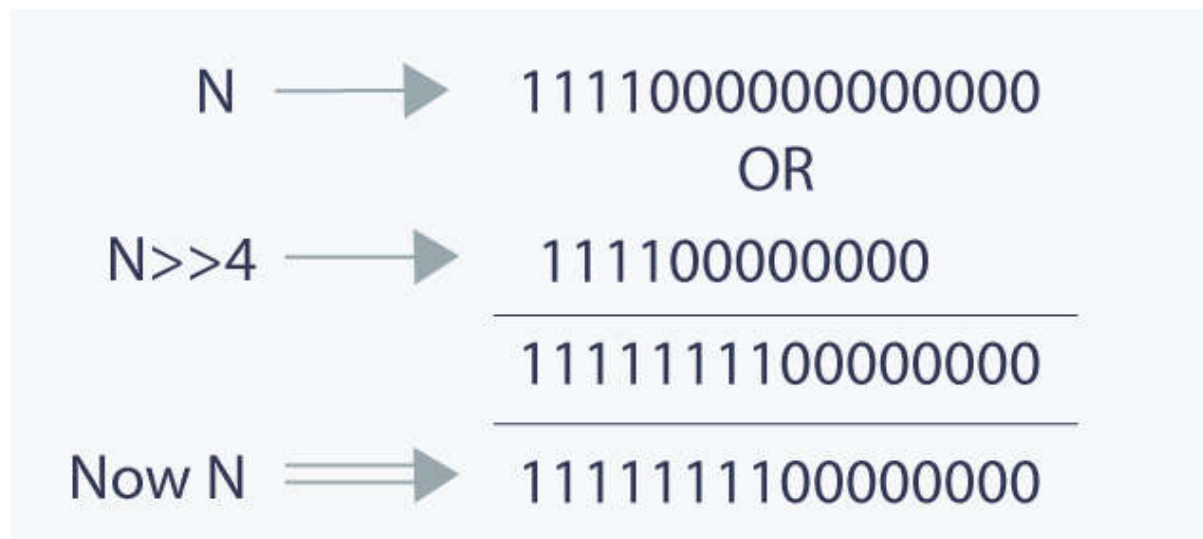
$N = N | (N \gg 2)$.



Now we will copy the 4 rightmost set bit to their adjacent right side.

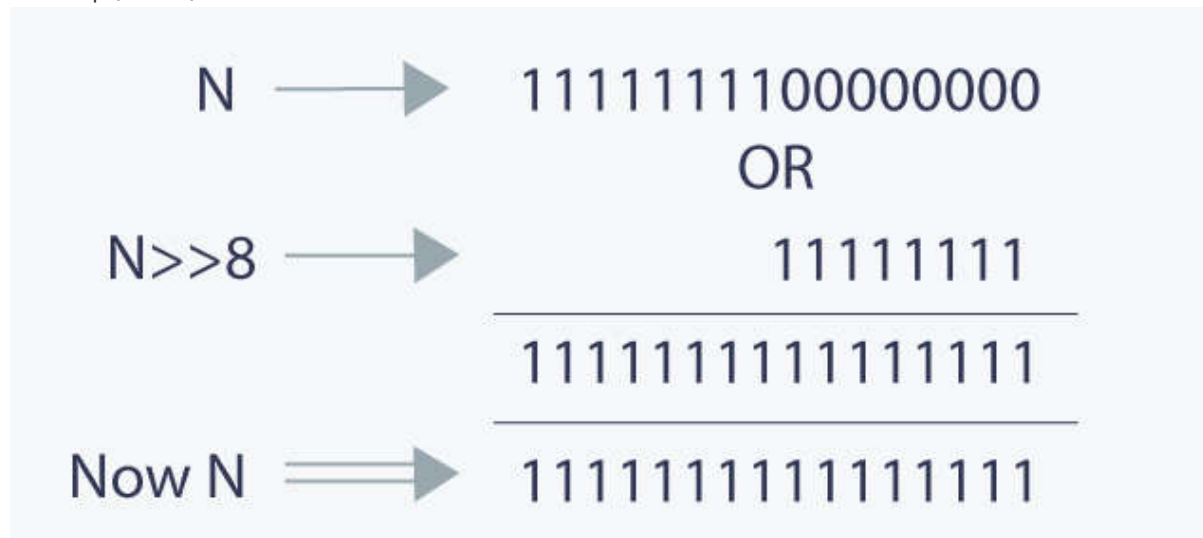
$N = N | (N \gg 4)$

?



Now we will copy these 8 rightmost set bits to their adjacent right side.

$N = N | (N \gg 8)$



Now all the right side bits of the most significant set bit has been changed to 1. This is how we can change right side bits. This explanation is for 16 bit integer, and it can be extended for 32 or 64 bit integer too.

Implementation:

```
long largest_power(long N)
{
    //changing all right side bits to 1.
    N = N | (N >> 1);
    N = N | (N >> 2);
    N = N | (N >> 4);
    N = N | (N >> 8);
}
```

```
//as now the number is 2 * x-1, where x is required answer, so adding 1 and dividing it by
2.
        return (N+1)>>1;
    }
```

Tricks with Bits:

1) $x \wedge (x \& (x-1))$: Returns the rightmost 1 in binary representation of x.

As explained above, $(x \& (x - 1))$ will have all the bits equal to the x except for the rightmost 1 in x. So if we do bitwise XOR of x and $(x \& (x-1))$, it will simply return the rightmost 1. Let's see an example.

$$x = 10 = (1010)_2$$

$$x \& (x-1) = (1010)_2 \& (1001)_2 = (1000)_2$$

$$x \wedge (x \& (x-1)) = (1010)_2 \wedge (1000)_2 = (0010)_2$$

2) $x \& (-x)$: Returns the rightmost 1 in binary representation of x

$(-x)$ is the two's complement of x. $(-x)$ will be equal to one's complement of x plus 1.

Therefore $(-x)$ will have all the bits flipped that are on the left of the rightmost 1 in x. So $x \& (-x)$ will return rightmost 1.

$$x = 10 = (1010)_2$$

$$(-x) = -10 = (0110)_2$$

$$x \& (-x) = (1010)_2 \& (0110)_2 = (0010)_2$$

3) $x \mid (1 \ll n)$: Returns the number x with the nth bit set.

$(1 \ll n)$ will return a number with only nth bit set. So if we OR it with x it will set the nth bit of x.

$$x = 10 = (1010)_2 \quad n = 2$$

$$1 \ll n = (0100)_2$$

$$x \mid (1 \ll n) = (1010)_2 \mid (0100)_2 = (1110)_2$$

Applications of bit operations:

1) They are widely used in areas of graphics ,specially XOR(Exclusive OR) operations.

2) They are widely used in the embedded systems, in situations, where we need to set/clear/toggle just one single bit of a specific register without modifying the other contents. We can do OR/AND/XOR operations with the appropriate mask for the bit position.

3) Data structure like n-bit map can be used to allocate n-size resource pool to represent the current status.

4) Bits are used in networking, framing the packets of numerous bits which is sent to another system generally through any type of serial interface.

To see a creative use of Bitwise operators, you can refer [this amazing article](#) , where the bit wise operations are smartly used while developing an online calendar for events.

Solve Problems

1
LIVE EVENTS

Tweet

评论 (64)

排序方式: 相关

登录/注册进行评论

Ankit Luthra 3 years ago

what will be the complexity of implementation of largest power of 2 \leq given number N explained above

▲ 1 票数 ● 回复 ● 信息 ● 永久链接



Prateek Garg ⚡ Author 3 years ago

Assuming all the bit wise operations as constant time operations (as they are nearly equal to constant), the time complexity of this algorithm can also be taken as constant.

▲ 3 票数 ● 回复 ● 信息 ● 永久链接



Anurag Maithani a year ago

I THINK COMPLEXITY WOULD BE $\log(n)$.

due to time taken to convert all the bits of the given no. to 1.

eg. (1010) to (1111).since every time we increase the size of shift in power of 2 .t herefore time taken should be $\log n$ base 2.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Abhinav Kr Singh 8 months ago

Yes i guess if N represents the number of bits in the number but $\log n$ base 2 to 16 bit number or even 32 bit number will be 4 to 5 so i guess lets say it to be a constant

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Vignesh Mahalingam 3 years ago

Code Monk rocks <3

▲ 3 票数 ● 回复 ● 信息 ● 永久链接

Duddupudi Sai Avinash 3 years ago

Small typo mistake in Left Shift (\ll) section :-

it is written1 $\ll n = 2n$

Corrected one is 1 $\ll n = 2^n$ (2 power n)

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Prateek Garg ⚡ Author 3 years ago

Thanks. It is fixed.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



competitivecoder 3 years ago

[developer:ptk23] Do we have some sample problems here at hackerearth.Can y

?

ou suggest some.except the past codemonk series problem

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Prateek Garg ⚡ Author 3 years ago

<https://www.hackerearth.com/problem/algorithm/aaryan-subsequences-and-great-xor/>

<https://www.hackerearth.com/problem/algorithm/subset-xor-4/>

<https://www.hackerearth.com/problem/algorithm/power-of-two-4/>

You can start with some easy problems :) .

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



competitivecoder 3 years ago

[developer:ptk23] Thanks :D

▲ 1 票数 ● 回复 ● 信息 ● 永久链接

Jack_1729 7 months ago

Good article please clear "2." written in the last code snippet.

▲ 1 票数 ● 回复 ● 信息 ● 永久链接

MD ASAD RUB 3 years ago

simple..brilliant..awesome!!

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Nitin Gaikwad 3 years ago

Nice...

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Shivasurya S 3 years ago

nice article! have learnt something useful.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

sanghpriya 3 years ago

superbbbb

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

vinayak kothari 3 years ago

really very good article for new coders

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Shyam Singh 3 years ago

A humble request:

I guess "leftmost" should be replaced by "rightmost" in many sentences in this tutorial. New Coders won't get confused.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Dynamo 3 years ago

Right.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Prateek Garg ⚡ Author 3 years ago

Yes, it was correct but not clear in some areas. It is fixed. Thanks for pointing it out :)

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Dynamo 3 years ago

you are confused!!!

Change "leftmost" term to "rightmost".

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

LIVE EVENTS

?

Khandkar Saeed Reza 3 years ago

can you tell how to improvise the code to print only the contiguous subsets? like if a=[1,2,3] subsets are (1),(2),(3),(1,2),(2,3),(1,2,3)....the set(1,3) will not be included..anyone ?

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Vatsal Sharma 编辑 2 years ago

```
I am writing for c++
for(int i=0;i<a.size();i++)
for(int j=i;j<a.size();j++)
cout<<j<<endl;
```

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Roshan Choudhary 2 years ago

This is for subsets and not subarrays...(1,3) is a subarray of the array (1,2,3)

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Satyendra Singh 3 years ago

I am new to the programming but these articles are awesome! Can any one tell me if i can download these!

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Shreyas Krishna 3 years ago

is the challenge open in all languages, python too?

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Prateek Garg ⚡ Author 3 years ago

yes, it is .

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Hitesh Kalwani 3 years ago

Very helpful for learning the bit operations , Thanks

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Syed Moinuddin Shibli 3 years ago

owsome

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Vignesh Mahalingam 3 years ago

In "finding the largest power of two"s implementation's 1st comment line: "//changing all left side bits to 1" is it correct?? Explain please.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Prateek Garg ⚡ Author 3 years ago

It should be changing all the "right" side bits to 1. It is fixed.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Shubhankar Dimri 3 years ago

isn't (tilde) 5 = -6? its written 2 here pls clarify

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Prateek Garg ⚡ Author 3 years ago

Actually here it is shown only for limited bits, but in system it flips all the bits(32 bits in this case) including the sign bit, that's why the answer is different in both the case S.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

vidyasagarvuna 3 years ago

Easy to refer the concepts DAA, CD and DLC.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

abhinav vinci 3 years ago

a very good tutorial, learned a lot

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Nitesh Singh 3 years ago

$x \& (-x)$: Returns the rightmost 1 in binary representation of x ? I am not able to understand ?

eg. $12 = (1100)$

$-12 = (0100)$

$(12) \& (-12) = (0100) = 4$

but the rightmost one is at 3rd position

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Prateek Garg ⚡ Author 3 years ago

Here indexing is from 0, so 1 is at 2nd position, which is $2^2 = 4$. It returns the value 2^x , where x is the index of rightmost one.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Naman Ahuja 2 years ago

:P

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



nishant gupta 2 years ago

Second That!!

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Abhinav Jha 3 years ago

Man this is seriously awesome. I am poor with bit manipulation and needed some good starting material. CodeMonk <3

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

As 3 years ago

At problem 5 this: $31 = 2 * 16 - 1$ should be: $31 = 2 * 5 - 1$.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

vinayak kothari 2 years ago

can anyone give the link from where to study bit mask + dynamic programming

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Ajith Panneerselvam 2 years ago

Really good work. Keep it up.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Shubham Aggarwal 2 years ago

Can you put 1s and 2s complement also..!!

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Atul Kumar Gupta 2 years ago

Applaudable work!!!

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

bhargav patel 2 years ago

?

all in one bit manipulation simple and awesome.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Shivendra Pratap Singh 2 years ago

in right shift operator 2^k should be written and replacement should be done by 0(not by 1 as given)

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Braj Mohan Jangid 2 years ago

Nice set of examples.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

ankur aggarwal 2 years ago

Amazing article, thanks Prateek :)

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Ishpreet Singh 编辑 2 years ago

Great Explanation Its was quite useful.....

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Jason Adams 2 years ago

I'm having trouble understanding this:

"(x-1) will have all the bits same as x, except for the rightmost 1 in x and all the bits to the right of the rightmost 1.

Let, $x = 4 = (100)_2$

$x - 1 = 3 = (011)_2$

Let, $x = 6 = (110)_2$

$x - 1 = 5 = (101)_2$

It might not seem obvious with these examples, but binary representation of (x-1) can be obtained by simply flipping all the bits to the right of rightmost 1 in x and also including the rightmost 1."

Which is the rightmost 1 here? is it the last bit? so for 4 the rightmost 1 will be 0, and for 3 it's 1? Also what do you mean by the "right of the rightmost 1?"

Thanks!

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



SAURABH AGARWAL 2 years ago

Hey!

The rightmost 1 means , when we start traversing from right the 1 which encountered first. Like it is the third bit from right in case of 4 and second bit from right in case of 6.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

SAURABH AGARWAL 2 years ago

In First Ques :To check that a given number is a power of 2 or not ?

If the number x is 3 then then binary representation of x-1 on flipping all the bits to the right of rightmost 1 in x and also including the rightmost 1 gives us 0 , which is wrong as the binary representation of 3-1=2 is 10.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接



Abhishek Anand 2 years ago

Just quoting ur words "The rightmost 1 means , when we start traversing from right the 1 which encountered first....."

so if $x=3[0011]$ then $x-1=2[0010]$as the rightmost 1 encountered here is at the 0th posn in 3. HOPE m clear !!

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Palash Bansal 2 years ago

?

Looks like it must be-
return (!x || !(x & (x - 1))); instead of return (x && !(x & (x - 1)));

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Ronald Andrean 2 years ago

Hello

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Rishabh Rathore 2 years ago

can anybody tell me what's the difference between >> and >>> in java?

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Keshav Sharma 2 years ago

while explaining left and right shift

say in left shift , there is a line : " Left shift is equivalent to multiplying the bit pattern with 2^k "

and also in right shift, it says dividing the bit pattern with 2^k
isn't it 2^k ?

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

jithender maroju 编辑 2 years ago

While explaining Right Shift (>>);, it says "Right shift operator is a unary operator which shift the some number of bits, in the given bit pattern, to the right and append 1 at the end"..

But it will not append with 1 , instead it will append with sign bit

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Zhefeng Gao 2 years ago

good!!learn a lot from this article!

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Jivnesh Sandhan 2 years ago

Thanks for contributing nice tutorial.

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Divyansh Gothwal 编辑 a year ago

I want to put one small and simple solution here for problem

5) Find the largest power of 2 (most significant bit in binary form), which is less than or equal to the given number N.

long largest_power(long n)

```
{
while(n&(n-1))
{
n=n&(n-1);
if(!n&(n-1))
{
return n;
}
}
return n;
}
```

What this do is instead of making everything 1 to the right side it reduces everything to right of most significant bit to 0 and final ans (n) will contain the required ans.

i believe this one is easy to understand

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

Immitation 10 months ago

Can ANYone please explain me this :

?

$x \mid (1 \ll n)$: Returns the number x with the n th bit set.

$(1 \ll n)$ will return a number with only n th bit set. So if we OR it with x it will set the n th bit of x .

$x = 10 = (1010)_2$ $n = 2$

$1 \ll n = (0100)_2$

$x \mid (1 \ll n) = (1010)_2 \mid (0100)_2 = (1110)_2$

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

[Jeetesh Saxena](#) 6 months ago

thanks a lot

▲ 0 票数 ● 回复 ● 信息 ● 永久链接

1
LIVE EVENTS


AUTHOR



Prateek Garg

 Student at DIT University

 Dehradun

 7 notes

TRENDING NOTES

[Python Diaries Chapter 3 Map | Filter | For-e](#)
[lse | List Comprehension](#)

written by Divyanshu Bansal

[Bokeh | Interactive Visualization Library | Us](#)
[e Graph with Django Template](#)

written by Prateek Kumar

[Bokeh | Interactive Visualization Library | Gr](#)
[aph Plotting](#)

written by Prateek Kumar

[Python Diaries chapter 2](#)

written by Divyanshu Bansal

[Python Diaries chapter 1](#)

written by Divyanshu Bansal

[more ...](#)

[关于我们](#)

[创新管理](#)

[能力评估](#)

[大学编程](#)

[维基开发者](#)

[博客](#)

[新闻](#)

[职业](#)

[联系我们](#)



网站语言： [中文](#) | [条款](#) | [隐私](#) | © 2018 HackerEarth

?