

DYNAMIC PROGRAMMING

Day 5

Youtube link:

Part 1: <https://www.youtube.com/watch?v=95o1IWeV7fY>

Part 2: <https://www.youtube.com/watch?v=8HOlj0RgcCU>

Contents:

1. Bitwise operations
2. Bitmasks
3. DP with bit masking

Bitwise Operations

Binary representation of numbers

`int x=9;`

Binary representation of 9 = 1001

Because $9 = 2^3 + 2^0$

We know, int is 32-bit .

9 = ...000000001001

`int x=7;`

`x = 0000000....00111`

Logical Operations

`&&` - AND

`||` - OR

`!` - NOT

These operations operate on boolean variables

```
if ( true && false ) {
```

```

....
}
if( 4 && 7) {
// Computer will first convert 4 & 7 to boolean (true/false) and
then it will operate
// 4 - true (All non-zero values are treated as true)
// 7 - true (All non-zero values are treated as true)
}
Similar for || & NOT

```

Bitwise Operations

- Bit-per-bit calculations

& operator (Bitwise AND operation)

Eg

A = 12 = (1100) [in binary]

B = 25 = (11001) [in binary]

$$\begin{array}{r}
 01100 \\
 \& 11001 \\
 \hline
 01000
 \end{array}$$

A&B = 8

| operator (Bitwise OR operation)

Eg

A = 12 = (1100) [in binary]

$B = 25 = (11001)$ [in binary]

$$\begin{array}{r} 01100 \\ 11\triangle 01 \\ \hline 11101 \end{array}$$

Decimal equivalent of $A \mid B$ is 29

\wedge operator (Bitwise XOR operation)

- Same bit (both 0 or both 1), then it will result in a 0 bit
- Different bit will result in a 1 bit ($0 \wedge 1 = 1$)

Eg

$A = 12 = (1100)$ [in binary]

$B = 25 = (11001)$ [in binary]

$$\begin{array}{r}
 01100 \\
 \wedge 11001 \\
 \hline
 10101
 \end{array}$$

$$A \wedge B = 21$$

$$A \wedge 0 = A$$

$$A \wedge A = 0$$

$$A \wedge (B \wedge C) = (A \wedge B) \wedge C$$

Try this problem -

<https://www.hackerrank.com/challenges/lonely-integer/problem>

~ operator (Bitwise NOT operation or 1's complement)

- Replace 0 by 1 and 1 by 0

Eg

$$B = 25 = (11001) \text{ [in binary]}$$

$$\begin{array}{r}
 0011001 \\
 \sim \hline
 1100110
 \end{array}$$

Note: Negative integers are represented in memory as their 2's complement form

i.e. $-x = 2$'s complement of x

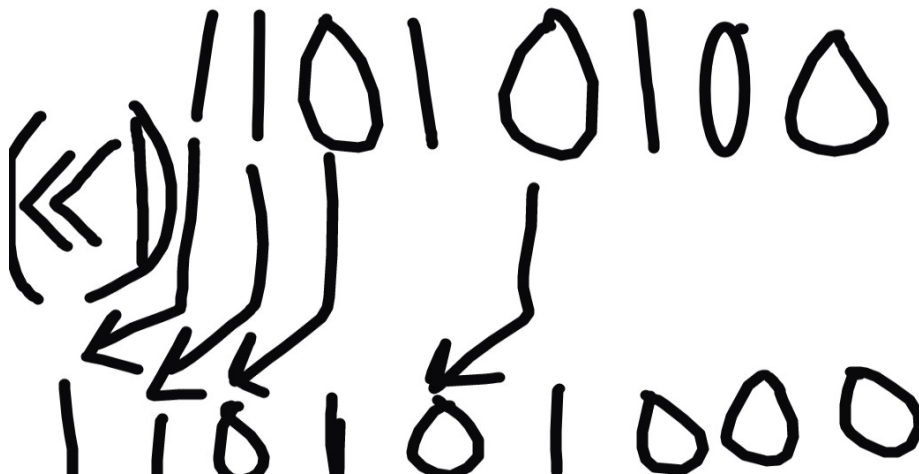
2's complement = 1's complement + 1

$-x = \sim x + 1 \Rightarrow \sim x = -(x+1)$

Eg. $\sim 25 = -26 = -25 - 1 = -(25+1)$

<< operator (Left shift operation)

$212 = (11010100)$



$212 \ll 1 = 424 = 212 * 2$

$212 \ll 2 = 848 = 212 * 4$

$x \ll b = ?$ (Left-shift number x by b bits)

Important property:

$x \ll b = x * (2^b)$

Preventing Overflows in Left Shift (Very Important)

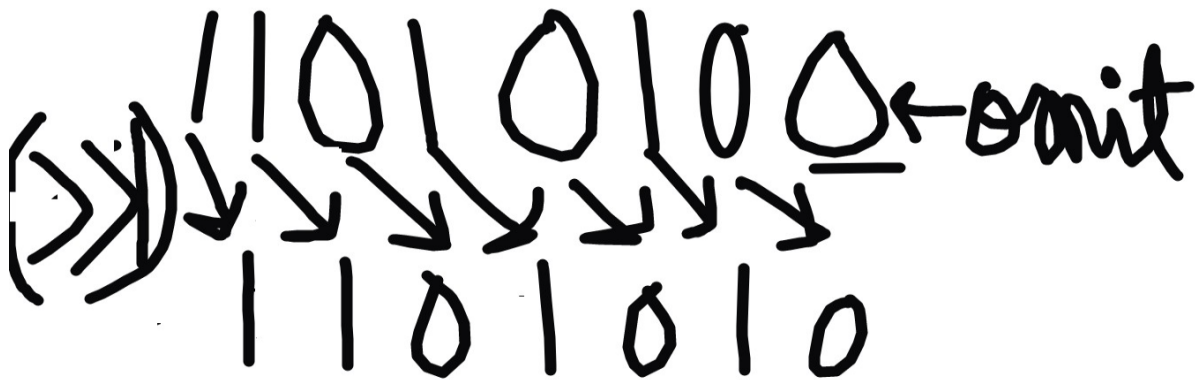
To prevent-overflows, use LL with constants
(to convert to long long)

Eg. $(1LL \ll 40)$ is correct

But $(1 \ll 40)$ is wrong (overflow)

>> operator (Right shift operation)

212 = (11010100)



$212 \gg 1 = (1101010) = 106 = 212/2$

$212 \gg 2 = 53 = 212 / 4$

$x \gg b = ?$ (Right-shift number x by b bits)

Important property:

$x \gg b = \text{floor}(x / (2^b))$

Important property: All bitwise operations are performed in $O(1)$

Applications of bitwise operations (Very very important)

1. Check whether x^{th} bit (from the right) of a number n is set (1) or not (0) ?

```
if ( n & ( 1<<x ) ) {
```

```
    //  $x^{\text{th}}$  bit is set
```

```
} else {
```

```
    //  $x^{\text{th}}$  bit is not set
```

```
}
```

$n =$ 01110011

$1 \ll x = 10000\dots 0$ (1 is at only the x-th position)

2. Set x^{th} bit (from the right) of a number n

$$n = n | (1 \ll x)$$

$1 \ll x = 10000\dots 0$ (1 is at only the x-th position)

3. Toggle x^{th} bit (from the right) of a number n
(Toggle means 0 becomes 1 and 1 becomes 0)

$$n = n \wedge (1 \ll x)$$

$1 \ll x = 10000\dots 0$ (1 is at only the x-th position)

4. Find LSB of n. Using this, also unset the LSB of n.
($O(1)$)

LSB = Lowest Set Bit



11010

↑

LSB 10

$$\text{LSB} = n \& (-n)$$

$$\begin{array}{r}
 n: 11010 \\
 \&(-n) 00110 \\
 \hline
 11000
 \end{array}$$

To unset LSB , subtract LSB

Method 1:

$$n = n - (n \&(-n))$$

Method 2:

$$n = n \wedge (n \&(-n))$$

5. Given number n, check whether it is a power of 2

Eg. n = 6, not a power of 2

n = 8, is a power of 2 [$8 = 2^3$]

$$\begin{array}{r}
 n: 10000 \\
 n-1: 01111 \\
 \hline
 10000
 \end{array}$$


```

if ( n & (n-1) ) {
// n is not a power of 2
} else if (n>0){
// n is a power of 2
}

```

// Corner case: When n=0, treat separately

Try this problem:

<https://www.hackerrank.com/challenges/counter-game/problem>

7. Given 2 equations on a & b. Given values of n & m. Find values of a & b that satisfy it: [Linkedin coding round 2020]

$$a + b = m$$

$$a \wedge b = n$$

$$1 \leq a, b, n, m \leq 10^9$$

$$a \leq b$$

(Link to a similar problem (Only constraints are different):

<https://codeism.contest.codeforces.com/group/qXv2tukHZE/contest/309847/problem/A>)

Solution

From half-adder (or binary addition)

$$S = a \wedge b$$

$$C = a \& b$$

$$a + b = a \wedge b + ((a \& b) \ll 1)$$

$$(a + b) - (a \wedge b) = ((a \& b) * 2)$$

$$a \& b = \text{floor} (((a + b) - (a \wedge b)) / 2)$$

$$\text{Let } k = a \& b = \text{floor} ((m - n) / 2)$$

Iterate through all bits and check the bit value of n & k,

$$\begin{array}{r}
 a \& b: 1010 \\
 a \wedge b: 1001 \\
 \hline
 a \cdot b: 10010
 \end{array}$$

HW questions

8. Find addition of 2 numbers using only bit-wise operations.

<https://leetcode.com/problems/sum-of-two-integers/>

9. Using the above, find multiplication of 2 numbers using only bit-wise operations

(Similar thing was discussed in number theory - think of something like iterative version of binary exponentiation)

10. Unset x^{th} bit of a number n ,

$$n = n \& \sim(1 \ll x)$$

Bitmasks

- Representation of sets (sets in Set theory) using an integer
- If x^{th} bit is set (1) in the integer , that element x is present in the set
- If x^{th} bit is unset (0) in the integer , that element x is not present in the set
- There are 2^n subsets of a set of n elements

$S = \{0, 1, 3, 5, 7\}$
10101011

- Finding the number of set bits (1-bits) in any integer x
(in $O(1)$)

`cout<<__builtin_popcount(x);`

Eg.

```
cout<<__builtin_popcount(9);  
// "1001" has 2 set bits
```

Now, try to solve this problem:

<https://codeforces.com/problemset/problem/1097/B>

Observations:

- Order of rotations doesn't matter
- We need to consider all the possible combinations of clockwise and anti-clockwise rotations
- We need to iterate through all the subset of a set that has n elements (n rotations)

Let us represent this set of rotations as bitmask

```
int vec[n];
```

```

for (int mask = 0; mask < (1<<n); mask++)
{
    // 1 - clockwise
    // 0 - anticlockwise
    // iterate through all bits of this number
    mask
    int sum=0; // final value of rotation
    (Assume clockwise - +ve)
    for( int i =0; i<n; i++)
    {
        if ( mask & (1<<i) )
        {
            sum=sum+vec[i];
        }
        else {
            sum = sum - vec[i];
        }
    }
    if (sum%360 == 0)
    {
        cout<<"YES";
        return 0;
    }
}

cout<<"NO";
return 0;

```

**Q. Given x, y, find whether x is a submask (subset) of y.
(in O(1))**

Eg.

If x = 1101

y = 1111

Then x is a submask (subset) of y

If x = 01101

y = 01110

Then x is not a submask(subset) of y

Method 1

```
if (x & y == x)
{
    // x is a subset of y
}
```

Method 2

```
if (x | y == y)
{
    // x is a subset of y
}
```

Q. Given any number n, iterate through all the submasks of n. (in decreasing order)

Eg. n=1010

1010, 1000, 0010, 0000

(1010 - 1 = 1001)

(1001 & 1010 = 1000)

(1000 -1 = 0111)

(0111 & 1010 = 0010)

n = 11011

11011, 11010, 11000, 10011, 10010, 10001.....

- Subtract 1 from each sub mask
- And then take bitwise and (&) with the original mask

```
for( int x=n; x>0; x = (x-1) & n)
{
```

```

    cout<<x<<'\\n';
}
cout<<0<<'\\n'; // treat 0 separately

```

Time complexity : $O(2^{\text{set_bits}})$

Q. Given any set of n elements, iterate through all the subsets (submasks) of all the subsets (submasks) of size n
 Given set can be written as 11111...1 (1's n times)

```

for( int i=0; i< (1<<n) ; i++)
{
    for( int x=i; x>0; x = (x-1) & i)
    {
        cout<<x<<'\\n';
    }
    cout<<0<<'\\n'; // treat 0 separately
}

```

Time complexity : $O(3^n)$

Proof: Let k =number of set bits

$$\begin{aligned}
 T(n) &= \sum^n C_k 2^k \\
 &= \sum^n C_k 2^k \\
 &= (1+2)^n = 3^n
 \end{aligned}$$

DP with bitmasking

- If some state of DP problem can be represented as a subset or a set, then we use DP with bitmasking

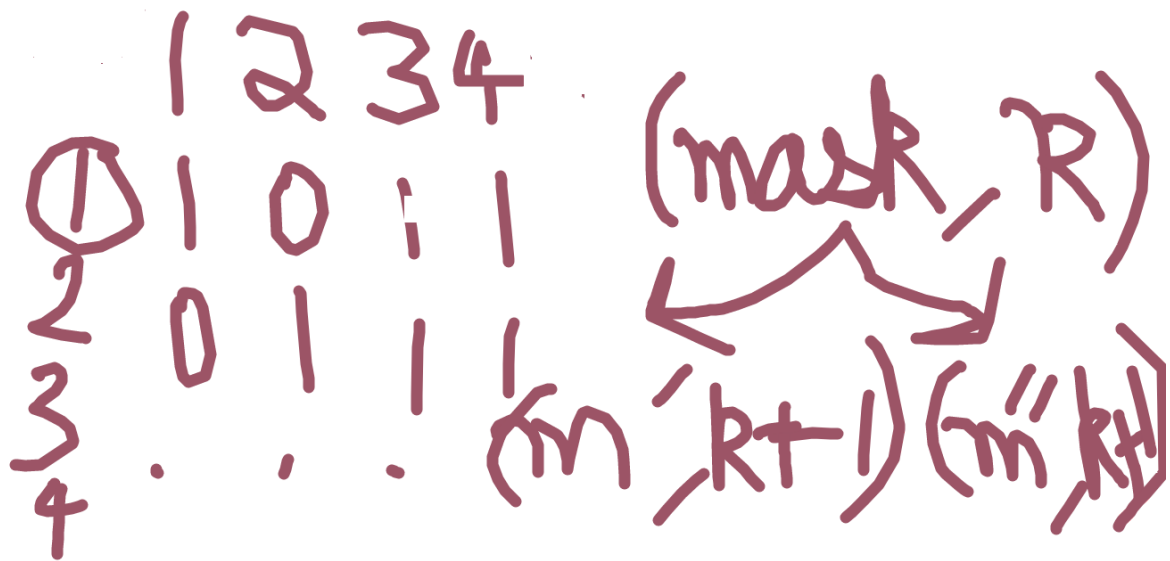
Problem 1: <https://www.spoj.com/problems/ASSIGN/>

- In questions involving bitmasks, the constraints are generally very low. ($n \leq 20$ or 25)
- Consider a set of n topics .

Let us consider in a ,mask, if i^{th} bit is 1 represents that i^{th} student has been given a subject.

If i^{th} bit is 0, then that student has not taken any subject yet.

Let k = smallest index of topic left to be distributed



```
int dp [ 1<<n ][ n ] // initially each value -1
int solve(int mask, int k)
{
```

```

int ans=0;
if (k == n) {
    return 1;
}
if (dp[mask][k]!=-1)
    return dp[mask][k];
for(i=0; i<n; i++)
{
    if(a[i][k]==1 && !(mask & (1<<i)) )
    {
        ans = ans + solve (mask | (1<<i), k+1 )
    }
}
return dp[mask][k]=ans;
}

// in main
solve(0, 0)

```

Time complexity: $O(n^2 \cdot 2^n)$ or $O(n \cdot 2^n)$

This solution will give MLE / TLE

We can reduce the state for k as follows:

```

int dp [ 1<<n ]
int solve(int mask)
{
    int k = __builtin_popcount(mask);
    // Now, k = set bits in mask calculated in O(1)
    // Effectively , k = index of topic to be assigned
    int ans=0;

```



```

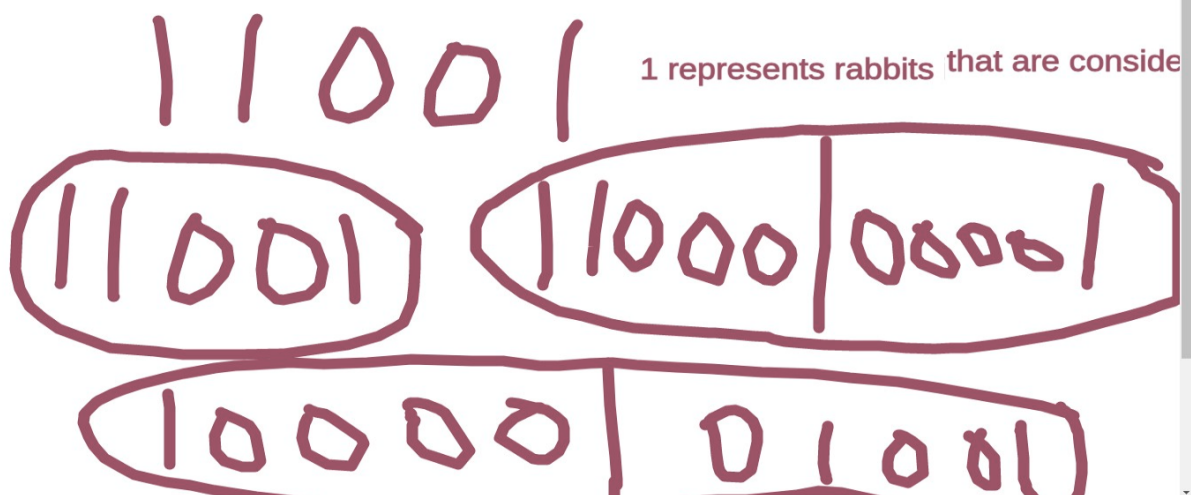
if (k == n) {
    return 1;
}
if (dp[mask] != -1)
    return dp[mask];
for(i=0; i<n; i++)
{
    if(a[i][k]==1 && !(mask & (1<<i)) )
    {
        ans = ans + solve (mask | (1<<i))
    }
}
return dp[mask]=ans;
}
// In main,
solve(0)

```

Time complexity: $O(2^n * n)$

PROBLEM Atcoder DP contest - U Grouping

https://atcoder.jp/contests/dp/tasks/dp_u



```

dp[1<<n] ; // initialize with -INF or -1e18

int solve (int mask)
{
    if ( dp[mask] != -INF)
    {
        return dp[mask];
    }
    // Case 1: All the set bits are in same group
    int ans=0;
    for(int i=0; i<n; i++)
    {
        for(int j=i+1; j<n; j++)
        {
            if ( (mask & (1<<i)) && (mask & (1<<j)) )
                ans += a[i][j];
        }
    }
    // Case 2: iterate through all submasks of given mask
    for( int i = mask&(mask-1); i>0; i=(i-1) & mask)
    {
        ans = max(ans, solve(i) + solve( mask ^ i ) );
    }
    return dp[mask]=ans;
}

```

// dp[mask] = maximum cost to build any possible grouping of rabbits which have 1 (set bit) in mask

Eg. dp[1001] = maximum cost to build any grouping of rabbits 0 and rabbit 3

```
// in int main()
cout<<solve( (1<<n) - 1 );
// because (1<<n) - 1 = 1111111....111 ( n times 1 )
```

HW questions

1. <https://www.hackerearth.com/practice/algorithms/dynamic-programming/bit-masking/practice-problems/algorithm/mehta-and-the-tricky-triplets/>
2. <https://leetcode.com/problems/number-of-ways-to-wear-different-hats-to-each-other/>
3. (For those who have studied graphs)
<https://www.hackerrank.com/challenges/synchronous-shopping>

Also, try long challenge problems here:

<https://codeism.contest.codeforces.com/>

Another trick for some number theory + bitmask problems

Any number n can be represented as a mask of its prime divisors.

Prime numbers = $\{2, 3, 5, 7, 11\}$

Eg. $n=6$, $6=2*3$, $n = 001100$

Sometimes, this helps to calculate GCD, LCM of 2 numbers effectively using bitwise $\&$ and bitwise $|$ operations respectively.

If you want to iterate through all the prime divisors of n , just iterate through all the set bits of n

If you want to iterate through all the divisors of n , just iterate through all the submasks of n