if $a \equiv b \bmod m$, $m | (a-b)$

Sub :

Day
Time : Date : / /

Modular Arithmatic

* $x \bmod y = x$ if $x < y$

* $x \bmod y = a$ $x > y$, a is remainder

finding the value of a negative whole Number in a given modulo.

→ The value of a negative number (a > b) in a given modulo is obtained by adding the modulus to the number until we get a positive number.

$-7 \bmod 2 = 1$

$-6 \bmod 4 = 2$

$-2 \bmod 3 = 1$

$-7+2 = -5$
$-5+2 = -3$
$-3+2 = -1$
$-1+2 = 1$

if $a < b$

$-a \bmod b = (b-a)$

$-13 \bmod 17 = (17-13) = 4$

Sub :

Time : Date : / /

Addition

$a + b \bmod y = (a+b) \bmod y$

$7+13 \bmod 4 = (7+3) \bmod 4$

Substraction

~~$a+b=$~~

$a - b \bmod y = (a-b) \bmod y$

$8-2 \bmod 3 = (8-2) \bmod 3 = 0$

$2-5 \bmod 9 = (2-5) \bmod 9 = -3 \bmod 9 = 6$

Multiplication

$a \times b \bmod y = (a \times b) \bmod y$.

\# $a \equiv (a \% m) \bmod m$

$b \equiv (b \% m) \bmod m$

$a+b \equiv (a \% m) + (b \% m) \bmod m$

$\equiv \{(a \% m) + (b \% m)\} \% m \pmod m$

$ab = (a \% m) \times (b \% m) \% m \pmod m$

Sub :

Day

Time : Date : / /

Tables In ~~Modulo Arithmetic~~.

Congruent

if $a \equiv b \bmod m$.

then $a \bmod m = b \bmod m$

Theorem-1

$\Rightarrow$ $a \equiv b \pmod m$ then $(a-b)$ should also be divisible by m.

$\Rightarrow$ $223 \equiv 103 \bmod m$. , $m = ?$

— Step-1 $223 \equiv 103 \bmod m$.

@ $a = 223$, $b = 103$.

$a - b = 223 - 103$.

$= 120$.

$\therefore$ m should be the any division of $a-b$.

G.C.D $= (a, b) = (-a, b) = (a, -b) = (-a, -b) = (|a|, |b|)$

Sub: Day: Time: Date: / /

$a \equiv (a \% m) \bmod m$.

Commutative law

$(a+b) \bmod m = (a \bmod m + b \bmod m) \bmod m$

$a-b \bmod m = (a \bmod m - b \bmod m) \bmod m$

Associative law:

$(a, b) \bmod m = (a \bmod m \times b \bmod m) \bmod m$

Theorem: 2 : If 'a' and 'b' is an integer and $(a, b) = d$ (a G.C.D), then $(a/d, b/d) = 1$

; if $(a, b) = 1$, $(a/d, b/d) = 1$

; if $a+b = c$, $a \pmod n + b \pmod n \equiv c \pmod n$

$a \times b = a$, $a \pmod n \times b \pmod n \equiv c \pmod n$

if $k \times a = k \times b \pmod c$, $\rightarrow a \equiv b \pmod{\frac{c}{gcd(k,c)}}$

Sub :

Day

Time : Date : / /

G.C.D $\left(\frac{30}{6}, \frac{78}{6}\right) = ?$

gcd of $(30, 78) = 6$

theorem - 2 $a_1 = 30$, $b = 78$

$d = 6$.

$\therefore \left(\frac{30}{6}, \frac{78}{6}\right) = \underline{1}$.

Some formula

1. $a \pm k \equiv b \pm k \pmod c$
2. $ak \equiv bk \pmod c$
3. $a^k = b^k \pmod c$ $k > 0$
4. $-a \equiv -b \pmod c$.

If $a_1 \equiv b_1 \pmod c$ $a_2 \equiv b_2 \pmod c$

1. $a_1 + a_2 \equiv b_1 + b_2 \pmod c$
2. $a_1 - a_2 \equiv b_1 - b_2 \pmod c$
3. $a_1 \times a_2 \equiv b_1 \times b_2 \pmod c$

If $a \equiv b \pmod n \rightarrow b \equiv a \pmod n$

$a \equiv b \pmod n$, $b \equiv c \pmod n$, $\rightarrow$

$a \equiv c \pmod n$

Sub :

Day

Time : Date : / /

Diophantine Equation

$$ax + by = c$$

$$x = x_0 + \left(\frac{b}{d}\right)t \qquad y = y_0 - \left(\frac{a}{d}\right)t$$

$$d = GCD(a, b)$$

$$1575x + 231y = 21$$

$a = 1575$, $b = 231$, $c = 21$

step 1: $GCD(1575, 231) = 21$

$$1575 = 6 \times 231 + 189$$

$$231 = 1 \times 189 + 42$$

$$189 = 4 * 42 + 21$$

$$42 = 2 \times 21 + 0$$

Sub :

Day

Time : Date : / /

Inverse Mod

* Inverse Modulus of a (mod b) is denoted by a^{-1} (mod b)

* Inverse Modulus exist if and only if
  ① GCD (a, b) = 1.

* $4^{-1} \bmod 9 = ?$

Step 1: GCD (4, 9) = 1.

Inverse mod exists.

$ax + by = 1$

Step 2: $1 = 9 - 2 \times 4$.

$= 4 \times (-2) + 9 \times (1)$.

Step - 3 $4^{-1} \pmod 9 = (-2) \bmod 9$.

$= 7$.

Sub :

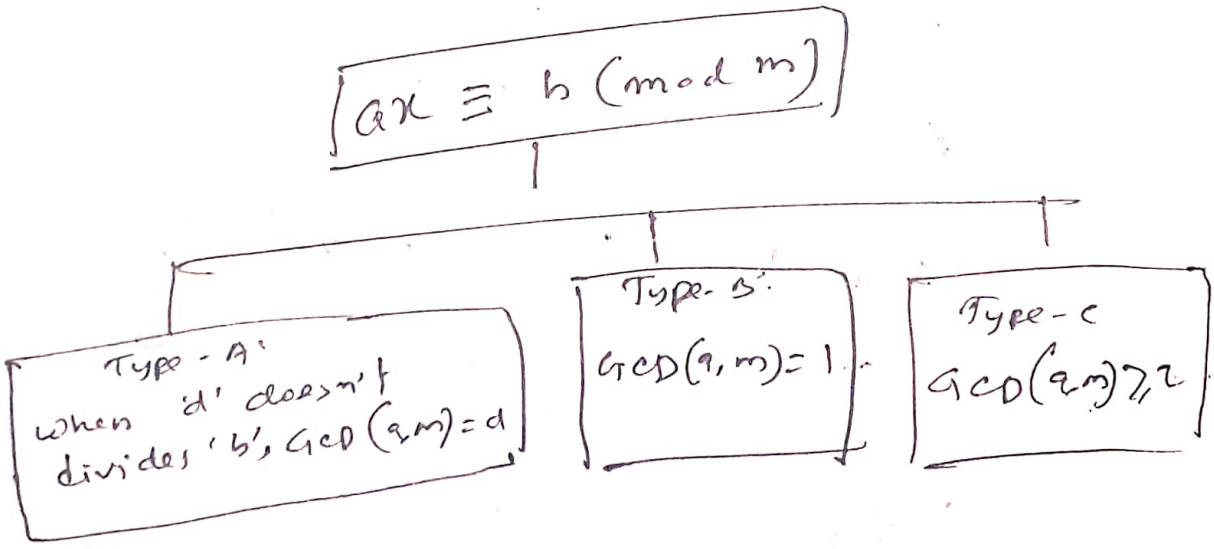
Day

Time : Date : / /

Linear congruence

$$ax \equiv b \pmod{m}$$

The linear congruence $ax \equiv b \pmod{m}$ has a solution if and only if $d \mid b$, where $d = (a, m)$



Sub :

Day

Time : Date : / /

Type A :

$14x \equiv 7 \mod 28$

$ax = b \pmod m$

$a = 14, \quad b = 7, \quad m = 28$

$\gcd(14, 28) = 18$

$d \mid b = ?$

$14 \mid 7 = X.$

7 is not divisable by 14

So, solution is doesn't exist.

Sub:

Day
Time: Date: / /

Prime factorization

Brute force (O(n))

```
void PrimFact (int N)
{
    for (int i=2; i<=N; i++)
        if (N % i == 0)
        { int cnt = 0;
          while (N % i == 0)
              cnt++;
              N /= i;
          cout << i << " " << cnt << endl;
        }
}
```

optimized: (O(sqrt(n)))

```
for (int i = 2; i*i <= N; i++)
{
}
if (N > 1)
  cout << N << " " << 1 << endl;
```

Sub :

Day

Time : Date : / /

Prefix Sum

⇒ sum of subarray of an array.

⇒ "for" loop is not efficient in worst case

⇒ We use precomputed value and store it in another array.

```
int P[n-1]
P[0] = 0
for (i=1, i<=n; i++)
    P[i] = P[i-1] + a[i];
```

⇒ Now, if we want to sum of a range (l, r)

① Sum of l to r will be = $P[r] - P[l-1]$

② Multiple l to r " " = $\frac{P[r]}{P[l-1]}$

③ Bitwise X-or " " = $P_r \oplus P_{l-1}$

Sub :

Dm

Time : Date : / /

Range ADD

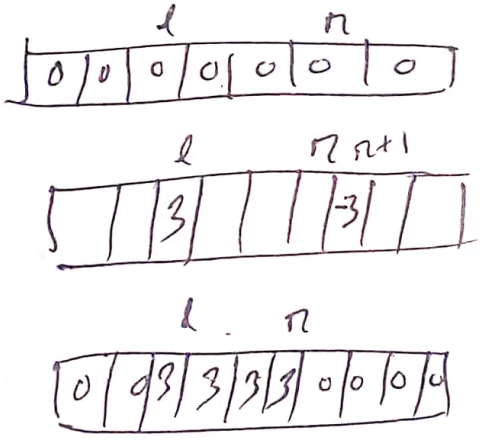
A query (l, r, v) will be given.

An array $\{0\}$ will be given. task is to assign the value of v in this range (l, r)

$$a_l = a_l + v$$

$$a_{r+1} = a_{r+1} - v$$

$\Rightarrow$ prefix sum of this array.



Sub :

Day

Time : Date : / /

Number Theory

1. divisibility:

→ divisible.

if $a \mid b$ and $b \mid c$, then $a \mid c$

if $a \mid b$ and $a \mid c$, $a \mid (b+c)$

$\Rightarrow a \mid (xb+yc)$

Sub :

Day

Time : Date : / /

Number of Divisor

$a \mid b \iff b = ka.$

```
for (i = 1 to √n)
    if (n % i == 0)
        i is a divisor
        n/i  "  "
```

∴ divisor (n) $\leq 2\sqrt{n}$.

Lowest prime factor

```
Sieve (int a, int b)
{
    is_composite[0] = is_composite[1] = 0;
    for (i = 2; i <= n; i++)  lowest_prime[i] = i;
    for (int i = 2; i*i <= n; i++)
    {
        if (is_composite[i] == false)
            for (int j = i*i; j <= n; j += i)
                if (is_composite[j] == false)
                {
                    is_composite[j] = true;
                    lowest_prime[j] = i;
                }
    }
    print " Lowest prime [i];
}
```

MD Sadman Hafiz

Sub :

Time : Date : / /

Number Theory Algorithm

Sieve and its variation

1. Sieve

problem statement:

- Input n, find All prime between 1 – n

short detail and approach:

- Take a visited array for marking
- Take the lowest prime and mark all of it's com multiple as composite
- continue the process till sqrt(n)
- unmarked are prime

implementation:

```
# define N 150
bool is composite[N];
vector<int> prime;

void sieve(int n)
{   for (i=2; i<n; i++)
    {   if (!is composite[i]) prime.push_back(i);
        for (int j = i*i; j<=n; j+=i)
            {   is composite[j] = true;
            }
    }
}
```

Sub :

Day

Time : Date : / /

Complexity :

Time:

Worst case : $O(N \log \log N)$

Average case : $O(N \log \log N)$

Best " : $O(N \log \log N)$

space: $O(N)$.

Application:

- Factorize a Number N
- Find Prime in Range $(N \to M)$
- Goldbach's conjecture.

Sub :

Day

Time : Date : / /

Linear Sieve

problem statement:

- optimize Sieve, , (linear)

Approach:

- As we see in sieve, in the inner loop, we cross out a number in multiple times.
- We can pick and mark these only once.
- Every ~~prim~~ composite number has at least one prime factor, we should take the smallest one. let, composite $a = i * p$ → smallest prime
- As p is the smallest, no prime less than p can divide i [i >= p]
- when we loop for every i, all primes not exceeding i is already recorded in the vector 'prime'
- So, if we only loop for all elements in prime, in the inner loop, break when the element divides i, we can pick composite exactly one.

prime[ ]

Sub :

<u>implementation:</u>

```
void sieve (int n)
{    for (i = 2; i < n; i++)
     {  if (! is composite[i]) prime.pushback(i)

        for (j = 0; j < prime.size() && i*prime[j] < n;
                                                j++)
     {
        is composite [i * prime [j]] = True;
            if (i % prime [j] == 0)
               break;
     }
   }
}
```

lp[1000] = {0}
if (lp[i] == 0) lp[i] = i minimum prime factor
We can store i also

lp[i * prime[j]] = pr[j];
if (pr[j] == lp[i])
break

<u>Example:</u>

(i = 10, p = 2 হলে
i % p == 0 | এর মানে হলো 10 কে 2 দিয়ে ভাগ করা যায়, তাহলে 10 এর multiple কেও 2 দিয়ে ভাগ করা যাবে

if i = 10, it means, all the prime less than or equal 10 has already in the vector, we can mark out 10 * 2, 10 and break. ~~Then 10 * 3, 10 * 5 break. 10 * 7~~

i >= p and i % prime[j] means Number from 2 - 10 are already checked and rest will be checked next.

Sub :

Day

Time : Date : / /

#complexity :

T : O(N)

Use :

- We can factorize any number only this in O(N)
-

Sub :

Day

Time : Date : / /

Segmented Sieve

problem Statement:

- Given a Rang $1 \leq R \leq R \leq 10^{10}$, $R-L <= 10^5$
- Find All prime between R, L.

Approach:

- By siene, we can only get upto 10^8 (global array)
- We'll generate sqrt(10^{10}) = 10^5 primes.
- Take one prime and cross out its multiples within that range
- We need to find the first composite within that range for particular prime no.
- ~~But if~~ base = curPrime * curPrime
- if base < L, We will want some operation
- in that case

```
if(base < L)
    base = ((L+curPrime-1)/curPrime) * curPrime
```

Sub :

Day

Time : Date : / /

Implementation:

```
Void seg sive ( int L, int R)
/* generate prime sqrt of (R) */ prime[]
bool is composite [R-L+1]

if (L == 1)
    is Composite [0] = true;

for (int j = 0; prime[j] * prime[j] <= R; i++)
{
        ll now = prime[j]
        base = prime[j] * prime[j];
    if (base < L)
      { base = ((L+ now -1)/now) * now;
      }
  for ( ll j = base; j <= R ; j += now )
          { is composite [j - L] = True;
}       }

for (ll i = 0; i <= R-L ; i++)
    { if ( ! is composite [i] )
            cout << L+i << endl;
    }
}
```

Sub :

Day

Time : Date : / /

→ ~~N log (log N)~~ ?

complexity : O (sqrt(R)) ?

Time

Space O(R-L+1)

Sub :

Day

Time : Date : / /

Bitwise Sieve

Problem: Reduce the memory required.

Approach: We use boolean flag for sieve, but
- instead we can use bitwise flag.
- as integer has 4 bytes. = 32 bit. We can use 32 bit as 32 marker / flag.

implementation:

```
int prime[10000]
int status[10000/32+1]
bool check(int index, int bitNo)
{   bool x;
    return x = status[index] & (1<<bitNo);
}
void set(int indx, int bitNo)
{   status[index] = status[index] | (1<<bitNo);
}

void sieve()
{   int i, j, sqrtN;
    sqrtN = (int)(sqrt(N))
    for (i = 3; i<=sqrtN ; i+=2)
    {   if (check(i/32, i%32) = false)
        {   for (j = i*i ; j <= N ; j += 2*i)
            {   set(j/32, j%32)
            }
        }
    }
```

Sub :

Date :

Time :

Date : / /

complexity :

Time - $O(n \log n)$

Space - $O(n)$

Sub :

Day

Time : Date : 7 / 7

Kth Prime : (Using Sieve)

```
#include <bits/stdc++.h>
using namespace std;
vector<int> primes;   // global.

bool arr[100000001];

void Sieve()
{
    int maxN = 100000000;
    arr[0] = arr[1] = true;
    for (int i = 2; i*i < maxN; i++)
    {
        if (!arr[i])
        {
            for (int j = i*i; j <= maxN; j += i)
                arr[j] = true;
        }
    }
    for (int i = 2; i <= maxN; i++)
        if (!arr[i])
            primes.push_back(i);
}
int main()
{
    int n;
    cin >> n;
    cout << primes[n-1];
}
```

Euler Totient function

- Problem statement:

- Euler Totient function $\phi(n)$ for an input n is the count of numbers in $\{1, 2, \dots n-1\}$ that are relatively prime to n. ($GCD = 1$)

যেমন $\phi(n) = n \frac{p_1 - 1}{p_1} \times \frac{p_2 - 1}{p_2} \dots \frac{p_k - 1}{p_k}$.

$N = p_1^{a_1} \times p_2^{a_2} \dots p_k^{a_k}$

Proof:

(i) Base case: $\phi(1) = 1$. → All are coprime.

(ii) when n is prime $\phi(p) = \phi(p) = p - 1$

(iii) n is power of prime $\phi(p^k)$, $k > 1$

then there are exactly p^k/p number between 1 and p^k that are divisible by p.

$\therefore \phi(p^k) = p^k - p^{k-1}$ → (p^k/p)

[p^k যেগুলো p এর multiple হবে সেগুলো বাদ]

Sub :

Day

Time : Date : / /

(iv) if a, b are relatively prime,

$$\phi(ab) = \phi(a) \cdot \phi(b)$$

$$= (a-1)(b-1)$$

else,

$$\phi(ab) = \phi(a) \cdot \phi(b) \cdot \frac{d}{\phi(d)}$$

$d = g.c.d(a, b)$.

Thus, using 1st three properties, we can compute, first, n can be written as

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_k}$$

[p_i prime factors of n]

$$\phi(n) = \phi(p_1^{a_1}) \cdot \phi(p_2^{a_2}) \cdots \phi(p_k^{a_k})$$

$$= (p_1^{a_1} - p_1^{a-1}) \cdot (p_2^{a_2} - p_2^{a_2-1}) \cdots (p_k^{a_k} - p_k^{a_k-1})$$

$$= p_1^{a_1}\left(1 - \frac{1}{p_1}\right) \cdot p_2^{a_2}\left(1 - \frac{1}{p_2}\right) \cdots p_k^{a_k}\left(1 - \frac{1}{p_k}\right)$$

$$= n$$

$$= p_1^{a_1} \cdot p_2^{a_2} \cdots p_k^{a_k} \left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_k}\right)$$

$$= n\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_k}\right)$$

Sub :

Page
Time : Date : / /

implementation:

```
int phi (int n)
{
    int result = n;
    for (int p = 2; p*p <= n; p++)
    {
        if (n % p == 0)
        {
            while (n % p == 0)
                n /= p;
            result -= result/p;
        }
    }
    if (n > 1)
        result -= result/n;
return result;
}
```

$n\left(1-\frac{1}{p_1}\right)$

Sub :

Day

Time Date : / /

Use :

- Sum of values of totient functions of all divisors of n is equals to n.

$$\sum_{d/n} \phi(d) = n.$$

Example : $n = 6$

factors = {1, 2, 3, 6}

$$n = \phi(1) + \phi(2) + \phi(3) + \phi(6)$$
$$= 1 + 1 + 2 + 2$$
$$= 6.$$

- The most important use :

if n, a are coprime

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

if n is prime

$$\phi(n) = n - 1$$

$$a^{n-1} = 1 \pmod{n}$$

∴ This is Fermat's little theorem

Sub :

Day

Time : Date : / /

Euler Totient from 1 to n.

```
void phi_1_to_n (int n)
{
    vector phi(n+1)
    for (int i = 0; i <= n; i++)
    {
        phi[i] = i;
    for (int i = 2; i <= n; i++)
    {   if (phi[i] == i)
        {  for (int j = i; j <= n; j += i)
            phi[j] -= phi[j]/i;
        }
    }
}
```

Day
Time : Date : / /

Sub :

Divisor Sum and Divisor count

Number of divisor

problem :

- given n, compute the number of divisors $d(n)$ and sum of divisors $\sigma(n)$.

Approach :

- We can decompose every composite number for in prime factor

$$60 = 2^2 \cdot 3 \cdot 5$$

- If a prime factor p appears e times, in prime factorization, then we can use the factor p upto e times in the subset.
- that means $(e+1)$ choices.

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdots p_k^{e_k}$$

$$d(n) = (e_1+1) \cdot (e_2+1) \cdots (e_k+1)$$

some facts :

Sub :

Day

Time : Date : / /

(i) if there is only one distinct prime divisor $n = p_1^{e_1}$, then there are obviously e_1+1 divisors $(1, p_1, p_1^2, \dots p_1^{e_1})$

(ii) If there are two distinct prime divisors $n = p_1^{e_1} \cdot p_2^{e_2}$, then we you can arrange a tabular.

1	1	p_2	p_2^2	$\dots$ $p_2^{e_2}$
p_1	p_1	$p_1 \cdot p_2$	$p_1 \cdot p_2^2$	$\dots$ $p_1 \cdot p_2^{e_2}$
p_1^2	p_1^2	$p_1^2 \cdot p_2$	$p_1^2 \cdot p_2^2$	$\dots$ $p_1^2 \cdot p_2^{e_2}$
$\vdots$				
$p_1^{e_1}$	$p_1^{e_1}$	$p_1^{e_1} \cdot p_2$	$p_1^{e_1} \cdot p_2^2$	$\dots$ $p_1^{e_1} \cdot p_2^{e_2}$

So, the number of divisors is $(e_1+1) \cdot (e_2+1)$

→ if more than two, same type of argument.

$\therefore \quad d(n) = (e_1+1) \cdot (e_2+1) \dots (e_k+1)$.

Sub :

Day

Time : Date : / /

Implementation:

```
int count_divisor (int n)
{
    sieve(n);
    int divisor = 1;
    for (int i = 0; i < prime.size(); i++)
    {
        if (n%. prime[i] == 0)
        {
            int cnt = 1;
            while (n%. prime[i] == 0)
            {
                n /= prime[i]
                cnt++;
            }
            divisor *= cnt;
        }
    }
    if (n>1)
        divisor *= 2;
    return divisor;
}
```

Sub :

Day

Time : Date : / /

Sum of divisor

problem: count the sum of the divisors of n.

Approach:

As before, if $n = p_1^{e_1}$.

$$\therefore \text{Sum} = 1 + p_1 + p_1^2 + \cdots p_1^{e_1}$$

$$= \frac{p_1^{e_1+1} - 1}{p_1 - 1}$$

if there is two factor

$$(1 + p_1 + p_1^2 \cdots p_1^{e_1}) \cdot (1 + p_2 + p_2^2 \cdots p_2^{e_2})$$

$$\frac{p_1^{e_1+1} - 1}{p_1 - 1} \cdot \frac{p_2^{e_2+1} - 1}{p_2 - 1}$$

$$\therefore \sigma(n) = \frac{p_1^{e_1+1} - 1}{p_1 - 1} \cdot \frac{p_2^{e_2+1} - 1}{p_2 - 1} \cdot \frac{p_k^{e_k+1} - 1}{p_k - 1}$$

Sub :

Day

Time : Date : / /

Implementation:

```
int divisorSum(int n)
{
    if (n<1)
        return 0;
    int sum = 1;
    Seive (n+1);
    for (int i = 0; primes[i]*primes[i] <= n; i++)
    {
        if(n%primes[i] == 0)
        {
            int cnt=1;
            while (n%primes[i] ==0)
            {
                n /= primes[i];
                cnt++;
            }
            sum *= (pow(primes[i], cnt)-1)/
                    (primes[i]-1));
        }
    }
    if (n>1)
        sum *= (pow(n,2)-1)/n-1);
    return sum;
}
```

Sub :

Day
Time : Date : / /

~~Bi~~

Binary Exponential (~~Big Mod~~)

problem statement:

- calculate a^n using only $O(\log n)$ instead of $O(n)$.

Approach:

- We can represent any power n in binary form.

$$3^{13} = 3^{(1101)_2} = 3^8 \cdot 3^4 \cdot 3^1$$

- Since the number n in a^n has exactly $\lfloor \log_2 n \rfloor + 1$ digits, in base 2, We only need to perform $O(\log n)$ multiplications, if we know the powers $a^1, a^2, a^4, a^8 \ldots a^{2^{(\log_2 n)}}$

- It's very easy, because every element is just square to previous. $3^1, 3^2, 3^4, 3^8$

$\therefore 3^{13} = 3^1 \cdot 3^4 \cdot 3^8 = 1594323$

Sub :

Day
Time : Date : / /

So $a^n = \begin{cases} 1 & \text{if } n == 0 \\ (a^{\frac{n}{2}})^2 & \text{if } n > 0 \text{ and } n \text{ even} \\ (a^{\frac{n-1}{2}})^2 * a & \text{if } n > 0 \text{ and } n \text{ odd} \end{cases}$

Implementation: (recursive)

```
long long binpow (long long a, long long b)
{
    if (b == 0)
        return 1;
    long long res = binpow (a, b/2);
    if (b % 2)
        return res * res * a;
    else
        return res * res;
}
```

Iterative:

```
long long binpow (long long a, long long b)
{
```

~~if (b==0)~~
~~return 1;~~

~~long long res = binpow (a, b/2);~~

```
    long long res = 1;
    while (b > 0)
    {   if (b & 1)      → // (b % 2 == 1)
            res = res * a;
        a = a * a;
        b >>= 1;        // b = b/2
    }
    return res;
}
```

odd হলে result এ সাথে গুণ হচ্ছে কেন?

- কারণ odd হলে binary bit 1 হয়। 1 হলে সেটা 1 দিয়ে গুণ হবে কিন্তু even হলে 0 হওয়ায় তাকে 0 দিয়ে গুণ হয় তাই উঠে যায়

- 1 1 0 1 (0 → even, 1 → odd)

8 4 (2) 1

X

complexity $O(\log_2 N)$

Use:

- It has important applications in many tasks unrelated to arithmatic. Since it can be used with any operations that have the property of associativity;

$$Z \cdot (X \cdot Y) = X \cdot (Y \cdot Z)$$

- BigMod (modular operation in large number)

Bigmod implementation (iterative) $(x^y) \% P$

```
int bigmod (ll x, ll y, ll P)
{   int res = 1;
    x = x % P; // if x is more than or equal to P
    if (x = 0), return 0; // if x is divisible by P
    while (y > 0)
    {   if (y & 1)
            res = (res * x) % P;
        x = (x * x) % P;
        y >>= 1;
    }
return res;
}
```

Sub :

Day

Time : Date : / /

Extended Euclidean Algorithm

problem :

find two integers x and y such that

$$ax + by = gcd(a, b)$$

bezout's identity — (x, y) bezout's pair

Approach :

- Normal Euclidean algorithm, works with remainder only.

$GCD(240, 46) = GCD(46, 240\%46) = GCD(46, 10)$

$GCD(46, 10) = GCD(10, 46\%10) = GCD(10, 6)$

$GCD(10, 6) = GCD(6, 10\%6) = GCD(6, 4)$

$GCD(6, 4) = GCD(4, 6\%4) = GCD(4, 2)$

$GCD(4, 2) = GCD(2, 4\%2) = GCD(2, 0)$

$GCD(2, 0) = 2.$

// Euclid.

```
int GCD (a, b)
{ if (b == 0)
    return a;
  else return (b, a % b);
}
```

Sub :

Day

Time : Date : / /

Introducing r and q

r = remainder, q = quotient.

let r_0 be a, r_1 be b.

$\therefore$ $q_i = \frac{r_{i-2}}{r_{i-1}}$, $r_i = r_{i-2} - q_i \times r_{i-1}$

Introducing x and y

As GCD(a, b) = ax + by, variable x and y will hold the coefficients.

$\therefore$ $r_i = ax_i + by_i$ → GCD actually a remainder

Initially $x_0, y_0 = (1, 0)$ $(x_1, y_1) = (0, 1)$

$(x_i, y_i) = ?$

$r_i = r_{i-2} - q_i \times r_{i-1}$, $r_i = ax_i + by_i$

$$r_i = (ax_{i-2} + by_{i-2}) - q_i (ax_{i-1} + by_{i-1})$$

$$= ax_{i-2} - q \times ax_{i-1} + by_{i-2} + q_i \times by_{i-1}$$

$$= a(x_{i-2} - q_i \times x_{i-1}) + b(y_{i-2} - q_i y_{i-1})$$

$$= a(x_i) + b(y_i)$$

Sub :

Day

Time : Date : / /

$\therefore x_i = x_{i-2} - q_i \times x_{i-1}$, $y_i = y_{i-2} - q_i \times y_{i-1}$

Now we can re-write the table

indx.	Quotient ~~Quotient~~	Remainder ~~Remainder~~	x_i	y_i
0		240	1	0
1		46	0	1
2	$240/46 = 5$	$240 - 5\times46 = 10$	$1 - 5\times0 = 1$	$0 - 5\times1 = -5$
3	$46/10 = 4$	$46 - 4\times10 = 6$	$0 - 4\times1 = -4$	$1 - 4\times-5 = 21$
4	$10/6 = 1$	$10 - 1\times6 = 4$	$1 - 1\times-4 = 5$	$-5 - 1\times21 = -26$
5	$6/4 = 1$	$6 - 1\times4 = 2$	$-4 - 1\times5 = -9$	$21 - 1\times-26 = 47$
6	$4/2 = 2$	$4 - 2\times2 = 0$	$5 - 2\times-9 = 23$	$-26 - 2\times47 = -120$
6				
9				

So the ans lies on the line $240\times-9 + 46\times47 = 2$.

Sub :

Day

Time : Date : / /

implementation : (iterative)

$$x_2 = x_{i-2},\ x_{i-1} = x_1$$
$$x = x_i$$

```
int ext_gcd(int A, int B, int *X, int *Y)
{
    int x2, y2, x1, y1;   // x_{i-2} y_{i-2} x_{i-1} y_{i-1}
    int x, y, r2, r1, q, r;   // x_i y_i r_{i-2} r_{i-1} q_i r_i

    x2 = 1, y2 = 0;
    x1 = 0, y1 = 1;

    for (r2 = A, r1 = B; r1 != 0; r2 = r1, r1 = r, x2 = x1, x1 = x, y2 = y1, y1 = y)
    {
        q = r2 / r1;
        r = r2 % r1;
        x = x2 - (q * x1);
        y = y2 - (q * y1);
    }
    *X = x2, *Y = y2;

    return r2;
}
```

given

swapping values. (iterati

Sub :

Day
Time : Date : / /

Uniqueness of solution:

→ solution is Not unique, We find a ~~pair (x, y) using~~ infinite number of pair (x_i, y_i)

$$\left(x + k\frac{b}{gcd(a,b)}, y - k\frac{a}{gcd(a,b)}\right)$$

Why ?

$$a\left(x + \frac{kb}{gcd(a,b)}\right) + b\left(y - \frac{ka}{gcd(a,b)}\right)$$

$$= ax + \frac{kab}{gcd(a,b)} + by - \frac{kab}{gcd(a,b)}$$

$$= ax + by$$

And gcd(a,b) is the smallest possible solution

- All the solutions are multiple of gcd(a,b) (bezout's identity)

Sub :

Time : Date : / /

Recursive Approach

$$ax + by = g$$

$$bx_1 + (a \bmod b)\, y_1 = g$$

Now $a \bmod b = a - \lfloor \frac{a}{b} \rfloor \cdot b$

$\therefore\ bx_1 + (a \bmod b)\, y_1 = g$

$\Rightarrow bx_1 + \left(a - \lfloor \frac{a}{b} \rfloor \cdot b\right) y_1 = g$

$\therefore\ g = a \cdot y_1 + b\left(x_1 - y_1 \lfloor \frac{a}{b} \rfloor\right)$

$\therefore\ x = y_1$

$y = x_1 - y_1 \lfloor \frac{a}{b} \rfloor.$

implementation :

```
int ext_gcd (int a, b, *x, *y)
{
    if (b == 0)
    {
        *x = 1
        *y = 0.
        return a;
    }
    int x1, y1;
    int gcd = ext_gcd (b, a%b, &x1, &y1);
    *x = y1;
    *y = x1 - y1 * (a/b);
    return gcd;
}
```

Sub :

Day

Time : Date : / /

Fermat's Little Theorem

problem:

if p is a prime number, then for any integer a, the number $a^p - a$ is an integer multiple of p.

$$a^p \equiv a \pmod{p}$$

$$a^p - a \equiv 0 \pmod{p}$$

special case:

If a is not divisible by p, fermat's little theorem is equivalent to the statement that $a^{p-1} - 1$ is an integer multiple of p.

$$a^{p-1} \equiv 1 \pmod{p}$$

Example:

let, a = 2, // an int which is not multiple of p

p = 17 // prime

$\therefore 2^{17-1} \equiv 1 \pmod{17}$

We got $65536 \% 17 \equiv 1$

$\therefore (65536 - 1)$ is multiple of 17.

Modular Multiplicative inverse

Sub :

Day

Time :

Date : / /

Use of fermat's little Theorem: (MMI)

We know, m is prime,

$$a^{m-1} \equiv 1 \pmod{m}$$

$$a \times a^{m-2} \equiv 1 \pmod{m}$$

here, $a^{m-2} \pmod{m}$ is the modular multiplicative inverse (MMI)

Sub :

Day

Time : Date : / /

Modular Multiplicative inverse

Statement:

A modular multiplicative inverse of an int a is an int x such that $a.x$ is congruent to 1 modulo m.

$$ax \equiv 1 \pmod{m}$$

- We can simply denote $x = a^{-1}$
- Value x should be in range $(1 \dots m-1)$
- A MMI exists if and only if A and M are relatively prime (coprime)

Approach: (when A, M are co-prime) $\gcd(a,b) = 1$

from ext- euclid,

$$ax + by = \gcd(a, b),$$

Now, $Ax + My = 1$ $(b = M, \gcd = 1)$

if we take M modulo

$$Ax + My \equiv 1 \pmod{M}$$

$$Ax \equiv 1 \pmod{M}$$ [My is multiple of M]

$$Ax \equiv 1 \pmod{M}$$

We can find x from ext- gcd.

Sub :

Time : Date : / /

#implementation:

```
{   int x, y;
    int g = ext_euclid(a, m, x, y)
    if (g != 1)
        cout << "No solution";
    else
    {   x = (x % m + m) % m
        cout << x << endl;
    }
```

x may be negative, x % m will also be negative, so we first mod and add m to make positive

complexity:

Time: O(log M)

space: O(log M)

Sub :

Day

Time : Date : / /

MMI when ~~Multiplicative~~ M is prime

From fermat's little theorem,

$$a^{M-1} \equiv 1 \pmod{M}$$

$$a^{M-2} \equiv a^{-1} \pmod{M}$$ [both side multiply a^{-1}]

$\therefore a^{M-2}$ is the MMI. modulo M.

Implementation:

We can simply use BIGMOD(Base, Power, Mod)

```
void modInverse(int A, int M)
{
    int g = gcd(A, M);
    if (g != 1)
        cout << "doesn't exist";
    else
        cout << BigMod(A, M-2, M);
}
```

Sub :

Day
Time : Date : / /

bigMod

```
int BigMod(int a, int n, int m)
{
    if (n == 0) return 1;

    if (n % 2 == 0)                    // m
    {
        int p = bigmod(a, n/2);
        return p*p;
        → (p*p)%m → modified.
    }
    else                               // m
    {
        int p = bigmod(a, (n-1)/2);
        return p*p*a;
        → (p*p*a)%m
    }
}
```

$$a^n = \begin{cases} 1, & \text{if } n = 0 \\ a^{n/2} \times a^{n/2} & \text{if } n \text{ is even} \\ a^{n-1/2} \; a^{(n-1)/2} \times a & \text{if } n \text{ is odd.} \end{cases}$$

Sub :

Day

Time : Date : / /

Chinease Remainder Theorem

Problem statement:

Given two sequences of numbers $A = [a_1, a_2 \cdots a_n]$ and $M = [m_1, m_2, \ldots m_n]$. find the smallest solution to the following linear congruence equations if if exists:

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$
$$\cdots$$
$$x \equiv a_n \pmod{m_n}$$

Approach : (Weak form)

Given two sequence $A = [a_1, a_2 \cdots a_n]$, $M = [m_1, m_2 \cdots m_n]$ where all elements of M are pairwise co-prime then there always exists an unique solution to $x \bmod L$, where $L = m_1 \times m_2 \times \cdots m_n$.

Finding a solution:

Let's just consider only 1st two equation

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$

- We will merge two equation into 1.

Sub :

Time : Date : / /

Since $gcd(m_1, m_2) = 1$

$\therefore m_1 p + m_2 q = 1$ [Bezout's identity]

——— (I)

From ext_gcd we can find p, q.

and solution is

$$x = a_1 m_2 q + a_2 m_1 p \pmod{m_1 m_2}$$

proof:

$$x = a_1 m_2 q + a_2 m_1 p$$

$x = a_1 (1 - m_1 p) + a_2 m_1 p$ using (I) eq.

$x = a_1 - a_1 m_1 p + a_2 m_1 p$

$x = a_1 + (a_2 - a_1) m_1 p$

$\therefore x \equiv a_1 \pmod{m_1}$

Similarly, $x = a_1 m_2 q + a_2 m_1 p \equiv a_2 \pmod{m_2}$

why mod $m_1 m_2$?

the solution x may not be the smallest. so we need to mod. by such no. which is divisible by both m_1 and m_2 ($lcm(m_1, m_2)$)

and, $lcm(m_1, m_2) = m_1 m_2$ [coprime]

Sub :

Day

Time : Date : / /

Proof of uniqueness: Let, we take two solution x_1 and x_2 ⟶

$$x_1 \equiv a_1 \pmod{m_1}$$

$$x_2 \equiv a_1 \pmod{m_1}$$

$$x_1 \equiv x_2 \pmod{m_1}$$

$$x_1 - x_2 \equiv 0 \pmod{m_1}$$

$\therefore m_1 \mid x_1 - x_2$ ⟶

Similarly we can show that $m_2 \mid x_1 - x_2$ ⟶

- That means difference of any two solution $(x_1 - x_2)$ is divisible by both m_1 and m_2 ⟶
- m_1 and m_2 coprime, that means

$$m_1 m_2 \mid (x_1 - x_2)$$

$$\therefore (x_1 - x_2) \equiv 0 \pmod{m_1 m_2}$$

$$x_1 \equiv x_2 \pmod{m_1 m_2}$$

- Now, $x_1 - x_2$ divisible by $m_1 m_2$, then how many solution exists $0 - (m_1 m_2 - 1)$?

1 - only one

That's why the solution is unique modulo $m_1 m_2$.

Sub :

Day

Time : Date : / /

We will then work with rest of the (n-1) equation

Implementation :

```
pair<int, int> weakCRT(vector<int> A, vector<int> M)
{
    if(A.size() != M.size()) return {-1, -1};
    int n = A.size()
    int a1 = A[0];
    int m1 = M[0];
    for (int i=1; i<n; i++)
    {
        int a2 = a[i];
        int m2 = m[i];
        int p, q;
        ext_gcd(m1, m2, &p, &q);
        int x = (a1 * m2 * q + a2 * m1 * p) % (m1*m2);
        a1 = x;
        m1 = m1 * m2;
    }
    if (a1 < 0)  a1 += m1
    return {a1, m1};
}
```

for rest of the equations

complexity

Time : O(n x log L);

Sub :

Day

Time : Date : / /

Strong ~~Weak~~ CRT

here, ~~no~~ elements of M is not pairwise coprime.

Existance of solution:

if $a_1 \equiv a_2 \pmod{g}$.

why.

$$x \equiv a_1 \pmod{m_1}$$

$$x - a_1 \equiv 0 \pmod{m_1}$$

$$m_1 \mid x - a_1$$

since m_1 divides $x - a_1$, divisor of m_1 also divides $x - a_1$

$$\therefore \quad x - a_1 \equiv 0 \pmod{g}$$

Similarly $x - a_2 = 0 \pmod{g}$

$$\therefore \quad a_1 - a_2 \equiv 0 \pmod{g}$$

$$\therefore \quad a_1 \equiv a_2 \pmod{g}$$

$$\therefore \quad g \mid a_1 - a_2$$

Finding Solution:

$m_1 p + m_2 q = g$ / bezout's identity

$\left(\frac{m_1}{g}\right) p + \left(\frac{m_2}{g}\right) q = 1$ ——— ①

we'll get p, q in ext-gcd($m_1/g, m_2/g, p, q$)

$$\boxed{x = a_1 \frac{m_2}{g} q + a_2 \frac{m_1}{g} p.}$$

why? from eq- ①

$$\frac{m_1}{g} p = 1 - \frac{m_2}{g} q.$$

$$\therefore x = a_1 \frac{m_2}{g} q + a_2 \left(1 - \frac{m_2}{g} q\right)$$

$$x = a_2 + (a_1 - a_2) \frac{m_2}{g} q.$$

As $g \mid a_1 - a_2$

$$\therefore x = a_2 + \frac{a_1 - a_2}{g} m_2 q.$$

$$\therefore x \equiv a_2 \pmod{m_2}$$

similarly $x \equiv a_1 \pmod{m_1}$

Sub :

Day
Time : Date : / /

Smallest ? LCM (m_1, m_2) (Explained earlier)

$$x \equiv a_1 \frac{m_2}{g} q + a_2 \frac{m_1}{g} p \pmod{LCM(m_1, m_2)}$$

Implementation:

```
pair<int, int> strong_CRT(Vector A, Vector M)
{
    if (A.size() != M.size()) return {-1, -1};
    int n = A.size();
    int a1 = A[0];
    int m1 = M[0];
    for(int i = 1; i < n; i++)
    {
        int a2 = A[i];
        int m2 = M[i];
        g = __gcd(m1, m2);
        if (a1 % g != a2 % g) return {-1, -1};
        int p, q;
        ext_gcd(m1/g, m2/g, &p, &q);
        int mod = m1/g * m2;
        int x = (a1 * (m2/g) * q + a2 * (m1/g) * p) % mod;
        a1 = x;
        if (a1 < 0) a1 += mod;
        m1 = mod;
    }
    return {a1, m1};
}
```

complexity = $O(n \times \log(L))$

Sub :

Day

Time : Date : / /

Binary Operation

① Left shift

$$x << i = x * 2^i$$

② Right shift

$$x >> i = \lfloor x * / 2^i \rfloor$$

* if we want to get 2^i number

$$1 << i = 1 \times 2^i$$

* check if i-th bit of a number

if we take a number which is known, and its ith bit is 1, and make AND with x. then we will get output 1, if ith bit of x is 1, otherwise 0.

$\therefore$ if $(x \;\&\; (1 << i))$

{ cout << 1

else } cout << 0 .

i's complement of x $\Rightarrow$ $(\sim x)$ (ith)

Sub :

Day
Time : Date : / /

Make ith Number of a bit = 1

like same as before $(x \text{ or } y)$

$(x \text{ or } (1 << i))$

how to make i-th bit = 0 ?

how to make i-th to $(i+6)$-th bit = 1 ?

take a number with i — th $(i+6)$ bit 1

The OR.

$x =$ 0 1 0 0 0 0 1 0 0 0 0 0 0 0

$y =$ 0 0 1 1 1 1 1 1 0 0

$x | y =$

Making $p = 2^{r+1} - 1$ $\quad q = 2^{l} - 1$ $\quad y = p - q$

$p =$ 1 1 1 1 1 1 1 1 1 1

$q =$ 1 1 1 1 1 1 1 1 0 0

$p = (1 << (r+1)) - 1$

$q = (1 << l) - 1$

$y = p - q$

$x | y = (x | y)$

Sub :

Subset actually can be represented as binary. 000 → ∅
001 → c
010 → b
100 → a

Day
Time : Date :

bit masking

→ Print all the subset of a set.

arr = [a, b, c]

2^n

```
for (int mask = 0; mask < (1<<n); mask++)
{
  for (int i = 0; i<n; i++)
  { if (mask & (1<<i)) // checks if ith bit 1,
      printf(arr[i])
  }
```