Number Theory

Sieve Of Eratosthenes

It is easy to find if some number (say N) is prime or not — you simply need to check if at least one number from numbers lower or equal sqrt(n) is divisor of N. This can be achieved by simple code:

```
boolean isPrime( int n ) {
if ( n == 1 ) return false; // by definition, 1 is no
t prime number
if ( n == 2 ) return true; // the only one even prime
for ( int i = 2; i * i <= n; ++i )
if ( n%i == 0 ) return false;
return true;
}
```

So it takes sqrt(n) steps to check this. Of course you do not need to check all even numbers, so it can be “optimized” a bit:

```
boolean isPrime( int n ) {
if ( n == 1 ) return false; // by definition, 1 is no
t prime number
if ( n == 2 ) return true; // the only one even prime
if ( n%2 == 0 ) return false; // check if is even
for ( int i = 3; i * i <= n; i += 2 ) // for each odd
number
if ( n%i == 0 ) return false;
return true;
}
```

So let say that it takes 0.5sqrt(n) steps*. That means it takes 50,000 steps to check that 10,000,000,000 is a prime

Problem?

If we have to check numbers upto N, we have to check each number individually. So time complexity will be **O(Nsqrt(N))**.

Can we do better?

Ofcourse! we can use a sieve of numbers upto N. For all prime numbers <= sqrt(N), we can make their multiple non-prime i.e. if **p** is prime, 2p, 3p, ..., floor(n/p)*p will be **non-prime**.

Animation

https://upload.wikimedia.org/wikipedia/commons/b/b9/Sieve_of_Eratosthenes_animation.gif

Sieve code

```
void primes(int *p){
    for(int i = 2;i<=1000000;i++)p[i] = 1;
    for(int i = 2;i<=1000000;i++){
        if(p[i]){
            for(int j = 2*i;j<=1000000;j+=i){
                p[j] = 0;
            }
        }
    }
    p[1] = 0;
    p[0] = 0;
    return;
}
```

Can we still do better?

Yeah sure! Here we don’t need to check for even numbers. Instead of starting the non-prime loop from 2p we can start from p^2.

Optimised code

```
void primes(bool *p){
    for(int i = 3;i<=1000000;i += 2){
        if(p[i]){
            for(int j = i*i;j <= 1000000; j += i){
            p[j] = 0;
            }
        }
    }
    p[1] = 0;
    p[0] = 0;
    return;
}
```

T = O(NloglogN)

Hence, we have signifiacntly reduced our complexity from N*sqrt(N) to approx linear time.

Segmented Sieve

```
void sieve(){
    for(int i = 0;i<=1000000;i++)p[i] = 1;
    for(int i = 2;i<=1000000;i++){
        if(p[i]){
            for(int j = 2*i;j<=1000000;j+=i)
            p[j] = 0;
        }
    }
    // for(int i=2;i<=20;i++)cout<<i<<" "<<p[i]<<endl;
}
int segmented_sieve(long long a,long long b){
    sieve();
    bool pp[b-a+1];
    memset(pp,1,sizeof(pp));
    for(long long i = 2;i*i<=b;i++){
        for(long long j = a;j<=b;j++){
```

```
            if(p[i]){
                if(j == i)
                continue;
                if(j % i == 0)
                pp[j-a] = 0;
            }
        }
    }
    int res = 1;
    for(long long i = a;i<b;i++)
    res += pp[i-a];
    return res;
}
```

Division

Let a and b be integers. We say a divides b, denoted by a|b, if there exists an integer c such that b = ac.

Linear Diophantine Equations

A Diophantine equation is a polynomial equation, usually in two or more unknowns, such that only the integral solutions are required. An Integral solution is a solution such that all the unknown variables take only integer values.
Given three integers a, b, c representing a linear equation of the form : ax + by = c. Determine if the equation has a solution such that x and y are both integral values.

General solution

```
(x, y) = (xo + b/d *t, yo – a/d *t)
```

Chinese Remainder Theorem

Typical problems of the form “Find a number which when divided by 2 leaves remainder 1, when divided by 3 leaves remainder 2, when divided by 7 leaves remainder 5” etc can be reformulated into a system of linear congruences and then can be solved using Chinese Remainder theorem.
For example, the above problem can be expressed as a system of three linear congruences:
“x ≡ 1 (mod 2), x ≡ 2 mod(3), x ≡ 5 mod (7)”.
x % num[0] = rem[0],
x % num[1] = rem[1],
....
x % num[k-1] = rem[k-1]

A Naive Approach is to find x is to start with 1 and one by one increment it and check if dividing it with given elements in num[] produces corresponding remainders in rem[]. Once we find such a x, we return it

Chinese remainder theorem

```
x =  ( ∑ (rem[i]*pp[i]*inv[i]) ) % prod
   Where 0 <= i <= n-1

rem[i] is given array of remainders

prod is product of all given numbers
```

```
prod = num[0] * num[1] * ... * num[k-1]

pp[i] is product of all but num[i]
pp[i] = prod / num[i]

inv[i] = Modular Multiplicative Inverse of
         pp[i] with respect to num[i]
```

Euler Phi Function

Euler's Phi function (also known as totient function, denoted by φ) is a function on natural numbers that gives the count of positive integers coprime with the corresponding natural number. Thus, φ(8) = 4, φ(9) = 6

The value φ(n) can be obtained by Euler's formula : Let n = $p1^{a1} * p2^{a2} * .... * pk^{ak}$ be the prime factorization of n. Then

φ(n) = n * (1- 1/p1) * (1- 1/p2) * ... * (1- 1/pk)

Code

```
int phi[] = new int[n+1];
for(int i=2; i <= n; i++) phi[i] = i; //phi[1] is 0

for(int i=2; i <= n; i++)
if( phi[i] == i )
for(int j=i; j <= n; j += i )
phi[j] = (phi[j]/i)*(i-1);
```

Properties

i. If P is prime then **φ(p^k) = (p-1)$p^{(k-1)}$**
ii. φ function is multiplicative, i.e. if (a,b) = 1 then **φ(ab) = φ(a)φ(b).**
iii. Let d1, d2, …dk be all divisors of n (including n). Then φ(d1) + φ(d2) + … + φ(dk) = n
For example: the divisors of 18 are 1,2,3,6,9 and 18. Observe that **φ(1) + φ(2) + φ(3) + φ(6) + φ(9) + φ(18) = 1 + 1 + 2 + 2 + 6 + 6 = 18**
iv. Number of divisors of n=$p1^{a1}.p2^{a2}…pn^{an}$:
d(n) = (a1+1) * (a2+1) * … (an + 1)
v. Sum of divisors:
S(n)= $(p1^{a1}-1)/(p1-1)$ $(p2^{a2}-1)/(p2-1)$ ....$(pn^{an}-1)/(pn-1)$

Wilson's theorem

If p is a prime, then (p — 1)! = -1 (mod p)

Problems
POWPOW2
http://www.spoj.com/problems/POWPOW2/

Problem

Given three integers a b n, $1 \le a,b,n \le 10^5$
$a^{(b^{f(n)})} \mod 1\,000\,000\,007$, where $f(n) = \binom{n}{0}^2 + \binom{n}{1}^2 + \cdots + \binom{n}{n}^2$.

Dealing with f(n)

The function f complicates the expression, but we can notice that $f(n)={}^{2n}C_n$. It's easy to find proofs online, e.g. here, so I'll skip that.

Reducing the exponents

$b^{(2n,n)}$ is a huge number and we need to reduce it to a more tractable number.

Euler's theorem states that if a and m are coprime, then $a^{\phi(m)}=1 \pmod m$, where $\phi(m)$ is Euler's totient function. This is useful because $a^y \equiv a^{(y \bmod \phi(m))} \pmod m$
The repeated $\phi(m)$factors in the exponent will yield a bunch of 1s).

$m=10^9+7$ which is a prime number, so $\phi(m)=m-1=10^9+6=2\times500000003$.

So, we have $a^{y \bmod 1000000006} \bmod 1000000007$.

The main difficulty of this problem is that our y is also an exponential, $y=b^{(2n,n)}$. In order to find the result, we need first to calculate $b^{(2n,n)} \bmod 1000000006$.

Finding $b^{(2n,n)} \bmod 1000000006$ when b is odd

Suppose b is odd. Then, we can apply Euler's theorem because b and 1,000,000,006 are coprime (recall that $b\le10^5$ so the 500000003 factor will always be coprime with b).

$b^{(2n,n)} \equiv b^{(2n,n) \bmod \phi(1000000006)} \bmod 1000000006$.

$\phi(1000000006)=\phi(2)\times\phi(500000003)=(2-1)\times(500000003-1)=500000002$

$500000002=2\times41^2\times148721500000002=2\times41^2\times148721$.

So, we need to find $(2n,n) \bmod 500000002$ which is not prime. Therefore, we need to use another tool: the Chinese Remainder Theorem (CRT). We can calculate

$(2n,n) \bmod 2$
$(2n,n) \bmod 41^2$
$(2n,n) \bmod 148721$

and use CRT to get the result modulo 500000002.

Finding $b^{(2n,n)} \bmod 1000000006$ when b is even

Unfortunately, if b is even, b and 1000000006 are not coprime.

Therefore, we need CRT again. Our modulus is the product of two primes: 2 and 500,000,003. So, we shall find $b^{(2n,n)}$ modulo 2 and 500,000,003 and use CRT to get the result modulo 1,000,000,006.

Note that when b is even the result modulo 2 is always 0. So, we only need to calculate the result modulo 500000003 and $\phi(500000003)=\phi(1000000006)$, so this part is equal to the case when b is odd. The only difference is using CRT.

Adding everything together

After finding $y=b^{(2n,n)} \bmod 1000000006$, we can calculate $a^y \bmod 1000000007$ normally to get the final result.

CODE

```
#include<bits/stdc++.h>
#define ll long long int

int t;
ll a, b, n;
ll fact[200005];
ll md = 1000000007;
long long int c_pow(ll i, ll j, ll mod)
{

    if (j == 0)
        return 1;
    ll d;
    d = c_pow(i, j / (long long)2, mod);
    if (j % 2 == 0)
        return (d*d) % mod;
    else
        return ((d*d) % mod * i) % mod;
}

ll InverseEuler(ll n, ll MOD)
{
    return c_pow(n, MOD - 2,MOD);
}

ll fact_14[1700][1700];
ll fact_B[150000];

ll min1(ll a, ll b) {
    return a > b ? b : a;
}


void calc_fact() {
    fact[0] = fact[1] = 1;
    ll tmd = 148721;
    for (int i = 2; i < 200003; ++i) {
        fact[i] = (fact[i - 1] * i);
        if (fact[i] >= (tmd))fact[i] %= (tmd);
    }
}

ll fact_41[200005];
ll fact_41_p[200005];

void do_func() {
    fact_41[0] = 1;
    fact_41_p[0] = 0;
    for (int i = 1; i < 200003; ++i) {
        ll y = i;
        fact_41_p[i] = fact_41_p[i - 1];
        while (y % 41 == 0) {
            y = y / 41;
            fact_41_p[i]++;
        }
```

```
        fact_41[i] = (y*fact_41[i - 1]) % 1681;
    }
}

ll fact_2[200005];
void do_func2() {
    fact_2[0] = 1;
    for (int i = 1; i < 200005; ++i) {
        fact_2[i] = (i*fact_2[i - 1]) % 2;
    }
}

ll get_3rd(ll n, ll r, ll MOD) {
    ll ans = (InverseEuler(fact[r], MOD)*InverseEuler(fact[n - r], MOD)) % MOD;
    ans = (fact[n] * ans) % MOD;
    return ans;
}

ll inverse2(ll m1, ll p1)
{
    ll i = 1;
    while (1)
    {
        if ((m1*i) % p1 == 1)
            return i;
        i++;
    }

}

ll chinese_remainder_2(ll n1, ll n2, ll n3)
{
    ll p1 = 2, p2 = 1681, p3 = 148721;
    ll m1, m2, m3;
    ll i1, i2, i3;
    ll m;
    ll ans;
    m = p1*p2*p3;
    m1 = m / p1; m2 = m / p2; m3 = m / p3;
    i1 = InverseEuler(m1, p1); i2 = inverse2(m2, p2); i3 = InverseEuler(m3, p3);
    //printf("i1 = %lld   i2 = %lld\n",i1,i2);
    ans = (n1*m1*i1) % m + (n2*m2*i2) % m + (n3*m3*i3) % m;
    ans = ans%m;
    return ans;
    //printf("%d\n",ans);
}

int main() {

    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    calc_fact();
    do_func();
    do_func2();
    cin >> t;
    while (t--) {
        cin >> a >> b >> n;
        if (a == 0 && b == 0) {
            cout << "1\n";
            continue;
```

```
        }
        if (b == 0) {
            cout << "1\n";
            continue;
        }
        ll a1 = (n == 0) ? 1 : 0;
        ll a2 = (fact_41[2 * n] * inverse2(fact_41[n],1681)) % 1681;
        a2 = (a2 * inverse2(fact_41[n], 1681)) % 1681;
        a2 = (a2 * c_pow(41, fact_41_p[2 * n] - 2 * fact_41_p[n], 1681)) % 1681;
        ll a3 = get_3rd(2 * n, n, 148721);
        //cout << a1 << " " << a2 << " " << a3 << "\n";
        ll ans = chinese_remainder_2(a1, a2, a3);
        if (ans == 0)ans = 500000002;
        ll y1 = c_pow(b, ans, md - 1);
        cout << y1 << "\n";
        ll z = c_pow(a, y1, md);
        cout << z << "\n";
    }

    return 0;
}
```

Best method for nCr

```
#include<iostream>
using namespace std;
#include<vector>

/* This function calculates (a^b)%MOD */
long long pow(int a, int b, int MOD)
{
    long long x=1,y=a;
    while(b > 0)
    {
        if(b%2 == 1)
        {
            x=(x*y);
            if(x>MOD) x%=MOD;
        }
        y = (y*y);
        if(y>MOD) y%=MOD;
        b /= 2;
    }
    return x;
}

/*  Modular Multiplicative Inverse
    Using Euler's Theorem
    a^(phi(m)) = 1 (mod m)
    a^(-1) = a^(m-2) (mod m) */
long long InverseEuler(int n, int MOD)
{
    return pow(n,MOD-2,MOD);
}

long long C(int n, int r, int MOD)
{
    vector<long long> f(n + 1,1);
```

```
    for (int i=2; i<=n;i++)
        f[i]= (f[i-1]*i) % MOD;
    return (f[n]*((InverseEuler(f[r], MOD) * InverseEuler(f[n-r], MOD)) % MOD)) % MOD;
}

int main()
{
    int n,r,p;
    while (~scanf("%d%d%d",&n,&r,&p))
    {
        printf("%lld\n",C(n,r,p));
    }
}
```