jeqcho's blog

Problem Solving Guide to Modular Combinatorics and Exponentiation

By jeqcho, history, 3 years ago,

Sometimes, you are asked to calculate the combination or permutation modulo a number, for example ${}^nC_k \mod p$. Here I want to write about a complete method to solve such problems with a good time complexity because it took me a lot of googling and asking to finally have the complete approach. I hope this blog can help other users and save their time when they solve combinatorics problem in Codeforces.

Example Problem

Find the value of ${}^nC_k$, $(1 \leq n, k \leq 10^6)$. As this number can be rather large, print the answer modulo p. $(p = 1000000007 = 10^9 + 7)$

Combination (binomial coefficients)

${}^nC_k$ means how many ways you can choose k items from an array of n items, also denoted as $\binom{n}{k}$. This is also known as binomial coefficients. The formula for combination is

$${}^nC_k = \frac{n!}{k!(n-k)!}$$

Sometimes, the denominator $k!(n-k)!$ is very large, but we can't modulo it since modulo operations can't be done independently on the denominator.
$\frac{n!}{k!(n-k)!} \mod p \neq \frac{n! \mod p}{k!(n-k)! \mod p}$. Now I will introduce the modular multiplicative inverse to solve this problem.

Modular multiplicative inverse

The modular multiplicative inverse x of a modulo p is defined as

$$a \cdot x \equiv 1 \pmod p$$

Here, I will replace x with $\text{inv}(a)$, so we have

$$a \cdot \text{inv}(a) \equiv 1 \pmod p$$

Getting back to the formula for combination, we can rearrange so that

$${}^nC_k = n! \cdot \frac{1}{k!} \cdot \frac{1}{(n-k)!}$$

Here, we can use $\text{inv}(a)$ as follows

$${}^nC_k \equiv n! \cdot \text{inv}(k!) \cdot \text{inv}((n-k)!) \pmod p$$

Now we can distribute the modulo to each of the terms by the distributive properties of modulo

$${}^nC_k \mod p = n! \mod p \cdot \text{inv}(k!) \mod p \cdot \text{inv}((n-k)!) \mod p$$

Now I will discuss on how to calculate $\text{inv}(a)$

Fermat's Little Theorem

You can easily remember this theorem. Let a be an integer and p be a prime number,

$$a^p \equiv a \pmod p$$

It is helpful to know that the p in the problem $(10^9 + 7)$ is indeed a prime number! We can rearrange the equation to get

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

Looking back at our equation for $\text{inv}(a)$, both equations equate to 1, so we can equate them as

$$a \cdot \text{inv}(a) \equiv a^{p-1} \pmod p$$

We can rearrange the equation to get

$$\text{inv}(a) \equiv a^{p-2} \pmod p$$

We now have a direct formula for $\text{inv}(a)$. However, we cannot use the `pow()` function to calculate a^{p-2} because a and p is a large number (Remember $1 \le n, k \le 10^6$) ($p = 10^9 + 7$). Fortunately, we can solve this using modular exponentiation.

Modular Exponentiation

To prevent integer overflow, we can carry out modulo operations **during** the evaluation of our new power function. But instead of using a while loop to calculate a^{p-2} in $O(p)$, we can use a special trick called **exponentiation by squaring**. Note that if b is an even number

$$a^b = (a^2)^{b/2}$$

Every time we calculate a^2, we reduce the exponent by a factor of 2. We can do this repeatedly until the exponent becomes zero where we stop the loop. This will give us a time complexity of $O(\log p)$ to calculate a^{p-2} because we halve the exponent in each step. For the case when b is odd, we can use the property

$$a^b = a^{b-1} \cdot a$$

We then store the trailing a into a variable. Then $b - 1$ is even and we can proceed as previously stated. We can repeatedly apply these two equations to calculate a^{p-2}. Here I will show you the implementation of this modified `powmod()` function to include modulo operations. `ll` is defined as `long long`

```
ll powmod(ll a, ll b, ll p){
    a %= p;
    if (a == 0) return 0;
    ll product = 1;
    while(b > 0){
        if (b&1){    // you can also use b % 2 == 1
            product *= a;
            product %= p;
            --b;
        }
        a *= a;
        a %= p;
        b /= 2;    // you can also use b >> 1
    }
    return product;
}
```

Then we can finally implement the $\text{inv}(a)$ function simply as

```
ll inv(ll a, ll p){
    return powmod(a, p-2, p);
}
```

Then, finally, we can implement ${}^nC_k$ as

```
ll nCk(ll n, ll k, ll p){
    return ((fact[n] * inv(fact[k], p) % p) * inv(fact[n-k], p)) % p;
}
```

We used the dp-approach for factorial where the factorial from 1 to n is pre-computed and stored in an array `fact[]`.

Time complexity

- Pre-computation of factorial: $O(n)$
- Calculation of ${}^nC_k$, which is dominated by modular exponentiation `powmod` : $O(\log p)$
- Total: $O(n + \log p)$

Reference

- Fermat's Little Theorem — wiki
- Modular Multiplicative Inverse — wiki
- Modular Multiplicative Inverse — cp-algorithm

Problems for you

- 300C - Beautiful Numbers (Example solution: 83862274)
- 717A - Festival Organization

Please comment below if you know similar problems.

I hope this blog will help you in your competitive programming journey.

Stay safe and thank you for reading.

#modular exponentiation, #modulo, modulo multiplication, #dp with modulus, #modular arithmetic, #modular_theory, #number theory, #combinatorics, #combination

+18 jeqcho 3 years ago 9

Comments (6)

Show archived | Write comment?

jeqcho

3 years ago, # | 0

Auto comment: topic has been updated by jeqcho (previous revision, new revision, compare).

→ Reply

redotter

3 years ago, # | ← Rev. 2 +1

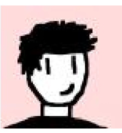
Good tutorial! It's worth mentioning that you can find modular inverse using Extended Euclidean Algorithm in $O(\log p)$, too.

Also, if n and k are small, you can calculate binomial coefficients with DP in $O(nk)$ without modular inverse.

And here are some problems:

- 1000D - Yet Another Problem On a Subsequence
- 1359E - Modular Stability
- 1091D - New Year and the Permutation Concatenation
- 1043F - Make It One
- 895D - String Mark

→ Reply



d-agrawal

3 years ago, # | +1

Thank you. This tutorial was very helpful and easy to understand.

→ Reply

raysk

3 years ago, # | 0

Thank you for tutorial!

→ Reply

3 months ago, # | ← Rev. 2 0

Shouldn't the first line of `powmod()` be `if(b == 0) return 1;` ?

My code wasn't passing without adding it in this problem x_CroNoS_x

My code wasn't passing without adding it in this problem.

→ Reply

7 weeks ago, # | 0

Viswanath.V

Firstly, thanks for this blog. I want to add something here. We can optimize the Time complexity a little bit in the code. Small observation: n! = n*(n-1)*(n-2)..(n-k)! So, we can calculate the factorial of n up to (n-k+1) (I would call it as a Partial_Factorial). this will narrow down our problem to find: (Partial_Factorial(n , k) % p * inv(fact[k] , p) %p) % p. Explaining this with an example, Consider 6C2. 6C2 = 6!/(4!*2!). I'm doing that like this: (6*5)/2!

→ Reply