

# Giai thừa modulo $p$

## Người viết:

- ▶ Nguyễn Minh Hiền - Trường Đại học Công nghệ, ĐHQGHN

## Reviewer:

- ▶ Phạm Công Minh - Trường THPT chuyên Khoa học Tự nhiên, ĐHQGHN
- ▶ Phạm Hoàng Hiệp - University of Georgia

## Giai thừa modulo $p$

### Giới thiệu

Khi giải các bài toán, đôi khi chúng ta sẽ gặp phải những công thức liên quan đến giai thừa, có thể ở trên tử, cũng có thể ở dưới mẫu như hệ số nhị thức.  $n!$  là một hàm số nguyên tăng khá nhanh, nên thường chúng ta được yêu cầu tính toán nó theo một modulo  $p$  nguyên tố nào đó. Để giải quyết vấn đề đó trong bài viết này, trước hết chúng ta đi đến một số ký hiệu sau:

- ▶  $n!$  ( $n$  giai thừa) là tích các số từ 1 đến  $n$ :  $n! = 1 \cdot 2 \cdot 3 \cdots n$
- ▶  $v_p(n)$  là số nguyên lớn nhất để  $p^{v_p(n)}$  là ước của  $n$ .

### Thuật toán "ngây thơ"

Cách đơn giản nhất để tính  $n! \bmod p$  là ta lần lượt nhân các số từ 1 đến  $n$ :

```
1 factorial[0] = 1;
2 for (int i = 1; i <= n; i++) {
3     factorial[i] = factorial[i - 1] * i % p;
4 }
```

Chú ý rằng nếu  $n \geq p$  thì  $n!$  luôn chia hết cho  $p$ .

Độ phức tạp:  $O(n)$ .

### Công thức Legendre

Phát biểu [công thức Legendre](#)  :

$$v_p(n!) = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \left\lfloor \frac{n}{p^3} \right\rfloor + \dots$$



Chú ý rằng khi  $p^k > n$  thì  $\left\lfloor \frac{n}{p^k} \right\rfloor = 0$ , ta không cần tính tiếp

► Chứng minh công thức Legendre

Code minh họa công thức Legendre:

```
1  int vp(int n, int p) {
2      int cnt = 0;
3      while (n) {
4          n /= p;
5          cnt += n;
6      }
7      return cnt;
8  }
```

Độ phức tạp:  $O(\log_p n)$

Bạn đọc có thể sử dụng công thức Legendre cùng sàng nguyên tố (đã chuẩn bị trước) để phân tích  $n! = p_1^{k_1} \cdot p_2^{k_2} \cdot \dots$  rồi tính  $n!$  theo modulo bất kỳ với thuật toán lũy thừa nhanh.

Dưới đây là code được tham khảo từ [Geeksforgeeks](#)  :

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N = 1000000; // 10^6
4  vector<int> primes;
5
6  // Công thức Legendre
7  int largestPower(int n, int p){
8      int x = 0;
9      while (n) {
10         n /= p;
11         x += n;
12     }
13     return x;
14 }
15
16 --
```

```

16 int binpow(int x, int y, int mod){
17     int res = 1;
18     x = x % mod;
19     while (y) {
20         if (y & 1)
21             res = (1LL * res * x) % mod;
22         y >>= 1;
23         x = (1LL * x * x) % mod;
24     }
25     return res;
26 }
27
28 // Sàng nguyên tố
29 void prepare(){
30     bool isPrime[N + 1];
31     memset(isPrime, 1, sizeof(isPrime));
32     isPrime[0] = isPrime[1] = 0;
33     for (int i = 2; i * i <= N; i++) {
34         if (isPrime[i]) {
35             for (int j = i * i; j <= N; j += i) {
36                 isPrime[j] = 0;
37             }
38         }
39     }
40     for (int i = 2; i <= N; i++) {
41         if (isPrime[i]) {
42             primes.push_back(i);
43         }
44     }
45 }
46
47 int modFact(int n, int mod){
48     int res = 1;
49     for (int p : primes) {
50         if (p <= n) {
51             int k = largestPower(n, p);
52             res = (1LL * res * binpow(p, k, mod)) % mod;
53         }
54         else {
55             break;
56         }
57     }
58     return res;
59 }
60
61 int main(){
62     prepare();
63     int n, mod;
64     //

```

```

04 |     cin >> n >> mod;
65 |     cout << modFact(n, mod);
66 |     return 0;
67 | }

```

Độ phức tạp thời gian:

- ▶ Chuẩn bị:  $O(N \log \log N)$
- ▶ Truy vấn:  $O(N)$

## Chia căn

- ▶ *Điều kiện sử dụng*:  $n < p \leq 2 \cdot 10^9$  ( $p$  không cần phải nguyên tố).  
Có thể chạy đến  $5 \cdot 10^{10}$  tùy thuộc vào lượng hằng số có thể sử dụng.
- ▶ *Ý tưởng*:
  - ▶ Chia đoạn  $[1; p]$  thành các đoạn riêng biệt có kích thước  $S$ , ta sẽ có tổng cộng  $\text{CNT} = \lfloor \frac{p}{S} \rfloor + 1$  đoạn.
  - ▶ Ta sẽ tính sẵn một mảng gồm CNT giá trị sau:  $0!, S!, (2S)!, \dots, (\lfloor \frac{p}{S} \rfloor \cdot S)!$  với độ phức tạp  $O(p)$ . Điều này không có ý nghĩa lắm. Vì thế, ta chỉ cần chạy code ra chương trình khác rồi lấy kết quả vào bài, bước này sẽ chỉ còn độ phức tạp  $O(\frac{p}{S})$  nhưng lại tăng độ dài của code!
  - ▶ Khi này, chọn  $k = \lfloor \frac{n}{S} \rfloor$ , ta có:  $n! = (k \cdot S)! \times \underbrace{(k \cdot S + 1) \dots n}_{\text{Duyệt mất tối đa } 10^6 \text{ phép tính}}$
- ▶ *Code C++ minh họa*:
  - ▶ Phần chuẩn bị:

```

1 | const int MOD = 1e9 + 7;
2 | const int S = 1e7;
3 |
4 | // Phần này thực hiện ở code khác
5 | // và lấy kết quả vào mảng fact[] trong code chính.
6 | void prepare(){
7 |     int tmp = 1;
8 |     for (int i = 1; i < MOD; i++){
9 |         tmp = 1LL * tmp * i % MOD;
10 |         if (i % S == 0) {
11 |             cout << tmp << ", ";
12 |         }
13 |     }
14 | }

```

Kết quả in ra màn hình là một dãy: 1, 641102369, 578095319, 5832229, ...  
Độ phức tạp phần chuẩn bị này là  $O(p)$ .

► Phần code chính:

```
1  const int MOD = 1e9 + 7;
2  const int S = 1e7;
3
4  // Dãy được copy từ kết quả in ra của code bên trên
5  int fact[] = {1, 641102369, 578095319, 5832229, ...};
6
7  int get_fact(int n) {
8      int t = n / S;
9      int res = fact[t];
10     for (int i = t * S + 1; i <= n; i++)
11         res = 1LL * res * i % MOD;
12     return res;
13 }
```


► Đánh giá:

- Độ phức tạp không gian:  $O\left(\frac{p}{S}\right)$
- Độ phức tạp thời gian:
  - Tiền xử lý (để lưu mảng `fact[]`):  $O\left(\frac{p}{S}\right)$
  - Truy vấn  $O(S)$

Tùy thuộc vào giới hạn bài toán, giới hạn độ dài của code, ta sẽ chọn  $S$  sao cho phù hợp.

## Biến đổi Fourier nhanh (FFT)

“

Trong bài này, ta sẽ sử dụng dạng Biến đổi số học - Number-theoretic transform (NTT) của FFT. Bạn đọc tham khảo tại [đây](#) .

Đây là thuật toán nhân hai đa thức bậc không vượt quá  $n$  với độ phức tạp  $O(n \log n)$ .

- Điều kiện sử dụng:  $n < p \leq 10^{12}$
- Ý tưởng: Ta sẽ tính  $n! \bmod p$  như sau:

Xét  $m = \lfloor \sqrt{n} \rfloor$  và đa thức  $P(x) = (x+1)(x+2)\dots(x+m) = \prod_{i=1}^m (x+i)$

Khi này:

$$n! = \underbrace{P(0) \cdot P(m) \cdot P(2m) \dots P((m-1)m)}_{\text{chỉ nh là } 1 \cdot 2 \cdot 3 \dots m^2} \cdot \underbrace{(m^2+1) \dots n}_{\text{tỉ nh đ ược trong } O(\sqrt{n})} \pmod{p}$$

## Cách 1: Sử dụng FFT đa điểm (FFT multipoint evaluation)



Cách này sử dụng rất nhiều kiến thức khó và mang giá trị về nghiên cứu là chính. Vì thế, để tham khảo chi tiết, bạn đọc tham khảo tại [đây](#) . Tuy nhiên bạn đọc có thể tham khảo các bài toán nhỏ của nó.

- *Kiến thức sử dụng:* Biến đổi Fourier nhanh (FFT), chia để trị, chia đa thức.
- Đầu tiên, ta sử dụng **FFT/NTT** để khai triển đa thức  $P(x)$  bên trên. Bạn đọc tham khảo bài tương tự của bước này: [960G - Bandit Blues - Codeforces](#) .
- Sau đó, ta tiếp tục sử dụng **FFT/NTT** để tính giá trị  $P(x) \bmod M$  tại  $m$  điểm:  $k \cdot m$  với  $k = 0..(m-1)$ . Đó cũng chính là bài toán [POLYEVAL - Codechef](#) hoặc [Library Checker - Multipoint Evaluation](#) .
- Chú ý rằng [POLYEVAL - Codechef](#) là bài toán dễ hơn, và bạn có thể sử dụng chia để trị với không gian lưu trữ  $O(p)$ . Để làm được bài [Library Checker - Multipoint Evaluation](#) , ta sử dụng thuật toán chia để trị và chia đa thức để đạt được không gian lưu trữ  $O(\sqrt{p})$ :
  - Ta thấy,  $P(x) = Q(x) \cdot (x - k) + P(k) \implies P(k) = P(x) \bmod (x - k)$ .
  - Giả sử cần tính  $P(x)$  tại  $x_1, x_2, \dots, x_n$ . Mỗi lần, ta chia  $P(x)$  thành 2 đa thức (Chú ý rằng, phép chia đa thức sẽ được chuyển thành nhân với đa thức nghịch đảo để sử dụng FFT):

$$P_0(x) = P(x) \bmod \left( (x - x_1)(x - x_2) \cdots (x - x_{\lfloor n/2 \rfloor}) \right)$$

$$P_1(x) = P(x) \bmod \left( (x - x_{\lfloor n/2 \rfloor + 1}) \cdots (x - x_n) \right)$$

$$\begin{aligned} P(x) &= \left( (x - x_1)(x - x_2) \cdots (x - x_{\lfloor n/2 \rfloor}) \right) \cdot Q_0(x) + P_0(x) \\ \implies P(x) &= \left( (x - x_{\lfloor n/2 \rfloor + 1}) \cdots (x - x_n) \right) \cdot Q_1(x) + P_1(x) \end{aligned}$$

$$\implies P(x_i) = \begin{cases} P_0(x_i) & \text{nếu } i \leq \lfloor n/2 \rfloor \\ P_1(x_i) & \text{còn lại} \end{cases}$$

- Bài toán trở thành: Tính  $P_0(x)$  tại  $x_1, \dots, x_{\lfloor n/2 \rfloor}$  và tính  $P_1(x)$  tại  $x_{\lfloor n/2 \rfloor + 1}, \dots, x_n$ .
- *Đánh giá:*
  - Độ phức tạp không gian:  $O(\sqrt{p})$
  - Độ phức tạp thời gian:  $O(\sqrt{p} \log^2 p)$

## Cách 2: Kết hợp sử dụng nội suy Lagrange

- *Kiến thức sử dụng:* Biến đổi Fourier nhanh (FFT), nội suy Lagrange (Lagrange Interpolation).
- *Ý tưởng:*

► Đặt  $P_d(x) = \prod_{i=1}^d (x + i)$  là đa thức bậc  $d$ .

► Đặt tập  $G_d(x) = \left\{ P_d(x), P_d(m+x), P_d(2m+x), \dots, P_d((m-1)m+x) \right\}$

Mục tiêu của ta là tính tập  $G_m(0)$ .

► Nhận thấy:  $P_1(0) = 1$  và

$$P_d(x) = \begin{cases} P_{d/2}(x) \cdot P_{d/2}(x + d/2) & \text{nếu } d \text{ chẵn} \\ P_{\lfloor d/2 \rfloor}(x) \cdot P_{\lfloor d/2 \rfloor}(x + \lfloor d/2 \rfloor) \cdot (x + d + 1) & \text{nếu } d \text{ lẻ} \end{cases}$$

► Như vậy để tính tập  $G_d(0)$ , ta cần tính tập  $G_{d/2}(0)$  và  $G_{d/2}(d/2)$ . Để tính tập  $G_{d/2}(d/2)$  từ tập  $G_{d/2}(0)$ , ta cần bài toán ngay dưới đây:

► Cho đa thức  $h(x)$  có bậc là  $d$  và biết  $h(0), h(1), \dots, h(d)$ . Ta cần tính  $h(x), h(x+1), \dots, h(x+d')$  với nội suy Lagrange.  $\forall k = 1, 2, \dots, d'$ , ta có:

$$\begin{aligned} h(x+k) &= \sum_{i=0}^d h(i) \prod_{j=0, j \neq i}^d \frac{x+k-j}{i-j} \\ &= \left( \prod_{j=0}^d (x+k-j) \right) \underbrace{\left( \sum_{i=0}^d \underbrace{\frac{h(i)}{i!(d-i)!(-1)^{d-i}}}_{\text{Hệ số bậc } i} \cdot \underbrace{\frac{1}{x+k-i}}_{\text{Hệ số bậc } x+k-i} \right)}_{\text{Sử dụng FFT/NTT ở đây, hệ số bậc } x+k} \end{aligned}$$

► Áp dụng với  $h(i) = P_d(i \cdot m)$  và  $x = d \cdot m^{-1}$ .

Trước đó, ta đã tính tập  $G_d(0) = \{h(0), h(1), \dots, h(d)\}$

Ta cũng sẽ tính tiếp được  $h(x+i) = P_d((x+i) \cdot m) = P_d(i \cdot m + d)$

Và đây cũng chính là tập  $G_d(d)$  cần tính!

► Code C++ minh họa (lược bỏ phần FFT)

```
int MOD;

// hàm trả về nghịch đảo x modulo MOD
long long inv(long long x);

// hàm trả về đa thức a * b
vector<long long> NTT(vector<long long> a, vector<long long> b);

void mul(long long &x, long long y){
    x = __int128(x) * y % MOD;
}

// Biết h(0), h(1), ..., h(d)
// Hàm trả về h(m), h(m+1), ..., h(m+cnt-1)
// m > d
vector<long long> Lagrange(vector<long long> h, long long m, int c
    int d = h.size() - 1;
```

```

// tính  $h[i] = (-1)^{(d-i)} h(i)/(i! (d-i)!)$ 
// hệ số  $x^i$ 
for (int i = 0; i <= d; i++){
    mul(h[i], (ifact[i] * ifact[d - i]) % MOD);
    if ((d - i) & 1)
        h[i] = (MOD - h[i]) % MOD;
}

//tính  $f[i] = 1/(m+i-d)$ 
// hệ số  $x^{(m + cnt - 1 - i)}$ 
vector<long long> f(d + cnt);
long long now = m - d;

for (int i = 0; i < d + cnt; i++)
    f[i] = inv(now+i);

// Nhân 2 đa thức và hệ số được lấy mod p
h = NTT(f, h);
h.resize(d + cnt);
// chỉ lấy hệ số  $x^{(m)}$  đến  $x^{(m+cnt-1)}$ 
h = vector<long long>(h.begin() + d, h.end());
now = 1;

for (int i = 0; i <= d; i++)
    mul(now, m - i);

mul(h[0], now);

for (int i = 1; i < cnt; i++){
    mul(now, m + i);
    mul(now, inv(m + i - d - 1));
    mul(h[i], now);
}

return h;
}

long long factorial(long long n, long long p){
    if (n >= p) return 0;
    if (n < 2) return 1;
    int s = __builtin_sqrtl(n);
    MOD = p;
    vector<long long> h{1, s + 1};
    for (int bit = __lg(s) - 1, d = 1; bit >= 0; bit--){
        // Hiện tại  $h(i) = (i * s + 1) * (i * s + 1) \dots (i * s +$ 
        // Tính  $h(d+1), \dots, h(2d)$ 
        auto nh1 = Lagrange(h, d + 1, d);
        // Tính  $h(d \cdot inv(s)), \dots, h(d \cdot inv(s) + 2d)$ 

```



```

66 // Như vậy, nh2(i) = (i * s + d + 1) * (i * s + d + 2) ...
67 auto nh2 = Lagrange(h, 1LL * inv(s) * d % mod, 2 * d + 1);
68 // h giờ đây là h(0), h(1), ..., h(2d)
69 h.insert(h.end(), nh1.begin(), nh1.end());
70 // d --> d * 2
71 d <= 1;
72
73 // Hiện tại h(i) = (i * s + 1) * (i * s + 2) ... (i * s +
74 // Còn nh2(i) = (i * s + d/2 + 1) * (i * s + d/2 + 2) ...
75 for (int i = 0; i <= d; i++)
76     h[i] *= nh2[i];
77 // Tại đây h(i) = (i * s + 1) * (i * s + 1) ... (i * s + c
78
79 // Nếu bit hiện tại của s là 1
80 if (s >> bit & 1){
81     d |= 1;
82     long long tmp = d;
83
84     // h[i] *= (d + i * s)
85     for (int i = 0; i < d; i++, tmp += s)
86         mul(h[i], tmp);
87
88     long long last = 1, tj = 1LL * s * d;
89
90     // last = (d*s+1)(d*s+2)...(d*s+d)
91     for (int i = 1; i <= d; i++)
92         tj++, last *= tj;
93
94     // Thêm biến last vào h
95     h.emplace_back(last);
96 }
97 }
98 long long ans = 1;
99
100 for (int i = 0; i < s; i++)
101     mul(ans, h[i]);
102
103 for (long long i = 1LL * s * s + 1; i <= n; i++)
104     mul(ans, i);
105
106 return ans;
107 }

```

► *Đánh giá:*

► Độ phức tạp không gian:  $O(\sqrt{n})$

► Độ phức tạp thời gian:

► Hàm Lagrange ở phần sử dụng sử dụng FFT có độ phức tạp  $O(N \log N)$  với  $N = \deg(h) + \text{cnt}$ .

Các vòng lặp khác chỉ có độ phức tạp tối đa đến  $O(N)$ .

► Hàm factorial có độ phức tạp lớn nhất ở phần sử dụng hàm Lagrange với tổng cộng:

$$O\left(s \log s + \frac{s}{2} \log \frac{s}{2} + \frac{s}{4} \log \frac{s}{4} + \dots\right) \quad \text{với} \quad s = \lfloor \sqrt{n} \rfloor. \quad \text{Chú ý rằng:}$$
$$s \log s + \frac{s}{2} \log \frac{s}{2} + \frac{s}{4} \log \frac{s}{4} + \dots < s \log s + \frac{s}{2} \log s + \frac{s}{4} \log s + \dots$$
$$= \log s \cdot \left(s + \frac{s}{2} + \frac{s}{4} + \dots\right)$$
$$< 2s \log s$$

► Điều này có nghĩa là bài toán có tổng độ phức tạp:  $O(\sqrt{n} \log n)$

## Bài tập luyện tập

[Hackerearth - Army Parade](#) 

[SPOJ - FACTMODP](#) 

## Tài liệu tham khảo

[GeeksforGeeks - Compute n! under modulo p](#) 

[FFT - multipoint evaluation](#) 

[Factorial mod prime - Prabowo Djonatan](#) 

[Codeforces - Fastest way to get factorial modulo a prime](#) 

[Codeforces - How to calculate n!%p for very large 'n' ?](#) 

Được cung cấp bởi [Wiki.js](#)