Tính $(a^m)^{b^n}$ % K

Mình biết là có thể có rất nhiều bạn AC bằng các cách khác nhau rất hay, đương nhiên là cũng rất nặng toán. Song mình chưa thấy có lời giải chính thức nào được đề xuất nên mình xin mạn phép bày ra ý kiến của mình cho bài toán này. Lời giải sau sẽ vô cùng trâu và không phù hợp cho những bạn nào thích code ngắn, giỏi toán hoặc anti mình :>

Chắc các bạn cũng biết sử dụng chia để trị để tính nhanh giá trị của lũy thừa số mũ nguyên rồi, nên mình sẽ không nhắc lại nữa và đi thẳng vào bài toán luôn

Trước tiên, chúng ta có thể thay a^m thành a^m % K mà không ảnh hưởng đến kết quả bài toán, gọi số này là c. Bỏ qua trường hợp c chia hết cho K, ta chỉ xét trường hợp còn lại

Để tính nhanh được biểu thức trên, cần tìm một số x sao cho $c^{b^n} \equiv c^x \ (mod\ K)$

Hay nói cách khác $c^{b^n - x} \equiv 1 \ (mod\ K)$

Mà ta đã biết: Nếu như $\gcd(a, b) = 1$ thì $a^{\varphi(b)} \equiv 1 \ (mod\ B)$ với φ là hàm phi Euler

Như vậy, nếu điều kiện $\gcd(c, K) = 1$ được thoả mãn thì $(a^m)^{b^n} \equiv (a^m)^{b^n \,\%\, \varphi(K)} \ (mod\ K)$

Ngược lại, nếu như $\gcd(c, K) > 1$ thì phải làm thế nào ?

Mình thấy trong code một số bạn có xử lí như sau:

$$v = \gcd(c, K)$$

$$ans = \left(\frac{c}{v}\right)^{b^n \% \varphi(K)} * v^{b^n} \% K$$

Về ý tưởng, các bạn đã đi theo đúng hướng, tách $c = v * u$ sao cho $\gcd(v, K) = 1$ rồi tính riêng u^{b^n} và v^{b^n}, song cách xử lí trên chưa hợp lí vì chưa chắc $\gcd(\frac{c}{v}, K) = 1$

Cho rằng chúng ta đã tách được $c = v * u$ ($\gcd(v, K) = 1$) thỏa mãn điều kiện, việc tính v^{b^n} đã vô cùng dễ dàng, chúng ta sẽ bàn về cách tính u^{b^n}

Có 2 trường hợp có thể xảy ra:

- Nếu tồn tại số z ($z \le b^n$) sao cho u^z chia hết cho K, tức là $z \le \log_2 K < 30$ (*) , tức là chỉ cần tìm b^t sao cho $t \le n$ và $b^t < 30$ và z^{b^t} chia hết cho K là được
- Ngược lại, nếu như không tồn tại số z nói trên thì việc tính toán vô cùng khó khan nếu chỉ dùng các phép toán trực tiếp. Vậy mình đề xuất cách xử lí như sau:

Xét phân tích tiêu chuẩn của K:

$$K = p_1^{t_1} p_2^{t_2} \dots p_h^{t_h}$$

Ta sẽ tính $\begin{cases} u^{b^n} \% p_1^{t_1} \\ u_{b^n} \% p_2^{t_2} \\ ... \\ u^{b^n} \% p_h^{t_h} \end{cases}$ rồi tính $u^{b^n} \% K$ dựa vào Chinese Remain Theorem

Lúc này ta cũng làm tương tự như c

Tách $u = p_i^{l_i} * q$ $(\gcd(q, p_i) = 1)$, do đó $q^{b_n} \equiv q^{b^n \% \varphi\left(p_i^{t_i}\right)} \ (mod\ p_i^{t_i})$ và chỉ cần xử lí phần $p_i^{l_i}$

- Nếu như $l_i * b^n \geq t_i$, rõ ràng u chia hết cho $p_i^{t_i}$
- Ngược lại chứng minh tương tự như $(*)$ ta có ngay $b^n \leq l_i * b^n < 30$ hoặc $l_i = 0$, có thể duyệt rất nhanh

Chốt lại, cách làm như sau:

B1: Tính $c = a^m \% K$

B2: Tách $c = v * u$ trong đó $\gcd(v, K) = 1$, tính nhanh giá trị $v^{b^n} \equiv v^{b^n \% \varphi(K)} \ (mod\ K)$

B3: Tính u^{b^n} bằng cách tính u^{b^n} cho từng thừa số nguyên tố của K và ghép lại sử dụng Định lý thặng dư trung hoa

Code mình tại đây:

```
#include <bits/stdc++.h>

using namespace std;

using ll = long long;


constexpr int N = 1e2 + 5;

ll a, b, m, n, mod;

struct Pair

{

    ll first, second, third;

    Pair(ll x = 0, ll y = 0, ll z = 0) : first(x), second(y), third(z) {}

};


void Read()

{

    cin >> a >> m >> b >> n >> mod;

}
```

```
ll Pow(ll a, ll b, ll mod)
{
    a %= mod;
    ll ans(1);
    for (; b; b >>= 1)
    {
        if (b & 1)
            ans = ans * a % mod;
        a = a * a % mod;
    }
    return ans;
}


ll ChineseRemainTheory(vector<Pair> s)
{
    ll ans(0), mod(1);
    for (auto i : s)
        mod *= i.second;
    vector<ll> pref(s.size()), suf(s.size());
    for (int i = 0; i < (int)s.size(); ++i)
    {
        pref[i] = i == 0 ? s[i].second : (pref[i - 1] * s[i].second % mod);
        suf[s.size() - 1 - i] = i == 0 ? s[s.size() - 1].second : (suf[s.size() - i] *
s[s.size() - 1 - i].second % mod);
    }
    for (int i = 0; i < (int)s.size(); ++i)
    {
        //cout << s[i].first << " " << s[i].second << " " << s[i].third << "\n";
        ll tmp(s[i].first), tmp2(1);
        tmp = (tmp2 = (i == 0 ? 1 : pref[i - 1]) * (i < (int)s.size() - 1 ? suf[i + 1]
: 1) % mod) * s[i].first % mod;
```

```
        (ans += tmp * Pow(tmp2, s[i].third - 1, mod)) %= mod;
    }
    return ans;
}

ll Get(ll mod, ll phi, ll factor, int cnt)
{
    ll v = Pow(a, m, mod);
    if (v == 0)
        return 0;
    ll tmp(0);
    while (v % factor == 0)
    {
        v /= factor;
        ++tmp;
    }
    ll ans = Pow(v, Pow(b, n, phi), mod);

    if (b != 1 && tmp != 0)
        for (int i = 0; i <= n; tmp *= b, ++i)
        {
            if (tmp >= cnt)
                return 0;
            tmp *= b;
        }

    return ans * Pow(factor, tmp, mod) % mod;
}

void Solve()
{
```

```
    vector<Pair> s;
    for (int i = 2; i * i <= mod; ++i)
        if (mod % i == 0)
        {
            int tmp(1), phi(1), cnt(0);
            while (mod % i == 0)
            {
                mod /= i;
                tmp *= i;
                ++cnt;
            }
            phi = tmp / i * (i - 1);
            s.emplace_back(Get(tmp, phi, i, cnt), tmp, phi);
        }
    if (mod != 1)
        s.emplace_back(Get(mod, mod - 1, mod, 1), mod, mod - 1);
    cout << ChineseRemainTheory(s) << '\n';
}

int32_t main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    Read();
    Solve();
}
```