Lyft Level 5 Challenge 2018 – Elimination Round

Problema D: *Divisors*

Prof. Edson Alves – UnB/FGA

Codeforces 1033D – Divisors

You are given n integers $a_1, a_2, \ldots, a_n$. Each of a_i has between 3 and 5 divisors. Consider $a = \prod a_i$ – the product of all input integers. Find the number of divisors of a. As this number may be very large, print it modulo prime number 998244353.

Entrada e saída

Input

The first line contains a single integer n $(1 \leq n \leq 500)$ *– the number of numbers.*

Each of the next n *lines contains an integer* a_i $(1 \leq a_i \leq 2 \times 10^{18})$*. It is guaranteed that the number of divisors of each* a_i *is between 3 and 5.*

Output

Print a single integer d *– the number of divisors of the product* $a_1 \cdot a_2 \cdot \ldots \cdot a_n$ *modulo 998244353.*

Exemplos de entrada e saída

Entrada	Saída
3 9 15 143	32
1 7400840699802997	4

Solução $O(n^2)$

- A restrição do número de divisores de um elemento a_i implica em apenas 4 cenários distintos, onde p e q são primos distintos
    1. $a_i = p^2$
    2. $a_i = p^3$
    3. $a_i = pq$
    4. $a_i = p^4$
- No primeiro caso, $\tau(p^2) = 2 + 1 = 3$, ou seja, a_i tem 3 divisores
- Nos casos 2 e 3 temos 4 divisores, pois $\tau(p^3) = 3 + 1 = 4$ e $\tau(pq) = (1 + 1)(1 + 1) = 4$
- No último caso $\tau(p^4) = 5$
- A solução portanto, depende da identificação destes casos e do devido tratamento dado a eles

Solução $O(n^2)$

- É possível determinar se um número n é um quadrado perfeito por meio de uma rotina baseada em busca binária com complexidade $O(\log n)$
- Esta rotina pode identificar os casos 1 e 4
- Uma rotina semelhante identifica se n é um cubo perfeito também em $O(\log n)$, e pode identificar o caso 2
- Nos casos 1, 2 e 4 os primos identificados pelas rotinas acima devem ser acumulados em um histograma que contém a fatoração do produto de todos os termos, de acordo com o expoente em questão

Solução $O(n^2)$

- O caso 3 é o mais difícil e que merece mais atenção e cuidado
- Uma vez que $a_i \leq 2 \times 10^{18}$, a fatoração de tais termos não pode ser feita por meio de um algoritmo *naive*
- Uma forma de obter esta fatoração é computar o maior divisor comum d entre todos os pares de números da forma $a = pq$
- Caso d seja um divisor próprio destes números e d não for uma chave do histograma da fatoração, ele deve ser registrado no histograma, inicialmente associado ao expoente zero

Solução $O(n^2)$

- Após este processamento, as chaves primas do histograma podem ser usadas numa tentativa de fatoração destes números
- Caso um número possa ser fatorado, os dois fatores devem atualizar o histograma
- Se o número não for fatorado, ele não compartilha primos com os demais número, exceto possivelmente com cópias idênticas de si mesmo
- Assim, tais números devem ser guardados em um segundo histograma

Solução $O(n^2)$

- Finalizado todos estes passos, a resposta pode ser computada a partir da entrada de ambos histogramas
- A resposta inicialmente é igual a 1
- Para todo par (p, k) do primeiro histograma, a resposta deve ser atualizada por meio de seu produto por $(k + 1)$
- Para todo par (x, c) do segundo histograma, a atualização deve ser por meio do produto por $(c + 1)^2$
- Ou seja, para cada conjunto de c repetições do número $x = p_j q_j$, a fatoração do produto dos a_i conterá os fatores p_j^c e q_j^c, e cada um contribui com um fator $(c + 1)$ no cálculo do número de divisores deste produto

Solução $O(n^2)$

```
ll is_square(ll n)
{
    ll a = 1, b = n;

    while (a <= b) {
        auto m = a + (b - a)/2;

        if (n/m == m and m*m == n)
            return m;
        else if (m < n/m)
            a = m + 1;
        else
            b = m - 1;
    }

    return -1;
}
```

Solução $O(n^2)$

```
ll is_cube(ll n)
{
    ll a = 1, b = n;

    while (a <= b) {
        auto m = a + (b - a)/2;

        if ((n/m)/m == m and m*m*m == n)
            return m;
        else if (m < (n/m)/m)
            a = m + 1;
        else
            b = m - 1;
    }

    return -1;
}
```

Solução $O(n^2)$

```
ll solve(const vector<ll>& as)
{
    map<ll, ll> fs, uniques;
    vector<ll> pqs;

    for (auto a : as)
    {
        auto s = is_square(a);

        if (s > 0)
        {
            auto p = is_square(s);

            // a = p^4 ou a = p^2
            p > 0 ?  fs[p] += 4 : fs[s] += 2;
        }
```

Solução $O(n^2)$

```
        else
        {
            auto c = is_cube(a);

            // a = p^3 ou a = pq
            c > 0 ?  (void) (fs[c] += 3) : pqs.push_back(a);
        }
    }

    for (auto x : pqs)
        for (auto y : pqs)
        {
            auto d = gcd(x, y);

            if (d > 1 and d < x and fs.count(d) == 0)
                fs[d] = 0;
        }
```

Solução $O(n^2)$

```
    for (auto x : pqs)
    {
        bool ok = false;

        for (auto [p, k] : fs)
            if (x % p == 0)
            {
                ++fs[p];
                ++fs[x / p];
                ok = true;
                break;
            }

        if (not ok)
            uniques[x]++;
    }
```

Solução $O(n^2)$

```
    ll ans = 1;

    for (auto [p, k] : fs)
        ans = (ans * (k + 1)) % MOD;

    for (auto [x, k] : uniques)
        ans = (ans * (k + 1)*(k + 1)) % MOD;

    return ans;
}
```