XXII Maratona de Programação (1° Fase)
Despojados

Daniel Saad Nogueira Nunes

O problema **Despojados** é sobre fatoração em números em primos e Análise Combinatória.

É sabido que todo número $x \in \mathbb{N}$ pode ser decomposto (fatorado) em fatores primos $p_0, \ldots, p_{n-1}$ tal que:

$$x = p_0^{i_0} \cdot p_1^{i_1} \cdot \ldots \cdot p_{n-1}^{i_{n-1}} = \prod_{k=0}^{n-1} p_k^{i_k}$$

Exemplos de fatoração:

$$6 = 2^1 \cdot 3^1$$

$$28 = 2^2 \cdot 7$$

$$6469693230 = 2^1 \cdot 3^1 \cdot 5^1 \cdot 7^1 \cdot 11^1 \cdot 13^1 \cdot 17^1 \cdot 19^1 \cdot 23^1 \cdot 29^1$$

Adicionalmente, a questão vinha com uma definição extra, a de número **despojado**.

Definição 1 (Número despojado)**.** *Um número $x \in \mathbb{N}$ é dito despojado se pode ser escrito como um produto de dois ou mais primos distintos sem repetição.*

De acordo com esta definição, 2 não é despojado, pois $2 = 2^1$, e portanto só possui um fator primo. 28 não é despojado, pois $28 = 2^2 \cdot 7$, ou seja, possui um fator primo elevado a uma potência maior que 1. Já $6469693230 = 2^1 \cdot 3^1 \cdot 5^1 \cdot 7^1 \cdot 11^1 \cdot 13^1 \cdot 17^1 \cdot 19^1 \cdot 23^1 \cdot 29^1$ é despojado, pois possui mais de um fator primo distinto, e nenhum fator está elevado a uma potência maior do que 1.

No entanto, o problema não está interessado em saber se um número é despojado ou não. O que interessa é saber quantos divisores de um número são despojados. Tomando em conta o número 28, sabemos que ele possui os divisores $D = \{1, 2, 4, 7, 14, 28\}$, dos quais apenas $D' = \{14\}$ é um divisor despojado, visto que $14 = 2 \cdot 7$.

Como calculamos os divisores de um número? Podemos usar a fatoração de números primos. Observe que

$$28 = 2^2 \cdot 7^1$$

Então, os divisores de 28 são:

$$1 = 2^0 \cdot 7^0$$

$$2 = 2^1 \cdot 7^0$$
$$4 = 2^2 \cdot 7^0$$
$$7 = 2^0 \cdot 7^1$$
$$14 = 2^1 \cdot 7^1$$
$$28 = 2^2 \cdot 7^1$$

Ou seja, os divisores são obtidos combinando os possíveis valores das potências. Mas a questão é mais específica, queremos saber quantos destes divisores são despojados. Portanto, não estamos interessados em nenhuma combinação que possua um primo com potência maior que 1.

Vamos generalizar este raciocínio. Suponha que um número $x \in \mathbb{N}$ seja fatorado em n números primos:

$$x = p_0^{i_0} \cdot p_1^{i_1} \cdot \ldots \cdot p_{n-1}^{i_{n-1}}$$

Só estamos interessados nas combinações de potências quando elas são menores ou iguais a 1, caso contrário, o divisor não pode ser despojado. Agora entra a parte de análise combinatória da questão. Quantas possibilidades de combinações de fatores primos podemos ter de modo que as potências dos mesmos sejam menores ou iguais a 1?

A resposta é 2^n, todos os subconjuntos possíveis. Dado que temos n primos distintos compondo x, temos 2^n possibilidades de arranjá-los de modo que suas potências sejam menores ou iguais a 1. Tomando como exemplo o $28 = 2^2 \cdot 7$, podemos arranjar os primos da seguinte maneira: $\{\emptyset, \{2^0 \cdot 7^0\}, \{2^1 \cdot 7^0\}, \{2^1 \cdot 7^1\}\}$. Mas ainda não terminamos, existem subconjuntos que possuem zero ou um fatores primos, mas eles não podem ser considerados divisores despojados, precisamos retirá-los da conta. Assim, dado que temos n primos distintos compondo um número x, o número de divisores despojados é

$$2^n - n - 1$$

Uma vez que temos n possibilidades de subconjuntos contendo apenas 1 fator primo e uma possibilidade de conjunto vazio.

Assim, a estratégia para resolução do problema é:

1. Fatore o número x em seus primos.
2. Conte quantos números primos distintos compões x e chame isso de n.
3. Retorne $2^n - n - 1$.

Solução

De acordo com a discussão anterior, obtemos o Algoritmo 1. Para não levar **TLE**, foi necessário fazer uma otimização. Como $x \leq 10^{12}$ um algoritmo força-bruta não vai passar. Desta forma, tentamos fatorar o número de acordo com todos os candidatos a primo menores que a raiz do número original. A partir do momento que o candidato a primo torna-se maior que a raiz do número original podemos parar a busca.

Isto funciona pois, se um número x não é primo, ele pode ser escrito como $x = p \cdot q$. E obrigatoriamente temos $2 \leq p \leq \sqrt{x}$ ou $2 \leq q \leq \sqrt{x}$. Assim, se não achamos um número até a raiz de x que divide x, sabemos que x é primo.

Algorithm 1: numeros-despojados

Input: x, número inicial
Output: d número de divisores despojados.

```
1  cont ← 1 // contador de fatores primos
2  k ← 2 // candidato a primo
3  raiz ← √x // raiz da entrada
   // enquanto o número não tiver sido fatorado por completo, ou o
      candidato a primo for menor ou igual a raiz de x, continue fatorando
4  while( (x > 1) ∧ (k ≤ raiz) )
5      if( x mod k = 0 )
           // Se x é divisível pelo primo k, incremente o contador de
              fatores primos
6          cont++
7          while( x mod k = 0 )
               // divida x pelo primo até não conseguir mais
8              x ← x/k
       // passe para o próximo candidato a primo
9      k++
   // Se a fatoração terminou com x > 1, significa que k > raiz e portanto,
      o que sobrou em x é primo
10 if( x > 1 )
11     cont++
12 return 2^cont − cont − 1
```

Complexidade

A solução tem tempo de pior caso $O(\sqrt{x})$, pois vamos até no máximo à raiz do número original, conforme linha 4.