Matemática

Funções Multiplicativas

Prof. Edson Alves
Faculdade UnB Gama

Funções Multiplicativas

Uma função é denominada função **aritmética** (ou **número-teórica**) se ela tem como domínio o conjunto dos inteiros positivos e, como contradomínio, qualquer subconjunto dos números complexos.

Uma função f aritmética é denominada função **multiplicativa** se

1. $f(1) = 1$
2. $f(mn) = f(m)f(n)$ se $(m, n) = 1$

Número de Divisores

Seja n um inteiro positivo. A função $\tau(n)$ retorna o número de divisores positivos de n.

Cálculo de $\tau(n)$

- Segue diretamente da definição que $\tau(1) = 1$
- Se $n = p^k$, para algum primo p e um inteiro positivo k, d será um divisor de n se, e somente se, $d = p^i$, com $i \in [0, k]$
- Assim, $\tau(p^k) = k + 1$
- Se $(a, b) = 1$ e p é um primo que divide ab, então ou p divide a ou p divide b

Cálculo de $\tau(n)$

- Se d divide ab, então ele pode ser escrito como $d = mn$, com $(m, n) = 1$
- Logo, $\tau(ab) = \tau(a)\tau(b)$, ou seja, $\tau(n)$ é uma função multiplicativa
- Considere a fatoração

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$$

- Portanto,

$$\tau(n) = \prod_{i=1}^{k} (\alpha_i + 1) = (\alpha_1 + 1)(\alpha_2 + 1) \dots (\alpha_k + 1)$$

Implementação de $\tau(n)$ em C++

```cpp
long long number_of_divisors(int n, const vector<int>& primes)
{
    auto fs = factorization(n, primes);
    long long res = 1;

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

    return res;
}
```

Cálculo de $\tau(n)$ em competições

- Em competições, é possível computar $\tau(n)$ em $O(\sqrt{n})$ diretamente, sem recorrer à fatoração de n
- Isto porque, se d divide n, então $n = dk$ e ou $d \leq \sqrt{n}$ ou $k \leq \sqrt{k}$
- Assim só é necessário procurar por divisores de n até $\sqrt{n}$
- Caso um divisor d seja encontrado, é preciso considerar também $k = n/d$
- Esta abordagem tem implementação mais simples e direta, sendo mais adequada em um contexto de competição

Implementação $O(\sqrt{n})$ de $\tau(n)$

```
long long number_of_divisors(long long n)
{
    long long res = 0;

    for (long long i = 1; i * i <= n; ++i)
    {
        if (n % i == 0)
            res += (i == n/i ? 1 : 2);
    }

    return res;
}
```

Soma dos Divisores

Um problema semelhantes ao anterior é determinar a soma de todos os divisores de *n*. Há dois
algoritmos possíveis, variantes dos dois anteriores. O primeiro deles é baseado na fatoração,
e é apresentado a seguir.

```
long long sum_of_divisors(int n, const vector<int>& primes)
{
    auto fs = factorization(n, primes);
    long long res = 1;

    for (const auto& f : fs)
    {
        int p = f.first;
        int k = f.second + 1;

        long long temp = 1;
```

Função Phi de Euler

A função Phi de Euler (`phi(n)`) retorna o número de inteiros positivos menores ou iguais a *n*
que são coprimos com *n*. É fácil ver que `phi(1) = 1` e que `phi(p) = p - 1`, se `p` é primo.
Menos óbvio são os fatos de que `phi(mn) = phi(m)phi(n)`, se $(m, n) = 1$ e que `phi(p^k) = p^{k - 1}(p - 1)`. Este dois últimos fatos nos permitem computar o valor de `phi(n)` a partir
da fatoração de *n*.

```
int phi(int n, const vector<int>& primes)
{
    if (n == 1)
        return 1;

    auto fs = factorization(n, primes);
```

Referências

1. Mathematics LibreTexts. [4.2 Multiplicativa Number Theoretic Functions]. Acesso em 10/01/2021.
2. Wikipédia. Arithmetic function. Acesso em 10/01/2021.
3. Wikipédia. Multiplicative function. Acesso em 10/01/2021.