Matemática

Teorema Fundamental da Aritmética

Prof. Edson Alves
Faculdade UnB Gama

Teorema Fundamental da Aritmética

- O Teorema Fundamental da Aritmética apresenta a relação fundamental dos números primos com todos os números naturais
- Ele afirma que todo $n > 1$ natural ou é primo ou é escrito de forma única (a menos de ordem) como o produto de primos
- Seja $n > 1$, p_i o i-ésimo número primo e $\alpha_i \geq 0$, para todo $i \in [1, k]$. Então

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

Fatoração de inteiros

- O conhecimento da fatoração (decomposição) de um natural n como produto de primos permite o cálculo de várias funções importantes, como MDC, MMC, número de divisores, soma dos divisões, função φ de Euler, etc
- A fatoração serve como alternativa para a representação do número, principalmente quando o número é muito grande (maior do que a capacidade de um `long long`, por exemplo)
- A fatoração pode ser implementada em $O(\pi(\sqrt{n}) \log n)$, se forem conhecidos os primos

```cpp
map<int, int> factorization(int n, const vector<int>& primes)
{
    map<int, int> fs;

    for (auto p : primes)
    {
        if (p * p > n)
            break;

        int k = 0;

        while (n % p == 0)
        {
            n /= p;
            ++k;
        }

        if (k)
            fs[p] = k;
    }

    if (n > 1)
        fs[n] = 1;

    return fs;
}
```

Fatoração de Fatoriais

- Uma aplicação importante da fatoração é a fatoração de fatoriais
- Os fatoriais crescem rapidamente, e mesmo para valores relativamente pequenos de n, o número $n!$ pode ser computacionalmente intratável
- A fatoração de $n!$ permite trabalhar com tais números e realizar algumas operações com os mesmos (multiplicação, divisão, MMC e MDC, etc)
- A função $E(n, p)$ retorna um inteiro k tal que p^k é a maior potência do primo p que divide $n!$

Exemplo de cálculo de $E(n, p)$

Para ilustrar o cálculo de $E(n, p)$ considere $n = 12$ e $p = 2$. A expansão de $12!$ é

```
1 x 2 x 3 x 4 x 5 x 6 x 7 x 8 x 9 x 10 x 11 x 12
```

É fácil observar que todos os múltiplos de 2 contribuem com um fator 2. Cancelando estes fatores obtém-se

```
1 x 1 x 3 x 2 x 5 x 3 x 7 x 4 x 9 x 5 x 11 x 6
```

Exemplo de cálculo de $E(n, p)$

Ainda restam ainda fatores 2 no produto, onde haviam originalmente os números $4, 8$ e 12. Isto acontece por, além de serem múltiplos de 2, os números $4, 8$ e 12 também são múltiplos de 2^2.

Eliminando os fatores 2 associados a 2^2 resulta em

```
        1 x 1 x 3 x 1 x 5 x 3 x 7 x 2 x 9 x 5 x 11 x 3
```

Exemplo de cálculo de $E(n, p)$

Mais 3 fatores foram eliminados, e sobrou ainda um fator, onde estava o 8. Isto acontece também porque 8 é múltiplo de 2^3. Eliminando este último fator, eliminamos um total de $6 + 3 + 1 = 10$. Portanto $E(12, 2) = 9$.

O exemplo acima fornece a expressão para o cálculo de $E(n, p)$, onde $p^r \leq n$:

$$E(n, p) = \left\lfloor \frac{n}{p} \right\rfloor + \left\lfloor \frac{n}{p^2} \right\rfloor + \ldots + \left\lfloor \frac{n}{p^r} \right\rfloor,$$

onde $\left\lfloor \frac{a}{b} \right\rfloor$ é a divisão inteira de a por b.

Implementação de $E(n, p)$ em $O(\log n)$

```
int E(int n, int p)
{
    int res = 0, base = p;

    while (base <= n)
    {
        res += n / base;
        base *= p;
    }

    return res;
}
```

Fatoração de $n!$ em $O(\pi(n) \log n)$

```
map<int, int> factorial_factorization(int n, const vector<int>& primes)
{
    map<int, int> fs;

    for (const auto& p : primes)
    {
        if (p > n)
            break;

        fs[p] = E(n, p);
    }

    return fs;
}
```

MDC, MMC e fatoração

Seja $a = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$ e $b = p_1^{\beta_1} p_2^{\beta_2} \ldots p_k^{\beta_k}$, com $\alpha_i, \beta_j \geq 0$ para todos $i, j \in [1, k]$. Então

$$(a, b) = p_1^{\min\{\alpha_1, \beta_1\}} p_2^{\min\{\alpha_2, \beta_2\}} \ldots p_k^{\min\{\alpha_k, \beta_k\}}$$

e

$$[a, b] = p_1^{\max\{\alpha_1, \beta_1\}} p_2^{\max\{\alpha_2, \beta_2\}} \ldots p_k^{\max\{\alpha_k, \beta_k\}}$$

Implementação do MDC

```cpp
int gcd(int a, int b, const vector<int>& primes)
{
    auto ps = factorization(a, primes);
    auto qs = factorization(b, primes);

    int res = 1;

    for (auto p : ps) {
        int k = min(ps.count(p) ? ps[p] : 0, qs.count(p) ? qs[p] : 0);

        while (k--)
            res *= p;
    }

    return res;
}
```

Implementação do MMC

```
int lcm(int a, int b, const vector<int>& primes)
{
    auto ps = factorization(a, primes);
    auto qs = factorization(b, primes);

    int res = 1;

    for (auto p : ps) {
        int k = max(ps.count(p) ? ps[p] : 0, qs.count(p) ? qs[p] : 0);

        while (k--)
            res *= p;
    }

    return res;
}
```

Número de Divisores

A fatoração de um número *n* também permite computar o número de divisores deste número: basta
fazer o produto de todos os expoentes da fatoração, somados cada um de uma unidade. Veja
o código abaixo.

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

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

        res *= (k + 1);
    }
```

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

PRIMES.UTM.EDU. How Many Primes Are There?. Acesso eme 08/11/2017.

WIKIPEDIA. Harmonic Number. Acesso em 08/11/2017.

WIKIPEDIA. Mertens' theorems. Acesso eme 08/11/2017.