Multiplicative Functions with Linear Sieve

Mazen Ghanayem

May 19, 2025

We often want to compute a **multiplicative function** $f(n)$ for all $1 \leq n < N$. Examples of multiplicative functions include:

- Euler's totient function $\phi(n)$
- Möbius function $\mu(n)$
- Sum of divisors $\sigma(n)$
- Number of divisors $d(n)$

Definition

A function f is **multiplicative** if:

- $f(1) = 1$
- $f(a \cdot b) = f(a) \cdot f(b)$ when $\gcd(a, b) = 1$

▶ Linear Sieve Overview

The **linear sieve** allows you to:

- Generate primes up to n in **O(n)** time
- Compute values of **any multiplicative function** in the same loop

---

■ Core Formula

Let p be a prime and i be a number already processed.
To compute $f(i \cdot p)$, there are 2 main cases:

♦ Case 1: $p \nmid i$ (i.e., p does not divide i)

- Then $\gcd(i, p) = 1$
- Since f is multiplicative:
$$f(i \cdot p) = f(i) \cdot f(p)$$

♦ Case 2: $p \mid i$ (i.e., p divides i)

- We already computed $f(i)$
- To compute $f(i \cdot p)$, use this formula:

$$f(i \cdot p) = \text{some rule based on } f(p^k)$$

- This depends on how your multiplicative function is defined on prime powers
- we can derive the formula of the $f(i \cdot p)$.

Suppose i has the form:

$$i = x \cdot p^{\text{cnt}[i]}$$

where p does **not** divide x (i.e., $\gcd(x, p) = 1$)

Then:

$$ip = x \cdot p^{\text{cnt}[i]+1}$$

Since x and p are coprime:

$$f(ip) = f(x) \cdot f(p^{\text{cnt}[i]+1})$$

But $f(x) = f(i)/f(p^{\text{cnt}[i]})$ So:

$$f(ip) = \frac{f(i)}{f(p^{\text{cnt}[i]})} \cdot f(p^{\text{cnt}[i]+1})$$

This simplifies to:

$$f(ip) = f\left(\frac{i}{p^{\text{cnt}[i]}}\right) \cdot f(p^{\text{cnt}[i]+1})$$

⊙□ General Code Template (Simple Version)

```cpp
std::vector<int> prime;
bool is_composite[MAXN];
int f[MAXN];  // replace with your function array

void sieve(int n) {
    std::fill(is_composite, is_composite + n, false);
    f[1] = 1;

    for (int i = 2; i < n; ++i) {
        if (!is_composite[i]) {
            prime.push_back(i);
            f[i] = f_p(1); // Set f(p^1)
        }

        for (int j = 0; j < prime.size() && i * prime[j] < n; ++j) {
            int p = prime[j];
            is_composite[i * p] = true;

            if (i % p == 0) {
                // p divides i: same prime as before
                f[i * p] = update_if_divides(i, p); // define this
                break;
            } else {
                // p does not divide i: coprime
                f[i * p] = f[i] * f[p];
            }
        }
    }
}
```

⊛ Example: Euler's Totient Function $\phi(n)$

- $\phi(p) = p - 1$
- If $p \mid n$: $\phi(p \cdot n) = \phi(n) \cdot p$
- If $p \nmid n$: $\phi(p \cdot n) = \phi(n) \cdot (p - 1)$

```cpp
int phi[MAXN];

void sieve_phi(int n) {
    std::fill(is_composite, is_composite + n, false);
    phi[1] = 1;

    for (int i = 2; i < n; ++i) {
        if (!is_composite[i]) {
            prime.push_back(i);
            phi[i] = i - 1;
        }

        for (int j = 0; j < prime.size() && i * prime[j] < n; ++j) {
            int p = prime[j];
            is_composite[i * p] = true;

            if (i % p == 0) {
                phi[i * p] = phi[i] * p;
                break;
            } else {
                phi[i * p] = phi[i] * (p - 1);
            }
        }
    }
}
```

⊛ Example: Möbius Function $\mu(n)$

- $\mu(1) = 1$
- $\mu(n) = (-1)^k$ if n is product of k distinct primes
- $\mu(n) = 0$ if n has a squared prime factor

```cpp
int mu[MAXN];

void sieve_mobius(int n) {
    std::fill(is_composite, is_composite + n, false);
    mu[1] = 1;

    for (int i = 2; i < n; ++i) {
        if (!is_composite[i]) {
            prime.push_back(i);
            mu[i] = -1;
        }

        for (int j = 0; j < prime.size() && i * prime[j] < n; ++j) {
            int p = prime[j];
            is_composite[i * p] = true;

            if (i % p == 0) {
                mu[i * p] = 0;
                break;
            } else {
                mu[i * p] = mu[i] * mu[p];
            }
        }
    }
}
```

△ To Add Your Own Function

1. Define $f(p^k)$
2. Set the rules for:
   - When $p \nmid i$: $f(i \cdot p) = f(i) \cdot f(p)$
   - When $p \mid i$: derive $f(i \cdot p)$ based on $f(p^k)$