Số học 2 - Số nguyên tố, Sàng Eratosthenes

Số học 2 - Số nguyên tố, Sàng Eratosthenes

Nguồn: HackerEarth và 1 số bài viết trên Wikipedia

Người dịch: Bùi Việt Dũng

Bạn có thể đọc phần 1 về Modulo & GCD ở đây.

Số nguyên tố (Prime Numbers)

Số nguyên tố là số nguyên lớn hơn 1 và có đúng 2 ước là 1 và chính nó.

Hợp số (Composite numbers) là số nguyên lớn hơn 1 và có nhiều hơn 2 ước.

Ví dụ, 5 là số nguyên tố vì 5 chỉ chia hết cho 1 và 5. Tuy nhiên, 6 là hợp số vì 6 chia hết cho 1, 2, 3 và 6.

Có rất nhiều phương pháp để kiểm tra một số nguyên có phải là số nguyên tố hay không.

Thuật toán "ngây thơ"

Ta sẽ duyệt hết tất cả các số từ 1 đến N và đếm số ước của N. Nếu số ước của N là 2 thì N là số nguyên tố, nếu không thì N không là số nguyên tố.

```
bool isPrime(int n) {
    for (int i = 2; i < n; i++)
        if (n % i == 0) {
            // n chia hết cho số khác 1 và chính nó.
            return false;
        }
    return n > 1;
}
```

Độ phức tạp của thuật toán: Độ phức tạp của thuật toán là $O(N)$ do ta phải duyệt hết các số từ 1 đến N.

Một thuật toán tốt hơn

Xét hai số nguyên dương N và D thỏa mãn N chia hết cho D và D nhỏ hơn $\sqrt{N}$. Khi đó $\frac{N}{D}$ phải lớn hơn $\sqrt{N}$. N cũng chia hết cho $\frac{N}{D}$. Vì thế, nếu N có ước nhỏ hơn $\sqrt{N}$ thì N cũng có ước lớn hơn $\sqrt{N}$. Do đó, ta chỉ cần duyệt đến $\sqrt{N}$.

```
bool isPrime(int n) {
    for (int i = 2; i*i <= n; i++)
        if (n % i == 0) return false;
    return n > 1;
}
```

Độ phức tạp của thuật toán: Độ phức tạp của thuật toán là $O(\sqrt{N})$ do ta phải duyệt từ 1 đến $\sqrt{N}$.

Sàng Eratosthenes (Sieve of Eratosthenes)

Sàng Eratosthenes dùng để tìm các số nguyên tố nhỏ hơn hoặc bằng số nguyên N nào đó. Nó còn có thể được sử dụng để kiểm tra một số nguyên nhỏ hơn hoặc bằng N hay không.

https://upload.wikimedia.org/wikipedia/commons/b/b8/Animation_Sieb_des_Eratosthenes_%28vi%29.gif

Nguyên lí hoạt động của sàng là vào mỗi lần duyệt, ta chọn một số nguyên tố và loại ra khỏi sàng tất cả các bội của số nguyên tố đó mà lớn hơn số đó. Sau khi duyệt xong, các số còn lại trong sàng đều là số nguyên tố.

Mã giả (Pseudo Code):

- Đánh dấu tất cả các số đều là số nguyên tố.
- Với mỗi số nguyên tố nhỏ hơn $\sqrt{N}$
  - Đánh dấu các bội lớn hơn nó là số nguyên tố.

```
void sieve(int N) {
    bool isPrime[N+1];
    for(int i = 0; i <= N;++i) {
        isPrime[i] = true;
    }
    isPrime[0] = false;
    isPrime[1] = false;
    for(int i = 2; i * i <= N; ++i) {
         if(isPrime[i] == true) {
             // Mark all the multiples of i as composite numbers
             for(int j = i * i; j <= N; j += i)
                 isPrime[j] = false;
        }
    }
}
```

Code trên được dùng để tìm các số nguyên tố nhỏ hơn hoặc bằng N.

Độ phức tạp của thuật toán:

Số lần lặp của vòng lặp trong là:

- Khi $i = 2$, vòng lặp trong lặp $\frac{N}{2}$ lần.
- Khi $i = 3$, vòng lặp trong lặp $\frac{N}{3}$ lần.
- Khi $i = 5$, vòng lặp trong lặp $\frac{N}{5}$ lần.
- ...

Độ phức tạp tổng: $N.(\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \ldots) = O(N \log N)$.

Phân tích thừa số nguyên tố với sàng Eratosthenes

Cách cài đặt:

Đầu tiên hãy xem xét thuật toán phân tích ra thừa số nguyên tố trong $O(\sqrt{N})$.

```
vector<int> factorize(int n) {
    vector<int> res;
    for (int i = 2; i * i <= n; ++i) {
        while (n % i == 0) {
            res.push_back(i);
            n /= i;
        }
    }
    if (n != 1) {
        res.push_back(n);
    }
    return res;
}
```

Tại mỗi bước ta phải tìm số nguyên tố nhỏ nhất mà N chia hết cho số đó. Do đó, ta phải biến đổi sàng Eratosthenes để tìm được số mình mong muốn trong $O(1)$.

```
int minPrime[n + 1];
for (int i = 2; i * i <= n; ++i) {
    if (minPrime[i] == 0) { //if i is prime
        for (int j = i * i; j <= n; j += i) {
            if (minPrime[j] == 0) {
                minPrime[j] = i;
            }
        }
    }
}
for (int i = 2; i <= n; ++i) {
    if (minPrime[i] == 0) {
        minPrime[i] = i;


```

```
    }
}
```

Bây giờ ta có thể phân tích một số ra thừa số nguyên tố trong $O(\log N)$.

```
vector<int> factorize(int n) {
    vector<int> res;
    while (n != 1) {
        res.push_back(minPrime[n]);
        n /= minPrime[n];
    }
    return res;
}
```

Điều kiện sử dụng phương pháp này là ta phải tạo được mảng có độ dài N phần tử.

Phương pháp này rất hữu ích khi ta phải phân tich nhiều số nhỏ ra thừa số nguyên tố. Ta không cần thiết phải sử dụng phương pháp này trong mọi bài toán liên quan đến phân tích một số ra thừa số nguyên tố. Ngoài ra, ta không thể sử dụng phương pháp này nếu N bằng 10^9 hay 10^12. Khi đó, ta chỉ có thể sử dụng thuật toán $O(\sqrt{N})$.

Tính chất thú vị: Nếu $N = p_1^{q_1}.p_2^{q_2}...p_k^{q_k}$ với $p_1, p_2, ..., p_k$ là các số nguyên tố thì N có $(q_1 + 1).(q_2 + 1)...(q_k + 1)$ ước phân biệt.

Sàng Eratosthenes trên đoạn

Đôi khi bạn phải tìm tất cả các số không phải trên đoạn $[1; N]$ mà là trên đoạn $[L; R]$ với R lớn.

Điều kiện sử dụng phương pháp này là bạn có thể tạo mảng độ dài $R - L + 1$ phần tử.

Cài đặt:

```
vector<bool> isPrime(R - L + 1, true);  // x là số nguyên tố khi và chỉ khi isP

for (long long i = 2; i * i <= R; ++i) {
    for (long long j = max(i * i, (L + i - 1) / i * i); j <= R; j += i) {
        isPrime[j - L] = false;
    }
}

if (1 >= L) {  // Xét riêng trường hợp số 1
    isPrime[1 - L] = false;
}

for (long long x = L; x <= R; ++x) {
    if (isPrime[x - L]) {
        // i là số nguyên tố
```

```
    }
}
```

Độ phức tạp của thuật toán là $O(\sqrt{R} * k)$ với k là hằng số.

Lưu ý: Nếu bạn chỉ cần kiểm tra tính nguyên tố của một hay một vài số thì ta không nhất thiết phải xây dựng sàng. Ta có thể sử dụng hàm sau để kiểm tra tính nguyên tố của một số.

```cpp
bool isPrime(int n) {
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}
```

Bài tập áp dụng:

- SPOJ - PRIME1
- VNOJ - NKABD