Lũy thừa nhị phân

Lũy thừa nhị phân

Tác giả:

- Cao Thanh Hậu - Trường Đại học Khoa học Tự Nhiên, ĐHQG-HCM

Reviewer:

- Nguyễn Minh Hiển - Trường Đại học Công nghệ, ĐHQGHN
- Nguyễn Minh Nhật - Trường THPT chuyên Khoa học Tự nhiên, ĐHQGHN

Lời nói đầu

Phép tính lũy thừa là một trong những phép tính cơ bản nhất, xuất hiện rất nhiều trong các công thức toán học và tất nhiên cũng có hàng tá các ứng dụng trong tin học. Vậy đâu là cách tính lũy thừa hiệu quả nhất trong lập trình và những ứng dụng của lũy thừa là gì, mời bạn đọc cùng tìm hiểu.

Thuật toán và cài đặt

Để gợi ý cho bạn đọc, hãy cùng giải câu đố sau: hãy tính 3^{10} bằng các phép nhân sao cho số phép nhân cần dùng là ít nhất. Trong mỗi phép nhân, bạn chỉ được sử dụng thừa số là 3 hoặc là kết quả của một phép nhân trước đó.

Nếu tìm được số phép nhân ít nhất là 4 thì xin chúc mừng bạn, đó là câu trả lời tối ưu. Cách thực hiện như sau:

- Lấy 3×3 để được 3^2.
- Lấy $3^2 \times 3^2$ để được 3^4.
- Lấy $3^4 \times 3$ để được 3^5.
- Cuối cùng, lấy $3^5 \times 3^5$ để được 3^{10}.

Vậy chỉ cần 4 phép nhân để tính 3^{10}, nhỏ hơn nhiều so với số phép nhân cần thực hiện nếu tính theo định nghĩa (tích của 10 thừa số 3).

Đây chính là ý tưởng cho thuật toán tính lũy thừa bằng phương pháp nhị phân.

Với mọi số tự nhiên b, công thức tổng quát như sau:

$$a^b = \begin{cases} 1, & \text{nếu } b = 0 \\ (a^{\frac{b-1}{2}})^2 \times a, & \text{nếu } b \text{ lẻ} \\ (a^{\frac{b}{2}})^2, & \text{nếu } b \text{ chẵn} \end{cases}$$

Dựa vào công thức trên, ta có hàm tính a^b như sau:

```
long long Pow(long long a, long long b) {
    if (!b) return 1;
    long long x = Pow(a, b / 2);
    if (b % 2 == 0)
        return x * x;
    else
        return x * x * a;
}
```

Dưới đây là một cách cài đặt khác không cần đệ quy. Trong cách này ta phân tích b thành tổng các lũy thừa của 2, đồng thời đạt được giá trị a^{2^k} từ giá trị $a^{2^{k-1}}$. Kết hợp lại, ta nhân a^{2^k} vào kết quả nếu 2^k là một số hạng trong các lũy thừa tạo nên tổng b.
Phân tích b dưới dạng nhị phân cho một cách nhìn rõ ràng hơn, ví dụ tính 3^{10} như sau:

$3^{1010_2} = 3^{1000_2} \times 3^{10_2}$

Cài đặt:

```
long long Pow(long long a, long long b) {
    long long ans = 1;
    while (b > 0){
        if (b % 2) ans = ans * a;
        a = a * a;
        b /= 2;
    }
    return ans;
}
```

Thuật toán có độ phức tạp là $\log(b)$.

Ứng dụng

Tính lũy thừa chia lấy dư

Khi $a > 1$ và b lớn thì a^b rất lớn, vì vậy người ta thường yêu cầu tính a^b chia lấy dư cho một số tự nhiên M nào đó.

Biết rằng phép nhân không làm ảnh hưởng đến phép chia lấy dư, vì vậy chỉ cần thêm các thao tác chia lấy dư mỗi khi thực hiện phép nhân, ta được hàm `Pow(a, b, M)` như

sau:

```
long long Pow(long long a, long long b, long long M) {
    long long ans = 1;
    while (b > 0){
        if (b % 2) ans = ans * a % M;
        a = a * a % M;
        b /= 2;
    }
    return ans;
}
```

Với các giá trị $a < 0$, có thể thay a bởi $a + M$ trước.

Một tính chất khác trong phép lũy thừa:

- Nếu m là số nguyên tố thì $x^n \equiv x^{n \bmod (m-1)} \pmod{m}$.
- Nếu m là hợp số, $x^n \equiv x^{n \bmod \phi(m)} \pmod{m}$

Trong các trường hợp b rất lớn so với m, có thể áp dụng tính chất này để tối ưu thời gian. Để hiểu hơn về các tính chất, các bạn có thể tham khảo định lý Fermat nhỏ và hàm phi Euler.

Một ví dụ điển hình ứng dụng tính chất bên trên là tính $a^{b^c} \mod M$ với M là một số nguyên tố. Khi đó, ta cần tính $a^{b^c \bmod (M-1)} \mod M$

Nhân lấy dư

Cần tính $a \times b \bmod m$.

Thoạt nhìn, đây là một vấn đề rất đơn giản, ta có thể lấy $a \bmod m$ rồi nhân với $b \bmod m$, rồi lại $\bmod m$.

Tuy nhiên trong một số trường hợp giá trị m khá lớn (ví dụ, $m = 10^{16}$), khi đó dù lấy a và b chia dư trước cho m, ta vẫn không thể thực hiện phép nhân a và b vì như thế sẽ quá giới hạn kiểu dữ liệu.

Vì vậy ta cần "nhân nhị phân" (một số tài liệu gọi là "nhân Ấn Độ"), với ý tưởng tương tự như lũy thừa nhị phân. Công thức như sau:

$$a \times b = \begin{cases} 0, & \text{nếu } b = 0 \\ 2 \times a \times \frac{b}{2}, & \text{nếu } b \text{ chẵn và lớn hơn } 0 \\ 2 \times a \times \frac{b-1}{2} + a, & \text{nếu } b \text{ lẻ và lớn hơn } 0 \end{cases}$$

Cài đặt:

```
long long Mul(long long a, long long b) {
    if (!b) return 0;
    long long x = Mul(a, b / 2);
```

```
    if (b % 2 == 0)
        return 2 * x % m;
    else
        return (2 * x + a) % m;
}
```

Tính số Fibonacci lớn - Phép nhân ma trận

Để hiểu rõ phần này, bạn đọc cần nắm các kiến thức cơ bản về nhân ma trận

Ứng dụng lũy thừa nhị phân kết hợp với nhân ma trận là một ứng dụng phổ biến, điển hình là bài toán tính số Fibonacci thứ n.

Dãy số Fibonacci được định nghĩa như sau:

$$f_i = \begin{cases} 0, & \text{nếu } i = 0 \\ 1, & \text{nếu } i = 1 \\ f_{i-2} + f_{i-1}, & \text{nếu } i \geq 2 \end{cases}$$

Bài toán đặt ra là tìm f_n với độ phức tạp $\log(n)$.

Một trong những cách thực hiện điều này là áp dụng kiến thức "nhân ma trận".

Từ ma trận A gồm 2 giá trị f_i và f_{i+1} , xây dựng ma trận kích thước 2×2 sao cho ma trận này nhân với A được ma trận mới gồm 2 giá trị f_{i+1} và f_{i+2}.

$$\begin{pmatrix} F_i \\ F_{i-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{i-1} \\ F_{i-2} \end{pmatrix}$$

Vì phép nhân ma trận có tính chất kết hợp nên ta dễ dàng áp dụng kĩ thuật lũy thừa nhị phân (thay số nguyên thành ma trận) để đạt được độ phức tạp $O(\log(n))$.

Luyện tập

Kiểm tra siêu máy tính

The last digit

LOCKER

Tham khảo

Bài viết được tham khảo từ Algorithms for Competitive Programming .