Lũy thừa nhị phân

Trần Việt Khoa
tvkhoa.husc@gmail.com,
Khoa Công nghệ thông tin, Đại học Khoa học Huế

Ngày 22 tháng 3 năm 2017

1 Phép toán đồng dư

Giả sử a và b là các số nguyên [1]. Ta nói rằng a đồng dư với b theo modulo m nếu hiệu của a và b chia hết cho m. Sử dụng ký hiệu được đề xuất bởi Gauss ta viết $a \equiv b \ (mod\ m) \Leftrightarrow m \mid a - b$.

Ví dụ: $2 \equiv 5 \ (mod\ 3),\ 7 \equiv 4 \ (mod\ 3),\ -1 \equiv 3 \ (mod\ 2)$.

Rõ ràng nếu hai số nguyên là đồng dư modulo m thì chúng có cùng số dư khi chia cho m.

Các phép toán số học với modulo

Phép cộng:

Nếu $a \equiv b \ (mod\ n)$ và $x \equiv y \ (mod\ n)$ thì $a + x \equiv b + y \ (mod\ n)$. Mở rộng ta có nếu $a \equiv b \ (mod\ n)$ thì $a + z \equiv b + z \ (mod\ n)$.

Ví dụ: $13 \equiv 1 \ (mod\ 3), 7 \equiv 4 \ (mod\ 3)$ nên $20 \equiv 8 \ (mod\ 4)$.

Phép trừ:

Nếu $a \equiv b \ (mod\ n)$ và $x \equiv y \ (mod\ n)$ thì $a - x \equiv b - y \ (mod\ n)$. Tương tự phép cộng ta cũng có nếu $a \equiv b \ (mod\ n)$ thì $a - z \equiv b - z \ (mod\ n)$.

Phép nhân:

Nếu $a \equiv b \ (mod\ n)$ và $x \equiv y \ (mod\ n)$ thì $a \times x \equiv b \times y \ (mod\ n)$, và nếu $a \equiv b \ (mod\ n)$ thì $a \times z \equiv b \times z \ (mod\ n)$.

Phép lũy thừa:

Công thức lũy thừa chính là triển khai từ công thức nhân và ta có nếu $a \equiv b \ (mod\ n)$ thì $a^k \equiv b^k \ (mod\ n)$.

Ví dụ: $2 \equiv 5 \ (mod\ 3)$, thì ta có $2^3 \equiv 5^3 \ (mod\ 3)$, thực vậy do $2 \equiv 5 \ (mod\ 3) \Leftrightarrow 2 \times 2 \equiv 5 \times 5 \ (mod\ 3) \Leftrightarrow 2^2 \times 2 \equiv 5^2 \times 5 \ (mod\ 3) \Leftrightarrow 2^3 \equiv 5^3 \ (mod\ 3)$.

Một số ví dụ về đồng dư

Ví dụ 1: Chứng minh rằng $2012^{2013} + 1$ chia hết cho 7.

Ta có $2012 \equiv 3 \ (mod\ 7)$, theo quy tắc lũy thừa ta có $2012^{2013} \equiv 3^{2013} \ (mod\ 7)$.

Mặt khác ta có: $3^2 = 9 \equiv 2 \ (mod\ 7) \Leftrightarrow 3^2 \times 3 \equiv 2 \times 3 \ (mod\ 7) \equiv 6 \ (mod\ 7)$. Trong modulo 7, con số -1 chính là con số 6, do đó $3^3 \equiv 6 \ (mod\ 7) \equiv -1 \ (mod\ 7)$.

Vậy $3^{2013}\ (mod\ 7) \equiv 3^{3\times 671}\ (mod\ 7) \equiv (-1)^{671}\ (mod\ 7) \equiv -1\ (mod\ 7)$.

Kết luận: $2012^{2013} + 1$ chia hết cho 7 □

Ví dụ 2: Tìm tất cả các số tự nhiên n sao cho $2015^n + n^{2015}$ chia hết cho 3.

Ta có $2015 \equiv 2\ (mod\ 3) \equiv -1\ (mod\ 3)$, do đó $2015^n \equiv (-1)^n\ (mod\ 3)$. Vì vậy nếu $2015^n + n^{2015}$ chia hết cho 3 thì $(-1)^n + n^{2015} \equiv 0\ (mod\ 3)$.

Chúng ta có n = 0, 1, hoặc 2 (mod 3). Hay nói cách khác n = 0, 1, hoặc -1 (mod 3).

Trường hợp 1: nếu n = 0 (mod 3) thì $(-1)^n + n^{2015} \equiv (-1)^n\ (mod\ 3)$. Do đó trường hợp này không thỏa mãn.

Trường hợp 2: nếu $n \equiv \pm 1\ (mod\ 3)$ thì $(-1)^n + n^{2015} \equiv (-1)^n + (\pm 1)^{2015} \equiv (-1)^n + 1\ (mod\ 3)$. Do đó để $(-1)^n + 1 \equiv 0\ (mod\ 3)$ thì n phải là số lẻ.

Như vậy $2015^n + n^{2015}$ chia hết cho 3 thì n phải thỏa mãn hai điều kiện, đó là $n \neq 0\ (mod\ 3)$ và $n \equiv 1\ (mod\ 2) \Leftrightarrow n \equiv \pm 1\ (mod\ 6)$ □

Ví dụ 3: Tìm hai chữ số tận cùng của số 2^{1000}.

Ta có $2^{10} \equiv 24\ (mod\ 100) \Leftrightarrow 2^{20} \equiv 24^2\ (mod\ 100) \equiv 76\ (mod\ 100)$.

Mặt khác ta nhận thấy rằng $76^k \equiv 76\ (mod\ 100)$ với mọi k=1,2,...

Do đó ta có $2^{1000} = 2^{20\times 50} \equiv 76^{50}\ (mod\ 100) \equiv 76\ (mod\ 100)$. Vậy hai chữ số cuối cùng của 2^{1000} là 76 □

2 Phép toán modulo trong ngôn ngữ lập trình

Phép modulo trên được sử dụng trong toán học, ví dụ trên ta dễ dàng tìm được hai số cuối của 2^{1000}. Tuy nhiên cũng bài toán như vậy cần giải bằng cách lập trình thì ta phải xem xét lại cách tính vì khó có thể xây dựng thuật toán heuristic như trên, nhưng ta lại vận dụng được khả năng tính nhanh của máy tính. Bài toán trên liên quan đến phạm vi tính toán của kiểu số (số nguyên) của ngôn ngữ lập trình. Một số ngôn ngữ lập trình có xử lý số lớn như Python, Java (BigInteger) thì ta không cần quan tâm, ví dụ thuật toán tính n!, kết quả Modulo cho $10^9 + 7$ ta viết trên Python như sau:

```
#function 1: Modulo M in each step
def factorial_1(n, M):
    ans=1
    while (n >=1):
        ans = (ans * n) % M
        n = n-1
    return ans
#function 2: Modulo M after repeater.
def factorial_2(n, M):
    ans=1
    while (n>=1):
        ans = ans * n;
        n=n-1
    ans = ans % M;
    return ans;
#Call
print(factorial(2150,1000000007))
print(factorial2(2150,1000000007))
```

Cả hai phương án đều cho cùng một kết quả: 776222743, bởi vì phép nhân (ans = ans*n) không bị tràn dữ liệu.

Đối ngôn ngữ lập trình (C, C++, Pascal) do hạn chế về kích thước dữ liệu (khoảng 9.10^{18} với kiểu long long). Thay vì cần trả lời chính xác một kết quả của phép tính số lớn nào đó, người ta chỉ cần lấy giá trị phần dư của nó khi chia cho một số M (M được lựa chọn theo hai yếu tố, một là đủ lớn để các phần dư có mật độ phân bố rộng hơn, hai là một số nguyên tố, vì thế $10^9 + 7$ luôn là một lựa chọn).

Tuy nhiên, nếu không nắm vững quy tắc sử dụng phép chia lấy phần dư modulo thì ta sẽ có kết quả sai cho các bài toán cần giải.

Các quy tắc sử dụng modulo (% trong C, C++, Pascal):

1. (a + b) % c = ((a % c) + (b % c)) % c.

2. (a * b) % c = ((a % c) * (b % c)) % c.

3. (a – b) % c = ((a % c) – (b % c)) % c.

Ví dụ sau cho ta thấy cách sử dụng phép modulo khi tính n!, kết quả trả về chia modulo cho M.

Thuật toán sau sai vì kết quả của phép toán ans=ans*n đã vượt ra khỏi phạm vi lưu trữ khi thực hiện trong vòng lặp, cho nên kết quả cuối cùng khi modulo cho M là không đúng.

```
long long factorial2(int n,int M)//WRONG APPROACH!!!
{
    long long ans=1;
    while(n>=1)
    {
        ans=ans*n;
        n--;
    }
        ans=ans%M;
    return ans;
}
```

Để có được thuật toán đúng phép modulo được thực hiện sau mỗi lần lặp, bạn có thể viết ans = ((ans % M)* (n % M)) % M, nhưng vì ans đã được modulo qua mỗi lần lặp và n nhỏ, giảm dần nên ta viết tắt như trên.

```
long long factorial(int n,int M)
{
    long long ans=1;
    while(n >=1)
    {
        ans=(ans*n)%M;
        n--;
    }
    return ans;
}
```

Chú ý:

- (a * b * c) % M = (((a * b) % M) * c) % M;
- (a + b + c) % M = (((a + b) % M) + c) % M.

• M phải là một số nguyên tố đến khi thực hiện phép Modulo ta không phải nhận một kết quả 0, ví dụ: M=12, (8*3) % M=0, Nếu M là số nguyên tố thì điều đó không xảy ra (trừ khi hoặc b bằng 0).

• Phép cộng và nhân dễ dàng áp dụng quy tắc trên, đối phép trừ ta chú ý điều sau:

- Thay vì dùng: a=a%c; b=b%c; ans = (a - b) % c;

- Bạn hãy dùng: a=a%c; b=b%c; ans = (a - b + c)% c;

- Có thể kiểm chứng cho a=121; b= 9113, c=7;

• Nếu a, b có kiểu int, $c = 10^9 + 7$ thì kết quả r=(a*b)%c phải nhỏ hơn c, tuy nhiên khi thực hiện phép a*b (thực hiện trước) có thể lớn hơn dữ liệu int, do vậy bạn phải ép kiểu (long long)(a*b) % c.

3 Lũy thừa nhị phân

Một trong những bài toán ta thường gặp khi tính toán vối số nguyên đó là hàm mũ (power) a^b [2]. Thuật toán ngây thơ cho hàm này là: $a^b = a \times a \times ... \times a$, với b lần nhân cho a. Tuy nhiên thuật toán này sẽ không hiệu quả khi tính với số b lớn. Phép lũy thừa nhị phân sẽ được sử dụng trong tình huống này. Nhận thấy $a^{b+c} = a^b \times a^c$ và $a^{2b} = a^b \times a^b = (a^b)^2$ sẽ được áp dụng để tính.

Ví dụ: tính hàm mũ với a=3, b= 9, biểu diễn b dưới dạng nhị phân b= 1001 ta có: $3^{1001_2} = 3^8 \times 3^1$, theo trên ta có $3^1 = 3, (3^1)^2 = 3^2 = 9; (3^2)^2 = 3^4 = 81; (3^4)^2 = 3^8 = 6561$. Vậy ta có $3^{1001_2} = 3^8 \times 3^1 = 6561 \times 3 = 19683$

Bản cài đặt bằng C++ cho ý tưởng trên:

```
long long binpow(long long a,long long b)
{
    long long res = 1;
    while (b){
        if (b & 1) res = res * a;
        a = (a * a);
        b >>= 1;
    }
    return res;
}
```

Thay vì tiếp cận như trên, bài toán này có thể tiếp cận bằng cách định nghĩa đệ quy như sau:

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

Bản cài đặt bằng C++ cho định nghĩa đệ quy trên:

```
long long binpow(long long a,long long b)
{
    if (b == 1) return a;
    long long res = binpow(a, b/2);
    if (b % 2) return res*res*a;
    else return res * res;
}
```

Với cài đặt này dễ dàng tính ra được độ phức tạp thuật toán là O(log n). Dựa trên thuật toán này ta viết đoạn mã cho thuật toán sau:

Thuật toán $a^b \ mod \ M$

```
int powMod(int a, int p, int M) {
   a %= M;
   long long res = 1;
   while (p) {
      if (p & 1) {
         res *= a;
         res %= M;
      }
      p >>= 1;
      a *= a;
      a %= M;
   }
   return res;
}
```

4 Thực hành

BÀI TOÁN (Số cuối cùng của a^b)[1]

Phát biểu

Cho hai số a và b, a và b đều không cùng bằng 0, Tìm số cuối cùng của a^b.

Dữ liệu vào

- dòng thứ nhất chứa số nguyên T ($T \leq 30$) là số testcase.

- T dòng tiếp theo mỗi dòng chứa hai số a và b $0 \leq a \leq 20, \ 0 \leq b \leq 2,147,483,000$.

Kết quả

- ứng với mỗi testcase in ra chữ số cần tìm.

Ví dụ

Dữ liệu vào	Kết quả
2 3 10 6 2	9 6

Lời giải

Sử dụng hàm powMod() với M=10 ở mục 3 ở trên.

BÀI TOÁN (Số cuối cùng của a^b)[2]

Phát biểu

Tương tự như bài toán trên cho hai số a và b, a và b đều không cùng bằng 0, Tìm số cuối cùng của a^b.

Dữ liệu vào

- dòng thứ nhất chứa số nguyên T ($T \leq 30$) là số testcase.

[1] http://vn.spoj.com/problems/LASTDIG/
[2] http://vn.spoj.com/problems/LASTDIG2/

- T dòng tiếp theo mỗi dòng chứa hai số a có d chữ số ($1 \leq d \leq 1000$) và b ($0 \leq b \leq 999 \times 10^{15}$).

Kết quả

- ứng với mỗi testcase in ra chữ số cần tìm.

Ví dụ

Dữ liệu vào	Kết quả
3 3 10 6 2 150 53	9 6 0

Lời giải

Do $a = \overline{d_{n-1} \ldots d_2 d_1 d_0} \equiv d_0 \ (mod\ 10)$ nên $a^b \ mod\ 10 = d_0^b \ mod\ 10$, sử dụng hàm powMod(), M=10 trên để tính kết quả cuối cùng.

BÀI TOÁN (Tổng của Z_n)[3]

Phát biểu

Cho hai số nguyên n và k. hãy tìm $(Z_n + Z_{n-1} - 2Z_{n-2}) \ mod\ 10000007$, với $Z_n = S_n + P_n$ và $S_n = 1^k + 2^k + 3^k + \cdots + n^k$, $P_n = 1^1 + 2^2 + 3^3 + \cdots + n^n$

Dữ liệu vào

- Gồm nhiều dòng, mỗi dòng là một testcase chứa hai số n ($1 < n < 200000000$) và k ($0 < k < 1000000$). Dòng kết thúc testcase chứa hai số 0 0.

Kết quả

- ứng với mỗi testcase in ra chữ số cần tìm trên mỗi dòng.

Ví dụ

Dữ liệu vào	Kết quả
10 3 9 31 83 17 5 2 0 0	4835897 2118762 2285275 3694

Lời giải

Triển khai công thức ta có $S = n^k + n^n + 2(n-1)^{n-1} + 2(n-1)^k$. Sau đó sử dụng hàm powMod(), M=10 trên để tính kết quả cuối cùng theo công thức.

Bài tập 1. http://www.spoj.com/problems/DIVISION/

Bài tập 2. http://vn.spoj.com/problems/CSQUARE/

[3]http://vn.spoj.com/problems/ZSUM/

5 Tài liệu tham khảo

Tài liệu

[1] Tô Niên Đông Vũ, **Modulo phần 1,** http://vuontoanblog.blogspot.com/2012/06/modulo1.html.

[2] Maxim Ivanov, **Binary Exponentiation,** https://e-maxx-eng.appspot.com/algebra/binary-exp.html.