

Thuật toán Euclid

Thuật toán Euclid

Tác giả:

- Nguyễn Đức Kiên, Trường Đại học Công nghệ, ĐHQGHN.

Reviewer:

- Nguyễn Minh Hiền - Trường Đại học Công nghệ, ĐHQGHN
- Phạm Hoàng Hiệp - University of Georgia
- Nguyễn Minh Nhật - Trường THPT chuyên Khoa học Tự nhiên, ĐHQGHN

Mở đầu

Một số ký hiệu toán học sử dụng trong bài viết

- Cho hai số nguyên a và b ($b \neq 0$). Nếu tồn tại số nguyên q sao cho $a = bq$ thì ta nói a chia hết cho b (ký hiệu $a : b$) hoặc b là ước của a (ký hiệu $b \mid a$).
- Cho ba số nguyên a , b và m ($m \neq 0$). Nếu tồn tại một số nguyên r sao cho $a = mq_1 + r$ và $b = mq_2 + r$ với q_1, q_2 là các số nguyên thì ta nói a đồng dư với b theo modulo m . Ký hiệu là $a \equiv b \pmod{m}$.
- Ký hiệu $\log_a(b)$ được hiểu là logarit cơ số a của b . Bài viết này sẽ sử dụng $\log(x)$ để thay cho $\log_2(x)$.

Ước chung lớn nhất

Đây là khái niệm tương đối quen thuộc với chúng ta.

Cho hai số tự nhiên a và b . Số nguyên dương d lớn nhất thỏa mãn $d \mid a$ và $d \mid b$ gọi là **ước chung lớn nhất (greatest common divisor - GCD)** của a và b . Ký hiệu là $\gcd(a, b)$ (**ƯCLN(a, b)** trong tiếng Việt) hoặc đơn giản hơn (a, b) .

$$\gcd(a, b) = \max\{d \in \mathbb{N}^* : (d \mid a), (d \mid b)\}$$

Về mặt toán học, với $k \neq 0$ thì $\gcd(0, k) = k$, và $\gcd(0, 0)$ không xác định. Tuy nhiên, để lập trình tiện lợi ta quy ước $\gcd(0, 0) = 0$.

Định nghĩa ƯCLN cũng có thể mở rộng cho số nguyên. Khi đó $\gcd(a, b) = \gcd(\|a\|, \|b\|)$

Có một vài cách để tìm ƯCLN của hai số a và b . Cách đơn giản nhất là ... duyệt từng số tự nhiên d một đến $\min\{a, b\}$ để kiểm tra điều kiện $d \mid a$ và $d \mid b$. Ngoài ra, trong toán học, ta cũng sử dụng phương pháp phân tích thành thừa số nguyên tố để tìm ƯCLN. Phương pháp này không hiệu quả lắm khi lập trình. Thay vào đó, chúng ta sẽ sử dụng thuật toán Euclid.

Thuật toán Euclid

Thuật toán này được trình bày trong tác phẩm "Cơ sở" (Elements) của Euclid vào khoảng năm 300 TCN, nhưng cũng có thể đã từng xuất hiện trước đó.

Thuật toán được thực hiện bằng cách liên tục áp dụng công thức sau cho tới khi ra kết quả:

$\gcd(a, b) = \begin{cases} a & \text{nếu } b = 0 \\ \gcd(b, a \bmod b) & \text{trong trường hợp còn lại} \end{cases}$

Chứng minh

Nếu d là ước của a và b , hiển nhiên nó cũng là ước của $a - b$.

Nếu d' là ước của b và $b - a$, hiển nhiên nó cũng là ước của $b + (a - b) = a$.

Do vậy với ba số $a, b, a - b$, nếu một số d bất kỳ là ước của một trong ba số trên thì sẽ là ước của hai số còn lại, tức là $\text{ƯC}(a, b) = \text{ƯC}(b, a - b)$. Điều này dẫn đến $\gcd(a, b) = \gcd(b, a - b)$.

Phép tính $a - b$ sau khi thực hiện $\lfloor \frac{a}{b} \rfloor$ lần thì sẽ thoả mãn $a \leq b$. Số a sau khi trừ đi $\lfloor \frac{a}{b} \rfloor$ lần b trở thành $a - b \lfloor \frac{a}{b} \rfloor = a \bmod b$.

Vậy $\gcd(a, b) = \gcd(b, a \bmod b)$ (đpcm).

Cài đặt

```

1 | int gcd(int a, int b)
2 | {
3 |     if (b == 0) return a;
4 |     return gcd(b, a % b);
5 | }
```

Hoặc ngắn hơn:

```

1 | int gcd(int a, int b)
2 | {
3 |     return (b ? gcd(b, a % b) : a);
4 | }
```

Độ phức tạp

Định lý Lamé: Thuật toán Euclid cần thực hiện ít hơn $5 \log_{10}(\min(u, v))$ lần chia lấy dư.

Thuật toán chạy chậm nhất khi $a = F_n, b = F_{n-1}$, với F_i là số Fibonacci thứ i . Khi đó thuật toán cần thực hiện $n - 2$ lần đệ quy.

Cải tiến

So với các phép toán khác, phép lấy phần dư (`%`) chậm hơn một chút dù vẫn có độ phức tạp là $O(1)$. Chúng ta có thể xây dựng một cách cài đặt khác không sử dụng phép toán này.

Ta có một số tính chất sau:

- $\gcd(2k, 2h) = 2 \gcd(k, h)$
- $\gcd(2k, 2h + 1) = \gcd(k, 2h + 1)$

Kết hợp với $\gcd(a, b) = \gcd(b, a - b)$ ta cài đặt như sau (Code tham khảo từ CP Algorithms):

```

1  int gcd(int a, int b)
2  {
3      if (!a || !b) return a | b;
4      int shift = __builtin_ctz(a | b);
5      a >>= __builtin_ctz(a);
6      do
7      {
8          b >>= __builtin_ctz(b);
9          if (a > b)
10             swap(a, b);
11             b -= a;
12     } while (b);
13
14     return a << shift;
15 }
```

Đoạn code trên thực hiện những công việc sau:

- Chia cả hai số a và b cho $shift$ là lũy thừa của 2, để hai số đều lẻ (Hàm `__builtin_ctz(k)` đếm số bit 0 tận cùng của k).
- Lúc này, ít nhất một trong hai số là lẻ. Liên tục chia số chẵn cho 2 để nó trở thành số lẻ, sau đó áp dụng $(a, b) = (b, a - b)$. Lặp lại bước trên tới khi một trong hai số là 0.
- Nhân kết quả (tạm gọi là ans) với $shift$, vì ta đã chia cả hai số này cho $shift$, vì rõ ràng $(shift, ans) = 1$.

Thuật toán cải tiến trên sẽ thực hiện chia $\log(a) + \log(b)$ lần trong trường hợp tệ nhất. Do vậy, độ phức tạp của thuật vẫn không đổi và là $O(\log(a, b))$.

Vài chú ý

- Thư viện `algorithm` của C++ có hỗ trợ hàm `__gcd(a, b)` để tìm ước chung lớn nhất của hai số a và b , cũng sử dụng thuật Euclid. Kể từ phiên bản `C++17`, thư viện `numeric` hỗ trợ thêm hàm `gcd(a, b)` với mục đích tương tự. Các hàm có sẵn này có thể được sử dụng để code ngắn gọn.
- Để tính bội chung nhỏ nhất (BCNN) của hai số, ta dùng công thức:

$$\text{lcm}(a, b) = \frac{a \times b}{\text{gcd}(a, b)}$$

Khi tính toán, công thức trên sẽ gây tràn số nếu $a \times b$ quá lớn, nhưng ta có thể giải quyết dễ dàng bằng cách thực hiện phép chia trước:

$$\text{lcm}(a, b) = \frac{a}{\text{gcd}(a, b)} \times b$$

Kể từ `C++17`, thư viện `numeric` cũng hỗ trợ cả hàm `lcm(a, b)` cho phép tính BCNN của hai số.

Thuật toán Euclid mở rộng

Với hai số tự nhiên a và b , thuật toán này được sử dụng để viết $d = \text{gcd}(a, b)$ dưới dạng **tổ hợp tuyến tính**. Nói cách khác, thuật toán này sẽ tìm một bộ giá trị nguyên (x, y) thoả mãn:

$$ax + by = d$$

Ví dụ:

$$\text{gcd}(55, 80) = 5 = 55 \times 3 + 80 \times (-2)$$

Các số x, y thoả mãn đẳng thức trên luôn tồn tại theo bổ đề sau:

Bổ đề Bézout: Với hai số nguyên a, b có ƯCLN là d , tồn tại hai số nguyên x và y thoả mãn $ax + by = d$. Hơn nữa, tất cả các số nguyên D có dạng $D = aX + bY$ đều là bội của d .

► Chứng minh

Ứng dụng trực tiếp của thuật toán này là các phương trình Diophantus, sẽ được thảo luận ở phần sau.

Mô tả thuật toán

Xét bài toán với hai số ban đầu là $a = A$ và $b = B$. Gọi d là ƯCLN của A và B .

Khi thực hiện thuật toán Euclid (không mở rộng) để tìm d , sau khi biến đổi hoàn tất ta thu được $a = d, b = 0$. Lúc này ta có $d = d \times 1 + 0 \times 0$, tức là $a = d, b = 0, x = 1, y = 0$.

Từ các giá trị a, b, x, y ở trên, ta truy lại các giá trị a, b ở bước trước và thay đổi các hệ số x, y để đẳng thức $d = ax + by$ đúng trong bước này.

Giả sử tại một bước ta có $a = a_0, b = b_0$. Đặt $a_0 = b_0q + r$ ($q, r \in \mathbb{N}, r < b_0$). Ta thấy $q = \left\lfloor \frac{a_0}{b_0} \right\rfloor$ và $r = a_0 \bmod b_0$.

Lại giả sử trước bước đó sau khi áp dụng thuật toán Euclid mở rộng, ta được bộ $a = b_0, b = r$ và các hệ số $x = x_1, y = y_1$ là .

Ta cần tìm các hệ số x_0, y_0 để: $a_0x_0 + b_0y_0 = d$.

$$\begin{aligned} d &= b_0x_1 + ry_1 \\ \Rightarrow d &= b_0x_1 + (a_0 - b_0q)y_1 \\ \Rightarrow d &= a_0y_1 + b_0(x_1 - qy_1) \end{aligned}$$

Liên tục cập nhật các hệ số x, y theo công thức trên tới khi thu được $a = A, b = B$ như ban đầu, ta sẽ thu được kết quả.

Cài đặt

```

1 // Hàm trả về ƯCLN của a và b đồng thời thay đổi giá trị của x,
2 int extEuclid(int a, int b, int& x, int& y)
3 {
4     if (b == 0)
5     {
6         x = 1;
7         y = 0;
8         return a;
9     }
10    int q = a / b;
11    int r = a - b * q;
12    int x1 = 0, y1 = 0;
13    int d = extEuclid(b, r, x1, y1);
14    x = y1;
15    y = x1 - q * y1;
16    return d;
17 }
```

Độ phức tạp

Thuật toán Euclid mở rộng thực tế chỉ là thêm một vài bước tính toán vào thuật toán Euclid thường nên độ phức tạp vẫn là $O(\log(\min\{a, b\}))$.


Phương trình Diophantus tuyến tính hai ẩn

Phương trình Diophantus (Diophantine function) tuyến tính hai ẩn có dạng như sau:

$$ax + by = c \quad (a, b, c \in \mathbb{Z})$$

Phương trình trên có vô số nghiệm (x, y) thực (trừ khi $a = b = 0, c \neq 0$, khi đó phương trình vô nghiệm). Tuy nhiên, ta chỉ quan tâm đến các nghiệm nguyên của phương trình.

Để cho ngắn gọn, bài viết sẽ sử dụng cụm từ "phương trình Diophantus" để chỉ phương trình Diophantus tuyến tính hai ẩn.

Bài tập áp dụng trực tiếp: [CEQU](#) 

Thuật toán tìm nghiệm

Khi $a = b = 0$, phương trình có nghiệm $x = k, y = h$ ($k, h \in \mathbb{Z}$) nếu $c = 0$ và vô nghiệm nếu $c \neq 0$.

Khi $a \neq 0, b = 0$ phương trình có nghiệm $x = \frac{c}{a}, y = k$ ($k \in \mathbb{Z}$) nếu $a \mid c$ và vô nghiệm nếu $a \nmid c$. Tương tự khi $a = 0, b \neq 0$.

Bây giờ ta chỉ xét các trường hợp $a \neq 0, b \neq 0$.

Tìm nghiệm tổng quát bằng phương pháp số học

Lưu ý: Phần dưới đây không thực sự liên quan tới thuật toán để giải bài này, đồng thời kết quả cũng khá phức tạp và không phải thứ chúng ta cần lúc này. Bạn đọc cân nhắc trước khi xem.

► Tìm nghiệm tổng quát bằng phương pháp số học

Tìm nghiệm bằng thuật toán

Ta đã biết phương trình chỉ có nghiệm nếu $\gcd(a, b) \mid c$. Nếu điều kiện này không thoả mãn, ta kết luận phương trình vô nghiệm.

Giả sử a, b là các số dương. Đặt $d = \gcd(a, b)$.

Sử dụng thuật toán Euclid mở rộng, ta có:

$$ax' + by' = d \quad (x', y' \in \mathbb{Z})$$

Nhân hai vế của phương trình với $\frac{c}{d}$ được:

$$a \left(x' \times \frac{c}{d} \right) + b \left(y' \times \frac{c}{d} \right) = c$$

Suy ra phương trình có nghiệm:

$$\begin{cases} x_0 = x' \times \frac{c}{d} \\ y_0 = y' \times \frac{c}{d} \end{cases}$$

Trường hợp a, b không dương, ta thay đổi dấu của x, y để thoả mãn đẳng thức.

Thay nghiệm x_0, y_0 trở lại phương trình, ta được:

$$ax_0 + by_0 = c \Rightarrow a \left(x_0 + k \times \frac{b}{d} \right) + b \left(y_0 - k \times \frac{a}{d} \right) = c \quad (k \in \mathbb{Z})$$

Từ đẳng thức này ta kết luận các nghiệm của phương trình có dạng:

$$\begin{cases} x = x_0 + k \times \frac{b}{d} \\ y = y_0 - k \times \frac{a}{d} \end{cases} \quad (k \in \mathbb{Z})$$

Chốt lại, để tìm nghiệm của một phương trình Diophantus, ta tìm các hệ số x', y' từ thuật toán Euclid mở rộng, rồi từ các hệ số này áp dụng vào các công thức trên để tính ra kết quả.

Cài đặt


Đoạn chương trình sau tìm **một** nghiệm nguyên của phương trình $ax + by = c$, với $a, b \neq 0$:

```

1  const pair <int, int> INVALID_ROOT = {INT_MAX, INT_MAX};
2
3  //Hàm trả về ƯCLN của a và b, biến đổi x, y thoả mãn ax + by =
4  int extEuclid(int a, int b, int &x, int&y)
5  {
6      if (b == 0)
7      {
8          x = 1;
9          y = 0;
10         return a;
11     }
12     int q = a / b;
13     int r = a - b * q;
14     int x1 = 0, y1 = 0;
15     int d = extEuclid(b, r, x1, y1);
16     x = y1;
17     y = x1 - q * y1;
18     return d;
19 }
20
21 //Tìm 1 nghiệm nguyên của phương trình ax + by + c = 0
22 pair <int, int> diophantineSolve(int a, int b, int c)
23 {
24     int x = 0, y = 0;
25     int d = extEuclid(a, b, x, y);
26     if (c % d != 0) return INVALID_ROOT;
27     x *= c / d;
28     y *= c / d;
29     if (a < 0) x = -x;
30     if (b < 0) y = -y;
31     return make_pair(x, y);
32 }
```

Một số bài toán liên quan

Đếm số nghiệm của phương trình Diophantus trong một khoảng cho trước

Bài tập áp dụng trực tiếp: [SGU 106](#) 

Tóm tắt đề bài: Đếm số cặp số nguyên x, y thoả mãn:

$$\begin{cases} ax + by = c \\ x_1 \leq x \leq x_2 \\ y_1 \leq y \leq y_2 \end{cases}$$

Các trường hợp có $a = 0$ hoặc $b = 0$ là tầm thường. Ta chỉ xét $a \neq 0$ và $b \neq 0$.

Ở phần trước, ta đã có công thức nghiệm tổng quát của các phương trình Diophantus từ một nghiệm bất kỳ:

$$\begin{cases} x = x_0 + k \times \frac{b}{d} \\ y = y_0 - k \times \frac{a}{d} \end{cases} (k \in \mathbb{Z})$$

Dễ thấy các nghiệm của bài toán lúc này chỉ phụ thuộc vào k . Bài toán trở thành tìm k sao cho x và y thoả mãn các điều kiện đã cho. Lúc này ta chỉ cần thay ngược biểu thức của x và y theo k vào rồi suy ngược k ra là xong.

Nếu bài toán yêu cầu liệt kê chi tiết các nghiệm này, ta cũng chỉ cần tăng k lên dần dần trong khoảng thoả mãn.

Tìm nghiệm có tổng dương nhỏ nhất

Bài toán này yêu cầu chúng ta tìm nghiệm x, y có $x + y$ dương nhỏ nhất.

Cộng từng vế của biểu thức nghiệm x và y theo k được:

$$x + y = x_0 + y_0 + k \times \frac{b-a}{d}$$

Dễ thấy nghiệm nhỏ nhất khi $k \times \frac{b-a}{d}$ nhỏ nhất. Tùy thuộc vào dấu của $b - a$, ta chọn k sao cho giá trị của biểu thức là cực tiểu.

Bài tập áp dụng: [Euclid Problem](#) . Ở bài này $c = d$.

Nghịch đảo modulo

Số tự nhiên γ được gọi là **nghịch đảo modulo** theo modulo m của một số tự nhiên a nếu $a\gamma \equiv 1 \pmod{m}$. Ký hiệu là $a^{-1} \pmod{m}$.

Ví dụ: $3 \equiv 7^{-1} \pmod{10}$

Không phải số tự nhiên nào cũng có nghịch đảo modulo; chẳng hạn, không có nghịch đảo modulo 4 của 2.

Xét phương trình Diophantus $ax + by = 1$. Khi phương trình có nghiệm (x_0, y_0) , ta có:

$$ax_0 + by_0 = 1 \Rightarrow ax_0 = 1 - by_0 \Rightarrow ax_0 \equiv 1 \pmod{b} \Rightarrow x_0 \equiv a^{-1} \pmod{b}$$

Ta thấy nghiệm x của phương trình là nghịch đảo modulo b của a . Qua đó ta cũng thấy, nghịch đảo modulo chỉ tồn tại khi và chỉ khi $(a, b) = 1$.






Nghịch đảo modulo thường được sử dụng trong những bài toán chia số lớn lấy phần dư, điển hình là các bài toán tính tổ hợp. Chẳng hạn:

$$C_n^k = \frac{n!}{k! \times (n-k)!} \equiv n! \times (k! \times (n-k)!)^{-1} \pmod{M}$$




(Lưu ý rằng công thức trên chỉ đúng nếu $k! \times (n-k)!$ nguyên tố cùng nhau với M với mọi k, n thoả mãn dữ liệu của đề)

Khi modulo M là số nguyên tố, để tiện lợi ta thường dùng định lý Fermat nhỏ để suy ra $x^{-1} \equiv x^{M-2} \pmod{M}$ rồi dùng lũy thừa nhanh để tính. Còn nếu M không nguyên tố, ta lại áp dụng thuật toán Euclid mở rộng để tìm nghịch đảo modulo qua phương trình $ax + My = 1$.

Bài tập áp dụng

- [UVA - Gift Dilemma](#) 
- [Codeforces - Ebony and Ivory](#) 
- [Codeforces - Beautiful Numbers](#) 
- [Codechef - Get AC in one go](#) 
- [VNOJ - VM 08 Bài 05 - Số nguyên](#) 

Tài liệu tham khảo

- Một loạt các bài viết trong mục Fundamentals, [CP Algorithms](#) 
- Wikipedia (phần chứng minh định lý Lamé và bổ đề Bézout)
- VNOI Wiki, [Nghịch đảo Modulo](#)  (bài viết cũ)
- [Post trên VNOI Forum của anh Tăng Khải Hạnh](#) 
- Slide về chủ đề này của thầy Lê Minh Hoàng (chưa tìm được nguồn)

Được cung cấp bởi [Wiki.js](#)