Bài tập lập trình 5

Sinh số nguyên tố lớn

Bài tập này nhằm sinh các số nguyên tố lớn. Các số này phải đảm bảo:

1. Số được sinh có độ dài 3072 bit,
2. Số được sinh ra phải ngẫu nhiên (khó dự đoán), và
3. Thời gian sinh mỗi số phải nhanh, thời gian trung bình nên nhỏ hơn 5 giây/1 số trên các máy tính thông thường.

Bạn không cần phải cài đặt lại thư viện số nguyên lớn mà có thể sử dụng thư viện số lớn có sẵn https://gmplib.org, *nhưng phải cài đặt hàm kiểm tra tính nguyên tố*. Các Mục 1, 2 và 3 dưới đây mô tả thuật toán sinh số nguyên tố lớn mà bạn nên sử dụng. Bạn phải trình bày thực nghiệm cho cài đặt của bạn như mô tả trong Mục 4.

1 Thuật toán sinh số nguyên tố lớn

Định nghĩa 1. Số nguyên dương $p > 1$ là **nguyên tố** nếu p không chia hết cho số nguyên dương nào ngoài 1 và p. Ngược lại p là **hợp số**.

Ta định nghĩa hàm phân phối của các số nguyên tố $\pi(n)$ là số lượng số nguyên tố nhỏ hơn hoặc bằng n.

Ví dụ 1. $\pi(10) = 4$ vì có đúng bốn số nguyên tố nhỏ hơn 10 là $2, 3, 5, 7$.

Định lý sau đây cho ta ước lượng xấp xỉ hàm $\pi(.)$.

Định lý 1.

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1.$$

Nói cách khác, giá trị $\pi(n)$ xấp xỉ bằng với $n/\ln n$ khi n lớn.

Ví dụ 2. Với $n = 10^9$ ta có

$$\pi(n) = 50847534 \qquad \text{và} \qquad n/\ln n = 48254942.$$

Sai số ở đây là 6%.

Vậy nếu ta lấy ngẫu nhiên một số nguyên dương k bit, xác suất để số này là số nguyên tố bằng $1/\ln 2^k$. Vậy về trung bình, ta cần $\ln 2^k$ lần thử để lấy được một số nguyên tố k bit.

Ví dụ 3. Nếu $k = 3072$ thì, trung bình lấy ngẫu nhiên

$$\ln 2^{3072} \approx 2130$$

số, ta sẽ có được một số nguyên tố k bit.

Từ phân tích ở trên, về trung bình thuật toán ngẫu nhiên dưới đây sẽ dừng sau 2130 bước lặp.

Thuật toán sinh số nguyên tố

1. Chọn ngẫu nhiên một số nhị phân p độ dài 3072 bit.

2. Đặt cả hai bit cao nhất và bit thấp nhất của p lên 1.

3. Kiểm tra xem p có là số nguyên tố.

4. Nếu có thì trả ra số nguyên tố p. Còn nếu không thì quay lại Bước 1.

Trong Bước 2 của thuật toán,

- ta đặt bit thấp nhất của p lên 1 đảm bảo rằng p là số lẻ;
- và ta đặt bit cao nhất của p lên 1 để đảm bảo rằng số p sinh ra thỏa mãn

$$2^{3072-1} < p \leq 2^{3072} - 1,$$

điều này cần thiết vì để đảm bảo an toàn cho thuật toán RSA thì các số nguyên tố sinh ra phải đủ lớn.

Các bước 1 và bước 3 sẽ được phân tích ở mục 2 và mục 3.

2 Sinh số ngẫu nhiên

Hệ điều hành GNU/Linux sinh các dãy bit ngẫu nhiên từ các nguồn ngẫu nhiên của hệ thống (các thao tác chuột, các thao tác bàn phím, các chương trình chạy…). Các dãy bit này được lưu trữ trong tệp tin `/dev/urandom`. Trong trường hợp các dãy bit sinh ra chưa đủ ngẫu nhiên do entropy nhỏ, hệ thống sẽ sử dụng một hàm giả ngẫu nhiên an toàn.

3 Thuật toán kiểm tra số nguyên tố

Thuật toán đơn định AKS (Agrawal–Kayal–Saxena primality test) cho phép kiểm tra liệu một số nguyên dương n có là nguyên tố trong thời gian $O(\log^{O(1)} n)$. Tuy nhiên, nó không được sử dụng nhiều trong mật mã vì thời gian chạy chậm. Người ta thường sử dụng thuật toán ngẫu nhiên Rabin-Miller kết hợp với một số bước tiền xử lý để tăng tốc thuật toán.

Thuật toán của Rabin-Miller là thuật toán kiểu Monte Carlo. Nó cho phép kiểm tra một số nguyên n có là nguyên tố hay không với một xác suất sai có thể làm nhỏ tuỳ ý. Thuật toán của bạn nên chọn xác suất sai nhiều nhất chỉ bằng $1/2^{128}$. Cụ thể, nếu thuật toán thông báo n là hợp số, vậy n chắc chắn là hợp số. Còn nếu thuật toán thông báo n là nguyên tố thì xác suất n là hợp số chỉ là $1/2^{128}$.

Phần còn lại trong mục này mô tả từng bước xây dựng và cải tiến thuật toán Rabin-Miller để kiểm tra một số có là nguyên tố.

Trước hết định lý sau đây cho một thuật toán đơn giản để kiểm tra một số là nguyên tố.

Định lý 2 (Fermat nhỏ)**.** *Nếu n là số nguyên tố thì, với mọi $1 \le a < n$,*

$$a^{n-1} = 1 \mod n.$$

Thuật toán như sau:

Thuật toán *CheckFermat(n)*:
Trả lại 0 nếu n là nguyên tố; 1 nếu n là hợp số.

1. **if** $(2^{n-1} \neq 1 \mod n)$ **return** 1; **else return** 0.

Tất nhiên, thuật toán trên có thể cho kết quả sai, nhưng chỉ có một kiểu sai: Nếu nó trả lại 0 thì vẫn có khả năng n là hợp số, nhưng nếu nó trả lại 1 thì chắc chắn n là hợp số.

Ví dụ 4. Với cả bốn số $n = 314,516,645$ và 1105, thuật toán *CheckFermat* cho kết quả là 0, dù các số này đều là hợp số.

Câu hỏi tự nhiên là: Số trường hợp mà thuật toán này cho kết quả sai có nhiều không? Thật ngạc nhiên là số trường hợp sai rất hiếm. Trong 10000 số nguyên dương đầu tiên, thuật toán chỉ nhầm 22 số.

Thuật toán *Witness* sau đây là một cải tiến của thuật toán trước dựa trên hai ý tưởng.

- Thay cơ sở 2 trong thuật toán *CheckFermat()* bằng một số a bất kỳ;
- Kết hợp với Định lý 3 dưới đây để hạn chế số trường hợp sai.

Định lý 3. *Nếu n là số nguyên tố lẻ và $e \ge 1$ thì phương trình*

$$x^2 = 1 \mod n^e$$

chỉ có hai nghiệm $x = 1$ và $x = -1$.

Thuật toán *Witness(a, n)*
Trả lại 0 *nếu* n *là nguyên tố; 1 nếu* n *là hợp số.*

1. Xét t và u thỏa mãn $t \geq 1$, u lẻ và $n - 1 = 2^t u$;

2. Gán $x_0 := a^u \mod n$;

3. **for** $i := 1, 2, \ldots, t$:

4. Gán $x_i := x_{i-1}^2 \mod n$;

5. **if** ($x_i = 1$ và $x_{i-1} \neq 1$ và $x_{i-1} \neq n - 1$) **return** 1;

6. **if** ($x_t \neq 1$) **return** 1; **else return** 0;

Thuật toán trên thực hiện tính $x_t = a^{n-1} \mod n$ bằng cách lặp lại việc tính $x_i^2 \mod n$ bắt đầu từ $x_0 = a^u \mod n$. Nếu $x_t = 1$ thì n là số nguyên tố. Vòng lặp có thể kết thúc sớm nếu

$$x_i = x_{i-1}^2 \neq \pm 1 \mod n$$

Bởi vì khi đó ta kết luận n là hợp số theo theo Định lý 3.

Thuật toán *Rabin-Miller* dưới đây lặp lại s lần tính *Witness(a,n)* để giảm xác suất sai.

Thuật toán *Rabin-Miller (n,s)*
Trả lại 0 *nếu* n *là nguyên tố; 1 nếu* n *là hợp số.*

1. **for** $j := 1, 2, \ldots, s$:

2. Chọn một số ngẫu nhiên a;

3. **if** (*Witness(a,n)* $= 1$) **return** 1;

4. **return** 0;

Xác suất sai của thuật toán *Rabin-Miller* được chỉ ra bởi định lý sau.

Định lý 4. *Với mỗi số nguyên lẻ* $n > 2$ *và số nguyên dương* s*, sai số của thuật toán Rabin-Miller*(n, s) *nhiều nhất là* 2^{-2s}.

Thuật toán sau đây thêm một vài bước tiền xử lý để giảm thời gian chạy của thuật toán *Rabin-Miller*.

Thuật toán kiểm tra xem liệu số nguyên dương n có là nguyên tố

1. Tính sẵn 400 số nguyên tố đầu tiên: $q_1, q_2, \ldots, q_{400}$;

2. **if** (n chia hết cho q_i nào đó) **return** "hợp số";

3. **if**(*CheckFermat*(n) = 1) **return** "hợp số";

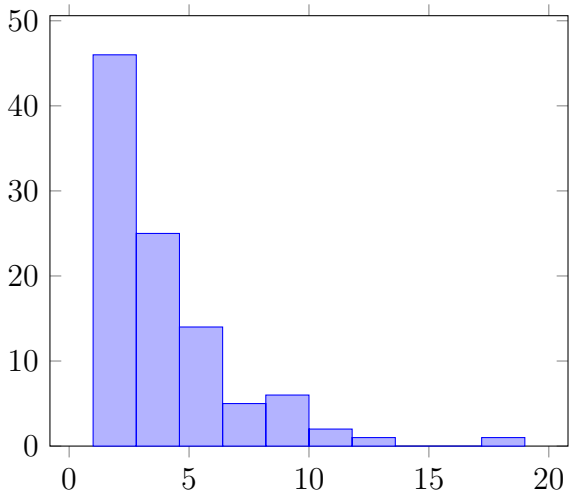
4. **if**(*Rabin-Miller*(n, 128/2) = 1) **return** "hợp số"; **else return** "nguyên tố";

Bước 1 và 2 nhằm *loại rất nhanh* các hợp số chia hết cho 400 số nguyên tố đầu tiên. Bước 3 *loại nhanh* một số lượng lớn các hợp số nhờ Định lý Fermat. Việc chọn cơ sở 2 giúp tính toán nhanh trên bit. Bước cuối cùng nhằm kiểm tra những số còn lại bằng thuật toán *Rabin-Miller*. Bước 4 cần nhiều tính toán.

Nếu thuật toán này thông báo "n là hợp số", vậy chắc chắn n là hợp số. Còn nếu nó thông báo "n là số nguyên tố", vậy xác suất n là hợp số chỉ là $1/2^{128}$.

4 Thử nghiệm

Dưới đây mô tả một thử nghiệm để sinh 100 số nguyên tố độ dài 3072 bit. Chương trình chạy trên máy laptop có bộ xử lý Intel Core i5-540M (2.53GHz, 3MB Cache), Ram 4Ghz, và chạy hệ điều hành GNU/Linux (Debian Jessie). Thời gian trung bình để sinh ra mỗi số khoảng 3 giây.



Hình 1: Mô tả histogram thời gian chạy để sinh 100 số nguyên tố. Cột mô tả số lượng số chạy trong thời gian chỉ ra bởi hàng.

5 Tài liệu tham khảo

1. Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, Charles E. Leiserson, *Introduction to Algorithms*, McGraw-Hill Higher Education 2001.

2. Victor Shoup, *A computational introduction to number theory and algebra*. Cambridge University Press 2006,

3. Jonathan Katz và Yehuda Lindell, *Introduction to Modern Cryptography*, Chapman & Hall/CRC, 2007.