ĐẠI HỌC CÔNG NGHỆ

ĐẠI HỌC QUỐC GIA HÀ NỘI

----ଓ✧ଌ----

VẤN ĐỀ KIỂM TRA CÁC SỐ NGUYÊN TỐ LỚN

BÁO CÁO MÔN HỌC

Học viên: Ma Trọng Khôi

Lớp : An Ninh Cơ Sở Dữ Liệu

Giảng viên: PGS. Trịnh Nhật Tiến

Hà Nội – 2013

Mục lục

Mục lục **1**

Chương 1 ..... **2**

Số nguyên tố **2**

Chương 2 ..... **4**

Kiểm tra số nguyên tố **4**

1. Phương pháp chia thử (trial division) 4
2. Phương pháp sàng 5
3. Định lý nhỏ Fermat 6
4. Các thuật toán kiểm tra số nguyên tố lớn 8
5. Số nguyên tố Mersenne, Dự án GIMPS 9

Kết luận **10**

Tài Liệu Tham Khảo **11**

Chương 1

Số nguyên tố

Một số nguyên tố là một số tự nhiên lớn hơn 1 và chỉ có hai ước duy nhất là số 1 và chính nó. Một số tự nhiên lớn hơn 1 và không phải là số nguyên tố thì được gọi là hợp số. Ví dụ, số 5 là số nguyên tố vì nó chỉ chia hết cho hai số là 1 và 5, ta cũng có số 6 là hợp số vì ngoài 1 và 6 nó còn chia hết cho 2 và 3.

Định nghĩa số nguyên tố đã được đề cập đế từ trước đây rất lâu. Trong một số tài liệu, số nguyên tố được đề cập đến đầu tiên bởi người Ai Cập cổ đại, tuy nhiên một số tài liệu khác lại nói Trung Quốc mới là nơi đầu tiên đưa ra khái niệm này[1]. Cả hai nền văn minh này đều không có nghiên cứu gì thêm. Nhứng nghiên cứu và ghi chép đầu tiên về số nguyên tố lại đên từ Hi Lạp (thế kỷ 300 trước công nguyên) của Euclid. Các lý thuyết cơ bản về số nguyên tố hay thuật toán sang Eratosthenes đã được giới thiệu từ thời kỳ này.

Sau thời đại của đế chế Hi Lạp, các nghiên cứu về số nguyên tố chỉ bắt đầu trở lại từ thế kỷ 17 cho đến hiện tại với khởi đầu là các các nghiên cứu của Fermat. Nhiều thuật toán xác định số nguyên tố nhanh hơn, hiệu quả hơn của nhiều nhà khoa học đã được ra đời, như phương phương pháp kiểm tra số Fermat có phải nguyên tố của Pepin (1877), lý thuyết của Proth (1878), phương pháp kiểm tra số nguyên tố Lucas-Lehmer (1856), và nhiều thuật toán được giới thiệu gần đây hơn như APRT-CL, ECPP, AKS.

Trong một thời gian dài, khả năng ứng dụng của số nguyên tố ngoài toán học được xem là rất hạn chế. Điều này đã được thay đổi từ thập kỷ 1970, khi khái niệm mã hóa sử dụng khóa công khai được phát minh. Thuật toán RSA sử dụng khóa được tính từ các số nguyên tố rất lớn có độ dài từ 100-200 ký tự. Nếu bây giờ bạn phát minh được một thuật toán phân tích thành thừa số nguyên tố nhanh và hiệu quả hơn thì rất nhiều hệ thống sử dụng khóa công khai sẽ gặp vấn đề

Từ năm 1951, với sự ra đời của máy tính, các số nguyên tố lớn nhất từ trước đến giờ đều lần lượt được tìm ra bởi máy tính. Bài toán làm sao tìm được các số nguyên tố lớn hơn dần được quan tâm và cũng là thách thức nằm ngoài vòng quay của toán học. Một số dự án như Great Internet Mersenne Prime Search hay các dự án tính toán phân

tán khác để tìm số nguyên tố lớn nhất bắt đầu trở nên phổ biến trong khoảng 15 năm trở lại đây.

Trong các phần tiếp theo ta sẽ lần lượt nghiên cứu một số thuật toán kiểm tra số nguyên tố từ cơ bản đến phức tạp.

Chương 2

Kiểm tra số nguyên tố

1. Phương pháp chia thử (trial division)

Một phương pháp đơn giản và thích hợp cho việc kiểm tra nguyên tố cho các số nhỏ là ta thử các phép chia cho các số nhỏ hơn. Giả sử cần kiểm tra một số tự nhiên n là nguyên tố hay hợp số. Ta tiến hành các phép chia thử lần lượt cho các số tự nhiên nhỏ hơn n và lớn hơn 1, nếu n chia hết cho ít nhất một số thì n là hợp số, còn nếu sau vòng lặp kiểm tra n không chia hết cho bất kỳ số nào thì n là số nguyên tố.

Tuy nhiên phương pháp này thực hiện khá nhiều phép tính dư thừa. Từ thời Hi Lạp cổ đại, người ta đã chứng minh được là một hợp số sẽ có thể phân tích được thành tích các thừa số nguyên tố nhỏ hơn nó và kết quả này là duy nhất. Như vậy ta chỉ cần phép chia thử với các số nguyên tố có giá trị nhỏ hơn căn bậc hai (sqrt(n)) của n. Việc xác định các số nguyên tố nhỏ hơn n lại làm nảy sinh vấn đề mới. Ta có thể đơn giản bằng cách chỉ chia thử cho các số lẻ (vì số nguyên tố đều là số lẻ trừ số 2). Đoạn code bên giới biểu diễn cho phương pháp trên, và vẫn hoạt động khá hiệu quả khi tiến hành các phép thử với số nguyên tố lớn hơn 1 tỷ chạy trên laptop cá nhân cho kết quả dưới 1s.

```cpp
/*Ma Trong Khôi*/
#include <iostream>
using namespace std;
bool isPrime (long long n) {
    if (n <= 3) return true;
    if ((n & 1) == 0 || (n % 3) == 0)  return false;
    long long i = 5;
    while (i * i <= n) {
        if ((n % i) == 0) return false;
        int j = i + 2;
        if (j*j <= n && (n % j) == 0) return false;
        i += 6;
    }
    return true;
}
int main() {
    long long n = 20056049013111;
    cout << isPrime(n);
    return 0;
}

```

2. Phương pháp sàng

Phương pháp sàng giải quyết bài toán liệt kê tất cả các số nguyên tố nhỏ hơn n.

2.1 Sàng Eratosthenes

Từ thế kỷ 3 trước công nguyên, Eratosthenes đã giới thiệu một phương pháp cũng mang tên ông cho bài toán này [4]. Phương pháp này hoạt động hiệu quả với n không quá lớn. Một đoạn code ví dụ cho thuật toán sàng:

```
4   bool arr[1000000];
5   int main() {
6       int n = 100;
7       for (int i = 4; i <= n; i+=2) arr[i] = true;
8
9       int i = 3;
10      while (i * i <= n) {
11          i += 2;
12          if (arr[i]) continue;
13          int j = i + i;
14          while (j <= n) {
15              arr[j] = true;
16              j += i;
17          }
18      }
19
20      for (int i = 2; i <= n; i++) {
21          if (!arr[i]) cout << i << ' ';
22      }
23      return 0;
```

2.2 Sàng Atkin

Một phương pháp sàng hiệu quả hơn, nhanh hơn, tố ít bộ nhớ hơn là sàng Atkin, được Atkin và Bernstein giới thiệu vào năm 2004 [4]. Tính đúng đắn và hiệu quả của nó đã được chứng minh trong bài báo của họ [5]. Việc cài đặt thuật toán cũng khá đơn giản, ta có thể tham khảo đoạn mã C++ như ở dưới.

```cpp
using namespace std;

void findPrimes(int arraySize);
bool isPrime[100000000];
int main()
{
    int arraySize = 10000000;

    int sqrtArraySize;
    sqrtArraySize = sqrt(arraySize);

    for (int x = 1; x <= sqrtArraySize; x++) {
        for (int y = 1; y <= sqrtArraySize; y++) {
            int n = 4 * x * x + y * y;
            if (n <= arraySize && (n % 12 == 1 || n % 12 == 5)) isPrime[n] = !isPrime[n];

            n = 3 * x * x + y * y;
            if (n <= arraySize && (n % 12 == 7)) isPrime[n] = !isPrime[n];

            if (x > y) {
                n = 3 * x * x - y * y;
                if (n <=arraySize && n % 12 == 11) isPrime[n] = !isPrime[n];
            }
        }//end 2nd FOR LOOP
    }//end 1st FOR LOOP

    for (int n = 5; n <= sqrtArraySize; n++) {
        if (isPrime[n]) {
            int omit = n * n;
            for (int k = omit; k <=arraySize; k +=omit)
                isPrime[k] = false;
        }
    }

    for (int x = 2; x <= arraySize; x++)
        if (isPrime[x])
            cout << x << "\t" << flush;

    return 0;
}
```

3. Định lý nhỏ Fermat

Mục này sẽ giới thiệu một phương pháp kiểm tra số nguyên tố nhanh và hiệu quả hơn, các phương pháp được giới thiệu hoạt động chủ yếu dựa trên định lý nhỏ Fermat [6].

Định lý: Nếu p là một số nguyên tố và a là một số tự nhiên đồng thời nguyên tố cùng nhau với p thì:

$$a^{p-1} \equiv 1 (\text{mod } p) \quad (1)$$

Định lý này đúng và thích hợp cho việc kiểm tra một số có phải là hợp số hay không (Ngoài ra người ta còn hay dùng một phương pháp khác do Wilson giới thiệu – nếu p là số nguyên tố thì (p-1)! ≡ -1(mod p)). Người ta thường sử dụng phương pháp này do việc tính kết quả của (1) rất nhanh. Đoạn mã dưới đây mô tả phép tính a^d chia lấy dư cho N, với độ phức tạp là một hàm log của d [6].

```cpp
using namespace std;

int main()
{
    int a,d,n,a2i,rs;
    cout << "a, d, N: ";
    cin >> a >> d >> n;
    cout << "a^d % N= ";
    rs = 1;
    a2i = a;
    while (d) {
        if (d & 1) rs = (rs * a2i) % n;
        d >>= 1;
        a2i = (a2i * a2i) % n;
    }
    cout << rs << endl;

    return 0;
}
```

Tuy nhiên định lý chỉ thỏa mãn điều kiện cần, còn ngược lại nếu một số a nguyên tố cùng nhau với p và thỏa mãn điều kiện (1) thì chưa chắc a đã là số nguyên tố, ví dụ như số 561 chia hết cho 3, nhưng ta có $2^{560} \equiv 1 \pmod{561}$. Các số có tính chất như vậy được gọi là giả số nguyên tố.

3.1 Kiểm tra số nguyên tố dựa theo số nguyên tố mạnh

Một số tự nhiên lẻ N là hợp số và $N - 1 = d \cdot 2^s$, d là số lẻ. N được gọi là giả số nguyên tố mạnh theo hệ số a nếu nó là giả số nguyên tố Euler theo hệ số a [6]. Và thỏa mãn điều kiện sau:

$$a^d \equiv 1 \pmod{N} \text{ hoặc } a^{d.2^s} \text{ , với mọi } r = 0, 1, 2, \ldots, s - 1.$$

Số lượng giả số nguyên tố mạnh là rất ít và phụ thuộc vào hế số a. Người ta đã chỉ ra được trong 10^{12} số tự nhiên đầu tiên chỉ có 101 số là số nguyên tố mạnh theo hệ số 2, 3, 5 [6]. Với hệ số 7 thì chỉ có 9 giả số nguyên tố mạnh trong 10^{12} số tự nhiên đầu tiên. Phương pháp sau đây cho phép kiểm tra nguyên tố với các số có giá trị nhỏ hơn $3,4 \cdot 10^{14}$, cụ thể là 341550071728321. Đoạn mã dưới mô tả lại phương pháp trong [6] khi áp dụng giả số nguyên tố mạnh để kiểm tra số nguyên tố.

```
long long d = N - 1, s= 0;
while(!(d & 1)) {
    d >>= 1;
    s++;
}

long long d1 = d;
int sw;
for (int i = 1; i < 8; i++) {
    sw = 0;
    long long a = base[i], prod = 1, a2j = a, d = d1;

    while (d) {        int base
        if (d & 1) prod = (prod * a2j) % N;
        d >>= 1;
        a2j = (a2j * a2j) % N;
    }

    if ((prod == 1) || (prod == (N - 1))) goto label1;
    else {
        for (int j = 1; j <= s - 1; j++) {
            prod = prod * prod % N;
            if (prod == 1) goto label2;
            if (prod == N - 1) goto label1;
        }
    }
    goto label2;
    label1: if (N < limit[i]) {
        sw = 1; goto label2;
    }
}
label2: if (sw == 0) {
    cout << N << " is composite" << endl;
} else {
    cout << N << " is prime" << endl;
}
```

4. Các thuật toán kiểm tra số nguyên tố lớn

Đã có rất nhiều thuật toán kiểm tra số nguyên tố lớn được phát triển, dưới đâylà một số thuật toán hiệu quả và thông dụng, ngoài ra còn rất nhiều thuật toán khác nữa.

Test	Developed in	Type	Running time
AKS primality test	2002	deterministic	$O(\log^{6+\varepsilon}(n))$
Elliptic curve primality proving	1977	deterministic	$O(\log^{5+\varepsilon}(n))$ *heuristica*
Miller–Rabin primality test	1980	probabilistic	$O(k \cdot \log^{2+\varepsilon}(n))$
Solovay–Strassen primality test	1977	probabilistic	$O(k \cdot \log^{3} n)$
Fermat primality test		probabilistic	$O(k \cdot \log^{2+\varepsilon}(n))$

Trong các thuật toán trên thì có thuật toán thứ 2 (ECPP), hiện tại đang được cho là nhanh nhất và được sử dụng phổ biến nhất. Vào năm 2009, số nguyên tố lớn nhất mà thuật toán này tìm ra được có khoảng 20 nghìn chữ số, số nguyên tố này được gọi là Mill's prime [7]. Thuật toán này đã được cải tiến nhiều, hiện tại có cải tiến đã giúp thuật toán đạt được độ phức tạp về thời gian là $O((\ln n)^4)$.

Thuật toán mới về kiểm tra số nguyên tố lớn đáng chú ý gần đây nhất có thể nêu ra là thuật toán AKS, giới thiệu vào năm 2004. Sau đấy thuật toán này cũng được cải tiến rất nhiều, đáng chú ý nhất là cải tiến của Bernstein vào năm 2004 giúp thuật toán đạt được độ phức tạp về thời gian là $O((\log n)^6)$.

5. Số nguyên tố Mersenne, Dự án GIMPS.

Các thuật toán như AKS hay ECPP, dù đã có nhiều cải tiến nhưng vẫn còn khá chậm khi kiểm tra các số nguyên tố cực lớn. Ví dụ như chương trình PRIMO, sử dụng thuật toán ECPP được giới thiệu vào năm 2004, có thể tìm ra được số nguyên tố 4769 chữ số sau khoảng 2000 giờ tính toán – hơn 2 tháng. Trong khi số nguyên tố lớn nhất người ta tìm được vào năm 2013 có 17 triệu chữ số.

Người ta nhận ra rằng các số có dạng: $2^n - 1$ thường có khả năng là số nguyên tố, số này còn được gọi là số Mersenne.

General Internet Mersenne Prime Search (GIMPS) là dự án được thành lập bởi Geoger Wolman với mục tiêu chuyên kiểm tra các số Mersenne lớn có phải là số nguyên tố hay không. Dự án sử dụng thuật toán Lucas-lelmer kết hợp với biến đổi Furious nhanh. Đến năm 2013 thì cả 11 chữ số nguyên tố lớn nhất từ trước đến giờ đều được tìm ra bởi GIMPS.

Kết luận

Rất nhiều phương pháp và thuật toán để giải nguyến các vấn đề liên quan đến số nguyên tố lớn đã được nghiên cứu và đề xuất. Tuy nhiên cho đến nay vẫn còn rất nhiều khó khăn và thách thức trong bài toán này cần giải quyết. Trong giới hạn của một bài báo cáo, người viết chỉ có thể giới thiệu được các nội dung được dánh giá là tổng quan nhất.

Các thuật toán được giới thiệu trong bài đều đã được người viết thực nghiệm lại, bạn đọc quan tâm có thể tham khảo thêm trên internet hoặc xem mã nguồn thực nghiệm của bài báo cáo này trong [12].

Xin chân thành cảm ơn.

Tài Liệu Tham Khảo

[1] http://primes.utm.edu/largest.html

[2] The Little Book of The Bigger Primes 2nd, *Paulo Ribenboim*

[3] http://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

[4] http://en.wikipedia.org/wiki/Sieve_of_Atkin

[5] http://cr.yp.to/papers/primesieves.pdf

[6] Prime Numbers A Computational Perspective 2nd

[7] http://mathworld.wolfram.com/MillsPrime.html

[8] http://mathworld.wolfram.com/EllipticCurvePrimalityProving.html

[9] Proving primality after Agrawal-Kayal-Saxena, *Daniel J. Bernstein*
http://cr.yp.to/papers/aks.pdf

[10] http://primes.utm.edu/bios/page.php?lastname=Woltman

[11] http://en.wikipedia.org/wiki/Lucas%E2%80%93Lehmer_primality_test

[12] https://github.com/khoimt/primetest