Thử tìm một thuật toán để xác nhận số nguyên tố

Phạm Đăng Long

Nội dung chính:

§1. Giới thiệu sơ lược về thuật toán
1. Khái niệm về thuật toán
2. Tính chất cơ bản của thuật toán
3. Trình bày thuật toán
4. Các cấu trúc cơ bản thuật toán
5. Một số ví dụ

§2. Tìm một thuật toán cho bài toán xác nhận số nguyên tố
1. Nêu bài toán
2. Một số thuật toán đã biết để giải câu bài toán chính và Thuật toán đề nghị.

Cụ thể:

§1. Giới thiệu sơ lược về thuật toán (Giành cho bạn nào muốn tìm hiểu về Thuật toán)

1. Khái niệm về Thuật toán

Thuật toán (Algorithm) là một gói thông tin về một dãy hữu hạn các chỉ thị khả thi để thực hiện theo một thứ tự xác định trong đó thì thu được kết quả mong muốn.

2. Tính chất cơ bản của thuật toán

- Số chỉ thị phải hữu hạn.
- Mỗi chỉ thị phải khả thi.
- Kết quả phải đúng đắn.
- Phải có tính chất phổ dụng.

Giá trị của một Thuật toán thể hiện ở kết quả đúng đắn hoặc gần đúng với độ chính xác cao và tổng thời gian thực hiện nhỏ và tốn ít tài nguyên, bộ nhớ.

3. Trình bày thuật toán

- Nêu từng bước của Thuật toan (còn gọi là Giải thuật).
- Vẽ lưu đồ, sơ đồ khối (Diagram hay Flowchart).
- Giả lập trình (Pseudo-Programming).

4. Các cấu trúc cơ bản thuật toán

- *Tuần tự:* Lần lượt thực hiện từ trên xuống, theo thứ tự Thuật toán đề ra.
- *Rẽ nhánh:*

  a/- Nếu ... thì ... Trái lại thì ...

  If <Điều kiện> then <Chỉ thị 1>;
  Else <Chỉ thị 2>;

  b/- Nếu ... thì ... (mà không cần Trái lại).

  If <Điều kiện> then <Chỉ thị 1>;
- *Nhẩy tự do* sau kiều kiện nào đó.

  Ví dụ: Nhập mật khẩu:

  Bước 1: Viết "Mật khầu = ".
  Bước 2: Nhập mật khẩu.
  Bước 3: Nếu sai thì nhẩy về Bước 1.
- *Ghép vài chỉ thị* thành một nhóm) sau một điều kiện nào đó.

  Ví dụ: Giải phương trình bậc nhất ax + b = 0:

  Nếu $a \neq 0$ thì {Cho x = -b/a; Viết "Đáp số: x = "; Viết giá trị của x}

- *Lặp:*

a/- *Có số lần cho trước (Duyệt):* Cho biến i chạy từ 1 đến n: Với mỗi i thực hiện thao tác.

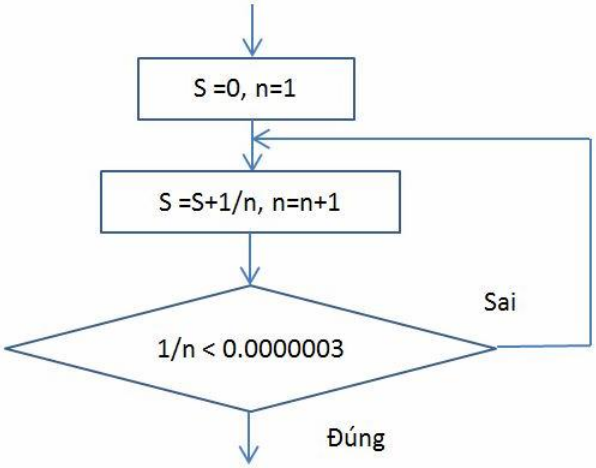
Ví dụ: Tính tổng $S = a_1 + a_2 + ... + a_{100}$.

Cho tổng S = 0.

Cho chỉ số i tăng dần từ 1 đên 100: Với mỗi i đó thực hiện chỉ thị $S = S + a_i$.

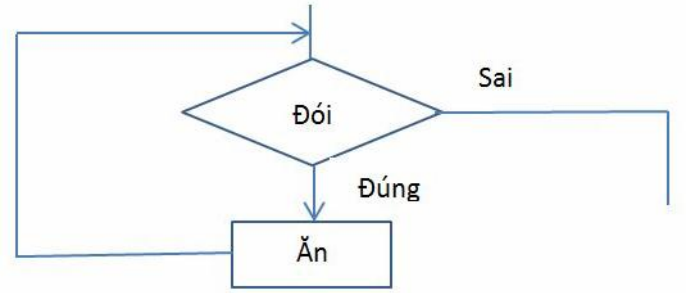
b/- *Hãy thực hiện chỉ thị cho đến khi gặp điều kiện dừng*.

Ví dụ: Tính tổng S = 1/1 + 1/2 + ... + 1/n + ... đến khi nào 1/n < 0.0000003.



c/- *Chừng nào còn thỏa mãn điều kiện thì còn thực hiện chỉ thị*.

Ví dụ: Còn đói thì còn ăn!



Những thuật toán dù phức tạp đến mấy cũng sẽ được thể hiện qua các cấu trúc cơ bản nêu trên ở những đoạn thích hợp. Những thuật toán nổi tiếng phải kể đến như là: Đệ quy, Truy hồi, Vét cạn quay lui, Săn sư tử trên sa mạc, Các thuật toán sắp xếp, Thuật toán Horner, Thuật toán Gauss, Ơclit và rất nhiều đã dùng trong Toán học hay các ngành khoa học kỹ thuật khác!

5. Một số ví dụ

Ví dụ 1: Thuật toán cái máy giặt (sau khi đã đặt chế độ giặt kỹ/dối và chất liệu vải)

Bước 1: Hút nước và đun nước.

Bước 2: Quay vật vã.

Bước 3: Tháo nước và đo độ đục của nước.

Bước 4: Nếu nước còn đục thì về Bước 1.

Bước 5: Quay ly tâm.

Bước 6: Tháo nước ra và dừng máy.

Ví dụ 2: Thuật toán Horner tính giá trị đa thức: $P = a_n x^n + a_{n-1}x^{n-1} + ... + ax + a_0$, với x cho trước.

$P = (...(a_n x + a_{n-1})x + a_{n-2})x + ...)x + a_0$,

Trình bày theo cách giả lập trình:

Input n và x;

For k = n downto 0 do (Input a_k);

Cho P = 0;

For k = n downto 0 do {P ← P*x + a_k; k = k – 1}.

Print "Đáp số: P = " và giá trị P. Hết.

Chẳng hạn: $5x^3 + 2x^2 + 7x + 8 = ((5x + 2)x + 7)x + 8$. Tính giá trị đa thức này với x = 6.
Khi đó quá trình thực hiện thuật toán như sau:

Bậc n = 3; x = 6;
Các hệ số $a_3 = 5$, $a_2 = 2$; $a_1 = 7$ và $a_0 = 8$;
P = 0;
k = 3; P = 0*6 + a_3 = a_3 = 5; k = 3 - 1 = 2;
k = 2; P = 5*6 + a_2 = 32; k = 2 - 1 = 1;
k = 1; P = 32*6 + a_1 = 199; k = 1 - 1 = 0;
k = 0; P = 199*6 + a_0 = 1194; k = 0 – 1 = -1;
Đáp số: P = 1194.

Xem thêm minh họa: **Cấu trúc lặp** (Youtube).

$2. Tìm một thuật toán cho bài toán xác nhận số nguyên tố

1. **Nêu bài toán:** Số tự nhiên n được gọi là số nguyên tố (Prime Number) khi n có đúng hai ước số là 1 và chính nó. Số tự nhiên n không là số nguyên tố, còn được gọi là hợp số (Combine Number).
   Số nguyên tố đóng vai trò quan trong trong nhiều lĩnh vực, đặc biệt là trong kyc thuật mật mã.
   Có nhiều bài toán liên quan đến khái niệm số nguyên tố, song có hai bài toán sau đây đước nhiều người chú ý:
   a/- Liệt kê các số nguyên tố trong một tập hợp số tự nhiên nào đó. Bài này đã có thuật toán Sàng Eratosthenes nổi tiếng, hầu như ai cũng biết!
   b/- Cho trước một số tự nhiên n tùy ý. Xét xem n có là số nguyên tố không?

2. **Một số thuật toán đã biết để giải câu b/ và Thuật toán đề nghị**.
   Cho trước một số tự nhiên n, có thể tương đối lớn đi.

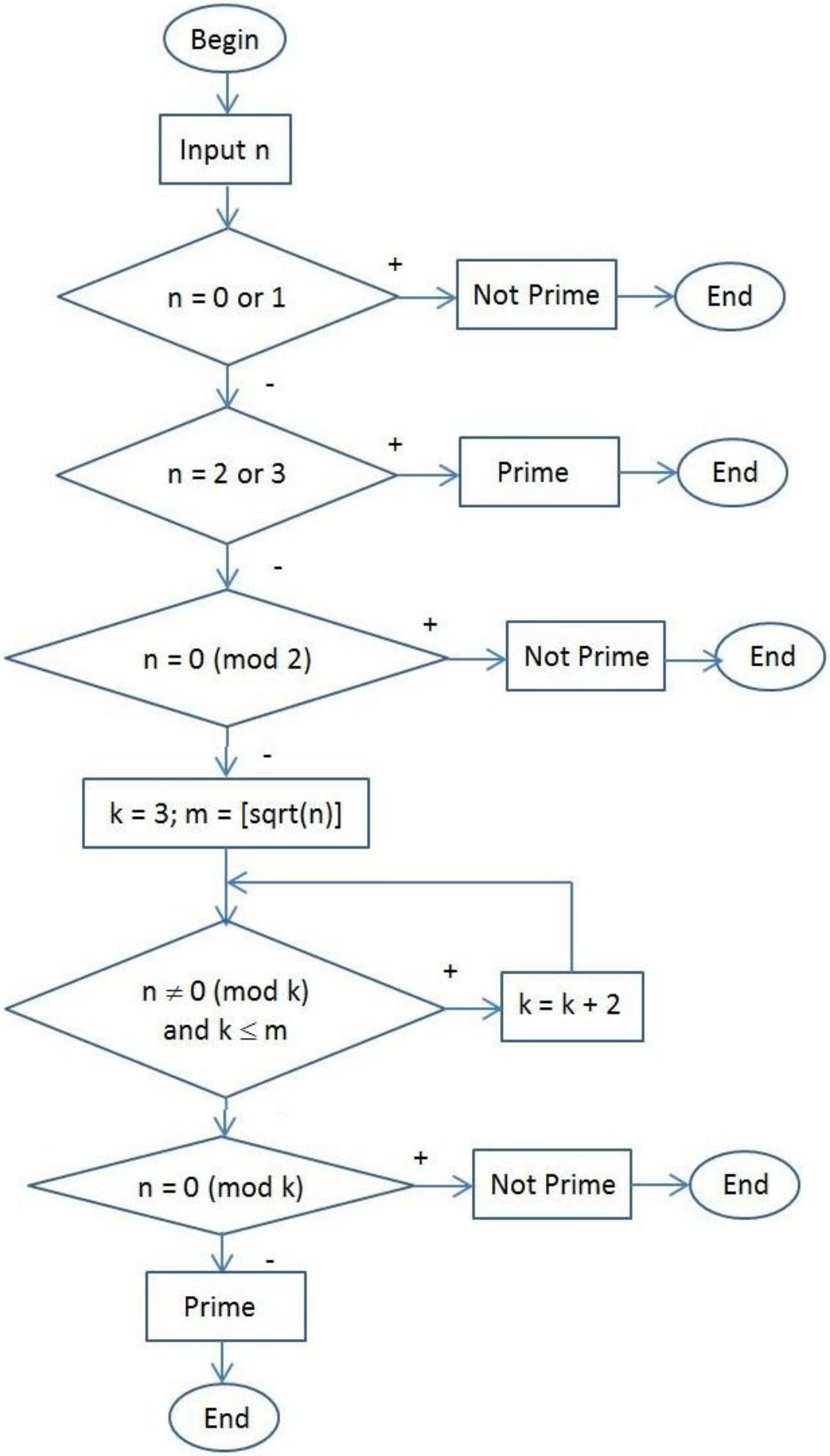
   Thuật toán 1: (Dựa theo định nghĩa số nguyên tố: Thử chia n cho các số k từ 2 đến n-1)
   Duyệt k từ 2 đến n - 1: Với mỗi k đó:
   Nếu n chia hết cho k thì {Số n là hợp số ; Thoát khỏi Thuật toán}
   Xuất kết quả n là số nguyên tố.
   (Còn gọi là thuật toán "ngây thơ", phải làm n – 2 phép chia, quá nhiều!).

   Thuật toán 2: (Theo tính chất: Mỗi số tự nhiên n không chia hết cho số lớn n/2).
   Gần giống như Thuật toán 1, chỉ khác mốc là [n/2], nhưng khá hơn, ít phép chia hơn.

   Thuật toán 3: (Dựa theo tính chất rằng mỗi số tự nhiên n không chia hết cho k > [$\sqrt{n}$], vì nếu chia hết thì thương là một số k' ≤ [$\sqrt{n}$] đã được xét trước đó rồi).
   Gần giống như Thuật toán 1, chỉ khác mốc là [$\sqrt{n}$], nhưng khá hơn nữa vì chỉ phải làm ít phép chia hơn!.
   - Vậy đã tốt chưa?
   - Vẫn chưa tốt vì phải thử chia cho nhiều số không cần thiết, đó là các số chẵn!

   Thuật toán 4: (Có lẽ sẽ tốt hơn nữa)
   Nếu n = 0 hoặc 1 thì kết luận ngay n là hợp số;
   Trái lại: Nếu n = 2 hoặc 3 thì n là số nguyên tố;
   Trái lại: Nếu n chẵn thì n là hợp;
   Trái lại (từ đây n là số lẻ lớn hơn 3): {
   Cho k = 3; Cho m = $\sqrt{n}$; (m có thể không nguyên)
   Chừng nào (k không chia hết n và k ≤ m) thì cho k tăng 2;
   Nếu k > m thì n là số nguyên tố;
   Trái lại n là hợp số.
   }
   (Chỉ phải làm [(m – 1)/2] phép chia!).

Thuật toán 4 trình bày dưới dạng lưu đồ như sau: (n, k và m là các biến nguyên)



Chú ý rằng khi đặt sẵn mốc m = [sqrt(n)] thì ta có thể dùng cấu trúc For … do như sau :

```
Input n ;
If  (n = 0) or (n = 1) then n is not prime;
Else
        If (n = 2) or (n = 3) then n is prime;
        Else
                If  n = 0 (mod 2) then n is not prime;
                Else  //từ đây n là số lẻ lớn hơn 3:
                        {
                        m = [ √n ]; //m nguyên
                        for (k = 3; k ≤ m; m + 2)
                                if (n = 0 (mod k)
                                        {
                                        n is not prime;
                                        return(); //Kết thúc
                                        }
                        n is not prime;
                         }
```

Áp dụng Thuật toán 4 :

Ví dụ 1. Với n = 91.

Ta đặt m = $\sqrt{91} \approx 9$. Vì 91 không rơi vào trường hợp đặc biệt nên ta thực hiện:
k = 3 ;
91 không chia hết cho k và k ≤ 9 nên k = k + 2 = 5.
91 không chia hết cho k và k ≤ 9 nên k = k + 2 = 7.
91 chia hết cho 7 nên điều kiện không thỏa mãn, k không được tăng thêm nữa, và phải ra ngoài vòng lặp.
Lúc này, do 91 = 0 (mod 7) nên kết luận 91 không là số nguyên tố.

Ví dụ 2. Với n = 97.

Ta đặt m = $\sqrt{97} \approx 9$. Vì 97 không rơi vào trường hợp đặc biệt nên ta thực hiện:
k = 3 ;
97 không chia hết cho k và k ≤ 9 nên k = k + 2 = 5.
97 không chia hết cho k và k ≤ 9 nên k = k + 2 = 7
97 không chia hết cho k và k ≤ 9 nên k = k + 2 = 9
97 không chia hết cho k và k ≤ 9 nên k = k + 2 = 11.
11 không còn ≤ m nữa, nên điều kiện không thỏa mãn, k không được tăng thêm nữa, và phải ra ngoài vòng lặp.
Lúc này, do điều kiên 97 ≠ 0 (mod 7) nên kết luận 97 là số nguyên tố.

Thuật toán 4 rất tốt nhưng vẫn chưa phải tối ưu, nhất là xác nhận số tự nhiên rất lớn. Nên chăng, ta thiết lập trước một dãy (p_k) gồm các số nguyên tố ≤ m, ở đây m = [$\sqrt{n}$]. Rồi khi cho k chạy từ 2 tới m, ta chỉ cần xét xem n có chia hết cho số nguyên tố p_k nào đó của dãy không! Để ý là trong khoảng từ p_2 tới m, lực lượng các số nguyên tố ít hơn lực lượng các số lẻ, nên số lần thử chia sẽ ít hơn!

Vậy phải cải tiến Thuật toán 4 thành Thuật toán 5, gồm hai pha sau:
Pha 1. Dùng Sàng Eratosthenes để rồi tạo ra dãy (p_k) các số nguyên tố trong khoảng từ 2 đến m.
Pha 2. Cải tiến bước nhảy k = k + 2 thành việc bước nhảy tự nhiên của chỉ số của dãy (p_k).

Thuật toán 5.

```
Input n ; Cho m = [ √n ]; //m nguyên
//Tạo ra dãy (pk) các số nguyên tố từ 2 đến m.
```

Cho i chạy từ 2 đến m: Cho $p_i = 1$; //Chuẩn bị "sàng", sau đó là sàng:
Cho i chạy từ 2 đến m//Duyệt từ 2 đến m
Nếu ($p_i = 1$) thì //Nếu nhãn là 1 thì cho các bội số của i thành 0.
{
Cho k = 2; //Hệ số để nhân với i
while (k*i <= m) {Cho $p_{i*k} = 0$; k = k+1} //Cho nhãn của bội của i bằng 0
}
//Bằng cách "sàng" đó, các só nguyên tố sẽ có nhãn là 1, hợp số có nhãn là 0.
//Thu lấy dãy (p_k) là các số nguyên tố: $p_1 = 2$, $p_2 = 3$, ...
Cho k = 0; //Chỉ số "lấy đà" của dãy (p_k)
Cho i chạy từ 2 đến m // i nào có nhãn là 1 thì chỉ số k tăng 1 đơn vị và cho $p_k = i$
Nếu ($p_i = 1$) thì {Cho k = k + 1; Cho $p_k = i$;}
Cho m = k; //Số lượng các số nguyên tố của dãy đến đây được ấn định.
//Phần chính của Thuật toán 5
If (n = 0) or (n = 1) then n is not prime;
Else
If (n = 2) or (n = 3) then n is prime;
Else
If n = 0 (mod 2) then n is not prime;
Else //từ đây n là số lẻ lớn hơn 3:
{
For (k = 2; $k \leq m$; k++) //k = 2 tức là $p_k = 3$
If (n = 0 (mod p_k) then {n is not prime; Exit; //Thoát}
n is prime;
}

Thuật toán 5 có 3 bước lớn là:

Bước 1. Nhập n và tính mốc m = [$\sqrt{n}$].

Bước 2. Dùng Sàng Eratosthenes để tìm dãy (p_k) các số nguyên tố $\leq$ m, đánh số lại thứ tự các số nguyên tố từ 1 đến và m (mốc mới) sẽ là số lượng các số nguyên tố $\leq$ m (mốc cũ). Như vậy, số phép thử chia sẽ ít hơn hẳn!

Bước 3. Duyệt k từ 2 đến m:
Nếu n = 0 (mod p_k) thì
{
Kết luận: n không phải là số nguyên tố;
Thoát khỏi Thuật toán;
}
Duyệt xong, mà không bị Thoát khỏi Thuật toán thì n là số nguyên tố.

Lấy lại ví dụ minh họa ở trên.
Với n = 97, m = 9;
//Sàng Eratosthenes cho dãy (p_k) cụ thể là $p_1 = 2$, $p_2 = 3$, $p_3 = 5$; $p_4 = 7$. Bây giờ m = 4.
//Vào phần duyệt:
Cho k chạy từ 1 đến 4: Không thấy lúc nào n chia hết cho p_k nên n is prime.
Với n = 91, m = 9.
//Sàng Eratosthenes cho dãy (p_k) cụ thể là $p_1 = 2$, $p_2 = 3$, $p_3 = 5$; $p_4 = 7$. Bây giờ m = 4.
//Vào phần duyệt:
Cho k chạy từ 1 đến 4:
Khi k = 4 thì 91 chia hết cho p_4 nên kết luận n là hợp số và kết thúc quá trình!
Ý tưởng thì vậy, nhưng không biết với số tự nhiên rất lớn khi chạy thời gian có bị đội lên không, , và có đủ bộ nhớ không?! Chắc là phải cần đến lập trình song song, với thuật toán mạnh hơn!

Hà Nội, ngày 9 tháng 5 năm 2019
Phạm Đăng Long