**Sàng Eratosthenes**

**Dẫn nhập**

Toán học trong lập trình là một mảng phổ biến và thú vị nhưng cũng khá phức tạp. Trong các bài học tới đây, chúng ta sẽ đi tìm hiểu về vấn đề này. Mở đầu, chúng ta sẽ cùng nhau đi tìm hiểu về các thuật toán kiểm tra số nguyên tố cũng như một kĩ thuật cho phép đánh dấu nhanh các số nguyên tố.

**Nội dung**

Để có thể tiếp thu bài học này một cách tốt nhất, các bạn nên có những kiến thức cơ bản về:

* [Biến](https://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban/bien-trong-c-variables-in-c-105), [kiểu dữ liệu](https://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban/kieu-ky-tu-trong-c-character-34), [toán tử](https://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban/toan-tu-so-hoc-toan-tu-tang-giam-toan-tu-gan-so-hoc-trong-c-operators-1145)trong C++
* [Câu điều kiện](https://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban/cau-dieu-kien-if-va-toan-tu-dieu-kien-if-statements-and-conditional-operator-1344), [vòng lặp](https://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban/vong-lap-for-trong-c-for-statements-1372), [hàm](https://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban/co-ban-ve-ham-va-gia-tri-tra-ve-basic-of-functions-and-return-values-1339)trong C++
* [Mảng trong C++](https://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban/mang-1-chieu-trong-c-arrays-1377)
* [Các kiến thức cần thiết để theo dõi khóa học](https://www.howkteam.vn/course/cau-truc-du-lieu-va-giai-thuat/cac-kien-thuc-can-thiet-de-theo-doi-khoa-hoc-4311)

Trong bài học ngày hôm nay, chúng ta sẽ cùng nhau tìm hiểu về:

* Phương pháp kiểm tra số nguyên tố
* Sàng Eratosthenes

**Phương pháp kiểm tra số nguyên tố**

**Khái niệm số nguyên tố**

Số nguyên tố là số tự nhiên lớn hơn 1 và không phải là tích của hai số tự nhiên nhỏ hơn. Nói cách khác, số nguyên tố là những số chỉ có 2 ước là 1 và chính nó.

Các nguyên số lớn hơn 1 và không phải là số nguyên tố thì là hợp số. Riêng 0 và 1 không phải là số nguyên tố cũng không phải là hợp số.

Ví dụ: 2, 3, 5, 11 là các số nguyên tố nhưng 6 không phải là số nguyên tố do 6 có 2 ước là 2 và 3

**Cách kiểm tra số nguyên tố**

**Thuật toán cơ bản**

Từ định nghĩa của số nguyên tố, ta có thể nghĩ đến việc duyệt tất cả các số từ 2 đến *n*. Nếu *n* chia hết cho bất cứ số nào nghĩa là *n* không phải số nguyên tố.

#include<bits/stdc++.h>

**using** **namespace** **std**;

**int** n;

**bool** isPrime(**int** num){

**for**(**int** i = 2 ; i < num ; ++i)

**if**(num % i == 0) **return** 0;

**return** 1;

}

**int** main(){

freopen("CTDL.inp","r",stdin);

freopen("CTDL.out","w",stdout);

**cin** >> n;

**if**(isPrime(n)) **cout** << n << " la so nguyen to" << endl;

**else** **cout** << n << " la hop so";

**return** 0;

}

Ta có thể thấy ngay, độ phức tạp của thuật toán trên là O(n). Vậy thì liệu có thuật toán nào tốt hơn không?

**Nhận xét**

Ta có một nhận xét sau về số nguyên tố: Nếu như *n* không phải là số nguyên tố thì tồn tại một ước nguyên tố của *n* nhỏ hơn hoặc bằng \sqrt{n}.

Thật vậy, giả sử *n* không phải là số nguyên tố mà các ước của nó đều lớn hơn \sqrt{n} thì tích của hai ước bất kì sẽ lớn hơn *n* (Vô lí).

**Thuật toán cải tiến**

Từ nhận xét trên, ta thấy chỉ cần duyệt các ước đến *√n* để kiểm tra *n* có phải số nguyên tố hay không.

#include<bits/stdc++.h>

**using** **namespace** **std**;

**int** n;

**bool** isPrime(**int** num){

**for**(**int** i = 2 ; i \* i <= num ; ++i)

**if**(num % i == 0) **return** 0;

**return** 1;

}

**int** main(){

freopen("CTDL.inp","r",stdin);

freopen("CTDL.out","w",stdout);

**cin** >> n;

**if**(isPrime(n)) **cout** << n << " la so nguyen to" << endl;

**else** **cout** << n << " la hop so";

**return** 0;

}

Khi này, độ phức tạp của thuật toán sẽ giảm xuống còn O(\sqrt{n}).

Trong đoạn code trên, có một chi tiết mà mình muốn các bạn lưu ý về vòng for. Tại sao điều kiện vòng lặp của mình là *i\times i\leq n* thay vì i\leq \sqrt{n}?

Lí do là vì nếu như viết i\leq \sqrt{n} thì tại mỗi lần lặp lại của vòng for, máy tính sẽ phải tính lại giá trị của \sqrt{n} khiến cho thuật toán chạy chậm hơn không cần thiết.

Đây là một kinh nghiệm cho các bạn khi viết code. Ví dụ thay vì viết i\leq n / j các bạn có thể viết i\times j\leq n.

**Sàng Eratosthenes**

Bây giờ, chúng ta có một yêu cầu như sau: Cho một số nguyên dương n (n\leq 10^{6}). Hãy đánh dấu tất cả các số nguyên tố trong phạm vi từ 2 đến n.

**Thuật toán cơ bản**

Với những kiến thức về cách kiểm tra số nguyên tố ở trên bên, chúng ta có thể nghĩ đến việc duyệt tất cả các số từ 2 đến n và dùng hàm kiểm tra số nguyên tố.

#include<bits/stdc++.h>

**using** **namespace** **std**;

**const** **int** MaxN = 1 + 1e6;

**int** n;

**bool** mark[MaxN];

**bool** isPrime(**int** num){

**for**(**int** i = 2 ; i \* i <= num ; ++i)

**if**(num % i == 0) **return** 0;

**return** 1;

}

**int** main(){

freopen("CTDL.inp","r",stdin);

freopen("CTDL.out","w",stdout);

**cin** >> n;

**for**(**int** i = 2 ; i <= n ; ++i)

**if**(!isPrime(i)) mark[i] = 1;

// Kết thúc vòng lặp, nếu mark[x] == 0 thì x là số nguyên tố và ngược lại

**return** 0;

}

Tuy nhiên, độ phức tạp của chương trình trên là O(n\times \sqrt{n}). Liệu có thuật toán nào cho thời gian tốt hơn không?

**Phương pháp**

Trên thực tế, đã có một phương pháp lọc các số nguyên tố được đưa ra bởi nhà toán học cổ đại người Hy Lạp Eratosthene. Vậy chính xác ông đã làm thế nào?

Ông lấy lá cọ, ghi tất cả các số từ 2 đến 100. Ông xét lần lượt từng số. Đầu tiên, gặp số 2 chưa bị chọc thủng, ông sẽ chọc thủng tất cả các số khác 2 trên lá cọ mà chia hết cho 2. Tiếp theo, gặp số 3 chưa bị chọc thủng, ông sẽ chọc thủng tất cả các số khác 3 trên lá cọ mà chia hết cho 3. Lúc này, số 4 đã bị chọc thủng nên ông bỏ qua và tiếp tục đến 5. Sau khi kết thúc, các số còn lại đều là số nguyên tố và lá cọ khi này giống như một cái sàng nên người ta gọi đây là “Sàng Eratosthene”.

Tại sao cách làm trên lại có thể loại bỏ tất cả các số nguyên tố? Ta thấy, qua mỗi bước, các số chia hết cho một số nguyên tố nào đó sẽ bị loại bỏ đi như ở bước 1 là các số chia hết cho 2, ở bước 2 là các số chia hết cho 3 nên đến cuối cùng, sẽ chỉ còn sót lại các số chia hết cho chính nó (chính là các số nguyên tố).

**Thuật toán cải tiến**

Từ những thông tin về sàng ở trên, ta có thể đưa ra chương trình như sau

#include<bits/stdc++.h>

using namespace std;

const int MaxN = 1 + 1e6;

int n;

bool mark[MaxN]={0};

int main(){

    freopen("CTDL.inp","r",stdin);

freopen("CTDL.out","w",stdout);

cin >> n;

    for(int i = 2 ; i \* i <= n ; ++i)

        if(!mark[i])

            for (int j = 2 ; i \* j <= n ; ++j)

                if (mark[i \* j] == 0) mark[i \* j] = 1;

    // Kết thúc vòng lặp, nếu mark[x] == 0 thì x là số nguyên tố và ngược lại

    for(int i = 2 ; i <= n ; ++i)

    if (mark[i] == 0)

        cout << i << " ";

    return 0;

}

Vậy thì độ phức tạp của thuật toán trên sẽ là bao nhiêu?

Ta thấy

* Khi *i=2* vòng lặp j lặp lại \frac{n}{2} lần
* Khi i=3 vòng lặp j lặp lại \frac{n}{2} lần
* Khi i=5 vòng lặp  lặp lại \frac{n}{5} lần
* …

Như vậy độ phức tạp tổng là O(\frac{n}{2} + \frac{n}{3} + \frac{n}{5} +...) và người ta đã chứng minh được \frac{n}{2} + \frac{n}{3} + \frac{n}{5} +... =n\times log(logn) . Do vậy, độ phức tạp của thuật toán là O(n\times log(logn))..

**Kết luận**

* Qua bài này chúng ta đã nắm được về **Sàng Eratosthenes**
* Bài sau chúng ta sẽ tìm hiểu về **Hàm mũ nhanh**
* Cảm ơn các bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên “**Luyện tập – Thử thách – Không ngại khó**”.

**Thảo luận**

Nếu bạn có bất kỳ khó khăn hay thắc mắc gì về khóa học, đừng ngần ngại đặt câu hỏi trong phần BÌNH LUẬN bên dưới hoặc trong mục [HỎI & ĐÁP](https://howkteam.vn/question) trên thư viện [**Howkteam.com**](https://howkteam.vn/) để nhận được sự hỗ trợ từ cộng đồng.