Bài 23. Phân tích thừa số nguyên tố trong C/C++

Bởi **Nguyễn Văn Hiếu** -

Bài số 21 trong 69 bài của khóa học Học C Không Khó

Bài toán phân tích thừa số nguyên tố, hay nói đầy đủ hơn là phân tích số tự nhiên N thành tích các thừa số nguyên tố là một bài tập lập trình cơ bản thường được sử dụng trong các bài thi nhập môn lập trình. Trong bài chia sẻ này, Lập trình không khó sẽ cùng các bạn đi tìm hiểu và giải quyết bài toán phân tích thừa số nguyên tố này nhé.

1. Đề bài phân tích thừa số nguyên tố

Nếu bạn để ý tên bài toán, "phân tích thừa số nguyên tố" đã mang sẵn ý nghĩa của bài toán. "thừa số" có ở trong câu: muốn tìm một thừa số chưa biết, ta lấy tích chia cho thừa số đã biết ^^; Chẳng hạn, 5 và 4 là các thừa số trong phép nhân 5×4 = 20. "nguyên tố" chỉ rằng các thừa số sẽ luôn là số nguyên tố, bạn thử mà xem!

Như vậy, để dễ hình dung, chúng ta sẽ có những ví dụ sau:

- $8 = 2^3$
- $100 = 2^2 * 5$
- $999 = 3^3 * 37$

Vậy mục tiêu của các bạn là: Nhập vào một số nguyên dương N, hãy phân tích số N thành tích các thừa số nguyên tố(chính là phần phía sau dấu bằng).

2. Ý tưởng giải bài toán phân tích thừa số nguyên tố

Rất đơn giản, giả sử bạn cần phân tích số N thành tích các thừa số nguyên tố. Bạn chỉ cần thực hiện chia số N cho các số nguyên tố trong đoạn [2; N]. Với mỗi số nguyên tố đó, đếm số lần mà số N chia hết. Tất nhiên, sau mỗi lần chia cho số i, số N của chúng ta sẽ giảm đi i lần.

Một ví dụ sinh động hơn, xét N = 300

300	2
150	2
75	5
15	5
3	3
1	

Khi đó, 300 = 2^2 * 5^2 * 3. Nhưng không phải khi N = 1 chúng ta sẽ dừng đâu nhé. Xét một ví dụ khác xem sao:

Giả sử N = 999, khi N chỉ còn 37, mà 37 là số nguyên tố, nên ta dừng quá trình chia.

999	3
333	3
111	3
37	37
1	

Khi đó, 999 = 3^3 * 37.

Nói một cách tổng quát hóa hơn, bạn sẽ dừng quá trình chia khi số chia lớn hơn N. Nói cách khác, chừng nào N chưa bằng 1, chúng ta tiếp tục quá trình chia.

3. Code phân tích thừa số nguyên tố

Lời giải triển khai với ngôn ngữ C:


```
0
1  #include <stdio.h>
2
3  int main(){
4      int n;
5      printf("\nNhap n = ");
6      scanf("%d", &n);
7      int dem;
8
```

```
    for(int i = 2; i <= n; i++){
        dem = 0;
        while(n % i == 0){
            ++dem;
            n /= i;
        }
        if(dem){
            if(dem > 1) printf("%d^%d", i, dem);
            else printf("%d", i);
            if(n > i){
                printf(" * ");
            }
        }
    }

}

```

Cũng với ý tưởng này, nhưng code bằng C++ sẽ như sau:

```

#include <iostream>
using namespace std;


int main(){
    int n;
    cout << "\nNhap n = ";
    cin >> n;
    int dem;

    for(int i = 2; i <= n; i++){
        dem = 0;
        while(n % i == 0){
            ++dem;
            n /= i;
        }
        if(dem){
            cout << i;
            if(dem > 1) cout << "^" << dem;
            if(n > i){
                cout << " * ";
            }
        }
```

```
    }

}

```

Cả 2 code trên khi chạy sẽ có output như nhau, đây là một kết quả chạy thử:

```

Nhap n = 126
2 * 3^2 * 7

```

Nếu bạn biết về cấu trúc ***từ điển***. Bạn có thể sử dụng chúng để đếm và lưu giữ lại để phục vụ khi cần về sau. Với cách này, bạn có thể sử dụng chỉ số mảng để đếm: Chẳng hạn như số 126, mảng đếm là mảng a, khi đó: a[2] = 1, a[3] = 2, a[7] = 1. Tuy nhiên, cách này có hạn chế là với số N lớn, sẽ tốn rất nhiều bộ nhớ.

```

#include <iostream>
using namespace std;

const int MAX = 10000;
int dem[MAX];
int main(){
    int N;
    cout << "\nNhap n = ";
    cin >> N;
    int n = N; // Tao ban sao cua N
    if(n > MAX){
        printf("Ban nhap so lon hon %d", MAX);
        return 0;
    }
    for(int i = 0; i <= n; i++) dem[i] = 0;
    for(int i = 2; i <= n; i++){
        while(n % i == 0){
            ++dem[i];
            n /= i;
        }
    }
    for(int i = 0; i <= N; i++){
        if(dem[i]){
            cout << i << " ^ " << dem[i] << "\n";
        }
```

```
    }
}

```

Chạy thử với N = 999 xem sao:

```

Nhap n = 999
3 ^ 3
37 ^ 1

```

Như vậy, 999 = 3^3 * 37.

Một cách tối ưu hơn và thuận tiện hơn là sử dụng cấu trúc *map* trong C++. Chúng ta có thể code như sau:

```

#include <iostream>
using namespace std;
#include <map>

int main(){
    int N;
    cout << "\nNhap n = ";
    cin >> N;

    map<int, int> m;
    for(int i = 2; i <= N; i++){
        while(N % i == 0){
            m[i]++;
            N /= i;
        }
    }

    for(auto it : m){
        cout << it.first << " " << it.second << "\n";
    }
}

```

26

Lưu ý: Lời giải này sử dụng biến auto chỉ có trong C++ 11 trở lên. Nếu bạn muốn chạy được thì cần thêm flag: std=c++11.

```
0
1 g++ -std=c++11 your_file.cpp -o your_program
2
```

Hoặc nếu bạn dùng Dev C++, hãy vào Tools -> Compile Options và chọn như ảnh sau:

Cài đặt C++11 cho Dev C++

26

Như vậy, tôi đã hoàn thành bài chia sẻ phân tích thừa số nguyên tố này. Hi vọng rằng các bạn đã có được những kiến thức thật bổ ích. Mọi thắc mắc, hãy để lại tại mục bình luận, tôi sẽ giúp bạn những gì có thể.

> Tham gia forum Lập Trình Không Khó để không bỏ lỡ những bài viết mới nhất của admin nhé.

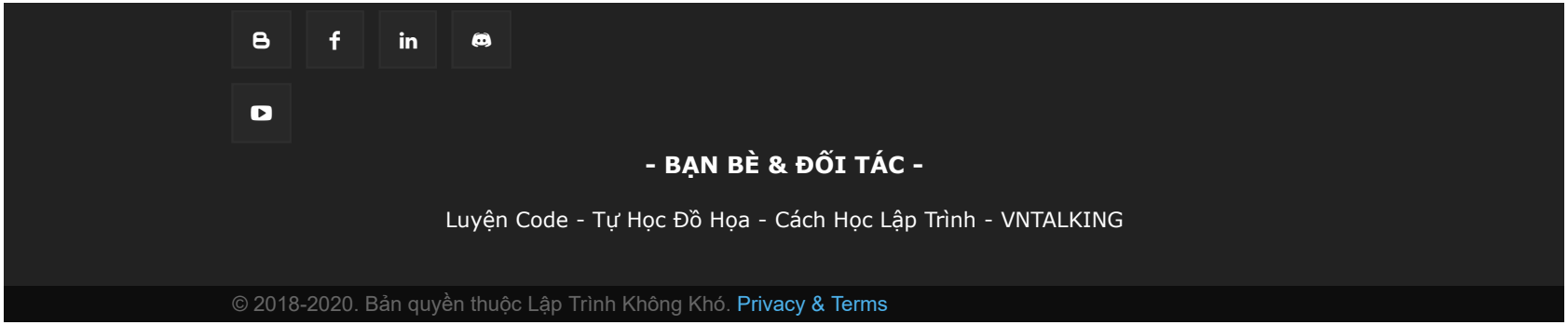
Các bài viết trong khóa học

Bài trước: Bài 22. Lệnh switch case trong C

Bài sau: Bài 24. Tìm số đảo ngược trong C/C++

Nguyễn Văn Hiếu

Sáng lập cộng đồng Lập Trình Không Khó với mong muốn giúp đỡ các bạn trẻ trên con đường trở thành những lập trình viên tương lai. Tất cả những gì tôi viết ra đây chỉ đơn giản là sở thích ghi lại các kiến thức mà tôi tích lũy được.


- BẠN BÈ & ĐỐI TÁC -
Luyện Code - Tự Học Đồ Họa - Cách Học Lập Trình - VNTALKING
© 2018-2020. Bản quyền thuộc Lập Trình Không Khó. Privacy & Terms

26