**Thuật toán kiếm tra số nguyên tố**

Trong bài này mình sẽ trình bày thuật toán để kiểm tra một số có phải là số nguyên tố hay không, sau khi giới thiệu xong thuật toán mình sẽ sử dụng ngôn ngữ C++ để giải mẫu cho các bạn. Trước tiên chúng ta tìm hiểu định nghĩa **số nguyên tố là gì** đã nhé.



**Mục lục**

* [1. Số nguyên tố là gì?](https://freetuts.net/thuat-toan-kiem-tra-so-nguyen-to-771.html#goto-h2-0)
* [2. Thuật toán kiểm tra số nguyên tố](https://freetuts.net/thuat-toan-kiem-tra-so-nguyen-to-771.html#goto-h2-1)
  + [Cách 1: Lặp từng phần tử với bước nhảy 1](https://freetuts.net/thuat-toan-kiem-tra-so-nguyen-to-771.html#goto-h3-0)
  + [Cách 2: Lặp từng phần tử với bước nhảy 2](https://freetuts.net/thuat-toan-kiem-tra-so-nguyen-to-771.html#goto-h3-1)
* [3. Lời kết](https://freetuts.net/thuat-toan-kiem-tra-so-nguyen-to-771.html#goto-h2-2)

**1. Số nguyên tố là gì?**

Theo định nghĩa của Wikipedia thì số nguyên tố là số tự nhiên lớn hơn 1, chỉ có 2 ước là 1 và chính nó. Theo định nghĩa này thì các số 2, 3, 5, 7, 11, ... là các số nguyên tố, trong đó số 2 là số nguyên tố chẵn duy nhất. Cũng như tính chất của số nguyên dương, chúng ta chỉ tìm thấy số nguyên tố nhỏ nhất chứ không thể tìm thấy số nguyên tố lớn nhất.

**Ví dụ**: 7 là số nguyên tố vì trong khoảng từ 2 - 6 không tồn tại số nào mà 7 chia hết cả.

**2. Thuật toán kiểm tra số nguyên tố**

Dựa vào định nghĩa của số nguyên tố chúng ta sẽ có một số cách giải như sau (*các ví dụ được viết bằng ngôn ngữ C++*):

*Bài viết này được đăng tại [free tuts .net]*

**Cách 1: Lặp từng phần tử với bước nhảy 1**

Giả sử cần kiểm tra số n có phải là số nguyên tố hay không thì các bước thực hiện như sau:

* **Bước 1**: Nhập vào n
* **Bước 2**: Kiểm tra nếu n < 2 thì kết luận n không phải là số nguyên tố
* **Bước 3**: Lặp từ 2 tới (n-1), nếu trong khoảng này tồn tại số mà n chia hết thì kết luận n không phải là số nguyên tố, ngược lại n là số nguyên tố.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **bool** laSoNguyenTo1(**int** n)  {      // Neu n < 2 thi khong phai la SNT      if (n < 2){          return false;      }        for (**int** i = 2; i < (n - 1); i++){          if (n % i == 0){              return false;          }      }        return true;  } |

**Hàm main**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | void main()  {  **int** n;      cout << "Nhap so can kiem tra";      cin >> n;        if (laSoNguyenTo1(n)){          cout << "La so nguyen to";      }      else {          cout << "Khong phai so nguyen to";      }  } |

**Cách 2: Lặp từng phần tử với bước nhảy 2**

Theo định nghĩa thì số 2 là số nguyên tố chẵn duy nhất, vì vậy ta sẽ loại 2 ra khỏi vòng lặp và trong thân vòng lặp chỉ kiểm tra các số lẻ mà thôi, cách này sẽ tối ưu hơn cách 1 rất nhiều.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | **bool** laSoNguyenTo2(**int** n)  {      // Neu n < 2 thi khong phai la SNT      if (n < 2){          return false;      }        // Neu n = 2 la SNT      if (n == 2){          return true;      }        // Neu n la so chan thi ko phai la SNT      if (n % 2 == 0){          return false;      }        // Lap qua cac so le      for (**int** i = 3; i < (n - 1); i += 2){          if (n % i == 0){              return false;          }      }        return true;  } |

**Hàm main**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | void main()  {  **int** n;      cout << "Nhap so can kiem tra";      cin >> n;        if (laSoNguyenTo2(n)){          cout << "La so nguyen to";      }      else {          cout << "Khong phai so nguyen to";      }  } |

**3. Lời kết**

Vẫn còn rất nhiều cách khác nhưng chung quy lại vẫn phải bám vào định nghĩa số nguyên tố là gì. Ví dụ trong vòng lặp điểm dừng sẽ là (n/2) thay vì (n-1) vì theo lý thuyết thì một số không bao giờ chia hết cho số một nửa của nó. Ví dụ số 9 thì số một nửa của nó là số (9 : 2 = 4), như vậy ta chỉ cần kiểm tra các số từ 2,3,4 mà thôi, còn các số 5,6,7,8 chắc chẵn 9 sẽ không chia hết.

|  |  |
| --- | --- |
| 1  2  3  4  5 | for (**int** i = 3; i <= (n/2); i += 2){          if (n % i == 0){                  return false;          }  } |

**Thuật toán tìm ước chung lớn nhất trong C/C++**

Trong bài này chúng ta sẽ tìm hiểu **thuật toán tìm ước chung lớn nhất**trong C++, bằng cách sử dụng vòng lặp, thuật toán Euclid và thuật toán loại trừ.



**Đề bài:**Nhập vào 2 số nguyên A và B, viết chương trình tìm ứng chung lớn nhất của 2 số đó.

**Kết quả:**

Input : A = 20, B = 12

Output : 4

Có rất nhiều cách để giải quyết bài toán này, trong bài viết chúng ta sẽ đi qua 3 cách để triển khai bài toán. Từ đó, lựa chọn ra cách tối ưu nhất.

*Bài viết này được đăng tại [free tuts .net]*

**Mục lục**

* [1. Tìm UCLN sử dụng vòng lặp](https://freetuts.net/thuat-toan-tim-uoc-chung-lon-nhat-2659.html#goto-h2-0)
* [2. Tìm UCLN bằng phương pháp trừ](https://freetuts.net/thuat-toan-tim-uoc-chung-lon-nhat-2659.html#goto-h2-1)
* [3. Tìm UCLN sử dụng thuật toán Euclid](https://freetuts.net/thuat-toan-tim-uoc-chung-lon-nhat-2659.html#goto-h2-2)
  + [Thuật toán Euclid](https://freetuts.net/thuat-toan-tim-uoc-chung-lon-nhat-2659.html#goto-h3-0)
  + [Thuật toán Euclid mở rộng (Extended Euclidean algorithm)](https://freetuts.net/thuat-toan-tim-uoc-chung-lon-nhat-2659.html#goto-h3-1)
* [4. Tìm UCLN bằng hàm có sẵn trong C/C++](https://freetuts.net/thuat-toan-tim-uoc-chung-lon-nhat-2659.html#goto-h2-3)

**1. Tìm UCLN sử dụng vòng lặp**

Đây là cách đơn giản nhất để cài đặt thuật toán tìm UCLN. Đối với thuật toán này chúng ta sẽ đi lặp lần các giá trị từ min(A, B) về 0 và kiểm tra giá trị một.

Các bước triển khai thuật toán này sẽ như sau: Chúng ta sẽ sử dùng vòng lặp for đẻ giải quyết bài tòán này.

* Lặp từ min(A,B) về 0.
* Với mỗi vòng lặp kiểm tra A và B có chia hết cho i hay không? Nếu chia hết trả về i.

Trình bài bài toán với ngôn ngữ C/C++ sẽ như sau :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include <bits/stdc++.h>  using namespace std;    //Hàm tìm ước chung lớn nhất  **int** UCLN(**int** A, **int** B) {      for(**int** i = min(A, B); i > 0; --i) {          if (A % i == 0 && B % i ==0)              return i;      }      //Không chạy tới đây vì A, B luôn chia hết cho 1  }  **int** main() {  **int** A,B;      A = 20;      B = 12;      cout << UCLN(A, B);  } |
| 1 | Output: 4 |

Đây là cách đơn giản nhất để giải quyết bài toán này, nhưng đối với dữ liệu lớn việc xử lý bằng cách này không hoàn toàn tối ưu. Độ phức tạp của thuật toán này là O(min(A, B)).

**2. Tìm UCLN bằng phương pháp trừ**

Ý tưởng của thuật toán này là trừ hai số A và B cho nhau tới khi hai số này bằng nhau. Lúc này ta sẽ tìm được ƯCLN của 2 số. Các bước triển khai thuật toán sẽ như sau:

* Kiểm tra liệu rằng A hoặc B có bằng 0 hay không ? Nếu bằng 0 trả về ƯCLN là A+B. Dừng chương trình.
* Lặp cho tới khi A = B. Với mỗi vòng lặp thì biến biến max(A, B) = giá trị max(A, B) - giá trị min(A, B).

Trình bài bài toán với ngôn ngữ C/C++ sẽ như sau :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | #include <bits/stdc++.h>  using namespace std;    //Hàm tìm UCLN  **int** UCLN(**int** A, **int** B) {      //Nếu A hoặc B = 0 thì UCLN = A+ B      if (A == 0 || B == 0)          return A + B;       //Lặp cho tới khi A = B      while(A != B) {          //Lấy số lớn trừ số bé.          if (A > B) {              A -= B;          }else{              B -= A;          }      }       // Trả về UCLB     // Lúc này A = B nên return về A hay B đều giống nhau      return A;  }  **int** main() {  **int** A,B;      A = 20;      B = 12;      cout << UCLN(A, B);  } |
| 1 | Output: 4 |

Thuật toán sẽ thực hiện lần lượt các bước như sau. Giả sử **A = 100, B = 30**.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | 100 =  100  - 30  70 = 70 - 30  40 = 40 - 30  //Đến đây là B < A nên nó sẽ lấy B = B - A  30 = 30 - 10  20 = 20 - 10  A = B = 10  reutrn 10 |

Ngoài thực hiện trừ A và B, chúng ta có thể thay dấu trừ thành chia dư. Kết quả trả về tương tự.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **int** UCLN(**int** A, **int** B) {        while(A \* B != 0) {          if (A > B) {              A %= B;          }else{              B %= A;          }      }      return A + B;  } |

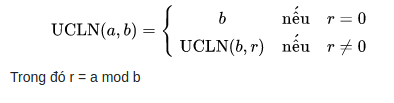
Đây là thuật toán tối ưu hơn so với thuật toán ban đầu, nhưng đây chưa phải là thuật toán tối ưu nhất.

**3. Tìm UCLN sử dụng thuật toán Euclid**

Giải thuật Euclid, hay Thuật toán Euclid, là một giải thuật giúp tính ước số chung lớn nhất (ƯSCLN) của hai số một cách hiệu quả. Trong phần này chúng ta sẽ đề cập giải thuật ở 2 khía cạnh cơ bản và mở rộng.

**Thuật toán Euclid**

[*Thuật toán Euclid*] là một giải thuật giúp chúng ta tìm ước chung lớn nhất của 2 số. Nó được triển khai dựa trên tính chất của UCLN đó là UCLN(A, B) = UCLN(B, A%B).



Giả sử chúng ta có 2 số **A = 20, B = 12**. Triển khai bằng thuật toán Euclid sẽ hoạt động như sau.

|  |  |
| --- | --- |
| 1  2  3  4 | UCLN(20, 12) = UCLN(12, 20%12) = UCLN(12, 8)  UCLN(12, 8) = UCLN(8, 12%8) = UCLN(8, 4)  UCLN(8, 4) = UCLN(4, 8%4) = UCLN(4, 0)  Vì B = 0 nên UCLN(4, 0) sẽ là 4 |

Ý tưởng triển khai thuật toán này sẽ quy nạp cho tới khi A % B = 0. Trình bài bài toán với ngôn ngữ C/C++ sẽ như sau :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | #include <bits/stdc++.h>  using namespace std;    **int** UCLN(**int** A, **int** B) {      if (B == 0)          return A;      return UCLN(B, A%B);  }  **int** main() {  **int** A,B;      A = 20;      B = 12;      cout << UCLN(A, B);  } |
| 1 | Output: 4 |

Đây là cách tối ưu nhất để giải các bài toán với dữ liệu lớn. Độ phực tạp của thuật toán này là O(logmax(A,B)).

**Thuật toán Euclid mở rộng (Extended Euclidean algorithm)**

**UCLN(A, B)**có một tính chất khá đặc biệt đó là luôn biểu diễn được ở dạng Ax + By = UCLN(A, B) trong đó *x, y* là hai số nguyên. Đây là một phần mở rộng của thuật toán Euclid cho phép chúng ta tìm ra hai số x và y thỏa mãn tính chất.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | #include <bits/stdc++.h>    using  namespace std;  **int** d, x, y;  void extendedEuclid(**int** A, **int** B) {      if (B == 0) {          d = A;          x = 1;          y = 0;      }      else {          extendedEuclid(B, A%B);  **int** temp = x;          x = y;          y = temp - (A/B)\*y;      }  }    **int** main() {      extendedEuclid(16, 10);      cout << "UCLN(16, 10) = " << d << endl;      cout << "x, y: " << x <<  ", " << y << endl;      return 0;  } |

Output: UCLN(20, 12) = 4

x, y: -1, 2

Độ phức tạp của thuật toán này là O(log(max(A,B)).

**4. Tìm UCLN bằng hàm có sẵn trong C/C++**

Ngoài cách tự viết các hàm tìm uớc chung lớn nhất, chúng ta còn có thể sử dụng hàm \_\_gcd có sẵn trong thư viện algorithm của C/C++.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #include<iostream>  #include<algorithm>  using namespace std;  **int** main() {  **int** A,B;      A = 20;      B = 12;      cout << \_\_gcd(A, B);  } |
| 1 | Output: 4 |

Đây là cách nhanh nhất để giải bài toán trong C/C++, ngoài tìm ước chung lớn nhất thư viện algorithm còn có nhiều hàm hỗ trợ khác cho giải các bài toán như max, min, sort,...Chúng ta có thể tham khảo thêm về thư viện [algorithm](https://freetuts.net/thuat-toan-tim-uoc-chung-lon-nhat-2659.html).

**Thuật toán tính lũy thừa nhanh trong C/C++**

Trong bài này chúng ta sẽ cùng nhau đi **tìm hiểu về thuật toán tính lũy thừa nhanh trong C/C++**. Thông thường đối với các bài toán tính lũy thừa chúng ta thường dùng hàm pow sẽ xử lý nhưng đối với các bài toán lớn sẽ mất nhiều thời gian hơn để xử lý.



Bài viết này sẽ đưa ra các cách để giải một bài toán tính lũy thừa một cách đơn giản và hiệu quả nhất.

**Đề bài:**Cho hai số nguyên a và b. Tính lũy thừa bậc b của a (a^b).

Input: a = 5, b = 10.

Output : 500

**Mục lục**

* [1. Sử dụng hàm pow](https://freetuts.net/thuat-toan-tinh-luy-thua-nhanh-trong-cc++-2681.html#goto-h2-0)
* [2. Sử dụng vòng lặp](https://freetuts.net/thuat-toan-tinh-luy-thua-nhanh-trong-cc++-2681.html#goto-h2-1)
* [3. Sử dụng công thức truy hồi](https://freetuts.net/thuat-toan-tinh-luy-thua-nhanh-trong-cc++-2681.html#goto-h2-2)

**1. Sử dụng hàm pow**

Đây là cách đơn giản nhất để tính lũy thừa bậc **b**của **a**. Trong thư viện cmath của C/C++ chúng ta có hàm pow, cho phép tính lũy thừa.

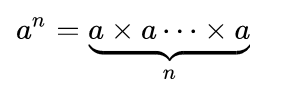
*Bài viết này được đăng tại [free tuts .net]*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | #include <cmath>  #include <iostream>  using namespace std;    **int** main() {  **long** **long** a, b, result;      a = 5;      b = 10000;      result = **pow**(a, b);      cout << result << endl;  } |

Đây là cách đơn giản và nhanh nhất để giải quyết bài toán.

**2. Sử dụng vòng lặp**

Trong trường hợp *b = n* là số nguyên dương, lũy thừa bậc **n**của **a**là tích của n thừa số bằng nhau.



Ngoài việc sử dụng hàm pow trong thư viện có sẵn ở C/C++, chúng ta còn có 1 cách khá là quen thuộc với những người mới làm quen với lập trình đó là sử dụng vòng for. Ý tưởng triển khai thuật toán này là lặp từ 1 tới *b*, với mỗi vòng lặp thực hiện nhân với *a*.

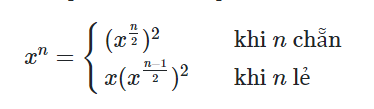
Triển khai bài toán với ngôn ngữ C/C++ như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #include <iostream>  using namespace std;  **long** **long** power(**long** **long** a, **long** **long** b) {  **long** **long** result = 1;      for(**int** i = 1; i <= b; i++) {          result \*= a;      }      return result;   }  **int** main() {  **long** **long** a, b, result;      a = 5;      b = 10;      result = power(a, b);      cout << result << endl;  } |

Thuật toán này hiệu quả với input là các số nhỏ. Giả sử các giá trị đầu vào vượt quá 10^8 thì chương trình chạy sẽ tốn rất nhiều thời gian và bộ nhớ, nên chúng ta sẽ có cách tối ưu hơn. Độ phức tạp của thuật toán này lên tới O(n).

**3. Sử dụng công thức truy hồi**

Đây là cách tối ưu nhất khi thực hiện tính lũy thừa mà không cần dùng tới hàm pow. Giả sử muốn tính x^n theo cách thông thường sẽ phải dùng tới **n**bước, tuy nhiên khi sử dụng công thức truy hồi thì sẻ giảm được 1 nửa. Ở đây chúng tá sẽ sử dụng công thức sau để triển khai bài toán.



Áp dụng công thức và triển khai thuật toán chúng ta sẽ có cách viết theo cách đệ quy như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | int sqr(int x) {      return x\*x;  }    int pow(int a, int b) {      if (b == 0) return 1;      else          if (b % 2 == 0)              return sqr(pow(a, b/2));          else              return a \* (sqr(pow(a, b/2)));  } |

Kết quả tương tự như các cách bên trên nhưng độ phức tạp nhỏ hơn rất nhiều chỉ O(log2(b)).

Trên đây là phần giới thiệu cũng như triển khai của các thuật toán tính lũy thừa trong C/C++. Đây cũng là những thuật toán hay được sử dụng cũng như rât hữu ích trong quá trình giải các bài toán tìm kiếm. Rất mong bài viết sẽ hữu ích cho bạn !

**Thuật toán kiểm tra số chẵn hay lẽ**

Bài này tương đối đơn giản nhưng mình cũng muốn đưa vào bởi khi bạn đọc bài này tức là bạn đang học kỹ thuật lập trình căn bản, nên mình nghĩ sẽ rất hữu ích cho bạn.



**Thuật toán kiểm tra số chẵn hay lẻ**

Để giải một bài toán trong lập trình thì ta phải xác dịnh được đề bài, công thức, cũng như là hướng giải. Ví dụ với bài này thì bạn phải phân tích dựa vào một định nghĩa như sau:

Một số được gọi là số chẵn khi nó chia hét cho 2, và được gọi là số lẻ khi không chia hết cho 2.

Ví dụ với số 4 thì nó là số chẵn vi khi chia nó cho 2 sẽ dư 0, còn số 5 là số lẻ vì khi chia cho 2 thì dư 1.

*Bài viết này được đăng tại [free tuts .net]*

Hầu như các ngôn ngữ lập trình cấp cao hiện nay đều sử dụng dấu % để chia lấy dư, và trong bài này mình sẽ đưa ra bài giải trong C++ nhé.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | #include <iostream.h>  using namespace std;  **int** main()  {  **int** n;        cout << "Nhap so can kiem tra";      cin >> n;        if (n % 2 == 0){          cout << n << " la so chan";      }      else {          cout << n << " la so le";      }        cout << endl;      cout << "Freetuts.net chuc ban hoc tot";  } |

Với các ngôn ngữ khác thì bạn chỉ cần thay đổi cú pháp là, còn về cách giải thì như nhau.

**Thuật toán kiểm tra năm nhuận**

Bài tiếp theo trong series thuật toán và giải thuật này chúng ta sẽ tìm hiểu cách kiểm tra năm nhuận bằng ngôn ngữ C++. Trước khi vào bài mình muốn nhấn mạng rằng nội dung của bài mình chỉ hướng dẫn cách kiểm tra trong lập trình, mình sẽ không giải thích ý nghĩa của năm nhuận cũng như chu kì của nó mà chỉ bám sát vào định nghĩa mà người xưa đã đưa ra mà thôi, sau này khi bạn tiếp cận bất kì một bài toán nào cũng phải dựa vào định nghĩa hoặc quy luật mà bài toán đó đang áp dụng để lập trình.



**Mục lục**

* [1. Năm nhuận là gì?](https://freetuts.net/thuat-toan-kiem-tra-nam-nhuan-773.html#goto-h2-0)
* [2. Thuật toán kiểm tra năm nhuận](https://freetuts.net/thuat-toan-kiem-tra-nam-nhuan-773.html#goto-h2-1)
  + [Hàm kiểm tra năm nhuận bằng C++](https://freetuts.net/thuat-toan-kiem-tra-nam-nhuan-773.html#goto-h3-0)
  + [Hàm main C++](https://freetuts.net/thuat-toan-kiem-tra-nam-nhuan-773.html#goto-h3-1)
* [3. Lời kết](https://freetuts.net/thuat-toan-kiem-tra-nam-nhuan-773.html#goto-h2-2)

**1. Năm nhuận là gì?**

**Năm nhuận** hay còn gọi là **năm nhuần**, là sự bổ sung thêm ngày và tháng để đảm bảo lịch phù hợp với các mùa của thời tiết. Theo lịch dương thì năm nhuân sẽ có tháng 2 là 29 ngày,còn theo lịch dương thì năm nhuận sẽ có thêm một tháng và ta hay gọi là tháng 13, mục đích họ tăng lên là để lịch phù hợp với các mùa.

Theo chu kì thì trái đất sẽ quay quanh mặt trời tổng cộng 365 ngày 5 giờ 48 phút 46 giây, như vậy mỗi năm sẽ dư ra gần 6h nên cứ 4 năm chúng ta lại dư mất một ngày, và nó sẽ rơi vào tháng 2 nên tháng này ta sẽ có 29 ngày để bám sát với chu kì quay.

Nói tóm lại mình có khái niệm năm nhuận đơn giản như sau: Năm nhuận là năm chia hết cho 4, trường hợp nếu vừa chia hết cho 4 vừa chia hết cho 100 thì nó phải chia hết cho 400 thì mới gọi là năm nhuận. Trong lập trình chúng ta sử dụng định nghĩa này để xây dựng ứng dụng kiểm tra năm nhuận. Khái niệm này vẫn có nhũng lúc sai vì thực tế 4h 48 phút 46 giây nhân lên 4 không thể bằng 1 ngày được.

*Bài viết này được đăng tại [free tuts .net]*

**2. Thuật toán kiểm tra năm nhuận**

Bám theo khái niệm năm nhuân là gì thì chúng ta có thuật toán kiểm tra năm nhuận như sau:

* Bước 1: Nhập số n cần kiểm tra
* Bước 2: Nếu n chia hết cho 100 và chi hết cho 400 thì nó là năm nhuận
* Bước 3: Nếu n chia hết cho 4 thì là năm nhuận
* Bước 4: Trả kết quả n không phải là năm nhuận

**Hàm kiểm tra năm nhuận bằng C++**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | void nam\_nhuan(**int** year)  {      if (year % 100 == 0)        {          if (year % 400 == 0){              cout << year << " la nam nhuan";          }          else{              cout << year << " khong phai la nam nhuan";          }      }      else if (year % 4 == 0) {          cout << year << " la nam nhuan";      }      else{          cout << year << " khong phai la nam nhuan";      }  } |

**Hàm main C++**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | void main()  {  **int** year;      cout << "Nhap nam can kiem tra: ";      cin >> year;        nam\_nhuan(year);  } |

**3. Lời kết**

Về khái niệm năm nhuận là gì thì bạn có thể mạng để đọc ở một số trang lớn khác, nhưng về thuật toán trong lập trình thì thường người ta hay sử dụng cách mà mình trình bày ở trên, nghĩa là năm nhuận là năm chia hết cho 4, trường hợp nso chia hết cho 100 thì phải chia hết cho 400.

**Thuật toán kiểm tra số hoàn hảo trong C++**

**Trong bài này mình sẽ giới thiệu đến các bạn một khái niệm mới đó chính là số hoàn hảo. Có lẽ đây là một khái niệm lạ đối với các bạn, vì nó không được phổ biến như [*số nguyên tố*], [*số chính phương*].**



Chúng ta sẽ tìm hiểu khái niệm về số hoàn hảo là gì, sau đó mình sẽ đưa ra ví dụ cho các bạn hiểu hơn về nó. Và cuối cùng mình sẽ viết thuật toán của nó kèm theo một ví dụ thực tế để các bạn có thể áp dụng được trong lập trình (*ngôn ngữ mình viết sẽ là ngôn ngữ C++*).

**Mục lục**

* [1. Số hoàn hảo là gì?](https://freetuts.net/thuat-toan-kiem-tra-so-hoan-hao-2919.html#goto-h2-0)
* [2. Thuật toán kiểm tra số hoàn hảo C++](https://freetuts.net/thuat-toan-kiem-tra-so-hoan-hao-2919.html#goto-h2-1)
  + [Thuật toán kiểm tra số hoàn hảo.](https://freetuts.net/thuat-toan-kiem-tra-so-hoan-hao-2919.html#goto-h3-0)
  + [Hàm main C++.](https://freetuts.net/thuat-toan-kiem-tra-so-hoan-hao-2919.html#goto-h3-1)

**1. Số hoàn hảo là gì?**

Vào thời trung cổ đại, các nhà toán học Hy Lạp đã đưa ra khái niệm số hoàn hảo.

Số hoàn hảo được hiểu đơn giản là số có tổng các ước số của nó bằng chính nó.

*Bài viết này được đăng tại [free tuts .net]*

Hãy cùng xem ví dụ sau đây:

* Số 6 là một số hoàn hảo vì tổng các ước số của nó bằng chính nó: 1 + 2 + 3 = 6.
* Số 12 không phải là số hoàn hảo vì tổng các ước số của nó lớn hơn chính nó: 1 + 2 + 3 + 4 + 6 = 16 (>12).

Nhờ vào các đặc điểm của số hoàn hảo, nhà toán học EuClid đã đưa ra một công thức cho các số "hoàn hảo" là các số chẵn:

Np = 2(p - 1) (2p - 1)

*Trong đó Np là số hoàn hảo, p là các số nguyên tố.*

Theo như công thức đưa ra, EuClid đã liệt kê 4 số hoàn hảo đầu tiên là: 6; 28; 496; 8128.

Đến nay vẫn chưa có nghiên cứu nào cho thấy số hoàn hảo có số lẻ và liệu có tồn tại vô số con số hoàn hảo hay không.

**2. Thuật toán kiểm tra số hoàn hảo C++**

**Giải thích thuật toán:** giả xử chúng ta có một số a là số cần kiểm tra.

1. Đầu tiên chúng ta tìm ra các ước số của a. Để làm được điều này chúng ta sẽ cần một vòng lặp For, lặp từ 1 đến a/2 với bước nhảy là 1 (i++).
2. Tiếp đến ta tạo một biến Sum, được dùng để tính tổng các ước số của a. Ta sẽ tạo một điều kiện trong vòng lặp For, nếu (a % i == 0) thì sum += i.
3. Và cuối cùng, kiểm tra xem nếu sum == a thì a chính là số hoàn hảo và ngược lại.

***\* Lưu ý:*** Các ước số của một số luôn bé hơn hoặc bằng 1/2 số đó. Vì vậy, trong ví dụ này các ước số của a luôn bé hơn hoặc bằng a/2.

**Thuật toán kiểm tra số hoàn hảo.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **bool** check(**int** a){  **int** sum = 0;//khai báo biến sum      for(**int** i=1;i<=a/2;i++){ //tạo vòng lặp for để tìm ước số của a          if(a%i==0)              sum+=i; //tổng các ước số của a      }      if(sum==a) return true; // trả về true      return false; // ngược lại trả về false  } |

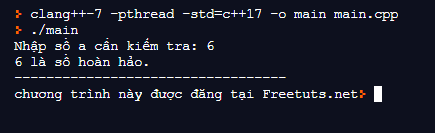
**Hàm main C++.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **int** main(){  **int** a;      cout<<"Nhập số a cần kiểm tra: ";      cin>>a;  // xử dụng hàm check() đã tạo để kiểm tra số a do người dùng nhập vào      if(check(a) ) // nếu a là số hoàn hảo thì hiển thị kết quả là số hoàn hảo          cout<<a<<" là số hoàn hảo.";      else          cout<<a<<" không phải là số hoàn hảo."; // và ngược lại hiển thị không phải là số hoàn hảo  } |

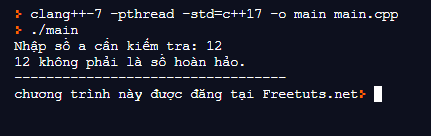
**Full code:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | #include<iostream>  using namespace std;  **bool** check(**int** a){  **int** sum = 0;//khai báo biến sum      for(**int** i=1;i<=a/2;i++){ //tạo vòng lặp for để tìm ước số của a          if(a%i==0)              sum+=i; //tổng các ước số của a      }      if(sum==a) return true; // trả về true      return false;  }  **int** main(){  **int** a;      cout<<"Nhập số a cần kiểm tra: ";      cin>>a;  // xử dụng hàm check() đã tạo để kiểm tra số a do người dùng nhập vào      if(check(a) ) // nếu a là số hoàn hảo thì hiển thị kết quả là số hoàn hảo          cout<<a<<" là số hoàn hảo.";      else          cout<<a<<" không phải là số hoàn hảo."; // và ngược lại hiển thị không phải là số hoàn hảo        cout<<"\n----------------------------------\n";      cout<<"chương trình này được đăng tại Freetuts.net";  } |

**Kết quả 1:**Kết quả trả về là số hoàn hảo.



**Kết quả 2:**Kết quả trả về không phải là số hoàn hảo.



Như vậy là chúng ta đã tìm hiểu xong thuật toán kiểm tra số hoàn hảo. Các bạn có thể sử dụng các cách khác nhau để thực hiện chương trình, hãy tạo cho mình một logic riêng. Điều đó rất có ích cho các bạn, chúc các bạn thực hiện thành công!!!

**Thuật toán kiểm tra số chính phương C++**

**Thuật toán kiểm tra số chính phương là một trong những thuật toán rất căn bản khi bắt đầu học lập trình. Nó giúp các bạn rèn luyện tính tư duy logic.**



Chúng ta sẽ cùng nhau tìm hiểu về khái niệm số chính phương. Sau đó sẽ viết thuật toán để kiểm tra số chính phương và cuối cùng sẽ là một ví dụ thực thế được viết bằng ngôn ngữ C++.

**Mục lục**

* [Số chính phương là gì?](https://freetuts.net/thuat-toan-kiem-tra-so-chinh-phuong-2921.html#goto-h2-0)
* [Thuật toán kiểm tra số chính phương.](https://freetuts.net/thuat-toan-kiem-tra-so-chinh-phuong-2921.html#goto-h2-1)
  + [Thuật toán kiểm tra số chính phương bằng vòng lặp.](https://freetuts.net/thuat-toan-kiem-tra-so-chinh-phuong-2921.html#goto-h3-0)
  + [Thuật toán kiểm tra số chính phương bằng hàm sqrt().](https://freetuts.net/thuat-toan-kiem-tra-so-chinh-phuong-2921.html#goto-h3-1)
* [Ví dụ thực tế.](https://freetuts.net/thuat-toan-kiem-tra-so-chinh-phuong-2921.html#goto-h2-2)

**Số chính phương là gì?**

Theo như chúng ta đã được học ở các cấp trung học, thì số chính phương được định nghĩa đơn giản như sau:

Số chính phương là số tự nhiên có căn bậc hai là một số tự nhiên, hay nói cách khác, số chính phương bằng bình phương của một số tự nhiên.

*Bài viết này được đăng tại [free tuts .net]*

**Ví dụ:**

* Số 4 là số chính phương vì 2^2 = 4. (hoặc sqrt(4) = 2).
* Tập các số chính phương như sau: CP = {4; 9; 16; 25; 26; ...}.

**Thuật toán kiểm tra số chính phương.**

Trong phần này, mình sẽ giới thiệu đến các bạn hai cách để viết thuật toán kiểm tra số chính phương:

1. Kiểm tra số chính phương bằng vòng lặp.
2. Kiểm tra số chính phương bằng hàm **sqrt()** trong thư viện **math.h**.

**Thuật toán kiểm tra số chính phương bằng vòng lặp.**

**Giải thích thuật toán:**

* Lặp i chạy từ 0 đến khi i\*i > n. Nếu i \* i = n thì n chính là số chính phương, sau đó kết thúc chương trình.
* Nếu i \* i > n thì n không phải là số chính phương.

**\* Lưu ý:** Trong vòng lặp cần có bước nhảy ++i, vì vậy hãy cho bước nhảy vào trong vòng lặp, nếu không vòng lặp sẽ không lặp đúng như mong muốn.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **bool** soCP(**int** n){  **int** i = 0;    while(i\*i <= n){          if(i\*i == n){              return true;          }          ++i;      }      return false;  } |

**Thuật toán kiểm tra số chính phương bằng hàm sqrt().**

**Giải thích thuật toán:**

* Cách thực hiện này đơn giản hơn rất nhiều so với cách sử dụng vòng lặp. Trong thư viện math.h có một hàm được dùng để tính căn bậc hai, đó chính là hàm sqrt().
* Chúng ta sẽ sử dụng hàm sqrt() để đặt điều kiện cho số n. Nếu sqrt(n) \* sqrt(n) = n, thì n chính là số chính phương và ngược lại.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **bool** soCP(**int** n){  **int** sqr = **sqrt**(n);      if(sqr\*sqr == n){          return true;      }      else return false;  } |

**Ví dụ thực tế.**

Trong phần ví dụ thực tế này, mình có một ví dụ như sau:

* Tạo mảng với số lượng phần tử do người dùng nhập.
* Viết hàm kiểm tra số chính phương và hiển thị tất cả các số chính phương có trong mảng.

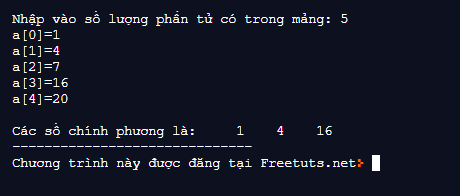
**Gợi ý:**

* Để in ra các số chính phương có trong mảng, chúng ta cần kiểm tra lần lượt các phần tử có trong mảng. Nếu số nào là số chính phương thì chúng ta sẽ in nó ra màn hình.
* Đầu tiên, chúng ta sẽ tạo hàm SoCP() để kiểm tra số chính phương.
* Tiếp đến, chúng ta sẽ tạo một mảng chứa các phần tử cần kiểm tra do người dùng nhập.
* Và cuối cùng, dùng vòng lặp for để lặp từng phần tử có trong mẳng. Cho vòng lặp lặp từ i = 0 đến i < n, với bước nhảy i++.

**Code Mẫu:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | #include <iostream>  #include <math.h>    using namespace std;    **bool** scp(**int** n){  **int** sqr = **sqrt**(n);      return (sqr\*sqr == n);  }    **int** main(){    **int** n;      do{          cout << "\nNhập vào số lượng phần tử có trong mảng: ";          cin >> n;      }while(n <= 0);    **int** a[n];        for(**int** i = 0;i < n;i++){          cout<<"a["<<i<<"]=";         cin >> a[i];      };        cout << "\nCác số chính phương là: \t";        for(**int** i = 0;i < n; i++){          if(scp(a[i])){              cout << a[i] << "    ";          }      }        cout<<"\n------------------------------\n";      cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong chương trình kiểm tra số chính phương có trong mảng. Cũng như kết thúc hướng dẫn thuật toán kiểm tra số chính phương. Chúc các bạn thực hiện thành công!!!

**Danh sách liên kết là gì? Sự khác nhau giữa DSLK và mảng**

**Trong bài này mình sẽ giới thiệu đến các bạn một khái niệm mới trong series giải thuật đó chính là danh sách liên kết.**



Chúng ta sẽ cùng nhau tìm hiểu danh sách liên kết là gì? sự khác nhau giữa danh sách liên kết với mảng. Một số loại danh sách liên kết thường gặp.

**Mục lục**

* [1. Danh sách liên kết là gì?](https://freetuts.net/danh-sach-lien-ket-la-gi-2974.html#goto-h2-0)
* [2. Sự khác biệt giữa danh sách liên kết và mảng](https://freetuts.net/danh-sach-lien-ket-la-gi-2974.html#goto-h2-1)
* [3. Một số loại danh danh sách liên kết thường gặp](https://freetuts.net/danh-sach-lien-ket-la-gi-2974.html#goto-h2-2)
  + [Danh sách liên kết đơn](https://freetuts.net/danh-sach-lien-ket-la-gi-2974.html#goto-h3-0)
  + [Danh sách liên kết đôi](https://freetuts.net/danh-sach-lien-ket-la-gi-2974.html#goto-h3-1)
  + [Danh sách liên kết vòng](https://freetuts.net/danh-sach-lien-ket-la-gi-2974.html#goto-h3-2)
* [4. Kết luận](https://freetuts.net/danh-sach-lien-ket-la-gi-2974.html#goto-h2-3)

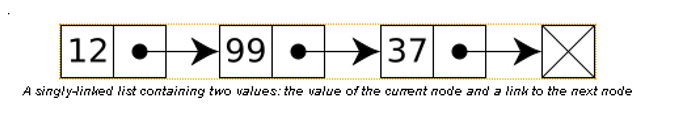
**1. Danh sách liên kết là gì?**

Danh sách liên kết có một số đặc điểm sau đây:

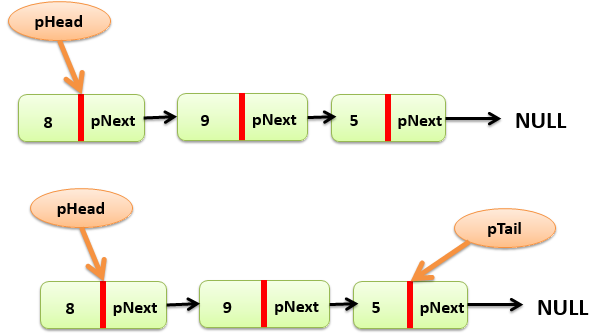
* Là một cấu trúc dữ liệu dùng để lưu trữ tập các phần tử rời rạc có thể co giãn một cách linh động.
* Kích thước của danh sách liên kết không cần định nghĩa trước, nó tự động thay đổi khi số phần tử trong danh sách thay đổi.
* Không giới giạn số lượng phần tử.
* Dễ dàng thực hiện thao tác: thêm, sửa, xóa.
* Truy xuất dữ liệu kiểu tuần tự.

Trong danh sách liên kết, mỗi phần tử còn được gọi là một node thường có ít nhất 2 thông số: Giá trị của node và mối liên kết tới node khác.

*Bài viết này được đăng tại [free tuts .net]*



Để quản lý danh sách liên kết ta thường quản lý node đầu (pHead), hoặc quản lý cả node đầu (pHead) và node cuối(pTail).



**2. Sự khác biệt giữa danh sách liên kết và mảng**

Danh sách liên kết và mảng đều được sử dụng với mục đích lưu trữ dữ liệu, tuy nhiên giữa hai kiểu lưu trữ này có một số ưu điểm và nhược điểm sau đây:

|  |  |
| --- | --- |
| **Mảng** | **Danh sách liên kết** |
| Phải biết trước số lượng phần tử, bị giới hạn bởi số lượng ban đầu cấp phát | Không cần biết trước, không bị giới hạn số lượng phần tử, tự động thay đổi kích thước |
| Truy suất ngẫu nhiên hoặc truy suất tuần tự | Chỉ truy suất tuần tự |
| Khó xóa phần tử trong mảng | Dễ dàng xóa phần tử trong danh sách |
| Khó thêm thêm phần tử | Dễ thêm phần tử |
| Dễ sắp xếp | Khó sắp xếp |
| Dễ tìm kiếm | Dễ tìm kiếm |

Như các bạn đã thấy, việc sử dụng danh sách liên kết rất linh hoạt so với mảng, chúng ta có thể sử dụng nó như một vùng lưu trữ vô hạn mà không cần phải khai báo giới hạn cho nó.

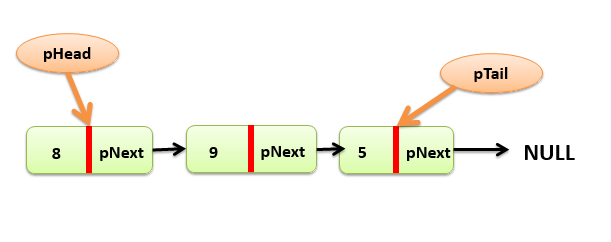
**3. Một số loại danh danh sách liên kết thường gặp**

Khi làm việc với danh sách liên kết, ta thường gặp các loại danh sách liên kết sau đây:

* Danh sách liên kết đơn
* Danh sách liên kết đôi
* Danh sách liên kết vòng

**Danh sách liên kết đơn**

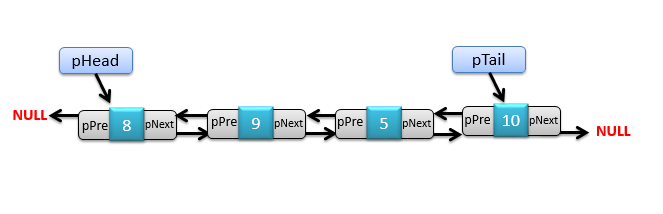
Danh sách liên kết đơn là một danh sách liên kết mà trong đó, mỗi phần tử liên kết với phần tử đứng sau nó trong danh sách.



Như hình trên, ở node thứ hai có liên kết với node thứ nhất thông qua pNext, tương tự như vậy node thứ ba liên kết với node thứ hai cũng thông qua pNext.

**Danh sách liên kết đôi**

Danh sách liên kết đôi là danh sách liên kết mà trong đó, mõi phần tử liên kết với phần tử đứng trước và đứng sau nó.

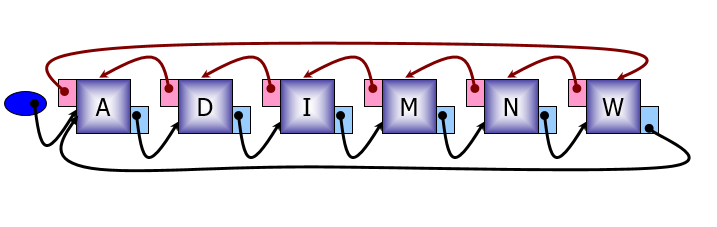


Tương tự như danh sách liên kết đơn, các phần tử đều liên kết với phần tử sau nó. Cộng thêm với đó là danh sách liên kết đôi cũng liên kết với phần tử trước nó nữa.

Các bạn có thể thấy mũi tên ở trong hình chỉ rõ sự liên kết giữa các node trong danh sách.

**Danh sách liên kết vòng**

Danh sách liên kết vòng cơ bản là danh sách liên kết đôi. Thay vào đó nó chỉ thêm một điều kiện là phần tử đầu (pHead) phải liên kết với phần tử cuối (pTail).



**4. Kết luận**

Trong bài viết này mình chỉ giới thiệu về khái niệm danh sách liên kết. Và so sánh danh sách liên kết với mảng để các bạn có thể nắm được các ưu điểm cũng như nhược điểm của nó. Mình cũng đã nói sơ qua về một số danh sách liên kết thường gặp, trong các bài tiếp theo chúng ta sẽ đi sâu hơn và chi tiết hơn về từng loại. Cách thức hoạt động, thêm, sữa, xóa các danh sách liên.