**PHẦN NÂNG CAO**

1. **Tìm kiếm**
2. ***Kiến thức***

* Tìm kiếm tuyến tính
* Tìm kiếm nhị phân

1. ***Bài tập***

* ***Bài 1***: Cho mảng a có n phần tử. Nhập vào phần tử x. Tìm trong mảng a có phần tử x hay không, nếu có đưa ra vị trí xuất hiện của x trong mảng a.
* ***Bài 2:*** Tìm số nguyên tố đầu tiên trong mảng 1 chiều các số nguyên.
* ***Bài 3***: Tìm số hoàn thiện cuối cùng trong mảng 1 chiều các số nguyên.

1. **Sắp xếp**
2. ***Kiến thức***

* Sắp xếp nổi bọt
* Sắp xếp chọn
* Sắp xếp chèn
* Sắp xếp trộn
* Sắp xếp nhanh

1. ***Bài tập***

* ***Bài 1***: Cho 2 mảng a và b. Trộn 2 mảng vào nhau và sắp xếp tăng dần.
* ***Bài 2***: Cho dãy số a có n phần tử. Hãy sắp xếp các số chẵn trong dãy theo thứ tự tăng dần, và các số lẻ theo thứ tự giảm dần.
* ***Bài 3***: Cho một dãy số nguyên a gồm n số nguyên. Hãy đưa ra dãy con không giảm dài nhất ở trong dãy a, nếu có nhiều dãy con dài nhất thì đưa ra dãy xuất hiện đầu tiên.

1. **Số học số nguyên. Số nguyên lớn. Kỹ thuật đếm cao cấp**
2. ***Kiến thức***
3. Biểu diễn số nguyên lớn bằng chuỗi kí tự

* Một cách hay để biểu diễn các số nguyên lớn trong C++ là sử dụng lớp chuỗi kí tự <string> trong C++. Các chữ số sẽ tương ứng với các kí tự trong chuỗi, và độ dài của các số khi đó sẽ phụ thuộc vào chương trình biên dịch của ngôn ngữ. Phương pháp biểu diễn này khá được ưa chuộng vì nó đơn giản, dễ hiểu, tuy nhiên trong một số trường hợp cụ thể, thời gian chạy của chương trình cài đặt số lớn bằng string sẽ khá lâu

1. Biểu diễn số nguyên lớn bằng mảng các kí tự

* Sử dụng một mảng để chứa các chữ số cũng là một phương án thường được sử dụng. Với phương pháp này, mỗi chữ số sẽ tương ứng với một kí tự trong mảng, đồng thời ta duy trì một biến đếm để kiểm soát số chữ số của số ban đầu. Với phương pháp này, tuy cài đặt có khó hơn, nhưng tốc độ chạy chương trình sẽ nhanh hơn, và chúng ta cũng không cần tới thao tác chuyển đổi giữa kí tự và chữ số như phương pháp đầu tiên.

1. ***Bài tập***

***Bài 1: Nhập xuất các số nguyên lớn***

* Cách biểu diễn bằng chuỗi kí tự: Trực tiếp nhập xuất

void input(bignum\_str &number)

{

cin >> number;

}

void output(bignum\_str number)

{

cout << number;

}

int main()

{

string a, b;

input(a);

input(b);

output(a);

output(b);

}

* Cách biểu diễn bằng mảng, nhập chuỗi bằng kiểu string, nhưng sau đó đưa lần lượt từng kí tự số vào vector và đổi thành chữ số luôn

// Nạp chồng toán tử trích luồng, dùng để nhập vào số lớn.

istream &operator >> (istream &cin, vi &number)

{

string s;

cin >> s;

number.clear();

for (int i = 0; i < s.size(); ++i)

number.push\_back(s[i] - '0');

return cin;

}

// Nạp chồng toán tử chèn luồng, dùng để in ra số lớn.

ostream &operator << (ostream &cout, const vi &a)

{

for (auto d: a)

cout << d;

return cout;

}

int main()

{

// Khi nhập xuất thì khai báo biến kiểu vi và dùng trực tiếp lệnh cin, cout.

vi a, b;

cin >> a >> b;

cout << a << endl << b;

return 0;

***Bài 2: So sánh hai số nguyên lớn***

* Phương pháp: Nguyên tắc so sánh hai số nguyên lớn khi sử dụng chuỗi kí tự như sau:
* Bước 1: Chuẩn hóa hai chuỗi bằng cách cân bằng độ dài của chúng. Nếu chuỗi nào ngắn hơn thì ta thêm kí tự 0 vào đầu chuỗi đó tới khi độ dài hai chuỗi bằng nhau.
* Bước 2: So sánh hai chuỗi sử dụng trực tiếp các toán tử >, <, >=, <=. ==, !=. Những toán tử này sẽ so sánh các chuỗi theo mã ASCII của từng cặp kí tự, và "vô tình" thứ tự ASCII lại chính là thứ tự đúng của các số.
* ***Sử dụng 2 hàm:***
* Hàm compare(a, b)  so sánh hai chuỗi s ố *a* và *b*. Nếu *a*<*b* thì trả về −1,  Nếu *a*>*b* thì trả về 1, còn *a*=*b* thì trả về 0.
* Hàm equal\_length(a, b) để cân bằng độ dài hai chuỗi số *a* và *b*.

void equal\_length(bignum\_str &a, bignum\_str &b)

{

while (a.size() < b.size())

a = '0' + a;

while (b.size() < a.size())

b = '0' + b;

}

int compare(bignum\_str a, bignum\_str b)

{

equal\_length(a, b);

if (a < b) // Có thể là a <= b.

return -1;

if (a > b) // Có thể là a >= b.

return 1;

return 0;

}

***Bài 3: Phép cộng hai số nguyên lớn***

* Cách giải:
* Bước 1: Cân bằng độ dài hai chuỗi bằng cách thêm kí tự 0 vào đầu chuỗi ngắn hơn.
* Bước 2: Cộng từng kí tự chữ số của hai chuỗi từ phải qua trái giống như quy tắc đặt tính ở tiểu học, phần nhớ được mang theo sang bên trái ở mỗi lần cộng. Sau mỗi lần cộng ở một hàng, ta thêm kí tự cuối của kết quả cộng hàng đó vào bên trái chuỗi kết quả.
* Bước 3: Nếu biến nhớ còn khác 0, viết thêm kí tự 1 vào bên trái chuỗi kết quả.
* Đối với chuỗi kí tự:

bignum\_str add(bignum\_str a, bignum\_str b)

{

equal\_length(a, b);

int carry = 0;

bignum\_str res;

for (int i = a.size() - 1; i >= 0; --i)

{

// Cộng hai chữ số cùng hàng và thêm biến nhớ từ hàng bên phải dồn lên.

int d = (a[i] - '0') + (b[i] - '0') + carry;

carry = d / 10; // Biến nhớ bằng kết quả hàng trước chia 10.

res = (char)(d % 10 + '0') + res; // Viết chữ số cuối của kết quả.

}

if (carry)

res = '1' + res;

return res;

}

* Đối với mảng chữ số:
* Đối với phương pháp mảng chữ số, ta làm hoàn toàn tương tự. Nhưng vì ban đầu các số được đưa vào mảng theo thứ tự ngược lại, nghĩa là chữ số hàng đơn vị của số ban đầu sẽ là phần tử đầu tiên của mảng, nên bước cân bằng độ dài hai mảng là không cần thiết nữa. Điều này giúp tiết kiệm thời gian chạy đáng kể.
* ***Cách giải:*** Sử dụng nạp chồng toán tử + đối với hai biến kiểu vi. Ngoài ra, ta cần thiết kế một hàm del\_zero() để xóa các chữ số 0 vô nghĩa ở đầu kết quả sau khi cộng xong. Hàm change() sẽ được tái sử dụng từ cài đặt của phép so sánh.

// Xóa các số 0 vô nghĩa ở đầu.

void del\_zero(vi &a)

{

reverse(a.begin(), a.end());

while (a.size() >= 2)

if (a.back() == 0)

a.pop\_back();

else

break;

reverse(a.begin(), a.end());

}

// Phép toán cộng.

vi operator + (vi a, vi b)

{

change(a, b);

int sz = a.size();

vi c;

int rem = 0;

for (int i = sz - 1; i >= 0; --i)

{

int x = a[i] + b[i] + rem;

rem = x / 10;

x %= 10;

c.push\_back(x);

}

c.push\_back(rem);

reverse(c.begin(), c.end());

del\_zero(c);

return c;

}

***Bài 4: Phép trừ 2 số nguyên lớn***

* Để cho đơn giản, chúng ta chỉ xét trường hợp lấy số lớn hơn trừ số nhỏ hơn. Nếu như *a*<*b* thì phải hoán đổi vị trí của chúng trước khi trừ, rồi đưa ra kết quả có thêm dấu - ở đằng trước.

Đối với phương pháp chuỗi, thuật toán như sau:

* Bước 1: Cân bằng độ dài hai chuỗi bằng cách thêm kí tự **'0'** vào đầu chuỗi ngắn hơn.
* Bước 2: Lầy từng cặp chữ số trừ đi nhau theo chiều từ phải qua trái giống như đặt tính, nếu kết quả bị âm thì cộng thêm 10 và nhớ −1 sang hàng phía trước.
* Bước 33: Nếu kết quả cuối cùng còn số 00 vô nghĩa ở bên trái thì xóa nó đi. Lưu ý chỉ được xóa đến khi kết quả còn 11 chữ số thì dừng lại.
* ***Trừ 2 số lớn kiểu chuỗi:***

bignum\_str diff(bignum\_str a, bignum\_str b)

{

equal\_length(a, b);

int d = 0, carry = 0;

bignum\_str res;

for (int i = a.size() - 1; i >= 0; --i)

{

d = (a[i] - '0') - (b[i] - '0') - carry;

// Tính toán biến nhớ cho hàng này.

if (d < 0)

{

d += 10;

carry = 1;

}

else

carry = 0;

// Thêm kí tự cuối cùng của kết quả trừ hàng vào đầu biến hiệu.

res = (char) (d + '0') + res;

}

// Xóa chữ số 0 vô nghĩa ở đầu kết quả. Nếu kết quả bằng 0 thì giữ lại một kí tự.

while (res.size() > 1 && res.front() == '0')

res.erase(res.begin());

return res;

}

* Trừ 2 số lớn kiểu mảng: Hàm change() được tái sử dụng từ đoạn chương trình so sánh hai số bên trên. Lưu ý, phải đảm bảo phép trừ được thực hiện bởi số lớn hơn trừ đi số bé hơn.

// Xóa các số 0 vô nghĩa ở đầu.

void del\_zero(vi &a)

{

reverse(a.begin(), a.end());

while (a.size() >= 2)

if (a.back() == 0)

a.pop\_back();

else

break;

reverse(a.begin(), a.end());

}

vi operator - (vi a, vi b)

{

change(a, b);

int sz = a.size();

vi c;

int rem = 0;

for (int i = sz - 1; i >= 0; --i)

{

int x = a[i] - b[i] - rem;

if (x < 0)

{

x += 10;

rem = 1;

}

else

rem = 0;

c.push\_back(x);

}

reverse(c.begin(), c.end());

del\_zero(c);

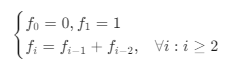
return c;

}

***Bài 5: Bài toán minh họa: Số Fibonaci thứ n***

**Đề bài**

Dãy số Fibonaci được định nghĩa theo công thức:



***Yêu cầu:*** Tìm số fibonaci thứ *n*?

***Input:***

* Một dòng duy nhất chứa số tự nhiên *n*.

***Ràng buộc:***

* 0≤ *n* ≤106

***Output:***

* In ra số Fibonacci thứ *n*.

***Sample Input:***

5

***Sample Output:***

8

**Ý tưởng**

Chắc chắn số fibonacci thứ *n* có thể sẽ vượt khỏi kiểu dữ liệu long long, mà đề bài lại không yêu cầu in ra kết quả mod cho giá trị nào cả. Do đó chúng ta cần sử dụng tới kiểu số lớn để tính toán ra số Fibonacci thứ *n*.

Chỉ cần sử dụng phép cộng số lớn để có thể đưa ra số Fibonacci thứ *n*. Ta dùng xâu hoặc dùng mảng lưu chữ số đều được.

***Độ phức tạp:*** *O*(*n*×*α*) với *α* là độ phức tạp phép cộng số lớn. Nó sẽ bằng với độ dài của các xâu cần cộng ở mỗi lượt.

**Code**

#include <bits/stdc++.h> //gọi hết các thư viện chuẩn

using namespace std;

typedef string bignum\_str;

bignum\_str fibonacci(int n)

{

if (n == 0)

return "0";

if (n == 1)

return "1";

bignum\_str f0 = "0", f1 = "1", fn;

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

{

fN = add(f0, f1);

f0 = f1;

f1 = fn;

}

return fn;

}

1. Tham lam
2. Kiến thức
3. Bài tập
4. Quy hoạch động
5. Quay lui
6. Vét cạn