**BÁO CÁO ĐỒ ÁN**

**---🙠**🕮**🙢---**

A picture containing logo

Description automatically generated

Thành viên trong nhóm: Nguyễn Anh Khoa – 20120118

Dũ Quốc Huy – 20120101

Môn học: Kĩ Thuật lập trình

Giáo viên hướng dẫn: Võ Hoài Việt

BIGINT

[0) Struct BigInt: 3](#_Toc73042603)

[1) Các phép toán cơ bản 3](#_Toc73042604)

[1.1) Phép cộng 3](#_Toc73042605)

[1.2) Phép trừ 4](#_Toc73042606)

[1.3) Phép nhân 5](#_Toc73042607)

[1.4) Phép chia và số dư 8](#_Toc73042608)

[a) Phép chia nhị phân: 8](#_Toc73042609)

[b) Phép dư nhị phân: 8](#_Toc73042610)

[c) Phép Chia hệ 10: 8](#_Toc73042611)

[d) Phép Mod hệ số 10: 9](#_Toc73042612)

[1.5) Phép chuyển từ hệ thập phân sang nhị phân: 11](#_Toc73042613)

[1.6) Phép chuyển từ hệ nhị phân sang thập phân: 12](#_Toc73042614)

[2) Các toán tử khác 12](#_Toc73042615)

[2.1) Phép dịch trái (<<) và phép dịch phải (>>): 12](#_Toc73042616)

[2.2) Phép and 14](#_Toc73042617)

[2.3) Phép or 14](#_Toc73042618)

[2.4) Phép xor 15](#_Toc73042619)

[2.5) Phép not 15](#_Toc73042620)

[3) Các hàm tiện ích 15](#_Toc73042621)

[3.1) Hàm trị tuyệt đối 15](#_Toc73042622)

[3.2) Hàm min/ max 16](#_Toc73042623)

[3.3) Hàm To\_Base 32, To\_Base 64, To\_Base 58 17](#_Toc73042624)

[a) Hàm to Base32: 17](#_Toc73042625)

[b)Base 58: 17](#_Toc73042626)

[c)Base 64: 18](#_Toc73042627)

[3.4) To\_string 19](#_Toc73042628)

[3.5) Isprime 20](#_Toc73042629)

[4) Các tiện ích khác 22](#_Toc73042630)

[4.1) Time 22](#_Toc73042631)

[4.2) Nhập xuất 23](#_Toc73042632)

[4.3) Các hàm thường sử dụng 24](#_Toc73042633)

# Struct BigInt:

Gồm hai phần: phần sign lưu dấu và phần chuỗi data để lưu dữ liệu số

# Các phép toán cơ bản

## Phép cộng

Phép Cộng: Đầu vào là 2 s1 và s2 có kiểu BigInt và biến hệ (10 tương ứng thập phân, 2 tương ứng nhị phân)

BigInt Add(BigInt s1, BigInt s2, string he);

Bước 1: Đưa 2 số về cùng độ dài maxlen với maxlen là độ dài của số lớn hơn cộng thêm 1 bằng hàm Add0 cho cả 2 số. Gán n=2 với hệ nhị phân và tương ứng n=10 với hệ thập phân. Khởi tạo biến carry (biến nhớ của phép cộng) . Một số BigInt s3 để lưu dữ liệu đầu ra

int n = he == "2" ? 2 : 10, temp1, temp2, carry = 0;

int maxlen = max(s1.data.length() + 1, s2.data.length() + 1);

s1.data = Add0(s1.data, maxlen);

s2.data = Add0(s2.data, maxlen);

BigInt s3;

Bước 2: Thực hiện phép lặp với số lần lặp là maxlen ( duyệt ngược từ kí tự cuối về kí tự đầu ⬄ cộng từ hàng đơn vị trở lên)

Bước 2.1: Dùng hàm GetDigit để lấy giá trị kiểu số nguyên của mỗi kí tự của cả 2 số s1 và s2

temp1 = GetDigit(s1, i);

temp2 = GetDigit(s2, i);

Bước 2.2: Cộng theo thứ tự của từng kí tự đồng thời cộng thêm biến carry ( biến nhớ), lưu vào trong biến kết quả (theo bài là s3.data)

s3.data = to\_string((temp1 + temp2 + carry) % n) + s3.data;

Bước 2.3: Tính toán lại biến carry (biến nhớ) để dùng cho phép cộng ở vị trí tiếp theo

carry = (temp1 + temp2 + carry) / n;

Bước 3: Xóa các số 0 dư thừa ở phép trước của kết quả phép cộng ( nếu có), trả kết quả ra là số có kiểu BigInt s3.

\*Lưu ý: hàm MenuAdd để xử lí những trường hợp có thể có ở phép cộng

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 1234567898765 + 56743829384748928746 | 56743830619316827511 |
| 2 | 2 10001011010111 + 1001111110110 | 11011011001101 |

## 1.2) Phép trừ

Phép Trừ: Đầu vào là 2 s1 và s2 có kiểu BigInt và biến hệ (10 tương ứng thập phân, 2 tương ứng nhị phân)

BigInt Minus(BigInt s1, BigInt s2, string he);

Bước 1: Xét độ dài; số trừ và số bị trừ

Bước 1.1: Đưa 2 số về cùng độ dài maxlen với maxlen là độ dài của số lớn hơn bằng hàm Add0 cho cả 2 số. Gán n=2 với hệ nhị phân và tương ứng n=10 với hệ thập phân. Một số BigInt s3 để lưu dữ liệu đầu ra. Một biến dau để xét dữ liệu đầu ra của phép trừ (âm và dương)

int n = he == "2" ? 2 : 10, temp1, temp2, carry = 0;

int maxlen = max(s1.data.length(), s2.data.length());

char dau = '\0';

s1.data = Add0(s1.data, maxlen);

s2.data = Add0(s2.data, maxlen);

BigInt s3;

Bước 1.2: So sánh dữ liệu cả 2 số, nếu số trừ lớn hơn số bị trừ thì dau=’-‘ tương ứng kết quả là số âm, đồng thời hoán đổi giá trị 2 số nhập va và ngược lại dau=’+’ nếu kết quả là số dương bằng cách sử dụng hàm Compare

if (s1.data.compare(s2.data) < 0) {

dau = '-';

string s = s1.data;

s1.data = s2.data;

s2.data = s;

}

Bước 2: Thực hiện phép lặp với số lần lặp là maxlen ( duyệt ngược từ kí tự cuối về kí tự đầu ⬄ trừ từ hàng đơn vị trở lên)

Bước 2.1: Dùng hàm GetDigit để lấy giá trị kiểu số nguyên của mỗi kí tự của cả 2 số s1 và s2

temp1 = GetDigit(s1, i);

temp2 = GetDigit(s2, i);

Bước 2.2: Nếu ở cùng thứ tự, giá trị của kí tự số trừ lớn hơn giá trị của kí tự số bị trừ thì ta tiến hành trừ có nhớ và ngược lại là phép trừ như bình thường. Đồng thời gán giá trị vào biến kết quả (s3.data)

if (temp1 < temp2)

{

s3.data = to\_string(temp1 + n - temp2) + s3.data;

s2.data[i - 1]++;

}

else s3.data = to\_string(temp1 - temp2) + s3.data;

Bước 3: Xét dấu kết quả và trả dữ liệu ra: Nếu là hệ thập phân và kết quả là một số âm, ta tiến hành gán dấu của dữ liệu đầu ra. Nếu là hệ nhị phân và kết quả là một số âm ta ghi ra số bù 2 của kết quả.

if (n == 10 && dau == '-') {

s3.data = Delete0(s3).data;

s3.sign = 1;

}

if (n == 2 && dau == '-')

s3.data = Bu2(s3).data;

return Delete0(s3);

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 9876562789182734 - 456789 | 9876562788725945 |
| 2 | 2 10101000110111 - 11101011111 | 10001011011000 |

\*Lưu ý: có hàm MenuMinus để xử lí các trường hợp có thể có của phép trừ

## 1.3) Phép nhân

1. Phép nhân với hệ nhị phân: Đầu vào là 2 số BigInt, trả ra kết quả là một số BigInt.

B1: Đưa hai số về cùng chiều dài bằng cách thêm 0 vào trước cả 2 số sao cho chiều dài mỗi số bằng tổng chiều dài cả hai số đầu vào,

int maxlen = s1.data.length() + s2.data.length();

s2.data = Add0(s2.data, maxlen);

s1.data = Add0(s1.data, maxlen);

Tạo một BigInt kết quả có chiều dài bằng với hai số sau khi thêm 0

BigInt kq;

kq.data = Add0(kq.data, maxlen);

B2: Thực hiện phép lặp với số lần bằng với số kí tự S1:

for (int i = 0; i < s1.data.length(); i++)

B2.1: Tiến hành kiểm tra bit cuối cùng của số thứ nhất (s1) nếu như số đó là 1, thì cộng thêm số thứ 2 vào kết quả

if (s1.data[maxlen - 1] == '1') { kq = Add(kq, s2, "2"); }

B2.2: Thực hiện phép dịch sang phải với số thứ nhất 1 bit , dịch trái với số thứ hai 1 bit, thực hiện lại phép lặp

s1 = dichphai(s1, 1);

s2 = dichtrai(s2, 1);

B3: Lấy số thứ 2 có được sau vòng lặp cộng vào kết quả. Ta thu được kết quả cuối cùng là kết quả của phép nhân

kq = Add(kq, s2, "2");

return kq;

1. Phép nhân với hệ thập phân: Nhận vào hai giá trị BigInt, thu được 1 giá trị Bigint. Chuỗi kết quả là phần data của kết quả(không kể sign)

B1: khởi tạo Zero và kết quả với giá trị 0. Kiểm tra dấu của s1 và s2, nếu khác nhau thì dấu của kết quả là – (1). Kiếm tra độ dài của s1 và s2 nếu một trong hai giá trị bằng không trả về zero. Khởi tạo biến đếm 1 và đếm 2 với giá trị 0, thêm số 0 vào trong kết quả với độ dài bài tổng chiều dài của 2 chuỗi s1 và s2

BigInt Zero, kq;

Zero.data = "0";

int len1 = s1.data.length();

int len2 = s2.data.length();

if (s1.sign != s2.sign) { kq.sign = 1; }

if (len1 == 0 || len2 == 0) { return Zero; }

int dem1 = 0;

int dem2 = 0;

B2: Thực hiện vòng lặp với số lần bằng chiều dài s1, có I là biến đếm và chạy từ cuối lên đầu chuỗi

for (int i = len1 - 1; i >= 0; i--)

B2.1: Tạo biến nhớ giá trị là 0, reset giá trị đếm 2 về 0

int nho = 0;

dem2 = 0;

B2.2 : Thực hiện vòng lặp con có j là biến đếm và cũng chạy từ cuối lên đầu chuỗi để tính giá trị của phần tử của chuỗi kết quả thứ (đếm 1+đếm 2)

for (int j = len2 - 1; j >= 0; j--)

1. nhân hai giá tri thứ i của s1 và thứ j của s2, cộng chuỗi kết quả thu được tại phần tử chuỗi kết quả thứ(đếm 1 + đếm 2) và cộng với giá trị biến nhớ(Chuỗi Kết quả ở vị trí đếm 1 + đếm 2 sẽ được tính bằng tổng nhiều lần, ví dụ như đếm 1=1, đếm 2=0 sẽ là vị trí 1, tương tự đếm 1=0, đếm 2 =1, cũng là chuỗi kết quả ở vị trí 1). Cả ba giá trị khi nhân với nhau đều phải trừ đi 48 vì số được lưu ở dạng char

kq.data[dem1 + dem2] = int(s1.data[i] - 48) \* int(s2.data[j] - 48) + int(kq.data[dem1 + dem2] - 48) + nho;

1. Tính giá trị nhớ mới bằng số vừa thu được chia cho 10.

nho = kq.data[dem1 + dem2] / 10;

1. Chuỗi kết quả ở vị trí đếm 1 + đếm 2 sẽ bằng kết quả ở vị trí đó vừa tính được ở trên, chia 10 và cộng vào 48. Sau đó tang giá trị biến đếm 2 lên 1 đơn vị

kq.data[dem1 + dem2] = (kq.data[dem1 + dem2] % 10) + 48;

dem2++;

B2.3: Nếu phần nhớ thu được > 0 thì sẽ cộng thêm vào trong vị trí đếm 1 + đếm 2 ( lúc này vi trí dịch lên 1 vì biến 2 vừa tăng). Tăng biến đếm 1 lên 1 đơn vị rồi kết thúc vòng lặp

if (nho > 0) { kq.data[dem1 + dem2] = kq.data[dem1 + dem2] + nho; }

dem1++;

B3: Đảo ngược chuỗi data của kết quả vừa thu được, rồi xóa phần số 0 ở đầu, trả về kết quả

string temp;

temp = Add0(temp, kq.data.length());

for (int i = 0; i < kq.data.length(); i++) {

temp[i] = kq.data[kq.data.length() - 1 - i];

}

kq.data = temp;

return Delete0(kq);

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 1234567 \* 123278 | 152194950626 |
| 2 | 2 10101000110111 \* 11101011111 | 1001101110010101101101001 |

## 1.4) Phép chia và số dư

### a) Phép chia nhị phân:

Đưa vào 2 số BigInt thu được kết quả là 1 số BigInt

B1: Tạo 1 biến thương có kiểu dữ liệu là BigInt và 1 biến temp cũng có kiểu dữ liệu tương tự

BigInt thuong;

BigInt temp;

B2: Bắt đầu một vòng lặp có số lần lặp bằng với chiều dài chuỗi thứ nhất(s1).

for (int i = 0; i < s1.data.length(); i++)

B2.1: Biến temp sẽ nhận thêm lần lượt các chữ số của số chia(s2), sau đó so sánh chiều dài temp và giá trị với s2, nếu chiều dài cái nào bé hơn thì thêm 0 sao cho ta được s2 và temp có chiều dài bằng nhau

B2.2: so sánh phần data của s2 và temp, nếu temp lớn hơn hay bằng data thêm vào data của thương số 1, rồi trừ temp cho s2, nếu không thì thêm vào số 0, sau đó khử 0 ở đầu của temp

B2.3: Khử 0 ở đầu s2

temp.data = temp.data + s1.data[i];

if (temp.data.length() > s2.data.length()) { s2.data = Add0(s2.data, temp.data.length()); }

else { temp.data = Add0(temp.data, s2.data.length()); }

B3: Sau khi đủ số lần lặp, phần thương chính là kết quả cần tìm.

return thuong;

### Phép dư nhị phân:

tương tự như phép chia, nhưng số dư trả về là phần temp còn lại

return temp;

### Phép Chia hệ 10:

Đầu vào là 2 số có kiểu dữ liệu BigInt

BigInt Div10(BigInt s1, BigInt s2);

Bước 1: Xét độ dài và khởi tạo mảng lưu bội của số bị chia

Bước 1.1: Đưa 2 số về cùng độ dài maxlen với maxlen là độ dài của số lớn hơn bằng hàm Add0 cho cả 2 số. Gán n=2 với hệ nhị phân và tương ứng n=10 với hệ thập phân. Khởi tạo biến carry (biến nhớ của phép cộng) . Một số BigInt s3 để lưu dữ liệu đầu ra

int maxlen = max(s1.data.length(), s2.data.length()), int k;

s1.data = Add0(s1.data, maxlen);

s2.data = Add0(s2.data, maxlen);

Bước 1.2: Tạo mảng lưu giá trị bội của số bị chia ( từ 0 -> 10) với kiểu dữ liệu BigInt

kb[0].data = '0';

for (int i = 1; i <= 10; i++)

{

kb[i] = Add(kb[i - 1], s2, "10");

}

Bước 2: Chạy vòng lặp với số lần lặp là maxlen. Dùng vòng lặp while để tìm thương của phép chia. Dùng các biến để lưu lại các giá trị dư ( dùng cho lần chia tiếp theo) và thương của phép chia.

for (int i = 0; i < maxlen; i++)

{

hold.data = hold.data + s1.data[i];

k = 1;

while (Compare(hold, kb[k]) >= 0)

k++;

s3.data = s3.data + to\_string(k - 1);

hold = Minus(hold, kb[k - 1], "10");

}

Bước 3: Xóa các số 0 thừa của thương (nếu có)(ở đây thương là BigInt s3) và trả về kết quả kiểu dữ liệu BigInt.

\*Lưu ý: có MenuDiv10 để xử lí trường hợp số âm, dương của dữ liệu đầu vào

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 19287635672891238475489387 / 98765 | 195288165573748174712 |
| 2 | 2 1010100 / 110 | 1110 |

### Phép Mod hệ số 10:

Đầu vào là 2 số có kiểu dữ liệu BigInt

BigInt Mod10(BigInt s1, BigInt s2);

Bước 1: Xét độ dài và khởi tạo mảng lưu bội của số bị chia

Bước 1.1: Đưa 2 số về cùng độ dài maxlen với maxlen là độ dài của số lớn hơn bằng hàm Add0 cho cả 2 số. Gán n=2 với hệ nhị phân và tương ứng n=10 với hệ thập phân. Khởi tạo biến carry (biến nhớ của phép cộng) . Một số BigInt s3 để lưu dữ liệu đầu ra

int maxlen = max(s1.data.length(), s2.data.length()), int k;

s1.data = Add0(s1.data, maxlen);

s2.data = Add0(s2.data, maxlen);

Bước 1.2: Tạo mảng lưu giá trị bội của số bị chia ( từ 0 -> 10) với kiểu dữ liệu BigInt

kb[0].data = '0';

for (int i = 1; i <= 10; i++)

{

kb[i] = Add(kb[i - 1], s2, "10");

}

Bước 2: Chạy vòng lặp với số lần lặp là maxlen. Dùng vòng lặp while để tìm thương của phép chia. Dùng các biến để lưu lại các giá trị dư ( dùng cho lần chia tiếp theo) và thương của phép chia.

for (int i = 0; i < maxlen; i++)

{

hold.data = hold.data + s1.data[i];

k = 1;

while (Compare(hold, kb[k]) >= 0)

k++;

s3.data = s3.data + to\_string(k - 1);

hold = Minus(hold, kb[k - 1], "10");

}

Bước 3: Xóa các số 0 thừa của dư (nếu có)(ở đây dư là hold) và trả về kết quả kiểu dữ liệu BigInt.

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 9876543210123456789 % 12345 | 2559 |
| 2 | 2 101011 % 11 | 1 |

## 1.5) Phép chuyển từ hệ thập phân sang nhị phân:

đầu vào là số muốn chuyển có kiểu dữ liệu BigInt và số nguyên x( ở đây x=2), dữ liệu trả về kiểu BigInt với data là số nhị phân ở dạng bù 2( bit đầu tiên là bit dấu, 0 là số dương, 1 là số âm ):

Sử dụng thuật toán đệ quy để thực hiện phép chia 2:

* Điều kiện dừng: khi dữ liệu đầu vào chỉ còn lại 0 thì ta ghi ra 0 và kết thúc vòng lặp đệ quy đó
* Phần đệ quy: thực hiện phép chia dữ liệu đầu cho 2 và cộng dồn số dư theo thứ tự số dư của phép đệ quy sau đừng trước số dư phép đệ quy trước (quy ước về chuyển hệ nhị phân)

if (s.data == "0") return "0";

else {

int du = 0, thuong = 0, i = 0;

BigInt s3;

s3.data = "";

for (i = 0; i < s.data.length(); i++) {

du = du \* 10 + GetDigit(s, i);

thuong = du / x;

du = du - thuong \* x;

s3.data = s3.data + to\_string(thuong);

}

s3 = Delete0(s3);

return ToBinary(s3, 2) + to\_string(du);

}

\*Lưu ý: nếu chuyển nhị phân số thập phân âm thì ta lấy số bù 2 của số nhị phân tìm được nếu chuyển trị tuyệt đối số đó.

Quy ước đầu ra: bit 0 ở đầu nếu là số dương và 1 nếu là số âm.

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 2 12345 | 011000000111001 |
| 2 | 10 2 -12345 | 100111111000111 |

## 1.6) Phép chuyển từ hệ nhị phân sang thập phân:

Yêu cầu dữ liệu đầu vào là số nhị phân lưu ở dạng bù 2 (bit đầu là bit dấu, 0 là số dương, 1 là số âm)

Dữ liệu đầu vào là số được lưu dưới kiểu dữ liệu BigInt với data là số nhị phân lưu ở dạng số bù 2, trả về giá trị có kiểu dữ liệu BigInt.

BigInt ToDecimal(BigInt s);

Bước 1: Khởi tạo biến phụ có dữ liệu là 2 (tương ứng với hệ nhị phân).

temp1.data = '2';

Bước 2: Sử dụng vòng lặp với số lần lặp là độ dài của dữ liệu đầu vào. Xét duyệt từng kí tự của dữ liệu đầu vào nếu là ‘1’ thì ta sử dụng phép mũ với 2^(độ dài xâu - 1- vị trí) và cộng dồn kết quả nhận được.

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

{

if (s.data[i] == '1') {

BigInt temp4;

temp4.data = to\_string(s.data.length() - 1 - i);

temp2 = Pow(temp1, temp4);

temp3 = Add(temp3, temp2, "10");

}

}

Bước 3: Trả về kết quả là tổng của các phép cộng ta thực hiện ở bước 2.

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 2 10 01101010111110111111 | 438207 |

# Các toán tử khác

## 2.1) Phép dịch trái (<<) và phép dịch phải (>>):

Sử dụng một hàm để dịch cho cả 2 hệ thập phân và nhị phân, cả phép dịch trái và phải. Dữ liệu đầu vào gồm số cần dịch, số lượng bit cần dịch, hệ của số cần dịch, phép toán dịch. Dữ liệu trả về có kiểu dữ liệu BigInt

BigInt MenuDich(BigInt s1, BigInt s2, string he, string pheptoan);

Trường hợp 1: Hệ thập phân

Bước 1: Chuyển sang hệ nhị phân bằng hàm ToBinary. Nếu là số âm ta phải chuyển sang số bù 2 bằng hàm Bu2

s3.data = ToBinary(s1, 2);

if (s1.sign == 1) s3 = Bu2(s3);

Bước 2: Chạy vòng lặp while để dịch bit, điều kiện dừng khi số lần dịch bằng 0

while (s2.data != "0")

Bước 2.2: Dịch phải (>>), ta xóa đi bit cuối cùng đồng thời giảm số lần dịch đi 1

if (pheptoan == ">>") {

s3.data.erase(s3.data.length() - 1 , 1);

s2 = Minus(s2, temp, "10");

}

Bước 2.2: Dịch trải (<<), ta thêm bit 0 vào cuối, đồng thời giảm số lần dịch đi 1

if (pheptoan == "<<") {

s3.data = s3.data + "0";

s2 = Minus(s2, temp, "10");

}

Bước 3: Chuyển kết quả của bước 2 sang hệ thập phân bằng hàm ToDecimal, và trả về kết quả vừa chuyển

return ToDecimal(s3);

Trường hợp 2: Hệ nhị phân:

Bước 1: Thực hiện giống như bước 2 trong trường hợp dịch bit hệ thập phân

Bước 2: Trả ra kết quả vừa tìm được ở bước 2 (không trả về hệ thập phân thì dữ liệu đầu vào dạng nhị phân)

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 5678 >> 2 | 1419 |
| 2 | 10 9876 << 3 | 79008 |
| 3 | 2 10101011 >> 4 | 1010 |
| 4 | 2 10110101 << 5 | 1011010100000 |

## 2.2) Phép and

B1: tìm chiều dài của chuỗi dữ liệu s1 và s2, sau đó thêm 0 vào số ngắn hơn cho 2 số có cùng chiều dài. Tạo 1 biến kết quả có chiều dài bằng với cả hai chuỗi sau khi thêm 0

int maxlen = (s1.data.length() > s2.data.length()) ? s1.data.length() : s2.data.length();

s1.data = Add0(s1.data, maxlen);

s2.data = Add0(s2.data, maxlen);

BigInt kq;

kq.data = Add0(kq.data, maxlen);

B2: Chạy vòng lặp I với số lần bằng độ dài chuỗi sau khi thêm 0, mỗi lần tại vị trí I nếu cả 2 nhận giá tri 1 thì giá trị tại vị trí I của kết quả cũng nhận giá trị 1

for (int i = 0; i < maxlen; i++) {

if ((s1.data[i] == '1') && (s2.data[i] == '1')) { kq.data[i] = '1'; }

}

B3: Sau khi hoàn thành các vòng lặp trả về Kết quả

## 2.3) Phép or

B1: tìm chiều dài của chuỗi dữ liệu s1 và s2, sau đó thêm 0 vào số ngắn hơn cho 2 số có cùng chiều dài. Tạo 1 biến kết quả

int maxlen = (s1.data.length() > s2.data.length()) ? s1.data.length() : s2.data.length();

s1.data = Add0(s1.data, maxlen);

s2.data = Add0(s2.data, maxlen);

BigInt kq;

B2: Chạy vòng lặp I với số lần bằng độ dài chuỗi sau khi thêm 0, mỗi lần tại vị trí I đều cộng thêm “1”, nếu cả 2 nhận giá tri 0 thì giá trị tại vị trí I của kết quả đổi thành giá trị “0”

for (int i = 0; i < maxlen; i++) {

kq.data = kq.data + "1";

if ((s1.data[i] == '0') && (s2.data[i] == '0')) { kq.data[i] = '0'; }

}

B3: Sau khi hoàn thành các vòng lặp trả về Kết quả

## 2.4) Phép xor

B1: tìm chiều dài của chuỗi dữ liệu s1 và s2, sau đó thêm 0 vào số ngắn hơn cho 2 số có cùng chiều dài. Tạo 1 biến kết quả

int maxlen = (s1.data.length() > s2.data.length()) ? s1.data.length() : s2.data.length();

s1.data = Add0(s1.data, maxlen);

s2.data = Add0(s2.data, maxlen);

BigInt kq;

B2: Chạy vòng lặp I với số lần bằng độ dài chuỗi sau khi thêm 0, mỗi lần tại vị trí I đều cộng thêm “1”, nếu cả 2 cùng nhận giá tri 0 hoặc cùng nhận 1 thì giá trị tại vị trí I của kết quả đổi thành giá trị “0”

for (int i = 0; i < maxlen; i++) {

kq.data = kq.data + "1";

if (s1.data[i] == s2.data[i]) { kq.data[i] = '0'; }

}

B3: Sau khi hoàn thành các vòng lặp trả về Kết quả

## 2.5) Phép not

Phép not với hệ nhị phân: Tạo BigInt kết quả có chiều dài bằng với chiều dài chuỗi đầu vào, duyệt qua từng bit, nếu bit 0 sẽ đổi thành 1, 1 sẽ đổi thành không

for (int i = 0; i < s1.data.length(); i++) {

s1.data[i] = '1' - s1.data[i] + 48;

}

return s1;

Với hệ thập phân là kết quả của phép trừ của -1 và số đầu vào

if (he == "10") { BigInt ammot; ammot.sign = 1; ammot.data = 1; return Minus(ammot, s1, "10"); }

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 2 101011 & 101010011 | 11 |
| 2 | 2 1010111 | 10111011 | 11111111 |
| 3 | 2 1001110 ^ 1010111 | 11001 |
| 4 | 2 100011 ~ | 11100 |

# Các hàm tiện ích

## 3.1) Hàm trị tuyệt đối

Đổi giá trị sign từ 1 về 0 với mọi số âm

if (s1.sign == true) { s1.sign = false; }

return s1;

## 3.2) Hàm min/ max

Tìm Max hoặc Min của 2 số: sử dụng một hàm chung để tìm min hoặc max cho cả 2 hệ nhị phân và hệ thập phân. (Sử dụng hàm phụ Compare)

void MinMax(BigInt s1, BigInt s2, string he, string pheptoan);

Trường hợp 1: Nếu là hệ thập phân:

Nếu phép toán là tìm max 2 số:

-Nếu một trong 2 số là số âm => ghi ra số kia

-Nếu 2 số cùng dương, sử dụng hàm Compare để so sánh 2 số là ghi ra số lớn hơn

-Nếu 2 số cùng âm, ta sử dụng hàm Compare cho trị tuyệt đối 2 số là ghi ra số có trị tuyệt đối nhỏ hơn( kèm dấu của nó)

-Nếu 2 số bằng nhau, ta ghi ra “2 so nhap vao bang nhau”

Nếu phép toán tìm min 2 số:

-Nếu một trong 2 số là số âm => ghi ra số âm

-Nếu 2 số cùng dương, sử dụng hàm Compare để so sánh 2 số là ghi ra số nhỏ hơn

-Nếu 2 số cùng âm, ta sử dụng hàm Compare cho trị tuyệt đối 2 số là ghi ra số có trị tuyệt đối lớn hơn( kèm dấu của nó)

-Nếu 2 số bằng nhau, ta ghi ra “2 so nhap vao bang nhau”

Trường hợp 2: Nếu là hệ nhị phân => làm tương tự như hệ nhị phân.

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 max 98765432 -123456 | 98765432 |
| 2 | 10 min 56789 98765 | 56789 |
| 3 | 2 max 1010111 10111 | 1010111 |
| 4 | 2 min 1001001 101010101010 | 1001001 |
| 5 | 10 max 123456789 123456789 | 2 so nhap vao bang nhau |

## 3.3) Hàm To\_Base 32, To\_Base 64, To\_Base 58

### a) Hàm to Base32:

Đưa vào 1 số BigInt dạng thập phân, đưa ra 1 số dưới dạng base 32.

B1: Tạo một số dữ liệu cơ bản như dạng bigint của 32, của kq và của biến trung gian

BigInt bahai;

BigInt kq;

BigInt c;

bahai.data = "32";

B2: Lặp cho đến khi s.data==0

while (s.data != "0")

B2.1: chia s cho 32 lấy dư rồi thêm 0 vào số dư cho đủ 2 chữ số

c.data = (Mod10(s, bahai)).data;

c.data = Add0(c.data, 2);

B2.2: So sánh số đó với 09, nếu nhỏ hơn thì xóa 0 của c và cộng trực tiếp vào kết quả. Nếu lớn hơn 09 thì đưa về kí tự bằng cách cộng thêm 55(A cách 10 55 đơn vị trong bảng mã). Gán s = s/10

if (c.data <= "09") { c = Delete0(c); kq.data = kq.data + c.data; }

else {

kq.data = kq.data + char(getnum(c.data, 2) + 55);

}

s = Div10(s, bahai);

B3: Thoát khỏi vòng lặp, đảo chuỗi kq và trả về

string temp;

temp = Add0(temp, kq.data.length());

for (int i = 0; i < kq.data.length(); i++) {

temp[i] = kq.data[kq.data.length() - 1 - i];

}

kq.data = temp;

return kq;

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 1234 To\_Base32 | 16I |

### b)Base 58:

Phép toán là **To\_Base58** .Dữ liệu đầu vào là số nguyên lớn ở dạng cơ số 10 được lưu dưới dạng BigInt, dữ liệu trả về dạng BigInt

BigInt ToBase58(BigInt s);

Bước 1: Thực hiện kết hợp 2 vòng lặp là while và for để tìm thương và dư khi chia cho 58, từ dư của mỗi phép chia ta chuyển sang kí tự thích hợp, thực hiện vòng lặp while cho đến khi thương còn lại là 0

while (s.data != "0")

{

int du = 0, thuong = 0;

BigInt s3;

s3.data = "";

for (int i = 0; i < s.data.length(); i++) {

du = du \* 10 + GetDigit(s, i);

thuong = du / 58;

du = du - thuong \* 58;

s3.data = s3.data + to\_string(thuong);

}

s3 = Delete0(s3);

s = s3;

st.data = base58[du] + st.data;

}

Bước 2: Trả về kết quả là BigInt st.

\*Lưu ý: nếu đầu vào chuyển dòng lệnh là số được lưu ở hệ cơ số 2, ta chỉ việc dùng hàm ToDecimal để chuyển hệ số 10 rồi thực hiện như trên.

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 1234 To\_Base58 | NH |
| 2 | 2 10011010010 To\_Base58 | NH |

## c)Base 64:

B1: Tạo một số dữ liệu cơ bản như dạng bigint của 64, của kq và của biến trung gian

BigInt saubon;

saubon.data = "64";

BigInt kq;

BigInt c;

B2: Lặp cho đến khi s.data==0

while (s.data != "0")

B2.1: chia s cho 64 lấy dư rồi thêm 0 vào số dư cho đủ 2 chữ số

c.data = (Mod10(s, saubon)).data;

c.data = Add0(c.data, 2);

B2.2: Đối chiếu số đó và gán cho phù hợp. Gán s = s/10

if (c.data <= "25") { kq.data = kq.data + char(getnum(c.data, 2) + 65); }

else {

if (c.data <= "51") { kq.data = kq.data + char(getnum(c.data, 2) + 71); }

else {

if (c.data <= "61") kq.data = kq.data + char(getnum(c.data, 2) - 4);

else {

if (c.data == "62") kq.data = kq.data + "+";

else kq.data = kq.data + "/";

}

}

}

s = Div10(s, saubon);

B3: Thoát khỏi vòng lặp, đảo chuỗi kq và trả về

string temp;

temp = Add0(temp, kq.data.length());

for (int i = 0; i < kq.data.length(); i++) {

temp[i] = kq.data[kq.data.length() - 1 - i];

}

kq.data = temp;

return kq;

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 1234 To\_Base64 | TS |

## 3.4) To\_string

B1: Tạo chuỗi kết quả. Bắt đầu vòng lặp đến khi n bằng 0

BigInt kq;

while (n != 0)

B2: lấy từng chữ số từ phải qua trái rồi thêm vào phần data. Gán n=n/10

int m = n % 10;

kq.data = kq.data + char(m + 48);

n = n / 10;

B3: Đảo chuỗi và xuất ra kết quả

string temp;

temp = Add0(temp, kq.data.length());

for (int i = 0; i < kq.data.length(); i++) {

temp[i] = kq.data[kq.data.length() - 1 - i];

}

kq.data = temp;

return Delete0(kq);

## 3.5) Isprime

1. Hàm random ra số BigInt, đưa vào độ dài tối đa của chuỗi số, đầu ra là 1 số BigInt có số chữ số bé hơn độ dài đưa vào

B1: Tạo 1 box có giá trị gồm 10 số :”0123456789”, tạo code kiểu BigInt lưu kết quả

BigInt code;

string box = "0123456789";

code.data = Add0(code.data, maxlen);

B2: Tìm độ dài ngẫu nhiên cho chuỗi số, sao cho độ dài ấy nhỏ hơn độ dài tối đa

int len = (rand() % (maxlen))+1;

B3: Chạy vòng lặp với số lần bằng số chữ số, cứ mỗi vòng sẽ chọn ra 1 số bất kì trong box kia làm giá trị

for (int i = 0; i < len; i++) {

code.data[i] = box[rand() % box.length()];

}

B4: Trả về số BigInt cần tìm

1. Hàm gcd: đầu vào 2 số BigInt(a và b, a>b), đầu ra 1 số BigInt là ước chung lớn nhất của cả 2

B1: Tạo số r

BigInt r;

B2: Lặp cho đến khi giá trị của b(b.data == 0) thì dừng lại. Trong mỗi vòng lặp, gán r bằng số bé hơn, số bé hơn lúc này sẽ bằng số dư của phép chia a cho b, và a sẽ nhận giá trị của r

while (b.data != "0") { r = b; b = Mod10(a, b); a = r; }

B3: Kết thúc vòng lặp, trả ra giá trị của a

1. Submod: Hàm hỗ trợ tính theo yêu cầu, đưa vào 3 số a,b,c

B1: Khai báo một số biến sử dụng, với b.data==”1” thì trả về giá trị a%c

BigInt two;

two.data = "2";

if (b.data == "1") { return Mod10(a, c); }

B2: Nếu không thỏa điều kiện ở bước 1, ta đặt BigInt x làm biến mới, rồi gán nó giá trị bằng Submod(a,mod10(a,b),c)

BigInt x = submod(a, Div10(b, two), c);

B3: kiểm tra b chẵn hay lẻ rồi tính submod theo 2 cách tương ứng

if (Mod10(b, two).data == "0") { return Mod10(Mul10(x, x), c); }

else { return Mod10(Mul10(Mod10(Mul10(x, x), c),a),c); }

1. Hàm phân tích: Đưa vào 2 số BigInt a và c, b được truyền tham chiếu

B1: khởi tạo biến.bắt đầu vòng lặp Khi data(a%2)==0 thì thoát khỏi vòng lặp. Mỗi lần trong vòng đọc có 1 biến đếm tăng 1 và a nhận giá trí a/2

b = 0;

BigInt two;

two.data = "2";

while (Mod10(a,two).data == "0") {

b++;

a = Div10(a, two);

}

B2: gán lại c=a

1. Hàm wn

B1: Tạo 1 số biến ban đầu. Tính giá trị số mũ và gán cho b, temp được gán cho submod(a,c,n)

int b;

BigInt c;

BigInt one;

one.data = "1";

phantich(Minus(n,one,"10"), b, c);

BigInt temp = submod(a, c, n);

B2: nếu temp.data==1 trả về đúng

if (temp.data == "1") { return true; }

B3: Nếu không thỏa điều kiện part 2. Ta tính temp 1=(temp\*temp)%n

else {

while (b >= 0) {

BigInt temp1 = Mod10(Mul10(temp, temp), n)

B4: Nếu temp1.data==”1” thì kiểm tra temp.data == Minus(n, one,"10").data trả về đúng, nếu không thỏa trả về sai

if (temp1.data == "1") {

if (temp.data == Minus(n, one,"10").data) { return true; }

else { return false; }

}

temp = temp1;

b--;

B5: kết thúc trả về sai

1. Hàm Prime

B1: Kiểm tra 1,2,3,4, số dương, số âm có thỏa mãn với điều kiện số nguyên tố

if (n.data < "1" || n.sign == 1) { return false; }

if (n.data == "2" || n.data == "3") { return true; }

B2: Tạo số randnum bằng 1 số bất kì sao cho randnum= 1 số Bigint bất kì % (n-3). Sau đó tạo 1 bigint a = n+2

BigInt randnum = Mod10(randomBig(n.data.length()), Minus(n, three, "10"));

BigInt a = Add(two, randnum,"10");

B3: Nếu gcd của 2 số a và n không nguyên tố cùng nhau thì không phải số nguyên tố. Nếu nguyên tố cùng nhau ta kiểm tra tiếp theo hàm wn

if (gcd(a, n).data != "1") { return false; }

else { return wn(a, n); }

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 263167 Isprime | Day la so nguyen to |
| 2 | 10 213456 Isprime | Day khong la so nguyen to |

Tham khảo : https://www.giaithuatlaptrinh.com/?p=278

# Các tiện ích khác

## 4.1) Time

Tìm thời gian thực thi phép toán bằng hàm có sẵn trong thư viện time.h:

Bước 1: Khai báo thêm thư viện time.h

#include <time.h>

Bước 2: Khai báo biến và tính thời gian thực thi phép toán: Nếu cuối dòng lệnh nhập vào có lệnh yêu cầu TIME thì ta ghi ra thời gian thực thi của dòng lệnh.

clock\_t start, end;

double duration;

{

Xử lí phép toán

}

end = clock();

if (tg == "TIME") {

duration = (double)(end - start) / CLOCKS\_PER\_SEC;

OutputTime(duration);

Hàm ghi ra thời gian:

void OutputTime(double d) {

fo << "\*Time Running: " << d << endl;

}

\*Lưu ý: chỉ những dòng lệnh đầu vào có lệnh TIME ở cuối thì mới ghi ra thời gian thực thi phép toán.

|  |  |  |
| --- | --- | --- |
| Dòng | Input.txt | Output.txt |
| 1 | 10 263167 Isprime TIME | Day la so nguyen to  \*Time Running: 0.216 |
| 2 | 10 2 7749 TIME | 01111001000101  \*Time Running: 0.001 s |

## 4.2) Nhập xuất

Đoc dữ liệu đầu vào từ file **Input.txt** gồm có n dòng ( chưa biết số lượng dòng), ta sử dụng vòng lặp while với câu lệnh !fi.eof() để kiểm tra xem đã là dòng cuối cùng của file chưa. Mỗi dòng ta đọc dưới dạng một chuỗi ký tự. Sau đó dùng hàm **Input** để cắt từng phần của dữ liệu đầu vào (hệ, số thứ nhất, phép toán, số thứ hai, lệnh yêu cầu ghi ra thời gian). Đồng thời với những trường hợp ngoại lệ về cách nhập vào (không có số thứ 2,…) ta sử dụng lệnh if để kiểm tra.

Hàm gọi chung để xử lí yêu cầu của dòng lệnh: **Operation** với dữ diệu đầu vào của hàm gồm he (hệ cơ số của số nguyên nhập vào), s1 (số thứ nhất), pheptoan (yêu cầu của dòng lệnh nhập vào), s2 (số thứ 2 nếu có) và tg (mặc định tg=”TIME” nếu muốn tìm thời gian thực thi). Sử dụng lệnh if … else để kiểm tra phép toán mà dòng lệnh đầu vào yêu cầu, xử lí và ghi ra kết quả bằng các hàm xuất dữ liệu.

Hàm xuất dữ liệu: **Output**  với dữ liệu đầu vào của hàm được lưu dưới dạng BigInt. Hàm sẽ kiểm tra dấu của dữ liệu vào nếu là số âm thì xuất dấu “-“ ở phía trước. Ngoài ra còn có một số dòng kết quả xuất trực tiếp ra file không thông qua hàm xuất nào ( xuất thời gian,… ).

## 4.3) Các hàm thường sử dụng

Hàm Add0: Là hàm thông qua vòng lặp tạo 1 chuỗi có độ dài bằng với độ dài mong muốn bằng cách thêm 0 vào trước chuỗi đó

string Add0(string s, int n) {

int temp = s.length();

for (int i = 0; i < n - temp; i++)

s = "0" + s;

return s;

}

Hàm Delete0: Là hàm ngược với hàm Add0 bằng cách xóa bớt các số 0 dư thừa ở đầu 1 chuỗi

BigInt Delete0(BigInt& s) {

while (s.data[0] == '0' && s.data.length() > 1)

{

s.data.erase(0, 1);

}

return s;

}