**Trường đại học Khoa học Tự nhiên**

**Khoa Công nghệ thông tin**

----- 🙡 🕮 🙣 -----

ĐỒ ÁN GIỮA KỲ

Đề tài: Số BIGINT

Giảng viên môn học: Võ Hoài Việt

Nguyễn Hoàng Lâm – 20120316

Trần Kiều Minh Lâm – 20120018

2020-2021

**Đề bài**: Kiểu dữ liệu số nguyên lớn có dấu gọi là BigInt có độ lớn từ 16 bytes (128 bits) trở lên gồm một số chức năng sau:

1) Chuyển đổi số BigInt từ hệ thập phân sang hệ nhị phân và ngược lại

2) Các operator=, operator+, operator-, operator\*, operator%

3) Các toán tử AND “&”, OR “|”, XOR “^”, NOT “~”

4) Các toán tử: dịch trái “<<, dịch phải >>.

5) Các hàm hỗ trợ: abs, min, max, pow, digits (số lượng ký tự số), to\_string, to\_base32, to\_base58, to\_base64, is\_prime

**Phần I: Phân tích đề:**

Kiểu dữ liệu integer trong ngôn ngữ C/C++ chỉ lưu trữ được 4 bytes cho hệ điều hành 32bit và 8 bytes cho hệ điều hành 64bit. Do đó những số nguyên lớn hơn ta phải dùng đến kiểu dữ liệu Big Integer, có thể lưu trữ từ 16 bytes trở lên, tương ứng với 128bit.

Để có thể lưu trữ kiểu dữ liệu lớn như vậy, chúng ta sẽ dùng struct BigInt để lưu trữ số nguyên lớn, trong đó sẽ có biến data (kiểu string) để lưu trữ số Big Integer. Mỗi 1 phần tử trong data sẽ biểu thị một chữ số trong số Big Integer. Như vậy nếu cần lưu trữ một số nguyên lớn hơn 65535 hay 216-1, ta chỉ cần một string có từ 5 phần tử trở lên.

Từ đó ta xây dựng các struct và các hàm chức năng tương đương để xử lí số Big Integer: hàm chuyển đổi hệ cơ số, integer to binary, binary to integer. Hàm khởi tạo, hàm xuất, các toán tử xung quanh nó.

**Phần II: Nhập và xuất:**

* Đọc dữ liệu từ file input.txt và xuất ra file output.txt.
* Mỗi dòng trong file input.txt tương ứng với 1 dòng trong file output.txt.
* Đọc từng dòng và phân tách từng phần tử trong 1 dòng. Có 3 trường hợp:

+ Phần tử index 0 và index 1 là 2 cơ số.

+ Phần tử index 0 là cơ số.

+ Phần tử index 1 là “~”.

**Phần III: Xử lí chức năng cơ bản**

**1. Struct BigInt.**

struct BigInt

{

int sign, base;

int numBit = 128;

string data;

BigInt& operator=(const BigInt &number) {

sign = number.sign;

base = number.base;

data = number.data;

return \*this;

}

void SetData(int numBase, string number) {

data = number;

base = numBase;

sign = 1;

if(base == 2 && number.size() >= numBit && number[0] == '1'){

sign = -1;

}

if (base == 10 && number[0] == '-'){

sign = -1;

data.erase(0, 1);

}

}

};

**Cấu tạo struct gồm**:

* Sign (kiểu integer): thông báo tính chất âm dương của số nhập vào, trong đó “1” là dương, “-1” là âm, “0” là số 0.
* Base(kiểu integer): là cơ số của số BigInt: “2” là nhị phân, “10” là thập phân.
* Numbit = 128: lấy số bit mặc định là 128 để xét bù 2.
* Data(kiểu string): lưu trữ dữ liệu của số nguyên lớn.
* Hàm setData(): khởi tạo dữ liệu cho 1 số BigInt.

**2. Chuyển đổi từ Binary sang Decimal và ngược lại:**

* **Hàm toBinary(BigInt number):** chuyển đổi 1 số BigInt từ hệ thập phân sang nhị phân. Hàm trả về một BigInt.

Cách hoạt động:

+ Đầu tiên ta đổi base sang 2.

+ Gán sign (dấu) của tham số cho kết quả.

+ ta chia lấy dư chữ số cuối cùng cho 2, sau đó chia cả số đó cho 2, kết quả lưu vào biến RES.

+ Sau khi chia xong ta đảo ngược toàn bộ chuỗi

+ Xét trường hợp nếu là số âm thì ta dùng bù 2: ta đảo tất cả các bit và sau đó cộng thêm “1”.

* **Hàm toDecimal(BigInt number):** chuyển đổi 1 số BigInt từ hệ nhị phân sang thập phân. Hàm trả về một BigInt.

Cách hoạt động:

+ Đầu tiên ta đổi base sang 10.

+ Gán sign (dấu) của tham số cho kết quả.

+ Nếu số âm thì dùng bù 2 để chuyển lại thành số dương.

+ cho vòng lặp chạy từ phải qua trái của dãy bit, nhân bit đó với biến

pow2 (int), mỗi lần lặp biến pow2 nhân 2.

+ Nếu số âm thì gán lại sign (dấu) ban đầu cho số kết quả.

**+Một số hàm hỗ trợ thêm:**

* **Compare(BigInt a, BigInt b):** hàm so sánh 2 số BigInt truyền vào.
  + Nếu a>b trả về 1.
  + a<b trả về -1.
  + a=b trả về 0.
* Hàm **deleteZero**: xóa những bit “0” không cần thiết ở đầu dãy bit.
* Hàm **addZero**: thêm những bit “0” vào đầu dãy bit để 2 dãy bit dài bằng nhau.
* **Abs:** trả về giá trị tuyệt đối của số truyền vào.
* **min, max**: trả về giá trị nhỏ hơn, lớn hơn của 2 số truyền vào.
* **Pow:** tính lũy thừa.
* **digits (số lượng ký tự số):** trả về số lượng chữ số.
* **to\_string**: chuyển số ra dạng string.
* **to\_base32, to\_base58, to\_base64**: chuyển số sang những hệ cơ số 32, 58, 64.
* **is\_prime**: kiểm tra một số liệu có phải là số nguyên tố hay không.
* Hàm trả về thời gian xử lí. Muốn sử dụng thì cmd thêm tham số checktime vào sau.
* Hàm trả về bộ nhớ sử dụng.

**Phần IV: Xử lí các toán tử**

**1. Các toán tử hai ngôi cơ bản**

* **Toán tử “+”:** sử dụng phép cộng như cộng số thập phân bình thường, cộng từ phải qua trái và nhớ 1. Ngoài ra áp dùng toán tử “-“ trong trường hợp toán hạng có số âm.
* **Toán tử “-“:** trừ như trừ hai số thập phân bình thường, trừ từ phải qua trái, nếu số bị trừ nhỏ hơn số trừ thì nhớ 1. Ngoài ra áp dùng toán tử “+“ trong trường hợp dương trừ cho âm.
* **Toán tử “\*”:**  nhân như nhân hai số thập phân bình thường, nhân số a cho từng chữ số của số b (từ phải qua trái), sau đó cộng các tích lại.
* **Toán tử “/”:** (chia lấy nguyên): chia như chia hai số thập phân bình thường, chia số bị chia, hạ từng chữ số xuống, từ trái qua phải.
* **Toán tử “%”:** (chia lấy dư) với hai phép nhân và chia lấy phần nguyên ta có thể tính phép đồng dư một cách dễ dàng: a % b = a - (a / b) \* b (với phép chia là chia lấy phần nguyên)

**2. Các toán tử 2 ngôi với bit**

* **Toán tử “>>”:** toán tử dịch phải, toàn bộ dãy bit sẽ dịch qua bên phải một số lần bất kì nhập từ bàn phím.

Cách hoạt động: dùng vòng lặp, cho lặp lại số lần bằng với số bit dịch phải. Môi lần lặp thì xóa bớt 1 bit ở bên phải ngoài cùng.

* **Toán tử “<<”:** toán tử dịch trái, toàn bộ dãy bit sẽ dịch qua bên trái một số lần bất kì nhập từ bàn phím.

Cách hoạt động: dùng vòng lặp, cho lặp lại số lần bằng với số bit dịch trái. Môi lần lặp thì thêm 1 bit “0” ở bên phải ngoài cùng.

* **Toán tử “&”:** toán tử AND:
  + 1 & 1 = 1
  + 1 & 0 = 0
  + 0 & 0 = 0
  + 0 & 1 = 0

Ta duyệt lần lượt qua từng phần tử của 2 số nhập vào, nếu 1&1 thì trả về 1 và làm tương tự như trên với từng bit còn lại. Các bit sẽ được dóng hàng sang phải bằng nhau.

* **Toán tử “|” :** toán tử OR:
* 1 | 1 = 1
* 1 | 0 = 1
* 0 | 1 = 1
* 0 | 0 = 0

Ta duyệt lần lượt qua từng phần tử của 2 số nhập vào, nếu 1|1 thì trả về 1 và làm tương tự như trên với từng bit còn lại. Các bit sẽ được dóng hàng sang phải bằng nhau.

* **Toán tử “^”:** toán tử XOR:
* 1 ^ 1 = 0
* 1 ^ 0 = 1
* 0 ^ 1 = 1
* 0 ^ 0 = 0

Ta duyệt lần lượt qua từng phần tử của 2 số nhập vào, nếu 1^1 thì trả về 0 và làm tương tự như trên với từng bit còn lại. Các bit sẽ được dóng hàng sang phải bằng nhau.

**3. Toán tử một ngôi:**

* **Toán tử “~”:** toán tử NOT:
* ~1 = 0
* ~0 = 1

Ta duyệt lần lượt qua từng phần tử của 2 số nhập vào, nếu ~1 thì trả về 0 và ~0 thì trả về 1.

**Phần V: Các thư viện và nguồn tham khảo**

**1. Các thư viện sử dụng trong chương trình**

* <iostream>
* <string>
* <cstdio>
* <sstream>

Một số thư viện dùng cho hàm tính thời gian.

* <ctime>
* <ratio>
* <chrono>

**2. Các nguồn tham khảo**

* <https://stackoverflow.com/>
* <https://www.geeksforgeeks.org/>
* <https://cpp.daynhauhoc.com/>
* <https://www.cplusplus.com/reference/>

Hàm tính thời gian

* <https://stackoverflow.com/questions/41077377/how-to-get-current-time-in-milliseconds/41077420?fbclid=IwAR2hULTT5OzFGK5LgThB2NPrmhn17XdbRw54bSp4pM-M9ab6WqIg290enlU>