ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Thực hành Kĩ thuật lập trình

ĐỒ ÁN MÔN HỌC

Big Integer

Sinh viên thực hiện

Hồ Văn Sơn
Trần Hữu Thiên

Tháng 5 năm 2021

Mục Lục

1. Mô tả Đồ án

Chương trình thực hiện các thao tác đối với 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`

Ngoài ra, chương trình thực thi đọc tham số dòng lệnh ở dạng command line và cho phép xuất ra thời gian xử lý và dung lượng bộ nhớ đã dùng trong mỗi thao tác hoặc hàm.

2. Kiểu cấu trúc `BigInt`

2.1. Khai báo kiểu cấu trúc `BigInt`

```
struct BigInt {
    int nData;
    uint16_t *pData;

    // Constructor
    BigInt();
    BigInt(int16_t x);

    // Destructor
    ~BigInt();

    // Copy assignment operator (couldn't be friend function)
    BigInt& operator = (const BigInt &y);

    // Resize BigInt
    void resize(int x);

    // Delete unused space at the end of dynamic array
    void fix();
};
```

Đây là cấu trúc `BigInt` mà nhóm đã xây dựng. Cụ thể các hàm như sau:

- Biến `nData` dùng để lưu độ lớn vùng nhớ của pData
- Biến `*pData` dùng để lưu `BigInt` theo đơn vị từng số nguyên dương 16-bit
- Hàm `BigInt()`: Khởi gán giá trị của cấu trúc `BigInt`

```
BigInt::BigInt() {
    nData = 1;
    pData = new uint16_t [1];
    pData[0] = 0;
}
```

- Hàm `BigInt(int16_t x)`: Gán cho `BigInt` giá trị của số nguyên 16-bit `x`

```
BigInt::BigInt(int16_t x) {
    nData = 1;
    pData = new uint16_t [1];
    pData[0] = x;
}
```

- Hàm `~BigInt()`: Xóa vùng nhớ đã cấp phát cho `BigInt`

```
BigInt::~BigInt() {
    if (nData && pData != NULL)
        delete []pData;
    nData = 0;
    pData = NULL;
}
```

- Toán tử gán (`operator=`):

```
BigInt& BigInt::operator = (const BigInt &y) {
    if (nData && pData != y.pData)
        delete []pData;
    nData = y.nData;
    if (nData) {
        pData = new uint16_t [nData];
        memcpy(pData, y.pData, sizeof(uint16_t) * nData);
    } else
        pData = NULL;
    return (*this);
}
```

  - Với phép gán struct `BigInt` thông thường, ta sẽ gán luôn con trỏ `pData` của hai struct đó cho nhau, và khi thực hiện các truy vấn đến 1 trong 2 struct đó, con trỏ của struct còn lại có thể bị thay đổi. Điều đó cũng xảy ra khi mà ta cài đặt `operator=` ở bên ngoài struct `BigInt`. Vì vậy ta xây dựng toán tử gán (`operator=`) ở bên trong struct `BigInt` để cấp 1 bộ nhớ mới cho `pData` của `BigInt`, giúp cho 2 struct của chúng ta trở nên độc lập.
  - Phép gán `operator=` trả về kết quả là một tham chiếu. Với cách này, ta có thể sử dụng chuỗi phép gán các `BigInt` (vd: `a = b = c`). Ta cũng có thể sử dụng cách trả về `void`, nhưng việc đó sẽ ngăn ta thực hiện thao tác trên.

- Hàm `resize()`: Tùy chỉnh kích thước của `BigInt`

```
void BigInt::resize(int x) {
    uint16_t *pTemp = new uint16_t [x];
    unsigned char padding_value = 0x00;
    padding_value =
        (nData && (pData[nData - 1] >> (BASE - 1) & 1)) ? 0xff : 0x00;
    memset(pTemp, padding_value, sizeof(uint16_t) * x);
    memcpy(pTemp, pData, sizeof(uint16_t) * min(x, nData));
    if (nData) delete []pData;
    pData = pTemp;
    nData = x;
}
```

  - Ta sử dụng `padding_value` là một biến kí tự để chèn vào những ô nhớ mà ta đã thêm vào, với `0xff` đại diện cho chuỗi 8 bit 1 và `0x00` đại diện cho chuỗi 8 bit 0 tương ứng với 2 trường hợp là số âm và số dương.

  - Sử dụng một hằng số `BASE` được định nghĩa trong file header, vì kiểu cấu trúc sử dụng đơn vị là số nguyên 16-bit nên gán `BASE` = 16.

- Hàm `fix()`: Xóa các ô nhớ không sử dụng ở cuối vùng nhớ

```
void BigInt::fix() {
    if (nData <= 1) return;
    uint16_t sign = (pData[nData - 1] >> (BASE - 1) & 1);
    for (int i = nData - 1; i >= 0; --i)
        if (pData[i] != sign * UINT16_MAX) {
            if ((pData[i] >> (BASE - 1) & 1) == sign)
                (*this).resize(i + 1);
            else
                (*this).resize(i + 2);
            return;
        }
    (*this).resize(1);
}
```

2.2. Đọc số `BigInt` từ `istream`

Nhóm đề xuất 2 cách đọc `BigInt` từ `istream` đó là:

- Đọc từ dạng nhị phân:

```
void readBinary(istream& is, BigInt &x) {
    string buffer;
    is >> buffer;
    x = Base2ToBigInt(buffer);
}
```

- Đọc từ dạng thập phân:

```
void readDecimal(istream& is, BigInt &x) {
    string buffer;
    is >> buffer;
    x = Base10ToBigInt(buffer);
}
```

Chi tiết các hàm `Base2ToBigInt` và `Base10ToBigInt` sẽ được giới thiệu ở mục sau.

2.3. Ghi số `BigInt` vào `ostream`

Nhóm đề xuất 2 cách ghi `BigInt` vào `ostream` như sau:

- Ghi bằng dạng nhị phân:

```
void writeDecimal(ostream &os, const BigInt &x) {
    os << to_string(x);
}
```

- Ghi bằng dạng thập phân:

```
void writeBinary(ostream &os, const BigInt &x) {
    os << BigIntToBase2(x);
}
```

Chi tiết các hàm `to_string` và `BigIntToBase2` sẽ được giới thiệu ở mục sau.

3. Các hàm bổ sung tính năng cho chương trình

3.1. Các hàm xét dấu BigInt

```
bool isZero(const BigInt &x) {
    return x.nData == 1 && x.pData[0] == 0;
}

bool isNegative(const BigInt &x) {
    return x.pData[x.nData - 1] >> (BASE - 1) & 1;
}

bool isPositive(const BigInt &x) {
    return !isZero(x) && !isNegative(x);
}
```

3.2. Hàm swap dùng cho 2 số BigInt

```
void swap(BigInt &x, BigInt &y) {
    BigInt temp;
    temp = x;
    x = y;
    y = temp;
}
```

3.3. Các toán tử so sánh 2 số BigInt

Bao gồm các toán tử: ==, !=, <, <=, >, >=

```
bool operator == (const BigInt &x, const BigInt &y) {
    if (x.nData != y.nData)
        return false;
    for (int i = 0; i < x.nData; ++i)
        if (x.pData[i] != y.pData[i])
            return false;
    return true;
}

bool operator != (const BigInt &x, const BigInt &y) {
    return !(x == y);
}

bool operator < (const BigInt &x, const BigInt &y) {
    int sign_x = isNegative(x),
        sign_y = isNegative(y);

    if (sign_x != sign_y)
        return sign_x > sign_y;
    if (x.nData != y.nData)
        return (x.nData < y.nData) ^ sign_x;
    for (int i = x.nData - 1; i >= 0; --i)
        if (x.pData[i] != y.pData[i])
            return x.pData[i] < y.pData[i];
    return false;
}

bool operator <= (const BigInt &x, const BigInt &y) {
    return x < y || x == y;
}

```

```
bool operator > (const BigInt &x, const BigInt &y) {
    return !(x <= y);
}

bool operator >= (const BigInt &x, const BigInt &y) {
    return !(x < y);
}
```

3.4. Hàm `divString`

Thực hiện phép chia string với một số nguyên.

Hàm này trả về kết quả của phép chia số `dividend` cho `divisor`, cùng với số dư của phép chia này là `remainder`.

```
string divString(string dividend, uint64_t divisor, uint16_t &remainder) {
    string quotient = "";
    uint64_t carry = 0;
    for (char pNum: dividend) {
        carry = carry * 10 + pNum - '0';
        if (quotient.empty() && carry < divisor)
            continue;
        quotient += (char)(carry / divisor + '0');
        carry %= divisor;
    }
    remainder = carry;
    return quotient;
}
```

3.5. Hàm `EQuotient`

Mục đích của hàm là tìm thương của phép chia `BigInt x` cho `BigInt y` (với điều kiện là kết quả của phép chia là nhỏ hơn $2^{16} - 1$). Hàm này được cài đặt nhằm mục đích tăng tốc độ xử lí sau này khi thực hiện các phép chia với các số lớn hơn.

```
uint16_t EQuotient(const BigInt &x, const BigInt &y) {
    uint16_t ret = 0;
    for (int i = BASE - 1; i >= 0; --i) {
        ret += (1 << i);
        if (y * ret > x)
            ret -= (1 << i);
    }
    return ret;
}
```

3.6. Hàm `to_int`

Mục đích của hàm `to_int` là để chuyển `BigInt x` về kiểu số nguyên 16-bit cho việc tính toán, với điều kiện được đảm bảo là `x.nData = 1`.

```
int16_t to_int(const BigInt &x) {
    if (x.nData == 0)
        return 0;
    return x.pData[0];
}
```

4. Chuyển đổi số BigInt từ hệ nhị phân sang hệ thập phân và ngược lại

4.1. Chuyển đổi số BigInt từ hệ nhị phân sang hệ thập phân

Ta sẽ làm theo tuần tự các bước như sau:

- Bước 1: Đọc một số biểu diễn ở dạng hệ nhị phân vào BigInt

  Ta có hàm Base2ToBigInt như sau:

```
BigInt Base2ToBigInt(string str) {
    BigInt ret;
    ret.resize((str.length() - 1) / BASE + 1);
    reverse(str.begin(), str.end());
    for (int i = 0; i < str.length(); ++i)
        if (str[i] == '1')
            ret.pData[i / BASE] |= (1 << (i % BASE));
    if (str.back() == '1')
        for (int i = 0; i < (BASE - str.length() % BASE) % BASE; ++i)
            ret.pData[ret.nData - 1] |= (1 << (BASE - 1) - i);
    return ret;
}
```

- Bước 2: Biểu diễn BigInt ở dưới dạng hệ thập phân

  Có thể thấy rằng, việc biểu diễn BigInt về dạng thập phân cũng giống như việc ta biểu diễn BigInt thành một chuỗi số trong hệ cơ số 10. Nhằm đảm bảo làm đúng yêu cầu đề bài, nhóm xây dựng hàm to_string như sau:

```
string to_string(const BigInt &x) {
    string ret = "";
    BigInt temp = abs(x);
    while (!isZero(temp)) {
        ret += (char)(to_int(temp % 10) + '0');
        temp /= 10;
    }
    if (isNegative(x))
        ret += '-';
    reverse(ret.begin(), ret.end());
    if (ret.empty())
        ret += '0';
    return ret;
}
```

4.2. Chuyển đổi số BigInt từ hệ thập phân sang hệ nhị phân

Ta sẽ làm theo tuần tự các bước như sau:

- Bước 1: Đọc một số biểu diễn ở dạng hệ thập phân vào BigInt

  Ta có hàm Base10ToBigInt như sau:

```
BigInt Base10ToBigInt(string str) {
    BigInt ret;
    ret.resize(1000000);
    bool negative = false;
    if (str[0] == '-') {
        negative = true;
        str.erase(str.begin());
    }
    ret.nData = 0;
    while (!str.empty())
        str = divString(str, 1ll*UINT16_MAX+1, ret.pData[ret.nData++]);
    ret.nData++;
    ret.resize(ret.nData);
    if (negative)
        ret = -ret;
    ret.fix();
    return ret;
}
```

- Bước 2: Biểu diễn BigInt ở dưới dạng hệ nhị phân

  Ta có hàm BigIntToBase2 như sau:

```
string BigIntToBase2(const BigInt &x) {
    string ret = "";
    bool sign = isNegative(x), flag = true;
    for (int i = x.nData - 1; i >= 0; --i)
        for (int j = BASE - 1; j >= 0; --j) {
            bool bit = (x.pData[i] >> j & 1);
            if (flag && bit == sign)
                continue;
            if (flag) {
                ret += (char)(sign + '0');
                flag = false;
            }
            ret += (char)(bit + '0');
        }
    if (ret.empty()) ret += (x.nData && x.pData[0]) ? '1' : '0';
    return ret;
}
```

5. Các toán tử thao tác với `BigInt`

5.1. Các toán tử tính toán số học (`operator`)

5.1.1. Toán tử cộng (`operator+`)

1. Tư tưởng thuật toán:

   Dựa trên phép cộng của các số trong base 10, ta có thể khái quát lại phép cộng các số `BigInt` như sau:

   - Bước 1: Khai báo `BigInt ret` và thực hiện thao tác resize để `ret` chứa đủ kết quả phép tính.
   - Bước 2: Khai báo `int32_t carry` làm biến nhớ cho phép cộng và 2 biến `int8_t sign_x, sign_y` để lưu dấu của `x` và `y`.
   - Bước 3: Thực hiện phép cộng `BigInt x` và `BigInt y` theo quy tắc tính phép cộng.
     - Bước 3.1: Khai báo `int32_t temp` để lưu kết quả của phép cộng từng đơn vị. Khởi gán `temp = carry`
     - Bước 3.2: Thực hiện phép cộng tại từng đơn vị của BigInt.
       + Cộng giá trị biến nhớ (`carry`) vào `temp`.
       + Cộng giá trị `x.pData[i]` vào `temp` (nếu `BigInt x` không tồn tại đơn vị thứ i thì `x.pData[i] = 0` nếu `x` không âm, hoặc `x.pData[i] = UINT16_MAX` nếu x âm, theo phương pháp bù 2)
       + Cộng giá trị `y.pData[i]` vào `temp` tương tự như `x.pData[i]`.
     - Bước 3.3: Cập nhật `carry`:
       Thực hiện thao tác: `carry = temp > UINT16_MAX`
     - Bước 3.4: Cập nhật `ret`:
       Thực hiện: `ret.pData[i] = temp - (carry ? UINT16_MAX + 1 : 0)`.
       Vì $0 \leq$ `temp` $\leq 2^{32} - 1$ do đó phép tính trên tương đương với:
       `ret.pData[i] = temp % (UINT16_MAX + 1)`.

2. Cài đặt:

```
BigInt operator + (const BigInt &x, const BigInt &y) {
    BigInt ret;
    ret.resize(max(x.nData, y.nData) + 1);

    int32_t carry = 0;
    int8_t  sign_x = isNegative(x),
            sign_y = isNegative(y);

    for (int i = 0; i < ret.nData; ++i) {
        int32_t temp = carry;
        temp += (int32_t)(i < x.nData ? x.pData[i] : sign_x*UINT16_MAX);
        temp += (int32_t)(i < y.nData ? y.pData[i] : sign_y*UINT16_MAX);
        carry = temp > UINT16_MAX;
        ret.pData[i] = temp - (carry ? UINT16_MAX + 1 : 0);
    }

    ret.fix();

    return ret;
}
```

5.1.2. Toán tử trừ (operator-)

1. Tư tưởng thuật toán:

   - Phép trừ 2 số `BigInt` có thể được xem như là phép cộng của số `BigInt` thứ nhất với số đối của số `BigInt` thứ hai. Vì vậy, ta sẽ cài đặt thêm 1 hàm để lấy số đối 1 số `BigInt`, từ đó ta sẽ dễ dàng cài đặt phép trừ 2 số `BigInt`.
   - Thuật toán tìm số đối (theo phương pháp bù 2):
     - Đảo ngược tất cả các bit của BigInt x.
     - Tính `x = x + 1`, tức là chuyển bit 0 thấp nhất thành 1 và tất cả bit 1 trước bit 0 thấp nhất thành 0.

2. Cài đặt:

   - Phép lấy số đối của `BigInt`:

```
BigInt operator - (const BigInt &x) {
    BigInt ret;
    ret = x;
    ret.resize(ret.nData + 1);

    for (int i = 0; i < ret.nData; ++i)
        ret.pData[i] = ~ret.pData[i];

    for (int i = 0; i < ret.nData; ++i) {
        if (ret.pData[i] != UINT16_MAX) {
            ret.pData[i]++;
            break;
        }
        ret.pData[i] = 0;
    }

    ret.fix();

    return ret;
}
```

   - Phép trừ 2 số `BigInt`:

```
BigInt operator - (const BigInt &x, const BigInt &y) {
    return x + (-y);
}
```

5.1.3. Toán tử nhân (operator*)

Để thuận tiện cho việc tính toán và đưa vấn đề tối ưu lên hàng đầu, ta cài đặt 2 định nghĩa cho `operator*` như sau:

1. Phép nhân 1 số `BigInt` với 1 số nguyên 32-bit

   (a) Thuật toán:

   - Bước 1: Đầu tiên ta khai báo `BigInt ret, temp_x` và `int32_t temp_y` và khởi tạo `temp_x, temp_y` lần lượt là giá trị tuyệt đối của `x` và `y`. Đồng thời `resize` cho `ret` để `ret` có đủ kích thước chứa kết quả của phép nhân.

- Bước 2: Ta thực hiện phép nhân như trong base10. Khi đó ta khai báo 1 biến tạm là `int64_t temp` lưu kết quả của phép nhân `temp_x.pData[i]` và `temp_y`. Sau đó ta cộng dồn các kết quả vào biến `ret`.
- Bước 3: Sau khi thu được kết quả là số dương `ret`, ta thực hiện việc lấy dấu cho `ret` bằng cách XOR các dấu của `x` với `y`. Sau đó thực hiện thao tác `fix` rồi trả về biến `ret` là kết quả của phép tính.

(b) Cài đặt:

```
BigInt operator * (const BigInt &x, const int32_t &y) {
    BigInt ret, temp_x;
    int32_t temp_y;
    temp_x = x; if (isNegative(x)) temp_x = -x;
    temp_y = abs(y);
    ret.resize(temp_x.nData + 2);
    for (int i = 0; i < temp_x.nData; ++i) {
        int64_t temp = 1ll * temp_x.pData[i] * temp_y;
        for (int k = 0; ; ++k) {
            temp += ret.pData[i + k];
            ret.pData[i + k] = temp % (UINT16_MAX + 1);
            temp /= (UINT16_MAX + 1);
            if (!temp) break;
        }
    }

    if (isNegative(x) ^ (y < 0))
        ret = -ret;
    else
        ret.fix();

    return ret;
}
```

2. Phép nhân 2 số `BigInt`:

(a) Thuật toán:

- Bước 1: Đầu tiên ta khai báo `BigInt ret, temp_x, temp_y` và khởi tạo `temp_x, temp_y` lần lượt là giá trị tuyệt đối của `x` và `y`. Đồng thời `resize` cho `ret` để `ret` có đủ kích thước chứa kết quả của phép nhân.
- Bước 2: Ta thực hiện phép nhân như trong base10. Khi đó ta khai báo 1 biến tạm là `int64_t temp` lưu kết quả của phép nhân `temp_x.pData[i]` và `temp_y.pData[i]`. Sau đó ta cộng dồn các kết quả vào biến `ret`.
- Bước 3: Sau khi thu được kết quả là số dương `ret`, ta thực hiện việc lấy dấu cho `ret` bằng cách XOR các dấu của `x` với `y`. Sau đó thực hiện thao tác `fix` rồi trả về biến `ret` là kết quả của phép tính.

(b) Cài đặt:

```
BigInt operator * (const BigInt &x, const BigInt &y) {
    BigInt ret, temp_x, temp_y;
    temp_x = x; if (isNegative(x)) temp_x = -x;
    temp_y = y; if (isNegative(y)) temp_y = -y;

    ret.resize(temp_x.nData + temp_y.nData + 1);
    for (int i = 0; i < temp_x.nData; ++i)
        for (int j = 0; j < temp_y.nData; ++j) {
            int64_t temp = 1ll * temp_x.pData[i] * temp_y.pData[j];
```

```
            for (int k = 0; ; ++k) {
                temp += ret.pData[i + j + k];
                ret.pData[i + j + k] = temp % (UINT16_MAX + 1);
                temp /= (UINT16_MAX + 1);
                if (!temp) break;
            }
        }

    if (isNegative(x) ^ isNegative(y))
        ret = -ret;
    else
        ret.fix();

    return ret;
}
```

5.1.4. Toán tử div (operator/)

Để thuận tiện cho việc tính toán và đưa vấn đề tối ưu lên hàng đầu, ta cài đặt 2 định nghĩa cho `operator/` như sau:

1. Phép chia `BigInt x` cho `int32_t y`

   (a) Thuật toán:

   - Bước 1: Khởi gán `BigInt quotient` để lưu thương trả về, `BigInt temp_x` là giá trị tuyệt đối của `x`, và `int64_t modulor` để lưu giá trị số dư trong phép chia.
   - Bước 2: Duyệt từ đơn vị cao nhất đến đơn vị thấp nhất trong `x`.
     - Bước 2.1: Dịch `quotient, modulor` sang trái 16 bit.
     - Bước 2.2: Thực hiện `modulor += temp_x.pData[i]` để lưu giá trị dư tại vị trí đang duyệt tới.
       + Nếu `modulor < abs(y)` thì bỏ qua, tiếp tục duyệt.
       + Ngược lại thì cập nhật `quotient` và `modulor`:

         ```
         quotient.pData[0] = modulor / abs(y);
         modulor = modulor  - 1ll * abs(y) * quotient.pData[0];
         ```

   - Bước 3: Xác định dấu của `quotient` bằng cách XOR các dấu của x và y. Thực hiện thao tác `quotient.fix()` sau đó trả về `quotient`.

   (b) Cài đặt:

```
BigInt operator / (const BigInt &x, const int32_t &y) {
    BigInt quotient, temp_x;
    temp_x = x; if (isNegative(x)) temp_x = -x;
    int64_t modulor = 0;

    for (int i = temp_x.nData - 1; i >= 0; --i) {
        quotient <<= BASE;
        modulor <<= BASE;
        modulor += temp_x.pData[i];
        if (modulor < abs(y))
            continue;
        quotient.pData[0] = modulor / abs(y);
        modulor = modulor - 1ll * abs(y) * quotient.pData[0];
    }

    if (isNegative(x) ^ (y < 0))
```

```
        quotient = -quotient;
    else
        quotient.fix();

    return quotient;
}
```

2. Phép chia `BigInt x` cho `BigInt y`

   (a) Thuật toán:

   - Bước 1: Khởi gán `BigInt quotient` để lưu thương trả về, `BigInt temp_x, temp_y` là giá trị tuyệt đối của `x, y`, và `BigInt modulor` để lưu giá trị số dư trong phép chia.
   - Bước 2: Duyệt từ đơn vị cao nhất đến đơn vị thấp nhất trong `x`.
     - Bước 2.1: Dịch `quotient, modulor` sang trái 16 bit.
     - Bước 2.2: Thực hiện `modulor.pData[0] = temp_x.pData[i]` để lưu giá trị dư tại vị trí đang duyệt tới.
       + Nếu `modulor < temp_y` thì bỏ qua, tiếp tục duyệt.
       + Ngược lại thì cập nhật `quotient` và `modulor`:

         ```
         quotient.pData[0] = EQuotient(modulor, temp_y);
         modulor = modulor - temp_y * quotient.pData[0];
         ```

   - Bước 3: Xác định dấu của `quotient` bằng cách XOR các dấu của x và y. Thực hiện thao tác `quotient.fix()` sau đó trả về `quotient`.

   (b) Cài đặt:

```
BigInt operator / (const BigInt &x, const BigInt &y) {
    BigInt quotient, modulor, temp_x, temp_y;
    temp_x = x; if (isNegative(x)) temp_x = -x;
    temp_y = y; if (isNegative(y)) temp_y = -y;
    for (int i = temp_x.nData - 1; i >= 0; --i) {
        quotient <<= BASE;
        modulor <<= BASE;
        modulor.pData[0] = temp_x.pData[i];
        modulor.fix();
        if (modulor < temp_y)
            continue;
        quotient.pData[0] = EQuotient(modulor, temp_y);
        modulor = modulor - temp_y * quotient.pData[0];
    }

    if (isNegative(x) ^ isNegative(y))
        quotient = -quotient;
    else
        quotient.fix();

    return quotient;
}
```

5.1.5. Toán tử mod (operator%)

Để thuận tiện cho việc tính toán và đưa vấn đề tối ưu lên hàng đầu, ta cài đặt 2 định nghĩa cho `operator%` như sau:

1. Phép mod `BigInt x` cho `int32_t y`

(a) Tư tưởng thuật toán:

Nếu ta gán $2^{16} = U$ là đơn vị của mỗi `BigInt` thì với số `BigInt x` ta có thể biểu diễn như sau:

$$\begin{aligned} &\texttt{x} = x_0 \cdot U^0 + x_1 \cdot U^1 + \ldots + x_i \cdot U^i \\ \Rightarrow\ &\texttt{x} \bmod \texttt{y} = (x_0 \cdot U^0 + x_1 \cdot U^1 + \ldots + x_i \cdot U^i) \bmod \texttt{y} \\ \Leftrightarrow\ &\texttt{x} \bmod \texttt{y} = (x_0 \cdot U^0 \bmod \texttt{y} + x_1 \cdot U^1 \bmod \texttt{y} + \ldots + x_i \cdot U^i \bmod \texttt{y}) \bmod \texttt{y} \end{aligned}$$

Khi đó ta có thuật toán như sau:

- Bước 1: Khai báo `int32_t modulor = 0` lưu số dư, `int32_t powBase` để lưu số mũ của đơn vị xét, và `BigInt temp_x` lưu giá trị tuyệt đối của `x`.
- Bước 2: Duyệt từng đơn vị của `temp_x` từ thấp nhất đến cao nhất. Mỗi lần duyệt ta thực hiện cập nhật `modulor` và `powBase`:

```
modulor = (1ll*modulor + 1ll*temp_x.pData[i]*powBase % y) % y;
powBase = 1ll * powBase * (1 << BASE) % y;
```

- Bước 3: Khai báo `BigInt ret` để lưu giá trị của `modulor`. Thực hiện `ret.resize(2)` để đủ kích thước và gán `modulor` cho `ret`.
- Bước 4: Thực hiện `ret.fix()` và lấy dấu cho `ret`, sau đó trả về kết quả là `ret`.

(b) Cài đặt:

```
BigInt operator % (const BigInt &x, const int32_t &y) {
    int32_t modulor = 0, powBase = 1;
    BigInt temp_x;
    temp_x = x; if (isNegative(x)) temp_x = -x;

    for (int i = 0; i < temp_x.nData; ++i) {
        modulor = (1ll * modulor
            + 1ll * temp_x.pData[i] * powBase % y) % y;
        powBase = 1ll * powBase * (1 << BASE) % y;
    }

    BigInt ret;
    ret.resize(2);
    ret.pData[0] = modulor % (1 << BASE);
    ret.pData[1] = modulor / (1 << BASE);
    ret.fix();
    if (isNegative(x))
        ret = -ret;

    return ret;
}
```

2. Phép mod `BigInt x` cho `BigInt y`

(a) Tư tưởng thuật toán:

- Bước 1: Khai báo `BigInt modulor` để lưu kết quả số dư, `BigInt temp_x, temp_y` lưu giá trị tuyệt đối của `x, y`
- Bước 2: Duyệt từng đơn vị của `x`, từ cao nhất đến thấp nhất.
  - Bước 2.1: Dịch `modulor` sang trái 16 bit
  - Bước 2.2: Gán `modulor.pData[0] = temp_x.pData[i]` và thực hiện thao tác `modulor.fix()`
    + Nếu `modulor < temp_y` thì tiếp tục duyệt

+ Ngược lại, gán `modulor` bằng kết quả của `modulor % temp_y`

- Bước 3: Xác định dấu cho `modulor` rồi trả về kết quả là `modulor`.

(b) Cài đặt:

```
BigInt operator % (const BigInt &x, const BigInt &y) {
    BigInt modulor, temp_x, temp_y;
    temp_x = x; if (isNegative(x)) temp_x = -x;
    temp_y = y; if (isNegative(y)) temp_y = -y;

    for (int i = temp_x.nData - 1; i >= 0; --i) {
        modulor <<= BASE;
        modulor.pData[0] = temp_x.pData[i];
        modulor.fix();
        if (modulor < temp_y)
            continue;
        int32_t temp = EQuotient(modulor, temp_y);
        modulor = modulor - temp_y * temp;
    }

    if (isNegative(x))
        modulor = -modulor;

    return modulor;
}
```

5.2. Các toán tử thao tác trên bit

☐ **Tư tưởng chung**: Duyệt từng đơn vị của 1 số (hoặc 2 số) `BigInt` rồi thực hiện các phép toán trên từng đơn vị.

5.2.1. Toán tử AND (operator&)

```
BigInt operator & (const BigInt &x, const BigInt &y) {
    BigInt temp_x, temp_y;
    temp_x = x; temp_y = y;
    temp_x.resize(max(temp_y.nData, temp_x.nData) + 1);
    temp_y.resize(temp_x.nData);
    for (int i = 0; i < temp_x.nData; ++i)
        temp_x.pData[i] &= temp_y.pData[i];
    temp_x.fix();
    return temp_x;
}
```

5.2.2. Toán tử OR (operator|)

```
BigInt operator | (const BigInt &x, const BigInt &y) {
    BigInt temp_x, temp_y;
    temp_x = x; temp_y = y;
    temp_x.resize(max(temp_y.nData, temp_x.nData) + 1);
    temp_y.resize(temp_x.nData);
    for (int i = 0; i < temp_x.nData; ++i)
        temp_x.pData[i] |= temp_y.pData[i];
    temp_x.fix();
    return temp_x;
}
```

5.2.3. Toán tử XOR (operator^)

```
BigInt operator ^ (const BigInt &x, const BigInt &y) {
    BigInt temp_x, temp_y;
    temp_x = x; temp_y = y;
    temp_x.resize(max(temp_y.nData, temp_x.nData) + 1);
    temp_y.resize(temp_x.nData);
    for (int i = 0; i < temp_x.nData; ++i)
        temp_x.pData[i] ^= temp_y.pData[i];
    temp_x.fix();
    return temp_x;
}
```

5.2.4. Toán tử NOT (operator~)

```
BigInt operator ~ (const BigInt &x) {
    BigInt ret;
    ret = x;
    for (int i = 0; i < ret.nData; ++i)
        ret.pData[i] = ~ret.pData[i];
    ret.fix();
    return ret;
}
```

5.3. Các toán tử dịch trái, phải

5.3.1. Toán tử dịch trái (operator<<)

1. Tư tưởng thuật toán:

   Với `nShift` là độ dời bit của `BigInt x` thì ta có thể gán 2 giá trị `nBlock = nShift / BASE` và `nBit = nShift % BASE`. Khi đó ta thực hiện dời `BigInt x` sang trái `nBlock` đơn vị (block 16-bit) và `nBit` bit, ta sẽ thu được kết quả của bài toán.

   - Bước 1: Khai báo `BigInt ret` lưu giá trị của `x` và 2 biến `int nBlock = nShift / BASE, nBit = nShift % BASE`.
   - Bước 2: Dịch `x` sang trái `nBlock` đơn vị:
     Duyệt các đơn vị của `x` từ cao nhất đến thấp nhất
     - Nếu `i >= nBlock` thì `ret.pData[i] = ret.pData[i - nBlock]`.
     - Ngược lại, `ret.pData[i] = 0`.
   - Bước 3: Dịch `x` sang trái `nBit` bit:
     Duyệt các đơn vị của `x` từ cao nhất đến thấp nhất
     - Dịch `ret.pData[i]` sang trái `nBit`.
     - Nếu `i > 0` thì thực hiện `ret.pData[i] |= (ret.pData[i - 1] >> (BASE - nBit)` để đảm bảo các bit bằng 1 không bị thất thoát.
       Trong đó `(ret.pData[i - 1] >> (BASE - nBit))` là phép toán lấy nBit bit cuối cùng của `ret.pData[i - 1]`.
   - Bước 4: Thực hiện `ret.fix()` và trả về kết quả là `ret`.

2. Cài đặt:

```
BigInt operator << (const BigInt &x, int nShift) {
    BigInt ret;
    ret = x;

```

```
    // Shift block
    int nBlock = nShift / BASE;
    if (nBlock) {
        ret.resize(ret.nData + nBlock);
        for (int i = ret.nData - 1; i >= 0; --i)
            ret.pData[i] = i >= nBlock ? ret.pData[i - nBlock] : 0;
    }

    // Shift bit
    int nBit = nShift % BASE;
    if (nBit) {
        ret.resize(ret.nData + 1);
        for (int i = ret.nData - 1; i >= 0; --i) {
            ret.pData[i] <<= nBit;
            if (i > 0)
                ret.pData[i] |= (ret.pData[i - 1] >> (BASE - nBit));
        }
        ret.fix();
    }

    return ret;
}
```

5.3.2. Toán tử dịch phải (operator>>)

1. Tư tưởng thuật toán:

   Với `nShift` là độ dời bit của `BigInt x` thì ta có thể gán 2 giá trị `nBlock = nShift / BASE` và `nBit = nShift % BASE`. Khi đó ta thực hiện dời `BigInt x` sang phải `nBlock` đơn vị (block 16-bit) và `nBit` bit, ta sẽ thu được kết quả của bài toán.

   - Bước 1: Khai báo `BigInt ret` lưu giá trị của `x` và 2 biến `int nBlock = nShift / BASE, nBit = nShift % BASE`.
   - Bước 2: Dịch `x` sang phải `nBlock` đơn vị. Duyệt các đơn vị của `x` từ thấp nhất đến cao nhất và thực hiện `ret.pData[i] = ret.pData[i + nBlock]`. Sau đó resize `ret` về kích thước `ret.nData - nBlock`.
   - Bước 3: Dịch `x` sang phải nBit bit:

     Duyệt các đơn vị của `x` từ thấp nhất đến cao nhất
     - Dịch `ret.pData[i]` sang phải `nBit`.
     - Thực hiện `ret.pData[i] |= (ret.pData[i + 1] & ((1ll << nBit) - 1) << (BASE - nBit)` để xác định vị trí các bit bằng 1 của đơn vị thứ `i`.
       Trong đó `(ret.pData[i + 1] & ((1ll << nBit) - 1)` là phép toán lấy nBit bit đầu tiên của `ret.pData[i + 1]`.
   - Bước 4: Thực hiện xác định dấu cho `ret` bằng cách duyệt các bit từ 0 tới nBit và thực hiện `ret.pData[ret.nData - 1] |= (1 << (BASE - i - 1))`
   - Bước 5: Thực hiện `ret.fix()` và trả về kết quả là `ret`.

2. Cài đặt:

```
BigInt operator >> (const BigInt &x, int nShift) {
    BigInt ret;
    ret = x;

    // Shift block
```

```
    int nBlock = nShift / BASE;
    if (nBlock) {
        for (int i = 0; i < ret.nData - nBlock; ++i)
            ret.pData[i] = ret.pData[i + nBlock];
        ret.resize(ret.nData - nBlock);
    }

    // Shift bit
    int nBit = nShift % BASE;
    if (nBit) {
        bool negative = isNegative(ret);
        for (int i = 0; i < ret.nData; ++i) {
            ret.pData[i] >>= nBit;
            if (i + 1 != ret.nData)
                ret.pData[i] |= (ret.pData[i + 1]
                    & ((1ll << nBit) - 1)) << (BASE - nBit);
        }
        if (negative)
        for (int i = 0; i < nBit; ++i)
            ret.pData[ret.nData - 1] |= (1 << (BASE - i - 1));
        ret.fix();
    }

    return ret;
}
```

6. Các hàm hỗ trợ

6.1. Hàm abs

```
BigInt abs(const BigInt &x) {
    BigInt ret;
    ret = x;
    if (isNegative(x))
        ret = -ret;
    return ret;
}
```

6.2. Hàm min, max

```
BigInt min(const BigInt &x, const BigInt &y) {
    BigInt temp_x, temp_y;
    temp_x = x; temp_y = y;
    if (temp_x > temp_y)
        swap(temp_x, temp_y);
    return temp_x;
}

BigInt max(const BigInt &x, const BigInt &y) {
    BigInt temp_x, temp_y;
    temp_x = x; temp_y = y;
    if (temp_x < temp_y)
        swap(temp_x, temp_y);
    return temp_x;
}
```

6.3. Hàm pow

Với 2 số `BigInt x` và `y`, ta chỉ chấp nhận tính toán được phép `x` lũy thừa `y` khi mà giá trị của `y` dương. Nếu `y` mang giá trị âm hoặc bằng 0, ta chấp nhận kết quả trả về là 1.

1. Tư tưởng thuật toán:

   Với `y` dương, ta luôn có thể biểu diễn `y` thành dạng như sau:

$$\begin{aligned} y &= 2^{k_1} + 2^{k_2} + \ldots + 2^{k_i} \\ \Rightarrow x^y &= x^{2^{k_1}+2^{k_2}+\ldots+2^{k_i}} \\ \Leftrightarrow x^y &= x^{2^{k_1}} \cdot x^{2^{k_2}} \cdot \ldots \cdot x^{2^{k_i}} \end{aligned}$$

   Vì vậy ta có thể đi tới một thuật toán tính `pow(x, y)` được mô tả như sau:

   - Khai báo `ret = 1, temp = x`
   - Thực hiện tính toán `y` ở dạng nhị phân, duyệt từ bit thấp nhất đến bit cao nhất của `y`, nếu gặp bit 1 thì thực hiện `ret *= temp`
   - Sau mỗi lần duyệt qua 1 bit thì thực hiện `temp *= temp`
   - Kết quả cuối cùng trả về giá trị của `ret`

2. Cài đặt hàm:

```
BigInt pow(const BigInt &x, const BigInt &y) {
    BigInt ret(1), temp;
    temp = x;
    if (isNegative(y)) return ret;
    for (int i = 0; i < y.nData; ++i) {
        for (int j = 0; j < BASE; ++j) {
            if (y.pData[i] >> j & 1)
                ret = ret * temp;
            temp = temp * temp;
            if (i == y.nData - 1
            && !(1ll * y.pData[y.nData - 1] & ~((1ll << j + 1) - 1)))
                break;
        }
    }
    return ret;
}
```

6.4. Hàm digits (số lượng ký tự số)

```
int digits(const BigInt &x, const int &base) {
    int ret = 0;
    BigInt temp;
    if (base == 2) temp = x;
    else temp = abs(x);
    if (base == 2)
        return BigIntToBase2(temp).length();
    return to_string(temp).length();
}
```

6.5. Các hàm chuyển đổi

6.5.1. Hàm `to_string`

Vốn dĩ hàm `to_string` được gọi là hàm chuyển đổi vì nó có chức năng tương tự như cách chúng ta cài một hàm tên là `to_base10`, tức là chuyển đổi `BigInt` thành dạng biểu diễn ở cơ số 10. Hàm được xây dựng như sau:

```
string to_string(const BigInt &x) {
    string ret = "";
    BigInt temp = abs(x);
    while (!isZero(temp)) {
        ret += (char)(to_int(temp % 10) + '0');
        temp /= 10;
    }
    if (isNegative(x))
        ret += '-';
    reverse(ret.begin(), ret.end());
    if (ret.empty())
        ret += '0';
    return ret;
}
```

6.5.2. Hàm `to_base32`

Với hàm này, nhóm sử dụng bảng chữ cái **RFC 4648** để chuyển đổi `BigInt` thành dạng biểu diễn cơ số 32:

The RFC 4648 Base 32 alphabet

Value	Symbol	Value	Symbol	Value	Symbol	Value	Symbol
0	A	8	I	16	Q	24	Y
1	B	9	J	17	R	25	Z
2	C	10	K	18	S	26	2
3	D	11	L	19	T	27	3
4	E	12	M	20	U	28	4
5	F	13	N	21	V	29	5
6	G	14	O	22	W	30	6
7	H	15	P	23	X	31	7
padding	=						

```
string to_base32(const BigInt &x) {
    string ret = "";
    BigInt temp = abs(x);
    while (!isZero(temp)) {
        int remainder = to_int(temp % 32);
        temp /= 32;
        ret += (char)(remainder + (remainder < 26 ? 'A' : '2' - 26));
    }
    if (isNegative(x))
        ret += '-';
    reverse(ret.begin(), ret.end());
    if (ret.empty())
        ret += '0';
    return ret;
}
```

6.5.3. Hàm `to_base58`

Với hàm này, nhóm sử dụng bảng chữ cái sau:

Byte	Character	Byte	Character	Byte	Character	Byte	Character
0	1	1	2	2	3	3	4
4	5	5	6	6	7	7	8
8	9	9	A	10	B	11	C
12	D	13	E	14	F	15	G
16	H	17	J	18	K	19	L
20	M	21	N	22	P	23	Q
24	R	25	S	26	T	27	U
28	V	29	W	30	X	31	Y
32	Z	33	a	34	b	35	c
36	d	37	e	38	f	39	g
40	h	41	i	42	j	43	k
44	m	45	n	46	o	47	p
48	q	49	r	50	s	51	t
52	u	53	v	54	w	55	x
56	y	57	z				

Base58 Mapping Table

Bởi vì bảng chữ cái này bỏ đi các kí tự: số 0, chữ I (hoa), chữ O (hoa), chữ l (thường) nên các dòng 8, 12, 14, 16 đã bỏ qua các kí tự này. Code của hàm này như sau:

```
1  string to_base58(const BigInt &x) {
2      string ret = "";
3      BigInt temp = abs(x);
4      while (!isZero(temp)) {
5          int remainder = to_int(temp % 58);
6          temp /= 58;
7          if (remainder < 9)
8              ret += (char)(remainder + '1');
9          else if (remainder < 16)
10             ret += (char)(remainder + 'A' - 9);
11         else if (remainder < 22)
12             ret += (char)(remainder + 'B' - 9);
13         else if (remainder < 33)
14             ret += (char)(remainder + 'C' - 9);
15         else if (remainder < 44)
16             ret += (char)(remainder + 'a' - 33);
17         else
18             ret += (char)(remainder + 'b' - 33);
19     }
20     if (isNegative(x))
21         ret += '-';
22     reverse(ret.begin(), ret.end());
23     if (ret.empty())
24         ret += '1';
25     return ret;
26 }
```

6.5.4. Hàm `to_base64`

Với hàm này, nhóm sử dụng bảng chữ cái sau:

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

```
string to_base64(const BigInt &x) {
    string ret = "";
    BigInt temp = abs(x);
    while (!isZero(temp)) {
        int remainder = to_int(temp % 64);
        temp /= 64;
        if (remainder < 26)
            ret += (char)(remainder + 'A');
        else if (remainder < 52)
            ret += (char)(remainder + 'a' - 26);
        else if (remainder < 62)
            ret += (char)(remainder + '0' - 52);
        else if (remainder == 62)
            ret += '+';
        else
            ret += '\\';
    }
    if (isNegative(x))
        ret += '-';
    reverse(ret.begin(), ret.end());
    if (ret.empty())
        ret += '0';
    return ret;
}
```

6.6. Hàm `is_prime`

1. Tư tưởng thuật toán:

   Với dữ liệu số nguyên lớn, nhóm sử dụng phương pháp kiểm tra Miller-Rabin để kiểm tra một số `BigInt` có phải là số nguyên tố hay không. Với cách cài đặt này thì vẫn có thể tồn tại sai số, nhưng cách cài đặt đảm bảo cho sai số là nhỏ nhất. Trước tiên, giải thuật kiểm tra Miller-Rabin được mô tả như sau:

   - Bước 1: Phân tích $n - 1 = 2^s \cdot m$ trong đó $s \geq 1$ và m là số tự nhiên lẻ
   - Bước 2: Chọn ngẫu nhiên số tự nhiên $a \in \{2, 3, ..., n - 1\}$
   - Bước 3: Đặt $b = a^m \ (mod\ n)$
   - Bước 4: Nếu $b \equiv 1 \ (mod\ n)$ thì trả về TRUE. Kết thúc
   - Bước 5: Cho k chạy từ 0 đến s:
     (a) Nếu $b \equiv -1 \ (mod\ n)$ thì trả về TRUE. Kết thúc.
     (b) Thay $b := b^2 \ (mod\ n)$
   - Bước 6: Trả về FALSE. Kết thúc.

2. Cài đặt:

   (a) Ta xây dựng hàm `powMod`. Kết quả là số dư của phép x lũy thừa y sau khi chia cho số mod.

```
BigInt powMod(const BigInt &x, const BigInt &y, const BigInt &mod) {
    BigInt ret(1), temp;
    temp = x;
    if (isNegative(y)) return ret;
    for (int i = 0; i < y.nData; ++i) {
        for (int j = 0; j < BASE; ++j) {
            if (y.pData[i] >> j & 1) {
                ret = ret * temp % mod;
            }
            temp = temp * temp % mod;
            if (i == y.nData - 1
            && !(1ll * y.pData[y.nData - 1] & ~((1ll << j + 1) - 1))
    )
                break;
        }
    }
    return ret;
```

   (b) Ta xây dựng hàm `Rand`, mục đích của hàm này là lấy ra ngẫu nhiên một số `BigInt` có giá trị nằm trong khoảng từ `2` đến `x - 1`

```
BigInt Rand(const BigInt &x) {
    BigInt ret;
    ret = x;
    if (isNegative(ret)) return ret;
    for (int i = ret.nData - 1; i >= 0; --i) {
        if (rand() % 2) {
            ret.pData[i] = rand() % x.pData[i];
            for (int j = i - 1; j >= 0; --j)
                ret.pData[j] = rand() % (1ll * UINT16_MAX + 1);
            break;
        }
        ret.pData[i] = x.pData[i];
    }
```

```
    ret.fix();
    if (ret < BigInt(2))
        ret = BigInt(2);
    return ret;
}
```

(c) Hàm `check` được cài đặt để kiểm tra `y` có phải là số nguyên tố hay không với một giá trị ngẫu nhiên của `temp`. Trong đó, `x, y, temp` và `step` lần lượt có vai trò tương đương với m, n, a và s trong phần mô tả thuật toán bên trên.

```
bool check(const BigInt &x, const BigInt &y, int step) {
    BigInt B1(1);
    BigInt temp = Rand(y - BigInt(2)), y1 = y - B1;
    temp = powMod(temp, x, y);
    if (temp == B1 || temp == y1)
        return true;
    for (int i = 0; i < step; ++i) {
        temp = temp * temp % y;
        if (temp == B1)
            return false;
        if (temp == y1)
            return true;
    }
    return false;
}
```

(d) Cài đặt hàm `is_prime`: Giá trị `nTest` là số lần ta thực hiện kiểm tra `x` nhờ hàm `check`.

```
bool is_prime(const BigInt &x) {
    BigInt B1(1), B4(4);
    if (x <= B1) return false;
    if (x < B4) return true;
    if (x.pData[0] % 2 == 0) return false;

    BigInt temp = x - B1;
    int step = 0;
    while (temp.pData[0] % 2 == 0) {
        temp /= 2;
        step++;
    }

    const int nTest = 10;
    for (int i = 0; i < nTest; ++i)
        if (!check(temp, x, step))
            return false;

    return true;
}
```

7. Cài đặt chương trình thực thi đọc tham số dòng lệnh dạng command line

7.1. Các hàm phân loại kiểu truy vấn

Chương trình của chúng ta gồm các chức năng cơ bản sau:

- Chuyển đổi số `BigInt` từ hệ nhị phân sang hệ thập phân
- Chuyển đổi số `BigInt` từ hệ thập phân sang hệ nhị phân
- Tính toán biểu thức số học với 2 số `BigInt`
- Các hàm `abs, min, max, pow` với 2 số `BigInt`
- Các hàm `digits, to_string, to_base32, to_base58, to_base64, is_prime` với 1 số `BigInt`

Ta cài đặt hàm `RunBigIntCommand` với mục đích là để phân loại truy vấn đưa vào thành một trong những chức năng cụ thể được liệt kê. Trong đó, `base` mang giá trị là 0 (đại diện cho base 10) hoặc 1 (đại diện cho base 2).

```
1  void RunBigIntCommand(ostream &os, string cmd) {
2      stringstream ss(cmd);
3      string buffer;
4
5      // get base
6      bool base;
7      ss >> buffer;
8      base = (buffer == "2");
9
10     if (isConvert(cmd)) {
11         ss >> buffer >> buffer;
12         TConvert(os, buffer, base);
13     } else if (is2Operator(cmd)) {
14         T2Operator(os, cmd, base);
15     } else {
16         string type;
17         ss >> type;
18         if (type == "~") TNot(os, cmd, base);
19         else if (type.substr(0, 3) == "abs") TAbs(os, cmd, base);
20         else if (type.substr(0, 3) == "min") TMin(os, cmd, base);
21         else if (type.substr(0, 3) == "max") TMax(os, cmd, base);
22         else if (type.substr(0, 3) == "pow") TPow(os, cmd, base);
23         else if (type[0] == 'd') TDigits(os, cmd, base);
24         else if (type.substr(0, 8) == "is_prime") TIsPrime(os, cmd, base);
25         else if (type.substr(0, 9) == "to_string") TToStr(os, cmd, base);
26         else if (type.substr(0, 9) == "to_base32") TToBase32(os, cmd, base);
27         else if (type.substr(0, 9) == "to_base58") TToBase58(os, cmd, base);
28         else if (type.substr(0, 9) == "to_base64") TToBase64(os, cmd, base);
29     }
30 }
```

Đoạn code trên sử dụng các hàm hỗ trợ việc kiểm tra truy vấn gồm:

- Hàm `isConvert`:
  Hàm này kiểm tra xem truy vấn đưa vào có phải là một lệnh chuyển đổi hệ cơ số hay không và được cài đặt như sau:

```
bool isConvert(const string &cmd) {
    stringstream ss(cmd);
    string pre = "", cur = "";
    int count = 0;
    while (ss >> cur) {
        count++;
        if (count > 3)
            return false;
        if (count == 1)
            pre = cur;
        if (count == 2 && !(pre == "2" && cur == "10")
        && !(pre == "10" && cur == "2"))
            return false;
    }
    return count == 3;
}
```

- Hàm `is2Operator`:
  Hàm này kiểm tra xem truy vấn đưa vào có phải là một lệnh tính toán số học giữa 2 số `BigInt` hay không.

```
bool is2Operator(const string &cmd) {
    stringstream ss(cmd);
    string ope;
    ss >> ope >> ope >> ope;
    if (ope == "+") return true;
    if (ope == "-") return true;
    if (ope == "*") return true;
    if (ope == "/") return true;
    if (ope == "%") return true;
    if (ope == "&") return true;
    if (ope == "|") return true;
    if (ope == "^") return true;
    if (ope == "<<") return true;
    if (ope == ">>") return true;
    return false;
}
```

7.2. Các hàm thực thi chức năng theo truy vấn

Sau khi phân loại ra các chức năng dựa trên truy vấn đưa vào, ta tiếp tục cài đặt các hàm để thực hiện các chức năng đó. Lưu ý rằng giá trị của `base` đã được định nghĩa bên trên. Cụ thể bao gồm các hàm:

1. Hàm `TConvert`: Hàm này thực hiện chức năng chuyển qua lại hệ thập phân và hệ nhị phân của số `BigInt`.

```
void TConvert(ostream &os, string cmd, bool base) {
    BigInt x;
    if (!base) {
        x = Base10ToBigInt(cmd);
        writeBinary(os, x);
    } else {
        x = Base2ToBigInt(cmd);
        writeDecimal(os, x);
    }
}
```

2. Hàm `T2Operator`: Hàm này thực hiện chức năng tính toán số học giữa 2 số `BigInt`.

```
void T2Operator(ostream &os, string cmd, bool base) {
    stringstream ss(cmd);
    string sx, sy, ope;
    ss >> sx >> sx >> ope >> sy;
    BigInt x, y, res;
    int ty;
    if (!base) {
        x = Base10ToBigInt(sx);
        if (ope == "<<" || ope == ">>")
            ty = Str2IntBase10(sy);
        else
            y = Base10ToBigInt(sy);
    } else {
        x = Base2ToBigInt(sx);
        if (ope == "<<" || ope == ">>")
            ty = Str2IntBase2(sy);
        else
            y = Base2ToBigInt(sy);
    }
    if (ope == "+") res = x + y;
    if (ope == "-") res = x - y;
    if (ope == "*") res = x * y;
    if (ope == "/") res = x / y;
    if (ope == "%") res = x % y;
    if (ope == "&") res = x & y;
    if (ope == "|") res = x | y;
    if (ope == "^") res = x ^ y;
    if (ope == "<<") res = x << ty;
    if (ope == ">>") res = x >> ty;
    if (!base)
        writeDecimal(os, res);
    else {
        string temp = BigIntToBase2(res);
        os << temp << endl;
    }
}
```

Trong đoạn code trên ta sử dụng các hàm `Str2IntBase10` và `Str2IntBase2` dùng để đổi chuỗi thành giá trị số được cài đặt như sau:

- Hệ thập phân:

```
int Str2IntBase10(const string &str) {
    int ret = 0;
    for (int i = 0; i < str.length(); ++i)
        ret = ret * 10 + str[i] - '0';
    return ret;
}
```

- Hệ nhị phân:

```
int Str2IntBase2(const string &str) {
    int ret = 0;
    for (int i = 0; i < str.length(); ++i)
        ret = (ret << 1) | (str[i] - '0');
    return ret;
}
```

3. Hàm `TNot`: Hàm thực hiện chức năng tính NOT số `BigInt`.

```
void TNot(ostream &os, string cmd, bool base) {
    stringstream ss(cmd);
    string sx;
    ss >> sx >> sx >> sx;
    BigInt x;
    if (!base) {
        x = Base10ToBigInt(sx);
        x = ~x;
        writeDecimal(os, x);
    }
    else {
        x = Base2ToBigInt(sx);
        x = ~x;
        writeBinary(os, x);
    }
```

(*) Trước khi đi vào các hàm phía sau, để tiện cho việc cài đặt, ta giới thiệu 2 hàm `get1Operand` như sau để lấy dữ liệu bên trong dấu ngoặc đơn đối với dữ liệu được đưa vào trong trường hợp xử lí các hàm hỗ trợ.

- Hàm `get1Operand`:

```
void get1Operand(string cmd, string &x, int length) {
    stringstream ss(cmd);
    ss >> x >> x;
    x.erase(0, length + 1); x.pop_back();
}
```

- Hàm `get2Operand`:

```
void get2Operand(string cmd, string &x, string &y, int length) {
    stringstream ss(cmd);
    ss >> x >> x >> y;
    x.erase(0, length + 1); x.pop_back();
    y.pop_back();
}
```

4. Hàm `TAbs`: Hàm tính giá trị tuyệt đối của `BigInt`.

```
void TAbs(ostream &os, string cmd, bool base) {
    string sx;
    get1Operand(cmd, sx, 3);
    BigInt x;
    if (!base) {
        x = Base10ToBigInt(sx);
        x = abs(x);
        writeDecimal(os, x);
    } else {
        x = Base2ToBigInt(sx);
        x = abs(x);
        writeBinary(os, x);
    }
}
```

5. Hàm `TMin`, `TMax`: Hàm tìm giá trị nhỏ nhất, giá trị lớn nhất trong 2 số `BigInt`.

```
void TMin(ostream &os, string cmd, bool base) {
    string sx, sy;
    get2Operand(cmd, sx, sy, 3);
    BigInt x, y;
    if (!base) {
        x = Base10ToBigInt(sx);
        y = Base10ToBigInt(sy);
        writeDecimal(os, min(x, y));
    } else {
        x = Base2ToBigInt(sx);
        y = Base2ToBigInt(sy);
        writeBinary(os, min(x, y));
    }
}

void TMax(ostream &os, string cmd, bool base) {
    string sx, sy;
    get2Operand(cmd, sx, sy, 3);
    BigInt x, y;
    if (!base) {
        x = Base10ToBigInt(sx);
        y = Base10ToBigInt(sy);
        writeDecimal(os, max(x, y));
    } else {
        x = Base2ToBigInt(sx);
        y = Base2ToBigInt(sy);
        writeBinary(os, max(x, y));
    }
}
```

6. Hàm `TPow`: Hàm tính phép lũy thừa `BigInt`.

```
void TPow(ostream &os, string cmd, bool base) {
    string sx, sy;
    get2Operand(cmd, sx, sy, 3);
    BigInt x, y;
    if (!base) {
        x = Base10ToBigInt(sx);
        y = Base10ToBigInt(sy);
        writeDecimal(os, pow(x, y));
    } else {
        x = Base2ToBigInt(sx);
        y = Base2ToBigInt(sy);
        writeBinary(os, pow(x, y));
    }
}
```

7. Hàm `TDigits`: Hàm tìm số ký tự số của 1 số `BigInt`.

```
void TDigits(ostream &os, string cmd, bool base) {
    string sx;
    get1Operand(cmd, sx, 6);
    BigInt x;
    if (!base)
        x = Base10ToBigInt(sx);
    else
        x = Base2ToBigInt(sx);
    os << digits(x, !base ? 10 : 2) << endl;
}
```

8. Hàm `TToStr`: Hàm chuyển đổi từ `BigInt` sang chuỗi ký tự (hoặc base 10).

```
void TToStr(ostream &os, string cmd, bool base) {
    string sx;
    get1Operand(cmd, sx, 9);
    BigInt x;
    if (!base)
        x = Base10ToBigInt(sx);
    else
        x = Base2ToBigInt(sx);
    os << to_string(x) << endl;
}
```

9. Hàm `TToBase32`: Hàm chuyển đổi từ `BigInt` sang base 32.

```
void TToBase32(ostream &os, string cmd, bool base) {
    string sx;
    get1Operand(cmd, sx, 9);
    BigInt x;
    if (!base)
        x = Base10ToBigInt(sx);
    else
        x = Base2ToBigInt(sx);
    os << to_base32(x) << endl;
}
```

10. Hàm `TToBase58`: Hàm chuyển đổi từ `BigInt` sang base 58.

```
void TToBase58(ostream &os, string cmd, bool base) {
    string sx;
    get1Operand(cmd, sx, 9);
    BigInt x;
    if (!base)
        x = Base10ToBigInt(sx);
    else
        x = Base2ToBigInt(sx);
    os << to_base58(x) << endl;
}
```

11. Hàm `TToBase64`: Hàm chuyển đổi từ `BigInt` sang base 64.

```
void TToBase64(ostream &os, string cmd, bool base) {
    string sx;
    get1Operand(cmd, sx, 9);
    BigInt x;
    if (!base)
        x = Base10ToBigInt(sx);
    else
        x = Base2ToBigInt(sx);
    os << to_base64(x) << endl;
}
```

12. Hàm `TIsPrime`: Hàm kiểm tra số `BigInt` có phải số nguyên tố hay không.

```
void TIsPrime(ostream &os, string cmd, bool base) {
    string sx;
    get1Operand(cmd, sx, 8);
    BigInt x;
    if (!base)
        x = Base10ToBigInt(sx);
    else
```

```
        x = Base2ToBigInt(sx);
    os << (is_prime(x) ? "true" : "false") << endl;
}
```

7.3. Đánh giá tốc độ, bộ nhớ chương trình sử dụng

- Mô tả: Xác định bộ nhớ đã sử dụng và thời gian từ đầu chương trình đến lệnh hiện tại.
- Ý tưởng:
  - Đánh giá tốc độ: Ta sử dụng hàm `clock()` để lấy thời gian trước khi chạy và sau khi chạy chương trình. Lấy hiệu hai thời gian đó ta được thời gian chạy của chương trình.
  - Đánh giá bộ nhớ sử dụng:
    + Trước tiên là khai báo các thư viện cần sử dụng:

      ```
      #include<windows.h>
      #include<Psapi.h>
      ```

    + Sau đó ta sử dụng các lệnh sau để có thể lấy được bộ nhớ trước khi chương trình thực hiện các truy vấn.

      ```
      auto myHandle = GetCurrentProcess();
      PROCESS_MEMORY_COUNTERS pmc;
      GetProcessMemoryInfo(myHandle, &pmc, sizeof(pmc));
      size_t mem_1 = pmc.PeakWorkingSetSize;
      ```

    + Nếu có truy vấn về thời gian và bộ nhớ đã sử dụng ta thực hiện thao tác `GetProcessMemoryInfo(myHandle, &pmc, sizeof(pmc))` rồi lấy `pmc.PeakWorkingSetSize` trừ đi `mem_1`, ta thu được kết quả là lượng bộ nhớ chương trình đã sử dụng cho đến truy vấn đó.
- Cài đặt:

  Dựa vào cơ sở phương pháp trên, ta cài đặt hàm `RunCommand` như sau:

```
void RunCommand(istream &is, ostream &os) {
    double time_1 = clock();
    auto myHandle = GetCurrentProcess();
    PROCESS_MEMORY_COUNTERS pmc;
    GetProcessMemoryInfo(myHandle, &pmc, sizeof(pmc));
    size_t mem_1 = pmc.PeakWorkingSetSize;

    string cmd;

    while (getline(is, cmd)) {
        if (cmd == "get_info") {
            GetProcessMemoryInfo(myHandle, &pmc, sizeof(pmc));
            os << "Excution time: "
               << (clock() - time_1) / CLOCKS_PER_SEC
               << "s\n";
            os << "Memory usage: "
               << pmc.PeakWorkingSetSize - mem_1
               << " Bytes\n";
        } else {
            RunBigIntCommand(os, cmd);
        }
    }
}
```

8. Tài liệu tham khảo

- StackOverFlow
- GeeksForGeeks
- Miller-Rabin | Wiki