**Trường Đại học Khoa học Tự nhiên – ĐHQG TPHCM**

**Khoa Công nghệ Thông tin**



**BÁO CÁO ĐỒ ÁN GIỮA KỲ**

**XỬ LÝ SỐ NGUYÊN LỚN**

**Giảng viên hướng dẫn:** TS. Võ Hoài Việt

**Nhóm sinh viên thực hiện:**

Nhan Gia Khâm – MSSV: 20120504

Hoàng Trọng Vũ – MSSV: 20120025

**Lớp:** CNTN2020

Thành phố Hồ Chí Minh – Tháng 5 năm 2021

**MỤC LỤC**

[I. Giới thiệu đồ án 4](#_Toc73023833)

[I.1. Nội dung đồ án 4](#_Toc73023834)

[I.2. Yêu cầu đồ án 4](#_Toc73023835)

[II. Ý tưởng thực hiện 5](#_Toc73023836)

[II.1. Cách lưu trữ 5](#_Toc73023837)

[II.2. Đọc dữ liệu 6](#_Toc73023838)

[II.3. Chuyển đổi hệ cơ số 7](#_Toc73023839)

[II.3.1. BigInt sang hệ nhị phân 7](#_Toc73023840)

[II.3.2. BigInt sang hệ thập phân 7](#_Toc73023841)

[II.3.2. Hệ thập phân sang hệ nhị phân và ngược lại 8](#_Toc73023842)

[II.3.3. BigInt sang hệ cơ số 32 8](#_Toc73023843)

[II.3.4. BigInt sang hệ cơ số 58 9](#_Toc73023844)

[II.3.5. BigInt sang hệ cơ số 64 10](#_Toc73023845)

[II.4. Xuất dữ liệu 11](#_Toc73023846)

[II.5. Toán tử gán 12](#_Toc73023847)

[II.6. Toán tử quan hệ 12](#_Toc73023848)

[II.6.1. Toán tử bằng, toán tử khác 12](#_Toc73023849)

[II.6.2. Toán tử bé hơn, toán tử lớn hơn hoặc hằng 12](#_Toc73023850)

[II.6.3. Toán tử lớn hơn, toán tử bé hơn hoặc bằng 13](#_Toc73023851)

[II.7. Toán tử số học 13](#_Toc73023852)

[II.7.1. Các trường hợp đặc biệt 13](#_Toc73023853)

[II.7.2. Toán tử cộng 14](#_Toc73023854)

[II.7.3. Toán tử trừ 14](#_Toc73023855)

[II.7.4. Toán tử nhân 15](#_Toc73023856)

[II.7.5. Toán tử chia lấy dư 15](#_Toc73023857)

[II.8. Toán tử thao tác bit 15](#_Toc73023858)

[II.8.1. Toán tử AND, OR, XOR 15](#_Toc73023859)

[II.8.2. Toán tử NOT 16](#_Toc73023860)

[II.8.3. Toán tử ShiftLeft 17](#_Toc73023861)

[II.8.4. Toán tử ShiftRight 17](#_Toc73023862)

[II.9. Các hàm hỗ trợ 17](#_Toc73023863)

[II.8.1. Hàm Abs 17](#_Toc73023864)

[II.8.2. Hàm Max, Min 18](#_Toc73023865)

[II.8.3. Hàm Digits 18](#_Toc73023866)

[II.8.4. Hàm Pow 18](#_Toc73023867)

[II.8.3. Hàm ToString 18](#_Toc73023868)

[II.8.4. Hàm IsPrime 18](#_Toc73023869)

[II.10. Tính thời gian xử lý và bộ nhớ đã dùng 19](#_Toc73023870)

[II.10.1. Tính thời gian xử lý 19](#_Toc73023871)

[II.10.2. Tính bộ nhớ đã dùng 20](#_Toc73023872)

[III. Cài đặt đồ án 21](#_Toc73023873)

[III.1. Môi trường lập trình 21](#_Toc73023874)

[III.2. Dữ liệu đầu vào, dữ liệu đầu ra 21](#_Toc73023875)

[III.3. Thực thi chương trình 21](#_Toc73023876)

[IV. Tài liệu tham khảo 23](#_Toc73023877)

# I. Giới thiệu đồ án

## I.1. Nội dung đồ án

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:

* Chuyển đổi số **BigInt** từ hệ thập phân sang hệ nhị phân và ngược lại.
* Các operator, operator, operator, operator, operator.
* Các toán tử AND “”, OR “”, XOR “”, NOT “”.
* Các toán tử shiftleft “”, shiftright “”.
* Chuyển đổi hệ cơ số: ToBase32, ToBase58, ToBase64.
* Các hàm hỗ trợ: Abs, Min, Max, Pow, Digits (số lượng ký tự số), ToString, IsPrime.

## I.2. Yêu cầu đồ án

Các yêu cầu được đặt ra như sau:

* Dùng kỹ thuật con trỏ và xử lý chuỗi để biễu diễn cấu trúc dữ liệu trên sao cho việc sử dụng bộ nhớ là tối ưu nhất.
* Đề xuất các thuật toán để thực thi các thao tác và hàm liên qua là tối ưu nhất về tốc độ và bộ nhớ.
* Chương trình được chấm tự động, chương trình thực thi đọc tham số dòng lệnh ở dạng command line:

*<MSSV1\_MSSV2.exe> <input.txt> <output.txt>*

* Cho phép xuất ra thời gian xử lý và dung lượng bộ nhớ đã dùng trong mỗi thao tác hoặc hàm.

# II. Ý tưởng thực hiện

Phần này trình bày ý tưởng của nhóm về cách lưu trữ dữ liệu, thuật toán đọc, xuất, chuyển đổi hệ cơ số, các phép toán thao tác bit và phép toán số học, các hàm hỗ trợ.

## II.1. Cách lưu trữ

Sử dụng struct **BigInt** để lưu trữ kiểu dữ liệu số nguyên lớn, với các thành phần như sau:

#define BLOCK uint16\_t

struct BigInt {

int8\_t sign;

int nBit;

int rBit;

int nBlock;

BLOCK\* data = NULL;

};

trong đó:

* *sign* bằng 1 nếu số **BigInt** không âm, bằng –1 nếu số **BigInt** âm.
* *nBit* là số bit vừa đủ để lưu biểu diễn nhị phân của ***giá trị tuyệt đối*** của số **BigInt.**
* *rBit* là số bit thực sự trong biểu diễn nhị phân của số **BigInt**.

***Lưu ý***:

* +  nếu số **BigInt** nhỏ hơn 128 bit.
  + Nếu số **BigInt** lớn hơn hoặc bằng 128 bit:
    -  nếu .
    -  nếu .
* *nBlock* là số block 2 bytes vừa đủ để lưu trữ *nBit* bits.
* *data* là con trỏ đến vùng nhớ có *nBlock* phần tử, mỗi phần tử có kích thước 2 bytes, lưu biểu diễn ***giá trị tuyệt đối*** của số **BigInt**trong hệ 216.

Nói cách khác, vùng nhớ do *data* trỏ đến lưu một dãy bit liên tục với độ dài *nBlock \* 16* bit, lưu trữ các bit trong biểu diễn nhị phân của ***giá trị tuyệt đối*** của số ***BigInt*** theo thứ tự như sau:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Phần tử | *data[0]* | | | | | | | | | | | | | | | | *data[1]…* | | |
| Lưu bit thứ | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 17 | 16 | … |

(*data*[*i*] lưu các bit thứ  đến )

***Các ví dụ minh họa:***

***Ví dụ 1:*** Lưu **“26375”** vào số **BigInt**.

Ta có **27375 = 1100111 000001112**

* *sign* = 1
* *rBit* = 128
* *nBit* = 15
* *nBlock* = 1
* *data*:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Phần tử | *data*[0] | | | | | | | | | | | | | | | |
| Dãy bit lưu | 0 | **1** | **1** | **0** | **0** | **1** | **1** | **1** | **0** | **0** | **0** | **0** | **0** | **1** | **1** | **1** |

***Ví dụ 2:*** Lưu **“–5”** vào số **BigInt**.

Ta có **5 = 1012**

Biểu diễn nhị phân của **-5** với 128 bits theo nguyên tắc bù 2 là

**11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111011**

* *sign* = –1
* *rBit* = 128
* *nBit* = 3
* *nBlock* = 1
* *data*:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Phần tử | *data*[0] | | | | | | | | | | | | | | | |
| Dãy bit lưu | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | **0** | **1** |

## II.2. Đọc dữ liệu

Đọc vào dữ liệu dưới dạng *chuỗi* theo các hệ cơ số tương ứng. Ý tưởng thuật toán như sau:

* ***Hệ nhị phân:***
  + Nếu chuỗi bắt đầu bằng ‘0’ thì *sign* = 1, bắt đầu bằng ‘1’ thì *sign* = –1.
  + Tìm vị trí kí tự ‘1’ đầu tiên của chuỗi (*start*).
  + Xác định *rBit*, *nBit* và *nBlock*. Cấp phát động vùng nhớ có kích thước  
    *nBlock* \* 2 bytes cho *data* quản lý.
  + Duyệt xuôi chuỗi từ vị trí *start* theo *i*, gán bit thứ *i – start* của dãy bit liên tục được lưu trong mảng *data* bằng kí tự thứ *i* trong chuỗi.
  + Nếu *sign* = –1, tức dãy bit ta vừa lưu vào mảng *data* là dãy bit biểu diễn số âm:
    - Tìm *biểu diễu bù 2* của dãy bit này với *rBit* bits
    - Cho *data* trỏ đến vùng nhớ mới chứa cách biểu diễn này, cập nhật lại *nBit, nBlock.*
* ***Hệ thập phân:***
  + *Số dương*:
    - Lần lượt chia chuỗi số cho 216 theo quy tắc tương tự như chia đại số, đưa các giá trị số dư vào mảng *data* và tăng *rBit*, *nBit, nBlock* một lượng phù hợp, gán *sign* = 1. Quá trình dừng khi chuỗi số chỉ còn toàn kí tự ‘0’.
  + *Số âm*:
    - Đọc phần giá trị tuyệt đối và lưu vào số **BigInt** với thuật toán như trên, gán *sign* = –1.

## II.3. Chuyển đổi hệ cơ số

Chuyển đổi số **BigInt** sang các hệ cơ số 32, 58, 64; chuyển đổi giữa hệ thập phân và nhị phân. Kết quả trả về của các hàm là một *chuỗi* chứa các kí tự biểu diễn.

### II.3.1. BigInt sang hệ nhị phân

Vì ta đang lưu dãy bit biểu diễn *giá trị tuyệt đối* của số **BigInt** trong mảng *data* nên khi chuyển số **BigInt** sang hệ nhị phân, ta phải tìm *biểu diễn bù 2* của dãy bit đó với *rBit* bit nếu *sign* của nó là –1.

Ý tưởng toán như sau:

* Nếu *sign* = –1:
  + Tìm biểu diễn bù 2 của số **BigInt** với *rBit* bit, lưu cách biểu diễn này vào một số **BigInt** mới.
  + Duyệt ngược theo *rBit* của số **BigInt** vừa thu được*,* lấy ra các bit lưu trong mảng *data* từ cao tới thấp và đưa vào chuỗi kết quả.
* Nếu *sign* = 1:
  + Duyệt ngược theo *nBit*, lấy ra các bit lưu trong mảng *data* từ cao tới thấp và đưa vào chuỗi kết quả.

### II.3.2. BigInt sang hệ thập phân

Ý tưởng thuật toán như sau:

* Lưu dấu của số **BigInt** vào một biến. Xem số **BigInt** là số dương.
* Lần lượt chia số **BigInt** cho 10 theo quy tắc tương tự như chia đại số, đưa các giá trị số dư vào chuỗi kết quả. Quá trình dừng khi số **BigInt** bằng 0.
* Nếu dấu đã lưu là –1 thì thêm kí tự ‘-’ vào cuối chuỗi kết quả.
* Đảo ngược chuỗi kết quả.

### II.3.2. Hệ thập phân sang hệ nhị phân và ngược lại

Hàm chuyển đổi số ở hệ thập phân sang hệ nhị phân và ngược lại có tham số là một *chuỗi*. Ý tưởng toán như sau:

* ***Hệ thập phân sang hệ nhị phân:***
  + Đọc chuỗi số ở hệ thập phân vào một số **BigInt**.
  + Chuyển số **BigInt** sang hệ nhị phân.
* ***Hệ nhị phân sang hệ thập phân:***
  + Đọc chuỗi ở hệ nhị phân vào một số **BigInt.**
  + Chuyển số **BigInt** sang hệ thập phân.

### II.3.3. BigInt sang hệ cơ số 32

Sử dụng bảng chữ cái *base32hex*. Ý tưởng thuật toán như sau:

* Nếu số **BigInt** là số âm, tìm biểu diễn bù 2 của dãy bit đang lưu trong mảng *data* với *rBit* bits.
* Duyệt từ bit thứ *rBit* – 1 về bit 0, lấy từng cụm 5 bits để đổi sang hệ cơ số 32, đưa vào chuỗi kết quả.
  + Nếu cụm cuối cùng không đủ 5 bits, thêm một vài bit 0 vào cuối cho đủ để đổi sang hệ cơ số 32.
* Số kí tự padding được tính bằng *pad* = 8 – (*rBit* % 40 + 4) / 5 và thêm *pad* kí tự ‘=’ vào chuỗi kết quả (để chuỗi kết quả có độ dài chia hết cho 8).

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

*Bảng chữ cái base32hex*

### II.3.4. BigInt sang hệ cơ số 58

***Lưu ý.*** Ta chỉ xét trường hợp đổi số **BigInt** ***không âm*** sang hệ cơ số 58.

Sử dụng bảng chữ cái Base58 chuẩn. Ý tưởng thuật toán như sau:

* Lần lượt chia số **BigInt** cho 58 theo quy tắc tương tự như chia đại số, từ số dư thu được, ta lấy ra chữ cái tương ứng trong bảng chữ cái Base58 và đưa vào chuỗi kết quả. Quá trình dừng khi số **BigInt** bằng 0.
* Đảo ngược chuỗi kết quả.

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

*Bảng chữ cái Base58*

### II.3.5. BigInt sang hệ cơ số 64

Sử dụng bảng chữ cái Base64 chuẩn. Ý tưởng thuật toán như sau:

* Nếu số **BigInt** là số âm, tìm biểu diễn bù 2 của dãy bit đang lưu trong mảng *data* với *rBit* bits.
* Duyệt từ bit thứ *rBit* – 1 về bit 0, lấy từng cụm 6 bits để đổi sang hệ cơ số 64, đưa vào chuỗi kết quả.
  + Nếu cụm cuối cùng không đủ 6 bits, thêm một vài bit 0 vào cuối cho đủ để đổi sang hệ cơ số 64.
* Số kí tự padding được tính bằng *pad* = 4 – (*rBit* % 24 + 5) / 6 và thêm *pad* kí tự ‘=’ vào chuỗi kết quả (để chuỗi kết quả có độ dài chia hết cho 4).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Value | Symbol |  | Value | Symbol |  | Value | Symbol |  | Value | Symbol |  | Value | Symbol |
| 0 | A | 13 | N | 26 | a | 39 | n | 52 | 0 |
| 1 | B | 14 | O | 27 | b | 40 | o | 53 | 1 |
| 2 | C | 15 | P | 28 | c | 41 | p | 54 | 2 |
| 3 | D | 16 | Q | 29 | d | 42 | q | 55 | 3 |
| 4 | E | 17 | R | 30 | e | 43 | r | 56 | 4 |
| 5 | F | 18 | S | 31 | f | 44 | s | 57 | 5 |
| 6 | G | 19 | T | 32 | g | 45 | t | 58 | 6 |
| 7 | H | 20 | U | 33 | h | 46 | u | 59 | 7 |
| 8 | I | 21 | V | 34 | i | 47 | v | 60 | 8 |
| 9 | J | 22 | W | 35 | j | 48 | w | 61 | 9 |
| 10 | K | 23 | X | 36 | k | 49 | x | 62 | + |
| 11 | L | 24 | Y | 37 | l | 50 | y | 63 | / |
| 12 | M | 25 | Z | 38 | m | 51 | z | *pad* | = |

*Bảng chữ cái Base64*

## II.4. Xuất dữ liệu

Xuất dữ liệu dưới dạng chuỗi cùng với hệ cơ số tương ứng:

* ***Hệ thập phân:***
  + Chuyển số **BigInt** sang hệ thập phân.
  + Xuất chuỗi.
* ***Hệ nhị phân:***
  + Chuyển số **BigInt** sang hệ nhị phân.
  + Xuất chuỗi.

## II.5. Toán tử gán

Nhóm đã xây dựng hàm *BigIntCpy*, tượng trưng cho operator= trong yêu cầu của đồ án. Mục đích của việc này là để thuận tiện hơn trong quá trình thực hiện việc gán số **BigInt** này bằng số **BigInt** khác theo từng mục đích, cụ thể:

* Khi muốn con trỏ *data* của số **BigInt** này trỏ đến cùng vùng nhớ với con trỏ *data* của số **BigInt** khác, ta dùng operator= có sẵn của kiểu dữ liệu *struct*.
* Khi muốn con trỏ *data* của số một **BigInt** trỏ đến vùng nhớ mới và sao chép nội dung từ vùng nhớ mà con trỏ *data* của số **BigInt** khác đến, ta dùng hàm *BigIntCpy.*

Các thành phần *sign, nBit, rBit, nBlock* sẽ được gán bằng nhau trong cả hai trường hợp trên.

## II.6. Toán tử quan hệ

Các phép so sánh lớn hơn, lớn hoặc hoặc bằng, bé hơn, bé hơn hoặc bằng, bằng, khác.

Giả sử hai toán hạng là .

### II.6.1. Toán tử bằng, toán tử khác

Ý tưởng thuật toán như sau:

* ***Toán tử bằng:***
  + Nếu  và : Trả về true.
  + Ngược lại: Trả về false.
* ***Toán tử khác:***
  + Nếu : Trả về false.
  + Ngược lại: Trả về true.

### II.6.2. Toán tử bé hơn, toán tử lớn hơn hoặc hằng

Ý tưởng thuật toán như sau:

* ***Toán tử bé hơn:***
  + Lần lượt xét các trường hợp về *sign*, *nBlock* và *data*[*i*] của .
  + Nếu :
    - Nếu : Trả về true.
    - Nếu : Trả về false.
  + Nếu  và :
    - Nếu : Trả về true nếu  và ngược lại.
    - Nếu : Trả về true nếu  và ngược lại.
  + Nếu  và :
    - Nếu : Trả về false.
    - Ngược lại: Trả về true.
  + Trả về false.
* ***Toán tử lớn hơn hoặc bằng:***
  + Nếu : Trả về false.
  + Ngược lại: Trả về true.

### II.6.3. Toán tử lớn hơn, toán tử bé hơn hoặc bằng

Ý tưởng thuật toán như sau:

* ***Toán tử lớn hơn:***
  + Nếu  hoặc : Trả về false.
  + Ngược lại: Trả về true.
* ***Toán tử bé hơn hoặc bằng:***
  + Nếu : Trả về false.
  + Ngược lại: Trả về true.

## II.7. Toán tử số học

Các toán tử cộng (), trừ (), nhân (), chia lấy dư ().

### II.7.1. Các trường hợp đặc biệt

Xử lý những trường hợp sau: cộng/trừ hai số cùng dấu; nhân/chia lấy dư hai số dương.

#### II.7.1.1. Cộng hai số cùng dấu

Ý tưởng thuật toán như sau:

* Thực hiện cộng theo quy tắc tương tự phép cộng đại số (cộng có nhớ).
* *sign* của kết quả phép toán là *sign* của toán hạng.

#### II.7.1.2. Trừ hai số cùng dấu

Ý tưởng thuật toán như sau: Giả sử hai toán hạng là 

* Nếu : Trừ cùng dấu  và gán *sign* của kết quả là –1.
* Nếu :
  + Thực hiện trừ theo quy tắc tương tự phép trừ đại số (trừ có nhớ).
  + *sign* của kết quả phép toán là 1.

#### II.7.1.3. Nhân hai số dương

Ý tưởng thuật toán như sau:

* Thực hiện nhân theo quy tắc tương tự phép nhân đại số.
* *sign* của kết quả là 1.

#### II.7.1.4. Chia lấy dư hai số dương

Ý tưởng thuật toán như sau: Giả sử hai toán hạng là 

* Nếu : Trả về kết quả là .
* Nếu :
  + Chừng nào :
    - Tìm số nguyên dương  lớn nhất sao cho 
    - .
  + Trả về .

### II.7.2. Toán tử cộng

Giả sử hai toán hạng là . Ý tưởng thuật toán như sau:

* Nếu  cùng dấu: Cộng cùng dấu .
* Nếu  trái dấu:
  + Nếu : Trừ cùng dấu .
  + Nếu : Trừ cùng dấu .

### II.7.3. Toán tử trừ

Giả sử hai toán hạng là . Ý tưởng thuật toán như sau:

* Nếu  cùng dấu: Trừ cùng dấu .
* Nếu  trái dấu: Cộng cùng dấu .

### II.7.4. Toán tử nhân

Giả sử hai toán hạng là . Ý tưởng thuật toán như sau:

* Nhân hai số dương .
* *sign* của kết quả bằng .

### II.7.5. Toán tử chia lấy dư

Giả sử hai toán hạng là . Ý tưởng thuật toán như sau:

* Chia lấy dư hai số dương .
* *sign* của kết quả bằng .

**Lưu ý.** Chỉ xét trường hợp *b* khác 0.

***Ví dụ:*** .

## II.8. Toán tử thao tác bit

Các toán tử ***AND*** (&), ***OR***(|), ***XOR***(^), ***NOT***(~), ***ShiftLeft*** (<<), ***ShiftRight*** (>>)

### II.8.1. Toán tử AND, OR, XOR

Trước khi thực hiện phép toán AND, OR, XOR giữa hai số **BigInt**, nếu toán hạng nào là số âm, ta phải tìm biểu diễn bù 2 của dãy bit lưu trong mảng *data* với *rBit* bit và lưu vào một số **BigInt** khác. Sau đó dùng số **BigInt** này để thực hiện phép toán.

***Ví dụ: Tính* –5 & 2**

Xét cách lưu trữ số **BigInt** –5:

* *sign* = –1
* *rBit* = 128
* *nBit* = 3
* *nBlock =* 1
* *data:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Phần tử | *data*[0] | | | | | | | | | | | | | | | |
| Biểu diễn | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

Ta tìm biểu diễn bù 2 của **–5** với 128 bits:

**11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111011**

Lưu dãy bit này vào một số **BigInt** mới như sau:

* *sign* = –1
* *rBit* = 128
* *nBit* = 128
* *nBlock* = 8
* *data*: Trỏ đến vùng nhớ chứa 8 phần tử, mỗi phần tử có kích thước 2 bytes, lưu dãy bit dài 128 bit ở trên

Sau đó sử dụng số **BigInt** này để thực hiện phép toán AND/OR/XOR với số **BigInt** 2.

Ý tưởng thuật toán như sau: Giả sử hai toán hạng là .

* Tạo một số **BigInt** lưu kết quả (*res*):
  + Xác định  dựa vào  và .
  + 
  + .
  + .
* Duyệt và thực hiện phép AND/OR/XOR thông thường với từng phần tử tương ứng của  và lưu vào .
* Nếu : tìm biểu diễn bù 2 của *res* với *res.rBit* bit.

### II.8.2. Toán tử NOT

Ý tưởng thuật toán như sau: Giả sử toán hạng là 

* Tạo số **BigInt** lưu kết quả (*res*).
* Nếu  là số âm:
  + .
  + .
* Nếu  là số không âm:
  + .
  + .

### II.8.3. Toán tử ShiftLeft

Ý tưởng thuật toán như sau: Giả sử ta thực hiện ShiftLeft *n* bit với số **BigInt** *a*

* Tạo số **BigInt** *res = a* (Sử dụng hàm *BigIntCpy*).
* Tăng *res*.*rBit*, *res.nBit* và *res.nBlock* một lượng vừa đủ, cấp phát lại vùng nhớ mà *data* trỏ đến (mở rộng ra).
* Tính *i = n / 16.*
* Duyệt ngược mảng *res.data* từ *nBlock* – 1 về *i* theo biến *x:*
  + 16 bit theo thứ tự từ cao tới thấpcủa *res.data*[*x*] sẽ là:
    - *(n % 16)* bit đầu (tình từ bit 0) của *res.data*[*x – i*]
    - *16 – (n % 16)* bit cuối (tính từ bit 15 về) của *res.data*[*x – i –* 1]
* Nếu , gán những phần tử thứ *i* – 1 về 0 của mảng *res.data* bằng 0.

**Lưu ý.** Ta chỉ tiến hành Shiftleft *n* với *n* là số nguyên không âm, lớn ***không quá***31 bit ().

### II.8.4. Toán tử ShiftRight

**Lưu ý.** Ta đang lưu dãy bit biểu diễn ***giá trị tuyệt đối*** của số **BigInt** vào mảng *data*. Do đó, khi dịch một số **BigInt** âm thì phải xử lý thêm một số thao tác khác.

Ý tưởng thuật toán như sau: Giả sử ta thực hiện ShiftRight *n* bit với số **BitInt** *a*

* Tạo số **BigInt** *res = a* (Sử dụng hàm *BigIntCpy*).
* Giảm *res.rBit*, *res.nBit* một lượng vừa đủ (đảm bảo *res.rBit* >= 128).
* Tính *i = n / 16.*
* Duyệt xuôi mảng *res.data* từ 0 đến *nBlock –* 1 *– i* theo biến *x.*
  + 16 bit theo thứ tự từ cao tới thấp của *res.data*[*x*] sẽ là:
    - (*n % 16*) bit cuối (tính từ bit 15 về) của *res.data*[*x + i*]
    - *16 –* (*n % 16*) bit đầu (tính từ bit 0) của *res.data*[*x + i +* 1]
* Nếu , gán những phần tử thứ *nBlock – i* đến *nBlock –* 1của mảng *res.data* bằng 0. Cấp phát lại mảng *res.data* để loại bỏ những phần tử thừa này.
* Nếu *a.sign* = –1:
  + Tìm số nguyên dương *k* lớn nhất sao cho .
  + Nếu : .

**Lưu ý.** Ta chỉ tiến hành ShiftRight *n* với *n* là số nguyên không âm, lớn ***không quá***31 bit   
().

## II.9. Các hàm hỗ trợ

Các hàm hỗ trợ*: Abs, Max, Min, Digits, Pow, ToString, IsPrime.*

### II.8.1. Hàm Abs

Lấy giá trị tuyệt đối như cách thông thường: Nếu số **BigInt** là số âm, ta gán lại *sign* = 1.

### II.8.2. Hàm Max, Min

Sử dụng toán tử quan hệ để trả về giá trị lớn nhất, giá trị nhỏ nhất giữa hai số **BigInt**.

### II.8.3. Hàm Digits

Trả về số lượng ký tự số của số **BigInt**.Ý tưởng thuật toán như sau:

* Khai báo biến đếm *cnt =* 0*.*
* Xem số **BigInt** là số dương.
* Lần lượt chia số **BigInt** cho 10 theo quy tắc tương tự như chia đại số, mỗi lần chia, ta tăng biến đếm *cnt* lên 1. Quá trình dừng khi số **BigInt** bằng 0.
* Số lượng kí tự số chính bằng *cnt*.

### II.8.4. Hàm Pow

Giả sử ta cần tính  với *a, b* là các số **BigInt.** Ý tưởng thuật toán tính lũy thừa như sau:

* Tạo số **BigInt** lưu kết quả là .
* Chừng nào :
  + Nếu  lẻ: .
  + .
* Trả về .

***Lưu ý:*** Chỉ xét trường hợp  và ta quy ước 00 = 1.

### II.8.3. Hàm ToString

Sử dụng hàm chuyển số **BigInt** sang hệ thập phân và trả về chuỗi kết quả.

### II.8.4. Hàm IsPrime

Xét riêng các trường hợp số **BigInt** không vượt quá 4, số **BigInt** là số chẵn.

Trong trường hợp số **BigInt** là số lẻ và lớn hơn hoặc bằng 5, sử dụng thuật toán **Miller – Rabin** để kiểm tra một số **BigInt** có phải là số nguyên tố hay không.

Giả sử ta tiến hành kiểm tra số **BigInt**  trong  lần. Ý tưởng thuật toán như sau:

* Phân tích  với  là số lẻ.
* Lặp *i* từ 1 đến :
  + Chọn ngẫu nhiên số nguyên dương  sao cho .
  + Nếu : Trả về false.
  + Tính .
  + Nếu  hoặc : Qua lần lặp *i* kế tiếp.
  + Lặp  từ 1 đến :
    - .
    - Nếu : Qua lần lặp *i* kế tiếp.
  + Trả về false.
* Trả về true.

Trong source code, số lần kiểm tra được đặt là 5.

## II.10. Tính thời gian xử lý và bộ nhớ đã dùng

Những hàm được nhóm xây dựng thêm các hàm để hỗ trợ việc tính thời gian xử lý và bộ nhớ như sau:

* Hàm *Input* với thao tác đọc dữ liệu.
* Hàm *Shift* với các thao tác dịch trái, dịch phải.
* Hàm *Not* với toán tử NOT.
* Hàm *Calculate* với các tính toán sử dụng toán tử AND, OR, XOR, cộng, trừ, nhân, chia lấy dư.
* Hàm *ToBaseN* với thao tác chuyển đổi số **BigInt** sang hệ cơ số khác.

Với các hàm hỗ trợ Abs, Max, Min, Digits, ToString, IsPrime, việc tính thời gian xử lý và bộ nhớ đã dùng được cài đặt ngay trong các hàm đó.

### II.10.1. Tính thời gian xử lý

Sử dụng thư viện *time.h* để tính thời gian xử lý theo cách dưới đây:

clock\_t start, end;

start = clock();

// code ...-

end = clock();

printf("Time = %gs", double(end - start) / CLOCKS\_PER\_SEC);

### II.10.2. Tính bộ nhớ đã dùng

Xây dựng hàm HeapUsed để xác định lượng bộ nhớ đã cấp phát trong phân vùng Heap:

|  |  |
| --- | --- |
| Hàm HeapUsed | Sử dụng |
| long long HeapUsed() {  \_HEAPINFO info = { 0, 0, 0 };  long long used = 0;  int rc;  while ((rc = \_heapwalk(&info)) == \_HEAPOK) {  if (info.\_useflag == \_USEDENTRY) {  used += info.\_size;  }  }  if (rc != \_HEAPEND && rc != \_HEAPEMPTY) {  used = (used ? -used : –1);  }  return used;  } | long long start, end;  start = HeapUsed();  // code ...  end = HeapUsed();  printf("Memory = %d bytes", max(end – start – 36, 0); |

# III. Cài đặt đồ án

## III.1. Môi trường lập trình

Nhóm sử dụng IDE **Visual Studio 2019** để cài đặt đồ án này.

## III.2. Dữ liệu đầu vào, dữ liệu đầu ra

Dữ liệu đầu vào được đọc từ file *input.txt*, kết quả được ghi ra file *output.txt (hai file này nằm trong cùng thư mục với file thực thi 20120504\_20120025.exe)*

Cách nhập dữ liệu vào file *input.txt* được trình bày trong file *Hướng dẫn sử dụng.docx.*

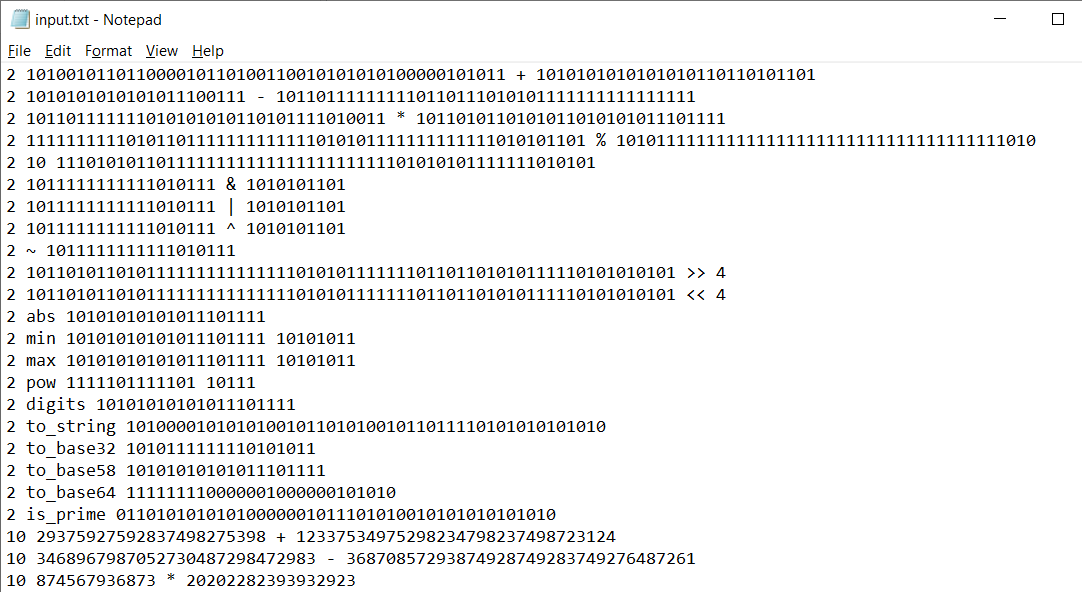
Thời gian xử lý và bộ nhớ đã dùng được in ra màn hình console

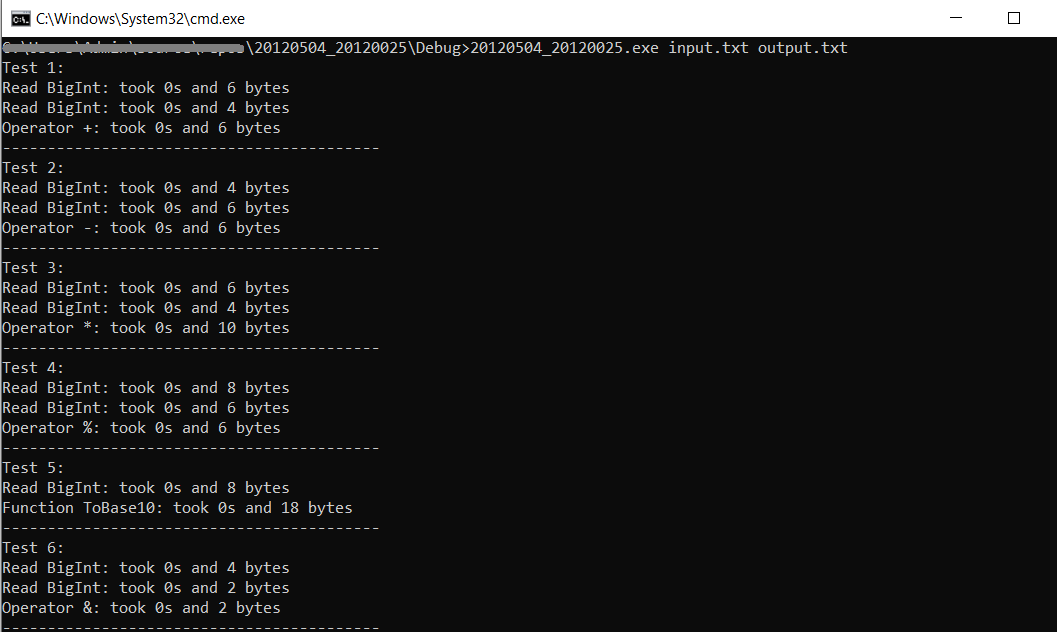
## III.3. Thực thi chương trình

Chạy chương trình bằng command line với cú pháp như sau:

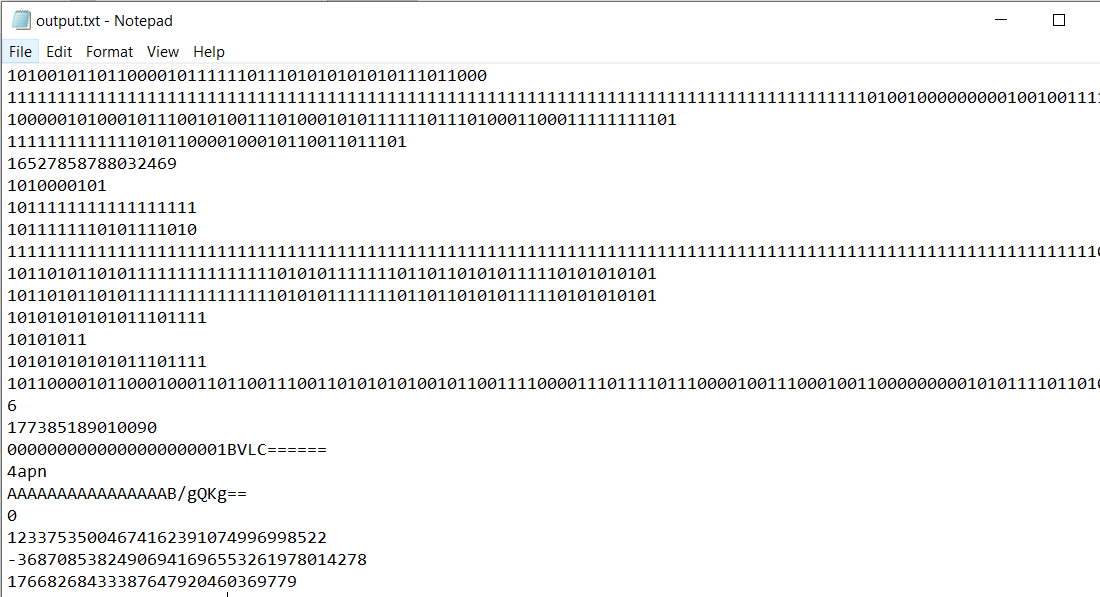
*20120504\_20120025.exe input.txt output.txt*

Hình ảnh minh họa kết quả thu được khi chạy chương trình:

File *input.txt*



*Thời gian xử lý, bộ nhớ đã dùng được in ra msàn hình console*



File *output.txt*

# IV. Tài liệu tham khảo

* Hệ cơ số 32: <https://en.wikipedia.org/wiki/Base32>
* Hệ cơ số 64: <https://en.wikipedia.org/wiki/Base64>
* Hệ cơ số 58: [The Base58 Encoding Scheme (ietf.org)](https://tools.ietf.org/id/draft-msporny-base58-01.html)
* Thuật toán Miller – Rabin: <https://vi.wikipedia.org/wiki/Ki%E1%BB%83m_tra_Miller-Rabin>
* Tính bộ nhớ đã dùng: <https://www.daniweb.com/posts/jump/650648>