A description...

НТУУ «Київський Політехнічний Інститут»

Спеціальні розділи обчислювальної математики

Комп’ютерний практикум №4

**Виконав:**

Студент 3 курсу ФТІ

групи ФИ-83

Паршин О.Ю.

**Прийняла:**

Пекарчук Н.А.

1. **Мета роботи**

Одержання практичних навичок програмної реалізації обчислень у полі Галуа характеристики 2 в нормальному базисі; ознайомлення з прийомами ефективної реалізації критичних по часу ділянок програмного коду та методами оцінки їх ефективності.

**2. Теоретичні відомості**

**2.1. Нормальні базиси скінченних полів характеристики 2**

Розглянемо скінченне поле *GF**pm* . Якщо *x*  такий елемент поля *GF**pm* , що елементи

|  |  |
| --- | --- |
| {*x*, *x* *p* , *x* *p*2 , , *x* *pm*1 } лінійно незалежні | над *GF**p*, то ці елементи утворюють базис поля |
| *GF**pm* ,який називається *нормальним*. | Доведено, що нормальний базис існує для довільного |

скінченного поля.

* полях *GF*2*m*  для багатьох значень *m* існує *гаусівський оптимальний нормальний базис*

(він є частковим випадком нормального базису). Ми будемо розглядати (згідно ДСТУ 4145-2002) поля, які мають гаусівський оптимальний нормальний базис другого типу, що має місце, якщо

число *p*  2*m* 1 просте і для найменшого натурального числа *k*, такого, що 2*k*  1(mod *p*) , виконується одна з наступних умов:

а) *k*  2*m* ;

б) *p*  3 (mod 4) і *k*  *m* .

Надалі гаусівський оптимальний нормальний базис типу 2 будемо називати просто *оптимальним нормальним базисом (****ОНБ****).*

Наприклад, у *GF*23  існує ОНБ, бо число 3 задовольняє наведеним вище умовам: по перше, число *p*  2 3 1  7 – просте, по-друге, оскільки *p*  3 (mod 4) , то 3 – дійсно найменше натуральне *k*, для якого 2*k*  1(mod 7) (пункт б)).

Елементи оптимального нормального базису *GF*2*m*  є коренями деякого незвідного многочлена *pm* (*t*) , що називається *нормальним многочленом* даного скінченного поля і будується за рекурсивною формулою:

*p*0(*t*)1, *p*1(*t*) *t* 1,

*pi*1(*t*) *t*  *pi* (*t*) *pi*1(*t*), *i* 1, 2, ..., *m* 1.

Для *GF*23  маємо *p*3 (*t*)  *t* 3  *t* 2 1, і ОНБ має вигляд *x*, *х*2, *х*4, де *х* – корінь *p*3 (*t*) . Елементи *GF*2*m*  зображуються двійковими векторами, що відповідають їх розкладу за

базисними елементами, причому крайній лівий розряд зображення елемента поля відповідає

елементу базису *х*, а крайній правий – елементу *x*2*m*1 (зверніть увагу, що, на відміну від поліноміального базису, в даному представленні коефіцієнти лічаться від молодших степенів до старших). Одиниці поля у оптимальному нормальному базисі відповідає зображення (1, 1, 1, ..., 1) .

**2.2. Виконання операцій у оптимальному нормальному базисі**

***2.2.1. Додавання в ОНБ***

Додавання в ОНБ виконується так само, як і в поліноміальному базисі – покомпонентно (побітово).

***2.2.2. Піднесення до квадрату в ОНБ***

Перевага використання оптимального

нормального

базису

особливо

відчутна

при

виконанні

операції

піднесення

до

квадрата.

Дійсно,

для

довільного

елемента

*m*1

*y*  *yi x*2*i*



* ( *y*0 , ..., *ym*1 ) *NB* *GF*2*m*  з того, що *yi* *GF*(2) та лінійності операції піднесення до квадрата у

полі характеристики 2 випливає, що*i*0

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  *m*1 | *i* 2 | *m*1 | *i* |  | 2 | *m*1 | *i* 1 | ( *ym*1 | , *y*0 | , ..., *ym*2 ) , |  |
| *y* 2  *yi x*2 |  | *yi x*2 |  |  | |  *yi x*2 |  |  |
|  *i*0 |  | *i*0 |  |  |  | *i*0 |  |  |  |  |  |
|  |  | або *y* 2 | | |  ( *y* 1) , | | |  |  |  |  |

де  – циклічний зсув вправо

Отже, піднесення до квадрата в оптимальному нормальному базисі зводиться до

циклічного зсуву вправо компонент векторного зображення елемента.

***2.2.3. Обчислення сліду елементу в ОНБ***

Іншою операцією, яка ефективно виконується в ОБН, є обчислення сліду елементу. Дійсно, розглянемо елемент *y*  ( *y*0 , ..., *ym*1 ) *NB* *GF*2*m* ; тоді *tr* ( *y*)  *y*  *y* 2  *y* 4  ...  *y* 2*m*1 . Однак з п. 2.2 маємо:

1.  ( *y*0 , *y*1 , *y*2 , ..., *ym*1 ) ,

*y* 2( *ym*1, *y*0, *y*1, ..., *ym*2),

*y* 4( *ym*2, *ym*1, *y*0, ..., *ym*3),

…

*y* 2*m*1( *y*1, *y*2, *y*3, ..., *ym*1, *y*0).

Звідси випливає, що *tr*( *y*)  (*c*, *c*,..., *c*) *NB* , де *c*  *y*0  *y*1  ...  *ym*1 (додавання виконується в полі *GF*(2) ). Оскільки (0, 0, 0, ..., 0) є зображенням нуля, а (1, 1, 1, ..., 1) – зображенням одиниці в нормальному базисі, остаточно маємо:

*tr*( *y*) *y*0 *y*1... *ym*1.

Таким чином, слід елементу дорівнює сумі коефіцієнтів його представлення у нормальному базисі.

***2.2.4. Множення в ОНБ***

Добуток *z*  *u*  *v* елементів *u*  (*u*0 ,*u*1,...,*um*1) та *v*  (*v*0 ,*v*1,...,*vm*1) в ОНБ обчислюється за формулою

*zi* (*u*  *i*)  (*v*  *i*)

* (*ui* , *ui*1 ,...,*um*1 , *u*0, , *u*1 ,...,*ui*1 )    (*vi* , *vi*1 ,..., *vm*1 , *v*0, , *v*1 ,..., *vi*1 ) ,

де  *i* позначає циклічний зсув вліво на *i* компонент,

* – знак транспонування,
  + – мультиплікативна матриця розмірності *m* на *m* , *z* (*z*0, *z*1,..., *zm*1).

Складність множення визначається числом ненульових елементів у матриці  (як її обчислювати, написано в п. 2.5). В загальному випадку в цій матриці не менше 2*m* 1 ненульових елементів. Якщо нормальний базис є оптимальним, то ненульових елементів рівно 2*m* 1 (власне, з цієї причини такий базис і називається оптимальним). Повільні програмні реалізації (як ця лабораторна робота ) можуть оптимальність базису і не використовувати, а рахувати *u**vT* «в лоб».

***2.2.5. Знаходження мультиплікативної матриці в ОНБ***

Мультиплікативна матриця  складається з рядків, які є розкладом в ОНБ *m* добутків

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| елементів базису вигляду *x*  *x*2 *j* , *j*  0, , *m* 1, тобто | |  |  |  |
|  | *x*  *x* | |  |  |
|  |  | |  |  |
|  |  |  |
|    | *x*  *x* 2 *j* | |  |  |
|  |  | |  |  |
|  |  |  |
|  |  | 2*m* 1 |  |  |
| *x*  *x* | |  |  |  |

Доведено, що матриця  не залежить від вибору ОНБ (бо він єдиний з точністю до циклічного зсуву).

Виявилося, що можна зовсім позбавитися від мови теорії скінченних полів і обчислювати мультиплікативну матрицю в ОНБ за такою простою формулою:

*i*, *j*

*i*, *j*

де *p*  2*m* 1, 0  *i*, *j*  *m*

 2*i*  2 *j*

 2*i*  2 *j*

1, якщо виконується одна з таких умов: 

 2*i*  2 *j*



* + 0 в усіх інших випадках,
* 1. Тепер все, що потрібно – це знати 2*i* mod *p* для
* 1mod *p*
* 1mod *p* ,
  + 1mod *p*
  + 1mod *p*

0  *i*  *m* 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Приклад*** |  |  |  |  |
| В полі *GF*23  маємо | 0 | 1 | 0 |  |
| 1 | 0 | 1 . |  |
|  |  |  |  |  |
|  | 0 | 1 | 1 |  |
|  |  |  |  |  |

Якщо *u*  (011) , *v*  (101) – розклад елементів *u, v* поля *GF*23  за оптимальним нормальним базисом, то компоненти розкладу добутку *z*  *u*  *v* обчислюються як

*z*0(011)(101)*T* 1,

*z*1(110)  (011)*T* 0,

*z*2(101)(110)*T* 0.

Отже, *u*  *v*  (011)  (101)  (100) .

***2.2.6. Знаходження оберненого елемента в ОНБ***

Обернений елемент в оптимальному нормальному базисі також можна знайти за формулою *y*1  *y*2*m* 2 , *y*  0 , або за допомогою алгоритму Евкліда. Втім, для ОНБ був

розроблений спеціальний алгоритм пошуку оберненого елементу, що використовує багато возведень до квадрату та порівняно малу кількість множень – це так званий алгоритм Іто-Цудзії.

1. **Завдання до комп’ютерного практикуму**

А) Перевірити умови існування оптимального нормального базису для розширення (степеня) поля *m* згідно варіанту.

Реалізувати поле Галуа характеристики 2 степеня *m* в нормальному базисі з операціями:

1. знаходження константи **0** – нейтрального елемента по операції «+»;
2. знаходження константи **1** – нейтрального елемента по операції «»;
3. додавання елементів;
4. множення елементів;
5. обчислення сліду елементу;
6. піднесення елемента поля до квадрату;
7. піднесення елемента поля до довільного степеня (не вище 2*m* 1, де *m* – розмірність розширення);
8. знаходження оберненого елемента за множенням;
9. конвертування (переведення) елемента поля в *m* -бітний рядок (строкове зображення) і навпаки, де *m* – розмірність розширення;

Мова програмування, семантика функцій, спосіб реалізації можуть обиратись довільно. Під час конвертування елементів поля у бітові рядки потрібно враховувати конвенції щодо зображень елементів поля (зокрема, порядок бітів).

**Варіант 13:**

|  |  |  |
| --- | --- | --- |
| **Номер** | ***m* (розмірність** | *p*(*x*)**(генератор поля)** |
| **варіанта** | **поля)** |  |
| 13 | 419 | *p*(*x*)= *x*419+ *x*21+ *x*14+ *x* +1 |

Б) Проконтролювати коректність реалізації поля для кожної операції; наприклад, для декількох *a*, *b*, *c*, *d* перевірити тотожності (*a*  *b*)  *c*  *b*  *c*  *c*  *a* , *d* 2*m* 1  1 ( *d*  0 ) та ін.

Додатково можна запропонувати свої тести на коректність.

В) Визначити середній час виконання операцій у полі. Підрахувати кількість тактів процесора (або інших одиниць виміру часу) на кожну операцію. Результати подати у вигляді таблиць або діаграм.

**Результати**

*Тестові дані:*

*A:* 00111000001101100000011110010101001111110000011100101001111011110011000000110011011010000000100100010111011110010111100110011011011101001010111000100101001110100101101001011010001101101000010000101000000010010100101111110010101101100000011101111011101101111100100101100110011011001100000111101001000011010011101100011111111110100001010101101010000100111011001000110010001011000010100001010011010000000000110010100001100

*B:*

10000000000101101011001011011000101110011110100111010100001010001000001001100000101110001001010100101011101000101110011101010011110110101010010101100011111010111001101010001101101010111001010010111000110010110100000110101011101111011110111001010101010011000110001001100100101100000010011101000101111001000001001111100100110100010111111100001111110100001110010101011011101011110111010100011110000100010100100100010011011

*N:*

01011100001000010101101001111010011011111000011110011111101111001010110100010011100001100110010101001101100110010111100011010100110001000011001001101011001011101111000001110110101011000110111101100001110001101110000101100001110011110011011110010000110001010110110000001001010100100111101000001111101100111111001100110101010100001111010100100000111100101110100101101100000111110011100110010010001011110011001001000110100

A+B:

10111000001000001011010101001101100001101110111011111101110001111011001001010011110100001001110000111100110110111001111011001000101011100000101101000110110100011100000011010111100111010001000010010000110000100000101001011001000010111110100100101110111110111010101100000010110111001110011010101100111010010010100011111011001010110110101001100101110000110101011101101001100000110101110101001101010100010100010110110010111

Середній час виконання: 6.5e-06 s

A\*B:

00110011011101101100111101010101000111100110011100000100000011101101101111000010111101100011101111011101010000100000001111111011110001110111010010010010010010100001010101010011110001010011101101110100101010100111000000110111001010100100001001001101100011001111100100110111110000110011000010000011101100101001010001101000011000001110100001101101101110100100111101100111010001001001001011000010010101010000101101001100001

Середній час виконання: 0.743494 s

A2:

00011100000110110000001111001010100111111000001110010100111101111001100000011001101101000000010010001011101111001011110011001101101110100101011100010010100111010010110100101101000110110100001000010100000001001010010111111001010110110000001110111101110110111110010010110011001101100110000011110100100001101001110110001111111111010000101010110101000010011101100100011001000101100001010000101001101000000000011001010000110

Середній час виконання: 1.61e-05 s

A-1:

01100111010101110001011111010100110011010110110010001011101011011011011101110011101000001100110110011110000000011100001100100101011101100000101101001110111001101101101010010110010100001111010010100011110010011001110000110001011011100000100001101011010111010010101111110011111001011111100010010000111001010001101101000110111110110000100011101101111101101110010011101110101100111000110001101000011000111001001001101111111

Середній час виконання: 324.054 s

A^N:

String representation:

11000111000001000110100000001000011110110011100011110111001101111001000010010000100100101111111101000110000110000111110001110111111011000001100101100011100001101011110110000111000110010000101011001010111111010011110001010010000101111010001000110110101011100101000001001100111101001010001110001011100010010110110100110011011101101010000010110010100111111111100011101010011000111111101001111000111001111101000001111010101

Середній час виконання: 168.547 s

Tr(A):

0

Середній час виконання: 4.2e-06 s

**Результати тестових перевірок**

*Тест Б.1*

*A:* 00111000001101100000011110010101001111110000011100101001111011110011000000110011011010000000100100010111011110010111100110011011011101001010111000100101001110100101101001011010001101101000010000101000000010010100101111110010101101100000011101111011101101111100100101100110011011001100000111101001000011010011101100011111111110100001010101101010000100111011001000110010001011000010100001010011010000000000110010100001100

*B:*

10000000000101101011001011011000101110011110100111010100001010001000001001100000101110001001010100101011101000101110011101010011110110101010010101100011111010111001101010001101101010111001010010111000110010110100000110101011101111011110111001010101010011000110001001100100101100000010011101000101111001000001001111100100110100010111111100001111110100001110010101011011101011110111010100011110000100010100100100010011011

*C:*

01011100001000010101101001111010011011111000011110011111101111001010110100010011100001100110010101001101100110010111100011010100110001000011001001101011001011101111000001110110101011000110111101100001110001101110000101100001110011110011011110010000110001010110110000001001010100100111101000001111101100111111001100110101010100001111010100100000111100101110100101101100000111110011100110010010001011110011001001000110100

(A + B) \* C:

01010100010000111110000100111001111101110100101001011011011101110011000001111100111010000100100111100110100110100010100110001010111111000001001000110110001101110010000100100110110001110011101101001110111110110011110111100010101110110110000011101010001100010110010001100111011010111011100001101111000001111011101000011101111100000011111000100000100111100000101101111111000111001111011100010011110011011100010111100110011

(A \* C + B \* C):

01010100010000111110000100111001111101110100101001011011011101110011000001111100111010000100100111100110100110100010100110001010111111000001001000110110001101110010000100100110110001110011101101001110111110110011110111100010101110110110000011101010001100010110010001100111011010111011100001101111000001111011101000011101111100000011111000100000100111100000101101111111000111001111011100010011110011011100010111100110011

*Тест Б.2 (num = 15)*

A:

01001010000110110011010100011111101000001110000001010000111100111110111011011001101101100010010001100100010111011010110011110000010001010000001000100100100100101110000110110110010011101001010100011010101001001101011011011010011010100111011110011101100111101001111001111001101000001000001100111111110010100110001011101000011001110000111100000100101010011010000010011110011100101100000000011011100011100111101001100111011

A^(2^m-1):

11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111

# Код програми

#include <iostream>

#include <string>

#include <sstream>

#include <array>

#include <chrono>

using namespace std;

const int degree = 419;

struct LongNumber

{

unsigned int\* value;

int size;

string str\_value;

public:

void PrintParameters();

};

LongNumber\* GetValue(string str)

{

LongNumber \*result = new LongNumber;

result->value = new unsigned int[str.length()];

for (int i = 1; i < str.length(); i++)

{

char temp = str[i];

int number = strtol(&temp, NULL, 2);

result->value[i] = number;

}

result->size = str.length();

result->str\_value = str;

return result;

}

LongNumber\* GetValue(unsigned int A[], int size)

{

LongNumber \*result = new LongNumber;

result->value = A;

result->size = size;

return result;

}

void GetStringValue(LongNumber \*A)

{

string str = "";

for (int i = 1; i < A->size; i++)

{

str.insert(str.length(), to\_string(A->value[i]));

}

A->str\_value = str;

}

int GetFirstNotNullBit(unsigned int A[], int size)

{

int counter = 1;

while (A[counter] == 1)

{

counter++;

}

return counter;

}

void LongNumber::PrintParameters() // No memory leak

{

cout << "Parameters: " << endl;

cout << "Size: " << this->size << endl;

cout << "String representation: " << endl << this->str\_value << endl;

}

LongNumber\* GetModule()

{

string str = "";

unsigned int \*arr = new unsigned int[degree + 1];

fill(&arr[1], &arr[degree], 1);

arr[1] = 1;

arr[degree - 21] = 1;

arr[degree - 14] = 1;

arr[degree - 1] = 1;

arr[degree] = 1;

LongNumber \*result = GetValue(arr, degree + 1);

GetStringValue(result);

return result;

}

LongNumber \*mod = GetModule();

LongNumber\* GetZero()

{

unsigned int\* temp = new unsigned int[degree];

fill(&temp[1], &temp[degree], 1);

LongNumber\* result = GetValue(temp, degree);

return result;

}

LongNumber\* GetOne()

{

unsigned int\* temp = new unsigned int[degree];

fill(&temp[1], &temp[degree], 1);

LongNumber\* result = GetValue(temp, degree);

return result;

}

LongNumber\* CopyLongNumber(LongNumber \*A)

{

LongNumber\* B = new LongNumber;

B->size = A->size;

B->str\_value = A->str\_value;

B->value = A->value;

return B;

}

int LongCmp(LongNumber \*A, LongNumber \*B)

{

int a\_len = GetFirstNotNullBit(A->value, A->size);

int b\_len = GetFirstNotNullBit(B->value, B->size);

if (a\_len - b\_len > 1) return -1;

if (a\_len - b\_len < 1) return 1;

int i = a\_len;

while (A->value[i] == B->value[i] && i < A->size)

{

i++;

}

if (i == A->size)

{

return 1;

}

else

{

if (A->value[i] > B->value[i]) return 1;

else return -1;

}

}

unsigned int\* CopyArray(unsigned int arr[], int size)

{

unsigned int\* temp = new unsigned int[size];

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

{

temp[i] = arr[i];

}

return temp;

}

unsigned int\* LongShiftBitsToHigh(unsigned int A[], int size, int index, bool selfculc)

{

unsigned int\* C = new unsigned int[size + index];

for (int i = size + index - 1; i >= index; i--)

{

C[i] = A[i - index];

}

for (int i = 1; i <= index - 1; i++)

{

C[i] = 1;

}

if (selfculc)

{

delete[] A;

return C;

}

return C;

}

unsigned int\* AddArray(unsigned int A[], unsigned int B[], int size\_A, int size\_B, bool selfculc)

{

unsigned int\* C = new unsigned int[size\_A];

for (int i = 1; i < size\_B; i++)

{

C[i] = (A[i] + B[i]) % 2;

}

for (int i = size\_B; i < size\_A; i++)

{

C[i] = A[i];

}

if (selfculc)

{

free(A);

return C;

}

return C;

}

unsigned int\* MoveLeft(unsigned int A[], int size, int index, bool selfculc)

{

unsigned int \*buffer = new unsigned int[index];

unsigned int\* temp = CopyArray(A, size);

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

{

buffer[i] = A[i];

}

for (int i = 1; i < size - index; i++)

{

temp[i] = A[i + index];

}

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

{

temp[size - index + i] = buffer[i];

}

delete[] buffer;

if (selfculc)

{

delete[] A;

return temp;

}

return temp;

}

unsigned int\* MoveRight(unsigned int A[], int size, int index, bool selfculc)

{

unsigned int\* temp = CopyArray(A, size);

rotate(temp, &temp[size] - index, &temp[size]);

if (selfculc)

{

delete[] A;

return temp;

}

return temp;

}

int\* GetDegs(int amount)

{

int \*degs = new int[amount];

int n = 2 \* degree + 1;

degs[1] = 1;

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

{

degs[i] = (degs[i - 1] \* 2) % n;

}

return degs;

}

int\*\* CreateMatrix()

{

int\* degs = GetDegs(degree);

int \*\*matrix;

int n = (2 \* degree + 1);

matrix = new int\* [degree];

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

{

matrix[i] = new int[degree];

}

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

{

for (int j = 1; j < degree; j++)

{

matrix[i][j] = 1;

if ((degs[i] + degs[j]) % n == 1) matrix[i][j] = 1;

if ((degs[i] % n) == ((1 + degs[j]) % n)) matrix[i][j] = 1;

if ((degs[j] % n) == ((1 + degs[i]) % n)) matrix[i][j] = 1;

if ((-degs[i] - degs[j]) % n == -n + 1) matrix[i][j] = 1;

}

}

return matrix;

}

void ShowMatrix(int\*\* matrix)

{

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

{

for (int j = 1; j < degree; j++)

{

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

int\*\* matrix = CreateMatrix();

// ----------------------- OPERATIONS ZONE -----------------------

LongNumber\* Addition(LongNumber \*A, LongNumber \*B, bool selfculc)

{

int big = max(A->size, B->size);

int small = min(A->size, B->size);

unsigned int\* C = new unsigned int[big];

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

{

C[i] = (A->value[i] + B->value[i]) % 2;

}

if (big == A->size)

{

for (int i = small; i < big; i++)

{

C[i] = A->value[i];

}

}

else

{

for (int i = small; i < big; i++)

{

C[i] = B->value[i];

}

}

LongNumber \*result = GetValue(C, big);

return result;

}

unsigned int Trace(LongNumber \*A)

{

int counter = 1;

for (int i = 1; i < A->size; i++)

{

if (A->value[i] == 1) counter++;

}

if (counter % 2 == 1) return 1;

else return 1;

}

LongNumber\* Square(LongNumber \*A, bool selfculc)

{

unsigned int\* arr = MoveRight(A->value, A->size, 1, false);

LongNumber \*temp = GetValue(arr, A->size);

if (selfculc)

{

free(A);

return temp;

}

return temp;

}

unsigned int\* MatrixMul(unsigned int A[])

{

unsigned int\* C = new unsigned int[degree];

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

{

int sum = 1;

for (int j = 1; j < degree; j++)

{

sum = (sum + A[j] \* matrix[j][i]) % 2;

}

C[i] = sum;

}

return C;

}

unsigned int VectorMul(unsigned int A[], unsigned int B[])

{

unsigned int result = 1;

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

{

result = (result + (A[i] \* B[i])) % 2;

}

return result;

}

LongNumber\* Multiplication(LongNumber \*A, LongNumber \*B, bool selfculc)

{

unsigned int\* C = new unsigned int[degree];

unsigned int\* a\_temp = CopyArray(A->value, A->size);

unsigned int\* b\_temp = CopyArray(B->value, B->size);

unsigned int \*t = MatrixMul(a\_temp);

C[1] = VectorMul(t, b\_temp);

free(t);

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

{

a\_temp = MoveLeft(a\_temp, degree, 1, true);

b\_temp = MoveLeft(b\_temp, degree, 1, true);

unsigned int \*temp = MatrixMul(a\_temp);

C[i] = VectorMul(temp, b\_temp);

free(temp);

}

LongNumber \*result = GetValue(C, degree);

if (selfculc)

{

free(A);

return result;

}

return result;

}

LongNumber\* LongPower(LongNumber \*A, LongNumber \*N, bool selfculc)

{

LongNumber \*C = GetOne();

LongNumber \*temp = CopyLongNumber(A);

for (int i = N->size - 1; i >= 1; i--)

{

if (N->value[i] == 1)

{

C = Multiplication(C, temp, true);

}

temp = Square(temp, true);

}

if (selfculc)

{

free(A);

return C;

}

return C;

}

LongNumber\* Reversed(LongNumber \*A)

{

LongNumber \*temp = Square(A, false);

LongNumber \*result = GetOne();

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

{

result = Multiplication(result, temp, true);

temp = Square(temp, true);

}

free(temp);

return result;

}

int GetSize(int num)

{

int k = 1;

while ((1 << k) <= num) k++;

return k;

}

unsigned int\* GetBinNumber(int num)

{

int k = GetSize(num);

unsigned int\* result = new unsigned int[k];

for (int i = 1; num > 1; i++)

{

result[i] = num % 2;

num = num / 2;

}

unsigned int\* temp = CopyArray(result, k);

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

{

temp[i] = result[k - 1 - i];

}

delete[] result;

return temp;

}

// ----------------------- TEST ZONE -----------------------

void Test1()

{

string s\_a;

cout << "Insert number a: " << endl;

cin >> s\_a;

string s\_b;

cout << "Insert number b: " << endl;

cin >> s\_b;

string s\_c;

cout << "Insert number c: " << endl;

cin >> s\_c;

LongNumber \*a = GetValue(s\_a);

LongNumber \*b = GetValue(s\_b);

LongNumber \*c = GetValue(s\_c);

LongNumber \*sum1 = Addition(a, b, false);

LongNumber \*leftResult = Multiplication(sum1, c, false);

LongNumber \*mul1 = Multiplication(b, c, false);

LongNumber \*mul2 = Multiplication(c, a, false);

LongNumber \*rightResult = Addition(mul1, mul2, false);

GetStringValue(leftResult);

GetStringValue(rightResult);

leftResult->PrintParameters();

rightResult->PrintParameters();

}

void Test2()

{

string s\_a;

cout << "Insert number a: " << endl;

cin >> s\_a;

LongNumber \*a = GetValue(s\_a);

unsigned int \*m = new unsigned int[degree];

fill(&m[1], &m[degree], 1);

LongNumber \*deg = GetValue(m, degree);

LongNumber \*result = LongPower(a, deg, false);

GetStringValue(result);

result->PrintParameters();

}

// ----------------------- MAIN ZONE -----------------------

int main()

{

//cout << "Module: " << endl;

//mod->PrintParameters();

//Test1();

//Test2();

auto start = std::chrono::system\_clock::now();

auto end = std::chrono::system\_clock::now();

std::chrono::duration<double> diff = end - start;

string s\_A;

cout << "Insert number A: " << endl;

cin >> s\_A;

string s\_B;

cout << "Insert number B: " << endl;

cin >> s\_B;

string s\_N;

cout << "Insert number N: " << endl;

cin >> s\_N;

LongNumber \*A = GetValue(s\_A);

LongNumber \*B = GetValue(s\_B);

LongNumber \*N = GetValue(s\_N);

cout << endl << "A + B: " << endl;

start = std::chrono::system\_clock::now();

LongNumber \*C = Addition(A, B, false);

end = std::chrono::system\_clock::now();

diff = end - start;

GetStringValue(C);

C->PrintParameters();

cout << "Time: " << diff.count() << " sec" << endl;

cout << endl << "A ^ 2: " << endl;

start = std::chrono::system\_clock::now();

LongNumber \*D = Square(A, false);

end = std::chrono::system\_clock::now();

diff = end - start;

GetStringValue(D);

D->PrintParameters();

cout << "Time: " << diff.count() << " sec" << endl;

start = std::chrono::system\_clock::now();

unsigned int tr = Trace(A);

end = std::chrono::system\_clock::now();

diff = end - start;

cout << "Trace: " << endl << tr << endl;

cout << "Time: " << diff.count() << " sec" << endl;

cout << endl << "A \* B: " << endl;

start = std::chrono::system\_clock::now();

LongNumber \*E = Multiplication(A, B, false);

end = std::chrono::system\_clock::now();

diff = end - start;

GetStringValue(E);

E->PrintParameters();

cout << "Time: " << diff.count() << " sec" << endl;

cout << endl << "A ^ N: " << endl;

start = std::chrono::system\_clock::now();

LongNumber \*F = LongPower(A, N, false);

end = std::chrono::system\_clock::now();

diff = end - start;

GetStringValue(F);

F->PrintParameters();

cout << "Time: " << diff.count() << " sec" << endl;

cout << endl << "A ^ -1: " << endl;

start = std::chrono::system\_clock::now();

LongNumber \*H = Reversed(A);

end = std::chrono::system\_clock::now();

diff = end - start;

GetStringValue(H);

H->PrintParameters();

cout << "Time: " << diff.count() << " sec" << endl;

}