

Варіант 3

Реалізація операцій у скінченних полях характеристики 2 (нормальний базис)

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.

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 « \cdot »;
- 3) додавання елементів;
- 4) множення елементів;
- 5) обчислення сліду елемента;
- 6) піднесення елемента поля до квадрату;
- 7) піднесення елемента поля до довільного степеня
- 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 3 = m (розмірність поля) = 179.

Спочатку відкриємо програму для перевірки обчислень. Візьмемо A , B , N , які згенерувала програма.

Calculator (Programmer mode) showing the following calculations:

- A =** 00001111100011011000101110100001000000001000100100111010101100100111000110011111000
- B =** 0100000101100111100111101010010111101010111001100011100101000000000101101100100001
- N =** 1101
- A + B =** 0111101110001100000101110001001000111100100000101110000110010011100100101011100111C
- A * B =** 0101001110010100000110000000000011000111100100111001000101010110110000110010100101C
- A^2 =** 011001111101001101000110100001000100111010000100001110111011001001110011011000001
- A^(-1) =** 00111010100111001001110001010101101100100010100111101000001101110100001110100100C
- A^N =** 00101011001000101011100001101100010110011100100101011001100110000101000000101000111

The status bar at the bottom indicates: ----Calculating!----

В результаті, отримаємо результати, які співпадають із значеннями у програмі:

```
-----Calculating!-----  
Addition: 01111011000100000101110001000011110001000111100000100100110010010101011100110010011101101000010101000001001110101111101000100001000100100110111011001  
our time: 0.88497309684753418  
our time: 0.9437825679779953  
Multiplication: 010111001010101110101110010000111111001010101100100001011110000000100001110000110100111001011001000000100101001011010101110010000000010000001101001011011010111  
our time: 0.88516240119934082  
our time: 1.8063142776489258  
Square: 010001001111000001110000010001110100010000111110110001001011001111110000111000000101110000111100010110001011010000100000000100010010011101001011001001110001100111100  
our time: 0.88978676795959473  
our time: 1.8293967723846436  
our time: 0.88237767219543457  
our time: 1.023707628250122  
our time: 0.88333873748779297  
our time: 1.0492217540740967  
our time: 0.10293769836425781  
our time: 1.04155492782529277  
our time: 0.8808424946459961  
our time: 0.9863255023956299  
our time: 0.08121109008789062  
our time: 0.9670872688293457  
our time: 0.87812690734863281  
our time: 0.9660420417785645  
our time: 0.88156275749206543  
our time: 0.9845292568206787  
our time: 0.87725644111633301  
our time: 0.9519217014312744  
our time: 0.87905030250549316  
our time: 0.9654076099395752  
our time: 0.88088088035583496  
our time: 0.9764316082000732  
our time: 0.88116426739501953  
our time: 0.961775541305542  
our time: 11.903401613235474  
Degree: 001010100100001010110101011001001100100001001001101101110100001011101110011110100010001101011011000010011011010001001011010010101000100110010101111000111100011110111011100111010  
  
our time: 11.903401613235474  
Degree: 0010101001000010101101010110010011001000010010011011011110100001011101100110010000101011101001010100101000100110010101111000111011110111001111010  
our time: 0.0  
Trace: 1
```

Текст програми:

```
import time
```

```
def calculate_time(func):  
    def inner(*args, **kwargs):  
        begin = time.time()  
        result = func(*args, **kwargs)  
        print("our time: ", time.time() - begin)  
        return result  
    return inner
```

```
class GaloisNumber:  
    def __init__(self, value, field_size):  
        self.value = value  
        self.field_size = field_size  
  
    def __str__(self):  
        return str(self.value)  
  
    def __add__(self, other):  
        first_number = self.value  
        second_number = other.value  
        if len(first_number) > len(second_number):  
            size = len(first_number)  
        else:  
            size = len(second_number)  
        result_bin = ""  
        for i in range(size):  
            temp = int(first_number[i]) + int(second_number[i])  
            result_bin += str(temp % 2)
```

```
return self.__class__(result_bin, self.field_size)
```

```
@staticmethod
```

```
def left_shift(number, shift):
```

```
    num_list = list(number)
```

```
    num_list = num_list[shift:] + num_list[:shift]
```

```
    result = ''
```

```
    for i in num_list:
```

```
        result += i
```

```
    return result
```

```
@calculate_time
```

```
def matrix_create(self):
```

```
    p = 2 * self.field_size + 1
```

```
    multiplicative_matrix = [[0]*self.field_size for _ in range(self.field_size)]
```

```
    for i in range(self.field_size):
```

```
        for j in range(self.field_size):
```

```
            if (2**i + 2**j) % p == 1:
```

```
                multiplicative_matrix[i][j] = 1
```

```
            elif (2**i - 2**j) % p == 1:
```

```
                multiplicative_matrix[i][j] = 1
```

```
            elif (-1 * 2**i + 2**j) % p == 1:
```

```
                multiplicative_matrix[i][j] = 1
```

```
            elif (-1 * 2**i - 2**j) % p == 1:
```

```
                multiplicative_matrix[i][j] = 1
```

```
            else:
```

```
                multiplicative_matrix[i][j] = 0
```

```
    return multiplicative_matrix
```

```

@calculate_time
def __mul__(self, other):
    result = ''
    multiplicative_matrix = self.matrix_create()
    for z in range(self.field_size):
        result_1 = [0 for _ in range(self.field_size)]
        result_2 = 0
        pre_1 = self.left_shift(self.value, z)
        pre_2 = self.left_shift(other.value, z)

        for i in range(self.field_size):
            for j in range(self.field_size):
                result_1[i] += int(pre_1[j]) * multiplicative_matrix[j][i]
            result_1[i] = result_1[i] % 2

        for i in range(self.field_size):
            result_2 += result_1[i] * int(pre_2[i])
        result_2 = result_2 % 2
        result += str(result_2)

    return self.__class__(result, self.field_size)

```

```

@calculate_time
def __pow__(self, power, modulo=None):
    result = self * self
    for i in range(int(power.value, 2) - 2):
        result = self * result
    return self.__class__(result, self.field_size)

```

```
@calculate_time
def double_pow(self):
    return bin(int(self.value, 2) >> 1)[2:]
```

```
@calculate_time
def tr(self):
    res = 0
    for i in range(len(self.value)):
        res += int(self.value[i])
    res = res % 2
    return res
```

```
first = GaloisNumber(input(), 179)
second = GaloisNumber(input(), 179)
degree = GaloisNumber(input(), 179)
print("-----Calculating!-----")
print("Addition: ", first + second)
print("Multiplication: ", first * second)
print("Square: ", first * first)
print("Degree: ", first ** degree)
print("Trace: ", degree.tr())
```

Можна побачити, що за допомогою функції `calculate_time` був обрахований час виконання для кожної операції.

Висновки:

В ході даної лабораторної роботи я навчилася працювати з операціями у полі Галуа характеристики 2 в нормальному базисі, зробила додавання в ОНБ, піднесення до квадрату, обчислення сліду, множення та знаходження мультиплікативної матриці.