Лабораторна робота №4

Мірошніченко І.О, ФБ-05

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 «□»;

3) додавання елементів;

4) множення елементів;

5) обчислення сліду елементу;

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;

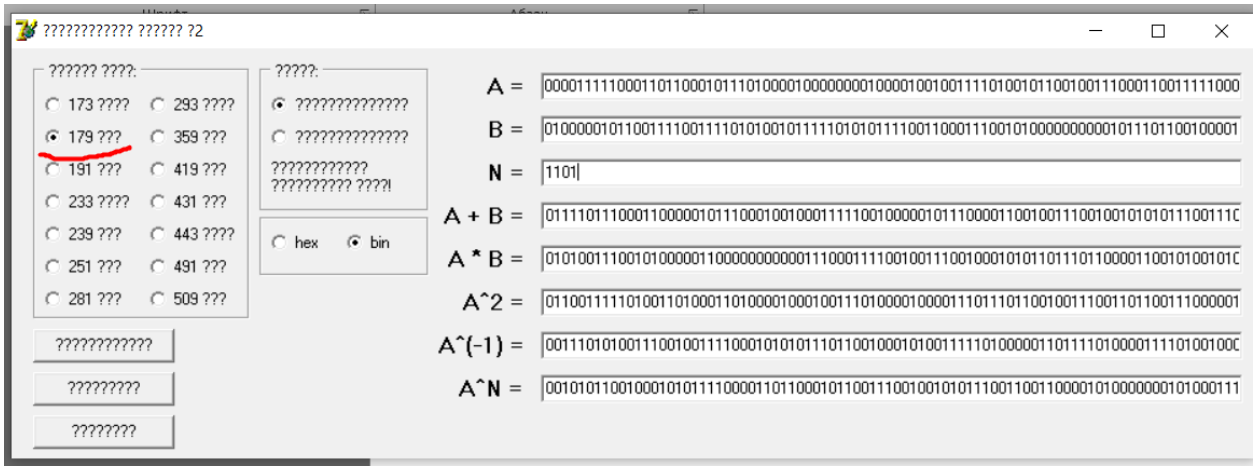
Мова програмування, семантика функцій, спосіб реалізації можуть обиратись довільно.

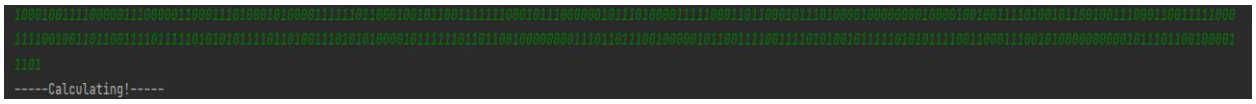
Під час конвертування елементів поля у бітові рядки потрібно враховувати конвенції щодо

зображень елементів поля (зокрема, порядок бітів).

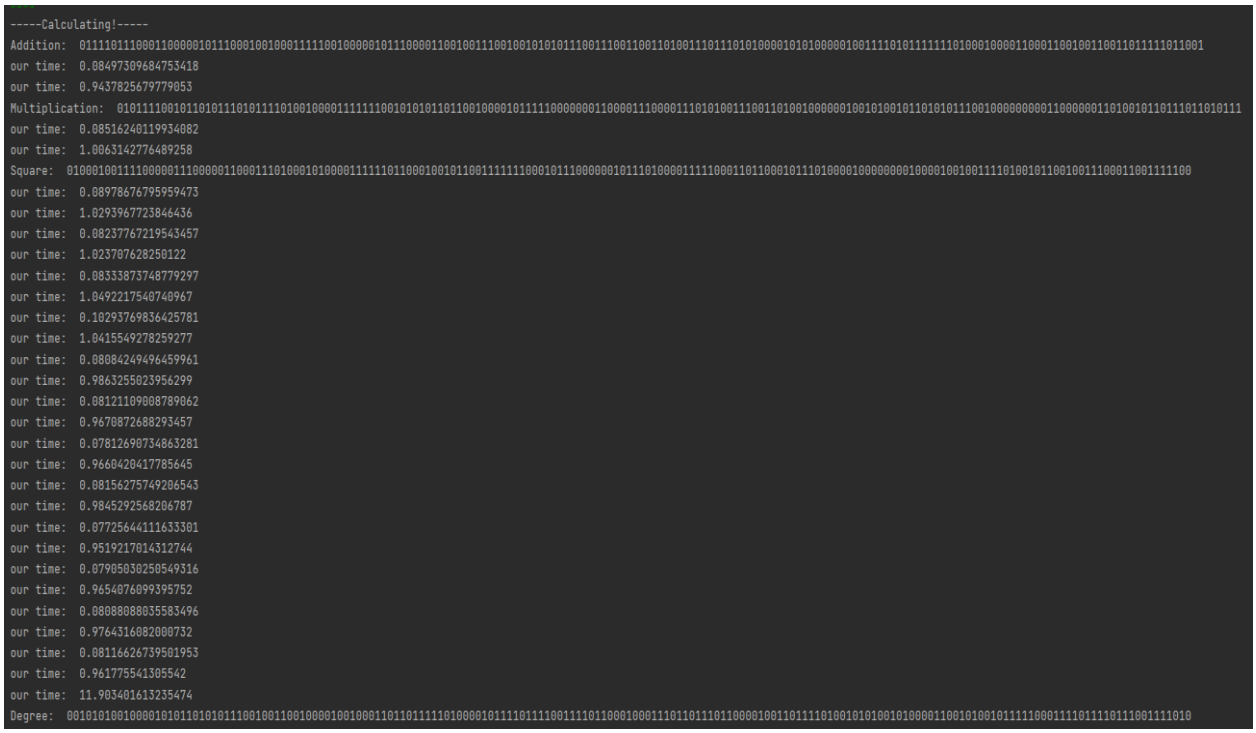
Варіант 3 = m(розмірність поля) = 179.

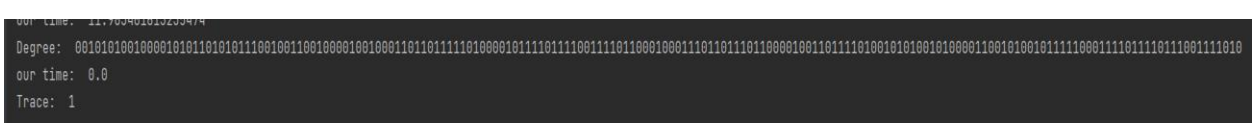
Спочатку відкриємо програму для перевірки обчислень. Візьмемо A , B, N, які згенерувала програма.





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





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.