СПЕЦІАЛЬНІ РОЗДІЛИ ОБЧИСЛЮВАЛЬНОЇ МАТЕМАТИКИ

Лабораторна 4

Терпило Софія

ФБ-06

Завдання до комп’ютерного практикуму:

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

Перевірити коректність операцій та знайти визначити середній час виконання

Хід роботи:

Мій варіант мав поле розмірності 233.

```python
def shift_cyclic_right(num, k):
    for x in range(k):
        num = num[-1] + num
        num = num[:-1]
    return num


def shift_cyclic_left(num, k):
    for x in range(k):
        num += num[0]
        num = num[1:]
    return num


def equalizes_lengths(first_num, second_num):
    if len(first_num) > len(second_num):
        while len(second_num) < len(first_num):
            second_num += '0'
    elif len(second_num) > len(first_num):
        while len(first_num) < len(second_num):
            first_num += '0'
    return first_num, second_num
```

```
def add_in_basis(first_num, second_num):
    if first_num == 0 or second_num == 0:
        return second_num if first_num == 0 else second_num
    if len(first_num) != len(second_num):
        first_num, second_num = equalizes_lengths(first_num, second_num)
    summary = ''
    for i in range(len(first_num)):
        temp = (int(first_num[i]) + int(second_num[i])) % 2
        summary += str(temp)
    return summary


def square_in_basis(num):
    return shift_cyclic_right(num, 1)


def trace_in_basis(num):
    trace = 0
    for x in range(len(num)):
        trace += int(num[x])
    trace %= 2
    return trace
```

```
def find_matrix_in_basis(dimensionality):
    p = 2 * dimensionality + 1
    matrix = []
    for i in range(dimensionality):
        matrix.append([])
        for j in range(dimensionality):
            l_i, l_j = 2 ** i, 2 ** j
            if (l_i + l_j) % p == 1 or (l_i - l_j) % p == 1 or (-l_i - l_j) % p == 1 or (-l_i + l_j) % p == 1:
                matrix[i].append(1)
            else:
                matrix[i].append(0)
    return matrix


def multiply_matrix_in_basis(u, v):
    u_length, v_length = len(u), len(v)
    if type(v) != list:
        v_length = 1
    product_matrix = []
    for i in range(v_length):
        summary = 0
        for j in range(u_length):
            if v_length == 1:
                summary += int(u[j]) * int(v[j])
            else:
                summary += int(u[j]) * int(v[i][j])
        product_matrix.append(summary % 2)
    return product_matrix
```

```python
def power_in_basis(num, pow, matrix, dimensionality):
    product = '1' * len(num)
    for i in range(len(pow)-1, -1, -1):
        if pow[i] == 1:
            product = multiply_in_basis(product, num, matrix, dimensionality)
        num = square_in_basis(num)
    return product


def inverse_in_basis(num, matrix, dimensionality):
    m_1 = str(bin(dimensionality - 1))[2:]  # m - 1
    inverse = num
    k = 1
    for i in range(1, len(m_1)):
        temp = inverse
        inverse = shift_cyclic_right(inverse, k)
        inverse = multiply_in_basis(inverse, temp, matrix, dimensionality)
        k *= 2
        if m_1[i] == 1:
            inverse = multiply_in_basis(num, square_in_basis(inverse), matrix, dimensionality)

            k += 1
    inverse = square_in_basis(inverse)
    return inverse


matrix = find_matrix_in_basis(233)
```

Контроль коректності:

a =
'11011001001001010011110101110001010100110101110010100100101111000110000110100100001011101001010010001110110001010010000111000011001111100111010111110010111101100001000100101010101111001001101000001110011100001100101010001110101010100'

b =
'10000011000111001001110000100000011011000011111110000111011010011001001001000111100010001010000101011001101100100100110101101001010110100001000111001000100111111111000011110010111000000110110010111001001101111011001010001000010101100'

c =
'11010110111001111101001001010010110111110111110001000011110111111001000000100101111111101100110010100011101111101011101111000110001001010100001010111101010001101101010010000011111111110101101110001011010010010110101100010001100011111101'

```python
# TEST: (a + b) * c = a * c + c * b
sum1_1 = add_in_basis(a, b)
prod1_1 = multiply_in_basis(c, sum1_1, matrix, 233)

prod2_1 = multiply_in_basis(a, c, matrix, 233)
prod2_2 = multiply_in_basis(b, c, matrix, 233)
sum2_1 = add_in_basis(prod2_1, prod2_2)
if prod1_1 == sum2_1:
    print('TEST : (a + b) * c = a * c + c * b => passed')
```

```
main
C:\Users\User\PycharmProjects\srom_lab4\venv\Scripts\python.exe C:\Users\User\Pychar
TEST : (a + b) * c = a * c + c * b => passed
```

Час виконання операцій:

```
addition => 0.00019599999999997397
multiplication => 3.1777251
power => 0.00011399999999994748
trace => 3.439999999965693e-05
inverse => 22.2256384
```