Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**Дисциплина: Криптографические протоколы**

Работу выполнила: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Д. Н. Баева

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Крамаренко

**Цель работы:** реализовать интерактивный калькулятор в поле Галуа с заданным пользователем образующим многочленом (сложение, умножение, деление, НОД, возведение в степень и таблица умножения).

**Ход работы:**

Класс GaloisFieldCalculator реализует операции над многочленами в поле Галуа. Реализация калькулятора состоит из применения следующих функций:

1) \_\_init\_\_(self, generator\_polynomial): это конструктор класса. Принимает один аргумент generator\_polynomial, который представляет собой генераторный многочлен. Далее инициализируется атрибут self.generator\_polynomial значением переданного генераторного многочлена.

2) Функция add(self, a, b) выполняет сложение многочленов a и b в поле Галуа. Возвращает результат (a + b) % P, дополняет многочлены a и b нулями до одинаковой длины. Затем поэлементно складывает коэффициенты многочленов и берет остаток от деления на образующий многочлен. Соответствующий код представлен на рисунке 1.

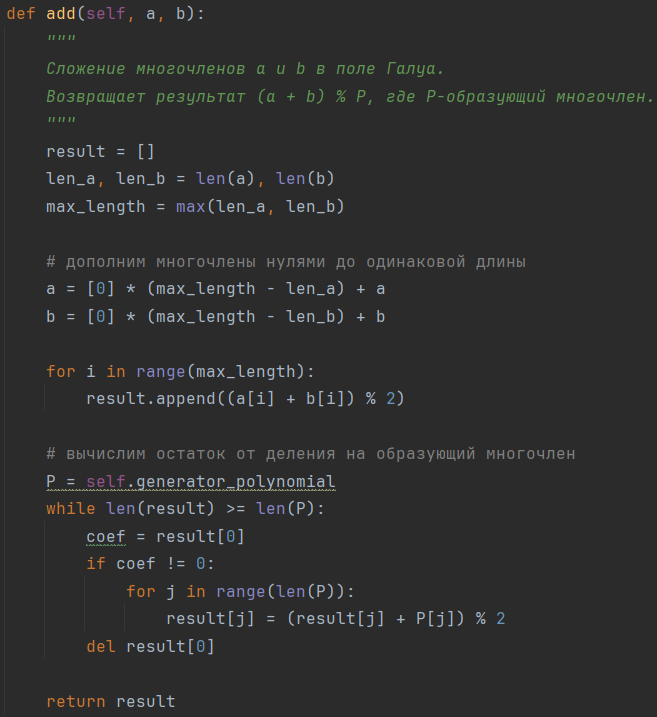


Рисунок 1 – Сложение многочленов.

3) Функция multiply(self, a, b) выполняет умножение многочленов a и b в поле Галуа. Возвращает результат (a \* b) % P. Создает многочлен result нулевой длины, достаточной для хранения произведения. Затем поэлементно перемножает коэффициенты многочленов a и b суммируя результаты и беря остаток от деления на образующий многочлен. Функция представлена на рисунке 2.

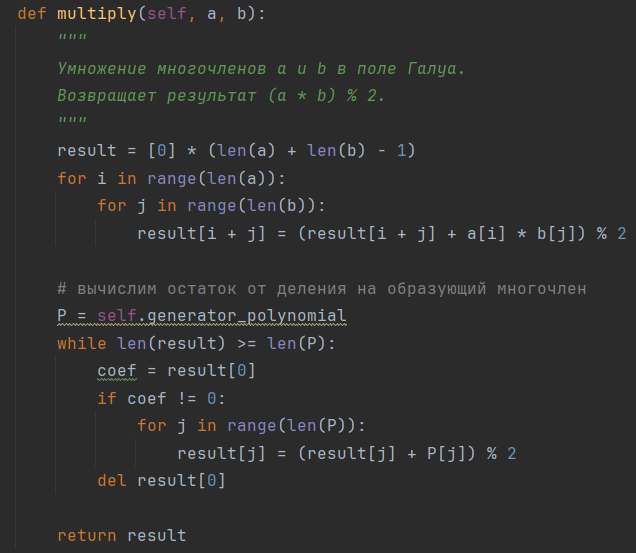


Рисунок 2 – Деление многочлена на многочлен.

4) Функция divide(self, a, b) выполняет деление многочленов a и b в поле Галуа. Возвращает частное (a // b) % P. Убирает ведущие нули у многочленов a и b и нормализует многочлены, деля их на первый коэффициент normalizer. Инициализирует ответ res нулями. Делит многочлены столбиком, обновляя коэффициенты частного. Если коэффициент не равен нулю, выполняет дополнительные операции. Фрагмент кода представлен на рисунке 3.

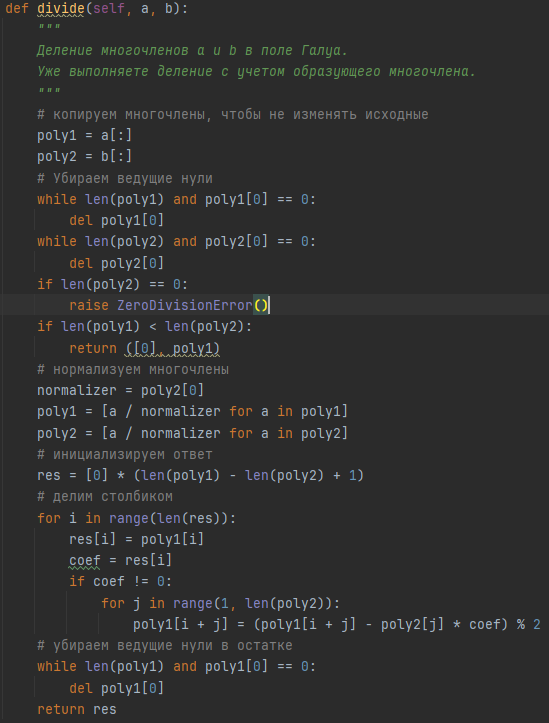


Рисунок 3 – Деление многочлена на многочлен.

5) Функция power(self, a, n) возводит многочлен a в степень n в поле Галуа. Возвращает результат (a \*\* n) % P. Инициализирует результат единицей [1]. Затем n раз умножает результат на многочлен a. Это отображено на рисунке 4.

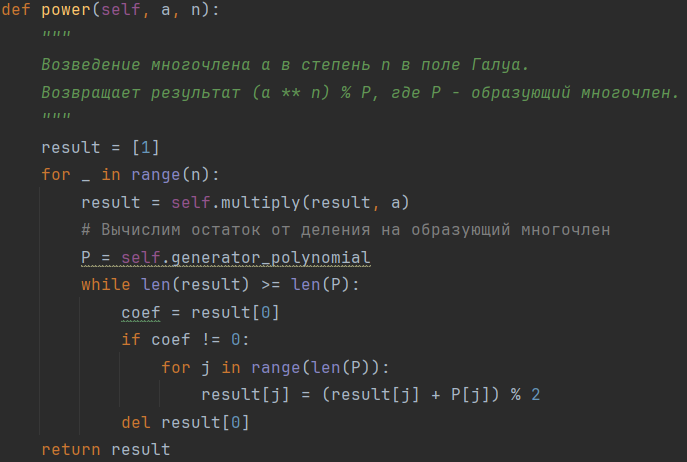


Рисунок 4 – Возведение многочлена в натуральную степень.

6) Функция multiplication\_table(self) создает таблицу умножения для элементов в поле Галуа. Возвращает двумерный список table. Для каждой пары элементов i и j (принимая, что поле 8-битное), вычисляет произведение многочленов [i] и [j] и добавляет его в соответствующую строку row. Соответствующий код представлен на рисунке 5.

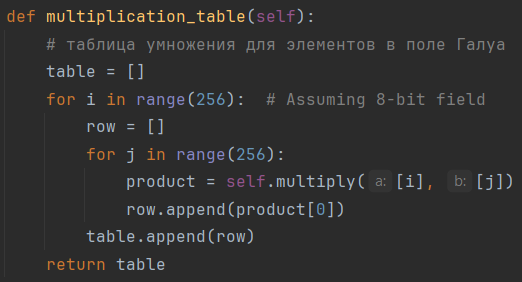


Рисунок 5 – Таблица умножения для элементов в поле Галуа.

В финале программы следует методы, запрашивающие у пользователя коэффициенты для образующего многочлена, многочлена f1, многочлена f2 и степени, в которые необходимо их возвести. После следует вызов соответствующих функций и вывода их результатов.

**Листинг программы**

Файл LW\_3.py

class GaloisFieldCalculator:

def \_\_init\_\_(self, generator\_polynomial):

self.generator\_polynomial = generator\_polynomial

def add(self, a, b):

"""

Сложение многочленов a и b в поле Галуа.

Возвращает результат (a + b) % P, где P-образующий многочлен.

"""

result = []

len\_a, len\_b = len(a), len(b)

max\_length = max(len\_a, len\_b)

# дополним многочлены нулями до одинаковой длины

a = [0] \* (max\_length - len\_a) + a

b = [0] \* (max\_length - len\_b) + b

for i in range(max\_length):

result.append((a[i] + b[i]) % 2)

# вычислим остаток от деления на образующий многочлен

P = self.generator\_polynomial

while len(result) >= len(P):

coef = result[0]

if coef != 0:

for j in range(len(P)):

result[j] = (result[j] + P[j]) % 2

del result[0]

return result

def multiply(self, a, b):

"""

Умножение многочленов a и b в поле Галуа.

Возвращает результат (a \* b) % 2.

"""

result = [0] \* (len(a) + len(b) - 1)

for i in range(len(a)):

for j in range(len(b)):

result[i + j] = (result[i + j] + a[i] \* b[j]) % 2

# вычислим остаток от деления на образующий многочлен

P = self.generator\_polynomial

while len(result) >= len(P):

coef = result[0]

if coef != 0:

for j in range(len(P)):

result[j] = (result[j] + P[j]) % 2

del result[0]

return result

def divide(self, a, b):

"""

Деление многочленов a и b в поле Галуа.

Уже выполняете деление с учетом образующего многочлена.

"""

# копируем многочлены, чтобы не изменять исходные

poly1 = a[:]

poly2 = b[:]

# Убираем ведущие нули

while len(poly1) and poly1[0] == 0:

del poly1[0]

while len(poly2) and poly2[0] == 0:

del poly2[0]

if len(poly2) == 0:

raise ZeroDivisionError()

if len(poly1) < len(poly2):

return ([0], poly1)

# нормализуем многочлены

normalizer = poly2[0]

poly1 = [a / normalizer for a in poly1]

poly2 = [a / normalizer for a in poly2]

# инициализируем ответ

res = [0] \* (len(poly1) - len(poly2) + 1)

# делим столбиком

for i in range(len(res)):

res[i] = poly1[i]

coef = res[i]

if coef != 0:

for j in range(1, len(poly2)):

poly1[i + j] = (poly1[i + j] - poly2[j] \* coef) % 2

# убираем ведущие нули в остатке

while len(poly1) and poly1[0] == 0:

del poly1[0]

return res

def power(self, a, n):

"""

Возведение многочлена a в степень n в поле Галуа.

Возвращает результат (a \*\* n) % P, где P - образующий многочлен.

"""

result = [1]

for \_ in range(n):

result = self.multiply(result, a)

# Вычислим остаток от деления на образующий многочлен

P = self.generator\_polynomial

while len(result) >= len(P):

coef = result[0]

if coef != 0:

for j in range(len(P)):

result[j] = (result[j] + P[j]) % 2

del result[0]

return result

def multiplication\_table(self):

# таблица умножения для элементов в поле Галуа

table = []

for i in range(256): # Assuming 8-bit field

row = []

for j in range(256):

product = self.multiply([i], [j])

row.append(product[0])

table.append(row)

return table

# ввод коэффициентов для образующего многочлена

gp\_coeffs = input("Введите коэффициенты для образующего многочлена через запятую: ")

generator\_polynomial = [int(coeff) for coeff in gp\_coeffs.split(",")]

calculator = GaloisFieldCalculator(generator\_polynomial)

# ввод коэффициентов для многочлена a

f1\_coeffs = input("Введите коэффициенты для многочлена f1 через запятую: ")

f1 = [int(coeff) for coeff in f1\_coeffs.split(",")]

# ввод коэффициентов для многочлена b

f2\_coeffs = input("Введите коэффициенты для многочлена f2 через запятую: ")

f2 = [int(coeff) for coeff in f2\_coeffs.split(",")]

# операции

sum\_result = calculator.add(f1, f2)

product\_result = calculator.multiply(f1, f2)

division\_result = calculator.divide(f1, f2)

print("Введите степень в которую нужно возвести многочлен f1: ")

k\_f1 = int(input())

print("Введите степень в которую нужно возвести многочлен f2: ")

k\_f2 = int(input())

power\_result\_f1 = calculator.power(f2, k\_f1)

power\_result\_f2 = calculator.power(f2, k\_f2)

multiplication\_table = calculator.multiplication\_table()

print("Сумма:", sum\_result)

print("Произведение:", product\_result)

print("Деление:", division\_result)

print("Возведение в степень многочлена f1:", power\_result\_f1)

print("Возведение в степень многочлена f2:", power\_result\_f2)

# print("Таблица умножения:")

# for row in multiplication\_table:

# print(row)