МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Арифметические операции в числовых полях

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ»

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Мызникова Сергея Анатольевича

Преподаватель, профессор ___________ В.А. Молчанов

подпись, дата

Саратов 2024

СОДЕРЖАНИЕ

1 Постановка задачи

Цель работы — изучение основных операции в числовых полях и их программная реализация.

Порядок выполнения работы:

1. Разобрать алгоритмы Евклида (обычный, бинарный и расширенный) вычисления НОД целых чисел и привести их программную реализацию.

2. Разобрать алгоритмы решения систем сравнений и привести их программную реализацию.

3. Рассмотреть метод Гаусса решения систем линейных уравнений над конечными полями и привести его программную реализацию.

2 Теоретические сведения

2.1 Алгоритм Евклида

Пусть даны целые числа $x_1 > x_2 > 0$. Для вычисления наибольшего общего делителя (x_1, x_2) существует алгоритм Евклида

$$r_{-1} = x_1;$$

$$r_0 = x_2;$$

$$r_{i-2} = d_i * r_{i-1} + r_i,\ 0 < r_i < r_{i-1},\ i = 1, \ldots, k;$$

$$r_{k-1} = d_{k+1} * r_k.$$

Тогда НОД$(x_1, x_2) = r_k$ и для его нахождения требуется выполнить $k + 1$ делений с остатком. Так как остатки выполняемых делений образуют строго убывающую последовательность, то этот процесс обязательно остановится в результате получения нулевого остатка деления.

2.2 Расширенный алгоритм Евклида

Если в алгоритме Евклида на каждом шаге вычислять кроме частного d_i и остатка r_i еще два значения u_i и v_i по правилу:

$$u_{-1} = 1,\ u_0 = 0;$$

$$v_{-1} = 0,\ v_0 = 1;$$

$$u_i = u_{i-2} - d_i * u_{i-1},\ i = 1, \ldots, k;$$

$$v_i = v_{i-2} – d_i * v_{i-1},\ i = 1, \ldots, k.$$

то такой алгоритм будет называться расширенным алгоритмом Евклида. Его суть состоит в том, чтобы дать линейное разложение наибольшего общего делителя $u_k * x_1 + v_k * x_2 = r_k =$ НОД(x_1, x_2). Данное уравнение играет важную роль в операциях модульной арифметики.

2.3 Бинарный алгоритм Евклида

Бинарный алгоритм Евклида – это ускоренный алгоритм для поиска наибольшего общего делителя двух чисел. Он основан на следующих свойствах:

• НОД(2·a, 2·b) = 2·НОД(a, b);

• НОД(2·a, 2·b + 1) = НОД(a, 2·b + 1);

• НОД(-a, b) = НОД(a, b).

2.4 Греко-китайская теорема об остатках

Пусть m_1, m_2, …, m_k - попарно взаимно простые целые числа и $M = m_1 m_2 \ldots m_k$. Тогда система линейных сравнений

$$\begin{cases} x \equiv a_1 (mod\, m_1) \\ x \equiv a_2 (mod\, m_2) \\ \quad\quad \ldots \\ x \equiv a_k (mod\, m_k) \end{cases}$$

имеет единственное неотрицательное решение по модулю M.

При этом, если для каждого $1 \le j \le k$ число $M_j = \frac{M}{m_j}$ и сравнение $M_j * x = a_j$ $(mod\ m_j)$ имеет решение z_j, то решением системы линейных сравнений является остаток по модулю M числа $x = M_1 z_1 + M_2 z_2 + \ldots + M_k z_k$.

2.5 Алгоритм Гарнера

Пусть все $m_i > 1$, $m = m_1 * \ldots * m_t$. Тогда любое число $0 \le x < m$ может быть однозначным образом представлено в виде

$x = x_0 + x_1 m_1 + x_2 m_1 m_2 + ... + x_{t-1} m_1 m_2 * ... * m_{t-1}$,

где $0 \le x_i < m_{i+1}$, $i = 0, 1, \ldots, t-1$.

Для x_i верно соотношение

$$x_i = \frac{r_i + 1 - (x_0 + x_1 m_1 + \dots + x_{i-1} m_i m_{i-1})}{m_1 * \dots * m_i} (mod\ m_{i+1})$$

Таким образом, x_i могут быть вычислены один за другим. Получившийся алгоритм носит имя Гарнера.

2.6 Метод Гаусса

Системой m линейных уравнений с n неизвестными $x_1, \dots, x_n$ называется выражение вида

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$

Решение систем осуществляется с помощью преобразований, которые сохраняют множество решений системы и поэтому называются равносильными.

Лемма 1. Следующие элементарные преобразования сохраняют множество решений любой системы линейных уравнений:

a) удаление из системы тривиальных уравнений;

b) умножение обеих частей какого-либо уравнения на одно и тот же ненулевой элемент поля;

c) прибавление к обеим частям какого-либо уравнения системы соответствующих частей другого уравнения системы.

Метод решения системы заключается в равносильном преобразовании ее в систему линейных уравнений с противоречивым уравнением или в разрешенную систему линейных уравнений вида:

$$\begin{cases} x_1 + \dots + \dots + a'_{1,r+1}x_{r+1} + \dots + a'_{1n}x_n = b'_1 \\ x_2 + \dots + a'_{2,r+1}x_{r+1} + \dots + a'_{2n}x_n = b'_2 \\ \dots \\ x_r + a'_{r,r+1}x_{r+1} + \dots + a'_{rn}x_n = b'_r \end{cases}$$

где $r \leq m$, так как в процессе элементарных преобразований исходной системы удаляются тривиальные уравнения. В этом случае неизвестные $x_1, \ldots, x_r$ называются разрешенными (или базисными) и $x_{r+1}, \ldots, x_n$ – свободными.

Преобразование системы в равносильную ей разрешенную систему осуществляется по методу Гаусса с помощью последовательного выполнения следующих Жордановых преобразований:

1) выбираем один из коэффициентов системы $a_{ij} \neq 0$;
2) умножаем i-ое уравнение системы на элемент a_{ij}^{-1};
3) прибавляем к обеим частям остальных k-ых уравнений системы соответствующие части нового i-ого уравнения, умноженные на коэффициент – a_{kj};
4) удаляем из системы тривиальные уравнения.

При этом выбранный ненулевой элемент a_{ij} называется разрешающим, строка и столбец, содержащие элемент a_{ij} также называются разрешающими.

3 Результаты работы

3.1 Алгоритмы: описание, псевдокод и временная сложность

3.1.1 Обычный алгоритм Евклида

Вход: два целых числа a, b.

Выход: НОД(a, b)

Шаг 1: пока a не равно 0 и b не равно 0 выполнить:

Шаг 1.1: если a > b, то положить a = a mod b и вернуться к шагу 1

Шаг 1.2: иначе положить b = b mod a и вернуться к шагу 1

Шаг 2: вывести a + b

Псевдокод:

```
Начало алгоритма
   Пока a не равно 0 и b не равно 0 выполнить:
      Если a больше b выполнить:
         a = a mod b
      Иначе выполнить:
         b = b mod a
   Вернуть a + b
Конец алгоритма
```

Временная сложность алгоритма. O(L(a)*L(b))

3.1.2 Расширенный алгоритм Евклида

Вход: два целых числа a, b.

Выход: НОД(a, b)

Шаг 1: положить x, xx, y, yy равными 1, 0, 0, 1

Шаг 2: пока b не равен 0 выполнить:

Шаг 2.1: положить q равным a // b

Шаг 2.2: положить a и b равными b и a mod b соответственно

Шаг 2.3: положить x и xx равными xx, x – xx*q

Шаг 2.4: положить y, yy равными yy, y – yy*q и вернуться к шагу 2

Шаг 3: вернуть a, x, y

Псевдокод:

```
Начало алгоритма
    Инициализировать x = 1, xx = 0, y = 0, yy = 1
    Пока b не равно 0 выполнить:
        q = целая часть от a / b
        a = b
        b = a mod b
        x = xx
        xx = x - xx * q
        y = yy
        yy = y - yy * q
    Вернуть a, x, y
Конец алгоритма
```

Временная сложность алгоритма. O(L(a)*L(b))

3.1.3 Бинарный алгоритм Евклида

Вход: два целых числа a, b.

Выход: НОД(a, b)

Шаг 1: если a = 0, вернуть b;

Шаг 2: если a = b или b = 0, вернуть a;

Шаг 3: если a = 1 или b = 1, вернуть 1;

Шаг 4: Если a и b четные, то вернуть 2*НОД(a/2, b/2);

Шаг 5: Если a четное и b нечетное, то вернуть НОД(a/2, b);

Шаг 6: Если a нечетное и b четное, то вернуть НОД(a, b/2);

Шаг 7: Если a и b нечетные, то выполнить:

Шаг 7.1: если b > a, то вернуть НОД((b-a)/2, a);

Шаг 7.2: иначе вернуть НОД((a-b)/2, b).

Псевдокод:

Начало алгоритма

Функция binary_Euclid(a, b)

    Если a равно 0 выполнить:

        Вернуть b

    Иначе если b равно 0 или a равно b выполнить:

        Вернуть a

    Иначе если a равно 1 или b равно 1 выполнить:

        Вернуть 1

    Иначе если a и b четные выполнить:

        Вернуть 2 * binary_Euclid (a / 2, b / 2)

    Иначе если a четное и b нечетное выполнить:

        Вернуть binary_Euclid (a / 2, b)

Иначе если a нечетное и b четное выполнить:

Вернуть binary_Euclid (a, b / 2)

Иначе если a и b нечетные выполнить:

Если b больше a выполнить:

Вернуть binary_Euclid ((b - a) / 2, a)

Иначе выполнить:

Вернуть binary_Euclid ((a - b) / 2, b)

Конец функции

Конец алгоритма

Временная сложность алгоритма. O(L(a)*L(b))

3.1.4 Греко-китайская теорема об остатках

Вход: список коэффициентов и список модулей

Выход: решение системы

Шаг 1: найти произведение всех модулей и записать в M

Шаг 2: создать список с элементами M / m[i]

Шаг 3: найти все решения z_j сравнений $M_j * x = a_j \pmod{m_j}$ и добавить в список z

Шаг 4: вычислить $x = M_1 z_1 + M_2 z_2 + \ldots + M_k z_k$

Псевдокод:

Начало алгоритма

Инициализировать M = 1

Для каждого числа num в m:

M = M * num

Инициализировать M_j как пустой список

Для каждого i от 0 до длины m:

Добавить M / m[i] в M_j

Инициализировать z как пустой список

Для каждого i от 0 до длины M_j:

Вычислить обратное по модулю mod_inverse(M_j[i], m[i])

Добавить (mod_inverse(M_j[i], m[i]) * a[i]) % m[i] в z

Инициализировать x = 0

Для каждого i от 0 до длины z:

x = x + M_j[i] * z[i]

Вернуть x % M

Конец алгоритма

Временная сложность алгоритма. $O(k^2 * \log^2(m_i) + k * \log^2(m_i))$, k – количество m_i

3.1.5 Алгоритм Гарнера

Вход: список коэффициентов и список модулей

Выход: решение системы

Шаг 1: для i = 2…k выполнить следующие действия

Шаг 1.1: установить $c_i = 1$

Шаг 1.2: для j = 1…i-1 выполнить

Шаг 1.2.1: установить $u = m_j^{-1} (\mathrm{mod}\ m_i)$

Шаг 1.2.2: установить $c_i = uc_i\ (\mathrm{mod}\ m_i)$

Шаг 2: установить $u = a_1$

Шаг 3: установить x = u

Шаг 4: для i = 2…k вычислить u = (a_i – x)c_i и x = x + u$\prod_{j=1}^{i-1} m_j$

Псевдокод:

Начало алгоритма

Инициализировать список с длиной m, заполненный нулями

Для каждого i от 1 до длины m:

c[i] = 1

Для каждого j от 0 до i:

Вычислить u как обратное по модулю mod_inverse(m[j], m[i])

c[i] = (u * c[i]) % m[i]

Инициализировать u как a[0]

Инициализировать x как u

Для каждого i от 1 до длины m:

Вычислить u = (a[i] - x) * c[i]

Если u меньше 0:

u = u + m[i]

u = u % m[i]

Инициализировать m_j = 1

Для каждого j от 0 до i:

m_j = m_j * m[j]

x = x + u * m_j

Вернуть x

Конец алгоритма

Временная сложность алгоритма. $O(m^2*L(m_i)*L(m_j) + m^2*L(m_i)*L(m_j))$

3.1.6 Метод Гаусса

Вход: матрица вместе со столбцом свободных членов, модуль, недостающие корни, если нужно

Выход: частное решение

Шаг 1: найти строку с ненулевым ведущим элементом

Шаг 2: поменять строки местами так, чтобы ведущая строка была выше

Шаг 3: привести ведущий элемент в строке к 1

Шаг 4: для каждой строки найти определить, насколько нужно умножить найденный ведущий элемент, чтобы обнулить в остальных строках

Шаг 5: обнулить все элементы в текущем столбце (помимо ведущего)

Шаг 6: удалить нулевые строки, если имеются

Шаг 7: увеличить индекс строки для следующей итерации

Шаг 8: обновить количество строк

Псевдокод:

Начало алгоритма

```
    n = количество строк в matrix
    m = количество столбцов в matrix - 1
    row_idx = 0
    Для каждого col_idx от 0 до m:
        pivot_row = None
        Для каждого i от row_idx до n:
```

Если элемент matrix[i][col_idx] не равен 0:

pivot_row = i

Прервать цикл

Если pivot_row равен None:

Перейти к следующей итерации цикла

Если pivot_row не равен row_idx:

Поменять местами строки matrix[row_idx] и matrix[pivot_row]

inv = обратное по модулю значение matrix[row_idx][col_idx] относительно mod

Для каждого k от 0 до m + 1:

matrix[row_idx][k] = (matrix[row_idx][k] * inv) % mod

Для каждого i от 0 до n:

Если i не равен row_idx и элемент matrix[i][col_idx] не равен 0:

factor = matrix[i][col_idx]

Для каждого k от 0 до m + 1:

matrix[i][k] = (matrix[i][k] - factor * matrix[row_idx][k]) % mod

matrix = список строк из matrix, где хотя бы один элемент в строке не равен 0

Увеличить row_idx на 1

n = количество строк в обновленной matrix

Вернуть matrix

Конец алгоритма

Временная сложность алгоритма. $O(m^2*n*log(mod))$

3.2 Тестирование алгоритмов

Обычный алгоритма Евклида

```
Выберите алгоритм:
1. Обычный алгоритм Евклида
2. Расширенный алгоритм Евклида
3. Бинарный алгоритм Евклида
4. Метод Гаусса
5. Алгоритм Гарнера
6. Китайская теорема об остатках
7. Выход
Ваш выбор: 1
Введите первое число: 132
Введите второе число: 516
НОД(132, 516) = 12
```

Рисунок 1. Тестирование обычного алгоритма Евклида

Расширенный алгоритм Евклида

```
Выберите алгоритм:
1. Обычный алгоритм Евклида
2. Расширенный алгоритм Евклида
3. Бинарный алгоритм Евклида
4. Метод Гаусса
5. Алгоритм Гарнера
6. Китайская теорема об остатках
7. Выход
Ваш выбор: 2
Введите первое число: 1016
Введите второе число: 666
Расширенный алгоритм Евклида для чисел 1016 и 666:
НОД(1016, 666) = 2
Линейное разложение: 2 = -137 * 1016 + 209 * 666
```

Рисунок 2. Тестирование расширенного алгоритма Евклида

Бинарный алгоритм Евклида

```
Выберите алгоритм:
1. Обычный алгоритм Евклида
2. Расширенный алгоритм Евклида
3. Бинарный алгоритм Евклида
4. Метод Гаусса
5. Алгоритм Гарнера
6. Китайская теорема об остатках
7. Выход
Ваш выбор: 3
Введите первое число: 678
Введите второе число: 918
НОД(678, 918) с помощью бинарного алгоритма = 6.0
```

Рисунок 3. Тестирование бинарного алгоритма Евклида

Метод Гаусса

```
Выберите алгоритм:
1. Обычный алгоритм Евклида
2. Расширенный алгоритм Евклида
3. Бинарный алгоритм Евклида
4. Метод Гаусса
5. Алгоритм Гарнера
6. Китайская теорема об остатках
7. Выход
Ваш выбор: 4
Введите количество строк матрицы: 3
Введите количество столбцов матрицы (включая правую часть): 4
Введите строку 1 через пробел: 2 3 4 5
Введите строку 2 через пробел: 1 1 1 6
Введите строку 3 через пробел: 3 4 5 7
Введите модуль для расчетов: 7
Решение системы методом Гаусса:
1*x1 + 0*x2 + 6*x3 = 6
0*x1 + 1*x2 + 2*x3 = 0
Введите значение для частной переменной x3: 3
Частное решение:
x1 = 2
x2 = 1
x3 = 3
```

Рисунок 4. Тестирование метода Гаусса

Алгоритм Гарнера

```
Выберите алгоритм:
1. Обычный алгоритм Евклида
2. Расширенный алгоритм Евклида
3. Бинарный алгоритм Евклида
4. Метод Гаусса
5. Алгоритм Гарнера
6. Китайская теорема об остатках
7. Выход
Ваш выбор: 5
Введите список коэффициентов a через пробел: 1 3 2
Введите список модулей m через пробел: 3 5 4
Решение системы по алгоритму Гарнера: x = 58
```

Рисунок 5. Тестирование алгоритма Гарнера

Греко-китайская теорема об остатках

```
Выберите алгоритм:
1. Обычный алгоритм Евклида
2. Расширенный алгоритм Евклида
3. Бинарный алгоритм Евклида
4. Метод Гаусса
5. Алгоритм Гарнера
6. Китайская теорема об остатках
7. Выход
Ваш выбор: 6
Введите список коэффициентов a через пробел: 1 3 2
Введите список модулей m через пробел: 3 5 4
Решение системы по Китайской теореме об остатках: x = 58.0
```

Рисунок 6. Тестирование греко-китайской теоремы об остатках

ЗАКЛЮЧЕНИЕ

В ходе работы били рассмотрены алгоритмы Евклида (обычный, расширенный, бинарный), некоторые из алгоритмов для решения систем сравнений, а именно, греко-китайская теорема об остатках и алгоритм Гарнера. Также был рассмотрен алгоритм Гаусса для решения систем линейных уравнений над конечным полем.

Для проверки работоспособности этих алгоритмов была реализована программа на языке Python. Программы были протестированы и была подтверждена правильность их работы. Также была произведена оценка временной сложности алгоритмов, приведены их псевдокоды.

ПРИЛОЖЕНИЕ А

Программа с реализованными алгоритмами

```
# Обычный алгоритм Евклида
def normal_Euclid(a, b):
    while a !=0 and b != 0:
        if a > b:
            a = a % b
        else:
            b = b % a
    return a + b
# Расширенный алгоритм Евклида
def extended_Euclid(a, b):
    x, xx, y, yy = 1, 0, 0, 1
    while b:
        q = a // b
        a, b = b, a % b
        x, xx = xx, x - xx*q
        y, yy = yy, y - yy*q
    return a, x, y
# Бинарный алгоритм Евклида
def binary_Euclid(a, b):
    if a == 0:
        return b
    elif b == 0 or a == b:
        return a
    elif a == 1 or b == 1:
        return 1
```

```
    elif a % 2 == 0 and b % 2 == 0:
        return 2 * binary_Euclid(a / 2, b / 2)
    elif a % 2 == 0 and b % 2 != 0:
        return binary_Euclid(a / 2, b)
    elif a % 2 != 0 and b % 2 == 0:
        return binary_Euclid(a, b / 2)
    elif a % 2 != 0 and b % 2 != 0:
        if b > a:
            return binary_Euclid((b - a) / 2, a)
        else:
            return binary_Euclid((a - b) / 2, b)
# Поиск обратного элемента
def mod_inverse(z, m):
    gcd, x, _ = extended_Euclid(z, m)
    return inv_elem(x, m)
def inv_elem(x, m):
    if x < 0:
        x = x + m
    return x % m
inv_elem(-5, 4)
# Китайская теорема об остатках
def chinese_theorem(a, m):
    M = 1
    for num in m:
        M *= num
    M_j = []
    for i in range(len(m)):
```

```
        M_j.append(M / m[i])
    z = []
    for i in range(len(M_j)):
        z.append((mod_inverse(M_j[i], m[i]) * a[i]) % m[i])
    x = 0
    for i in range(len(z)):
        x += M_j[i] * z[i]
    return x % M
# Алгоритм Гарнера
def Garner_algorithm(a, m):
    c = [0] * len(m)
    for i in range(1, len(m)):
        c[i] = 1
        for j in range(i):
            u = mod_inverse(m[j], m[i])
            c[i] = (u * c[i]) % m[i]
    u = a[0]
    x = u
    for i in range(1, len(m)):
        u = (a[i] - x) * c[i]
        if u < 0:
            u += m[i]
        u %= m[i]
        m_j = 1
        for j in range(i):
            m_j *= m[j]
        x += u * m_j
```

```
    return x
# Метод Гаусса
def gauss_jordan_modul(matrix, mod):
    n = len(matrix)
    m = len(matrix[0]) - 1
    row_idx = 0
    for col_idx in range(m):
        # Найдем строку с ненулевым ведущим элементом
        pivot_row = None
        for i in range(row_idx, n):
            if matrix[i][col_idx] != 0:
                pivot_row = i
                break
        if pivot_row is None:
            continue
        # Меняем строки местами, если нужно
        if pivot_row != row_idx:
            matrix[row_idx],   matrix[pivot_row]   =   matrix[pivot_row],
matrix[row_idx]
        # Приведение ведущего элемента к единице
        inv = mod_inverse(matrix[row_idx][col_idx], mod)
        for k in range(m + 1):
            matrix[row_idx][k] = (matrix[row_idx][k] * inv) % mod
        # Обнуление всех элементов в текущем столбце, кроме ведущего
        for i in range(n):
            if i != row_idx and matrix[i][col_idx] != 0:
                factor = matrix[i][col_idx]
```

```
                for k in range(m + 1):
                    matrix[i][k] = (matrix[i][k] - factor * matrix[row_idx][k]) % mod
        # Удаление нулевых строк
        matrix = [row for row in matrix if any(row[:-1])]
        # Увеличиваем индекс строки для следующей итерации
        row_idx += 1
        # Обновляем количество строк n, так как могли удалить нулевые строки
        n = len(matrix)
    return matrix
# Приведение матрицы к виду для поля
def reverse_matrix(matrix, m):
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            matrix[i][j] = inv_elem(matrix[i][j], m)
    return matrix
def algorithm_interface():
    def input_numbers():
        a = int(input("Введите первое число: "))
        b = int(input("Введите второе число: "))
        return a, b
    def input_chinese_theorem():
        a = list(map(int, input("Введите список коэффициентов a через пробел: ").split()))
        m = list(map(int, input("Введите список модулей m через пробел: ").split()))
        return a, m
```

```
def input_matrix():
    rows = int(input("Введите количество строк матрицы: "))
    cols = int(input("Введите количество столбцов матрицы (включая правую часть): "))
    matrix = []
    for i in range(rows):
        row = list(map(int, input(f"Введите строку {i + 1} через пробел: ").split()))
        matrix.append(row)
    mod = int(input("Введите модуль для расчетов: "))
    return matrix, mod
def explain_extended_Euclid(a, b, gcd, x, y):
    explanation = f"Расширенный алгоритм Евклида для чисел {a} и {b}:\nНОД({a}, {b}) = {gcd}\n"
    explanation += f"Линейное разложение: {gcd} = {x} * {a} + {y} * {b}\n"
    return explanation
def explain_gauss(matrix, mod):
    print("Решение системы методом Гаусса:")
    num_vars = len(matrix[0]) - 1
    num_rows = len(matrix)
    for row in matrix:
        print(" + ".join(f"{el}*x{i + 1}" for i, el in enumerate(row[:-1])) + f" = {row[-1]}")
    partial_solution = {}
    for i in range(num_rows, num_vars):
        value = int(input(f"Введите значение для частной переменной x{i + 1}: "))
```

```
            partial_solution[f'x{i + 1}'] = value
        complete_solution = [0] * num_vars
        for row in reversed(matrix):
            lead_idx = None
            for i in range(len(row) - 1):
                if row[i] != 0:
                    lead_idx = i
                    break
            if lead_idx is None:
                continue
            result = row[-1]
            for i in range(lead_idx + 1, len(row) - 1):
                if f'x{i + 1}' in partial_solution:
                    result -= row[i] * partial_solution[f'x{i + 1}']
                else:
                    result -= row[i] * complete_solution[i]
            if mod != 0:
                result = (result % mod + mod) % mod
            if row[lead_idx] != 0:
                complete_solution[lead_idx] = result // row[lead_idx] if mod == 0 else (result * mod_inverse(row[lead_idx], mod)) % mod
        for i in range(num_rows, num_vars):
            complete_solution[i] = partial_solution[f'x{i + 1}']
        print("Частное решение:")
        for i in range(num_vars):
            print(f"x{i + 1} = {complete_solution[i]}")
    while True:
```

```
        print("\nВыберите алгоритм:")
        print("1. Обычный алгоритм Евклида")
        print("2. Расширенный алгоритм Евклида")
        print("3. Бинарный алгоритм Евклида")
        print("4. Метод Гаусса")
        print("5. Алгоритм Гарнера")
        print("6. Китайская теорема об остатках")
        print("7. Выход")
        choice = int(input("Ваш выбор: "))
        if choice == 1:
            a, b = input_numbers()
            result = normal_Euclid(a, b)
            print(f"НОД({a}, {b}) = {result}")
        elif choice == 2:
            a, b = input_numbers()
            gcd, x, y = extended_Euclid(a, b)
            explanation = explain_extended_Euclid(a, b, gcd, x, y)
            print(explanation)
        elif choice == 3:
            a, b = input_numbers()
            result = binary_Euclid(a, b)
            print(f"НОД({a}, {b}) с помощью бинарного алгоритма = {result}")
        elif choice == 4:
            matrix, mod = input_matrix()
            matrix = reverse_matrix(matrix, mod)
            result_matrix = gauss_jordan_modul(matrix, mod)
```

```
            explain_gauss(result_matrix, mod)
        elif choice == 5:
            a, m = input_chinese_theorem()
            result = Garner_algorithm(a, m)
            print(f"Решение системы по алгоритму Гарнера: x = {result}")
        elif choice == 6:
            a, m = input_chinese_theorem()
            result = chinese_theorem(a, m)
            print(f"Решение системы по Китайской теореме об остатках: x =
{result}")
        elif choice == 7:
            print("Выход.")
            break
        else:
            print("Некорректный выбор, попробуйте снова.")
algorithm_interface()
```