МИНОБРНАУКИ РОССИИ
ФГБОУ ВО «СГУ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ В ЧИСЛОВЫХ ПОЛЯХ

ЛАБОРАТОРНАЯ РАБОТА

студента 5 курса 531 группы
направления 100501 — Компьютерная безопасность
факультета КНиИТ
Окуньков Сергей Викторович

Проверил
профессор ______________ В. А. Молчанов

Саратов 2023

СОДЕРЖАНИЕ

1 Постановка задачи . . . 3
2 Теоретические сведения . . . 4
2.1 Алгоритм поиска наибольшего общего делителя . . . 4
2.1.1 Алгоритм Евклида . . . 4
2.1.2 Расширенный алгоритм Евклида . . . 4
2.1.3 Бинарный алгоритм Евклида . . . 5
2.2 Алгоритмы решения систем сравнений . . . 5
2.2.1 Греко-китайская теорема об остатках . . . 5
2.2.2 Алгоритм Гарнера . . . 6
2.3 Метод Гаусса решения систем линейных уравнений над конечными полями . . . 6
3 Результаты работы . . . 9
3.1 Описание алгоритмов Евклида вычисления НОД целых чисел . . . 9
3.2 Описание алгоритмов решения систем сравнений . . . 11
3.3 Код программы, реализующей рассмотренные алгоритмы . . . 14
3.4 Результаты тестирования программ . . . 19
ЗАКЛЮЧЕНИЕ . . . 21

1 Постановка задачи

Цель работы - изучение основных операций в числовых полях и их программная реализация.

Порядок выполнения работы:

1. Разобрать обычный, бинарный и расширенный алгоритмы Евклида вычисления наибольшего общего делителя 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 поиска наибольшего общего делителя

2.1.1 Алгоритм Евклида

Алгоритм Евклида вычисления наибольшего общего делителя целых чисел a и $b > 0$ состоит из следующих этапов. Положим $a_0 = a$, $a_1 = b$ и выполним последовательно деления с остатком a_i на a_{i+1}:

$$a_0 = a_1q_1 + a_2, 0 \leq a_2 < a_1,$$

$$a_1 = a_2q_2 + a_3, 0 \leq a_3 < a_2,$$

$$\dots$$

$$a_{k-2} = a_{k-1}q_{k-1} + a_k, 0 \leq a_k < a_{k-1},$$

$$a_{k-1} = a_kq_k.$$

Так как остатки выполняемых делений образуют строго убывающую последовательность $a_1 > a_2 > \dots > a_k \geq 0$, то этот процесс обязательно остановится в результате получения нулевого остатка деления. Легко видеть, что НОД(a_0, a_1) = НОД(a_1, a_2) = $\cdots$ = НОД(a_{k-1}, a_k) = a_k. Значит, последний ненулевой остаток a_k = НОД(a, b).

2.1.2 Расширенный алгоритм Евклида

Расширенный алгоритм Евклида позволяет не только вычислять наибольший общий делитель целых чисел a и $b > 0$, но и представлять его в виде НОД$(a, b) = ax + by$ для некоторых $x, y \in Z$. Значения x, y находятся в результате обратного прохода этапов алгоритма Евклида, в каждом из которых уравнение разрешается относительно остатка a_i, который представляется в форме $a_i = ax_i + by_i$ для некоторых $x_i, y_i \in Z$.

В результате получается следующая последовательность вычислений:

$$a_0 = a, \qquad a_0 = ax_0 + by_0,$$

$$a_1 = b, \qquad a_0 = ax_1 + by_1,$$

$$a_2 = a_0 - a_1q_1, \qquad a_2 = ax_2 + by_2,$$

$$a_3 = a_1 - a_2q_2, \qquad a_3 = ax_3 + by_3,$$

$$\dots \qquad \dots$$

$$a_i = a_{i-2} - a_{i-1}q_{i-1}, \qquad a_i = ax_i + by_i,$$

$$\dots \qquad \dots$$

$$a_k = a_{k-2} - a_{k-1}q_{k-1}, \qquad a_k = ax_k + by_k,$$

$$0 = a_{k-1} - a_k q_k, \qquad 0 = ax_{k+1} + by_{k+1}$$

В правом столбце все элементы $a_k, a_{k-1}, a_{k-2}, ..., a_1, a_0$ представляются в виде $a_i = ax_i + by_i$. Очевидно, что $x_0 = 1, y_0 = 0, x_1 = 0, y_1 = 1$ и выполняются равенства: $a_i = a_{i-2} - a_{i-1}q_{i-1}, x_i = x_{i-2} - x_{i-1}q_{i-1}, y_i = y_{i-2} - y_{i-1}q_{i-1}$. Отсюда последовательно получаются искомые представления всех элементов $a_k, a_{k-1}, a_{k-2}, \dots, a_1, a_0$ и, в частности, представление НОД$(a, b) = a_k = ax_k + by_k$.

2.1.3 Бинарный алгоритм Евклида

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

1. НОД($2 \cdot a, 2 \cdot b$) = 2$\cdot$ НОД(a, b);
2. НОД($2 \cdot a, 2 \cdot b + 1$) = НОД($a, 2 \cdot b + 1$)
3. НОД($-a, b$) = НОД(a, b).

2.2 Алгоритмы решения систем сравнений

2.2.1 Греко-китайская теорема об остатках

Теорема. Пусть $m_1, m_2, \dots, m_k$ – попарно взаимно простые целые числа и $M = m_1 m_2 \cdots m_k$. Тогда система линейных сравнений

$$\begin{cases} x \equiv a_1 (mod\, m_1) \\ x \equiv a_2 (mod\, m_2) \\ \dots \\ x \equiv a_k (mod\, m_k) \end{cases} . \tag{1}$$

имеет единственное неотрицательное решение по модулю M.

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

2.2.2 Алгоритм Гарнера

Пусть $M = \prod_{i=1}^{k} m_i$, числа $m_1, \ldots, m_k$ попарно взаимно просты, и $c_{ij} \equiv m_i^{-1}(mod\, m)_j, i \neq j, i, j \in 1, \ldots, k$. Тогда решение системы может быть представлено в виде

$$x = q_1 + q_2 m_1 + q_3 m_2 + \cdots + q_k m_1 \ldots m_k,$$

где $0 \leq q_i < m_i, i \in 1, \ldots, k$, и числа q_i вычисляются по формулам

$$q_1 = u_1(mod\, m_1)$$

$$q_2 = (u_2 - q_1)c_{12}(mod\, m_2)$$

$$\ldots$$

$$q_=(((u_k - q_1)c_{1k} - q_2)c_{2k} - \cdots - q_{k-1})c_{k-1k}(mod\, m)$$

2.3 Метод Гаусса решения систем линейных уравнений над конечными полями

Пусть $P = (P, +, \times, 1, 0)$ — произвольное поле.

Системой n линейных уравнений с m неизвестными $x_1, \ldots, x_m$ называется выражение вида:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = b_1 \ (1) \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m = b_2 \ (2) \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nm}x_m = b_n \ (n), \end{cases} \quad . \qquad (2)$$

где $(1), (2), \ldots, (n)$ — линейные уравнения с неизвестными $x_1, \ldots, x_m$, коэффициентами $a_{11}, a_{12}, \ldots, a_{nm} \in P$ (первый индекс указывает номер уравнения, второй индекс – номер неизвестного) и свободными членами $b_1, \ldots, b_n \in P$ (индекс – номер уравнения). При этом числа $a_{11}, a_{12}, \ldots, a_{nm}$ называются также коэффициентами системы и $b_1, \ldots, b_n$ — свободными членами системы.

Система называется однородной, если $b_1 = \cdots = b_n = 0$.

Система (2) кратко записывается в виде

$$\sum_{j=1}^{m} a_{ij}x_j = b_i(i = 1, \ldots, n).$$

Определение. Решением системы (2) называется такой упорядоченный набор $\zeta_1, \ldots, \zeta_m \in P$ из m элементов, что при подстановке в уравнения $(1) - (n)$ значений $x_1 = \zeta_1, \ldots, x_m = \zeta_m$ получаются верные равенства $\sum_{j=1}^{m} a_{ij}\zeta_j = b_i(i = 1, \ldots, n)$. Такое решение сокращенно записывается в виде элемента $\zeta = (\zeta_1, \ldots, x_m = \zeta_m)$ множества P^n.

Множество всех решений системы (2) обозначается символом $R(2)$.

Определение. Система (2) называется совместной, если у нее есть решения, и несовместной в противном случае. При этом совместная система называется определенной, если она имеет единственное решение, и неопределенной в противном случае.

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, т.е. являются равносильными:

1. удаление из системы тривиальных уравнений,
2. умножение обеих частей какого-либо уравнения на одно и тот же ненулевой элемент поля,
3. прибавление к обеим частям какого-либо уравнения системы соответствующих частей другого уравнения системы.

Метод решения системы (2) заключается в равносильном преобразовании ее в систему линейных уравнений с противоречивым уравнением или в разрешенную систему линейных уравнений вида:

$$\begin{cases} x_1 + \cdots + \cdots + a'_{1,r+1}x_{r+1} + \cdots + a'_{1,m}x_m = b'_1 \ (1) \\ x_2 + \cdots + a'_{2,r+1}x_{r+1} + \cdots + a'_{2,m}x_m = b'_2 \ (2) \\ \cdots \\ x_r + a'_{r,r+1}x_{r+1} + \cdots + a'_{r,m}x_m = b'_r \ (r), \end{cases} \quad . \tag{3}$$

где $r \leq n$, 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 $x_{r+1}, \ldots, x_m$ — свободными.

Преобразование системы (2) в равносильную ей разрешенную систему (3) 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 (здесь $k = 1, \ldots, n, k \neq i$) соответствующие части нового i-ого уравнения, умноженные на коэффициент — a_{kj};
4. удаляем из системы тривиальные уравнения (нулевые строки);

При этом выбранный ненулевой элемент a_{ij} называется разрешающим, строка и столбец, содержащие элемент a_{ij}, также называются разрешающими. Такие действия удобнее осуществлять над таблицей коэффициентов системы (2), которая представляется в виде:

$$\overline{A} = \begin{pmatrix} a_{11} & \cdots & a_{1m} & b_1 \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & \cdots & a_{nm} & b_n \end{pmatrix}$$ и называется матрицей системы (2).

Конечной целью применения метода Гаусса к системе линейных уравнений (2) является преобразование с помощью Жордановых преобразований системы (2) в равносильную ей разрешенную систему (3).

Матрица $\overline{A'}$ такой разрешенной системы (3) имеет вид:

$$\overline{A'} = \begin{pmatrix} 1 & 0 & \cdots & 0 & a'_{1,r+1} & \cdots & a'_{1m} & b'_1 \\ 0 & 1 & \cdots & 0 & a'_{2,r+1} & \cdots & a'_{2m} & b'_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \\ 0 & 0 & \cdots & 1 & a'_{r,r+1} & \cdots & a'_{rm} & b'_r \end{pmatrix}$$

Единичные столбцы матрицы $\overline{A'}$ будем называть разрешенными (или базисными), остальные столбцы с коэффициентами a'_{ij} — свободными. Строки, содержащие единицы базисных столбцов, называются разрешенными. Матрица называется разрешенной, если все ее строки разрешенные.

Применение метода Гаусса к системе линейных уравнений (2) в матричной форме равносильно преобразованию матрицы $\overline{A}$ этой системы в эквивалентную ей разрешенную матрицу $\overline{A'}$. При этом на каждом шаге метода Гаусса в преобразуемой матрице с помощью элементарных преобразований формируется новый единичный столбец.

3 Результаты работы

3.1 Описание алгоритмов Евклида вычисления НОД целых чисел

Алгоритм 1 - алгоритм Евклида

Вход: целые числа a, b.

Выход: $d =$ НОД (a, b).

Шаг 1. Положить $a_0 = a, a_1 = b, i = 1$.

Шаг 2. Найти остаток a_{i+1} от деления a_{i-1} на a_i.

Шаг 3. Если $a_{i+1} = 0$, то положить $d = a_i$. Иначе — положить $i = i + 1$ и вернуться к шагу 2.

Шаг 4. Результат: $d =$ НОД (a, b).

Псевдокод:

```
Алгоритм Евклида(a, b):
если b = 0 то
    вернуть a
иначе
    вернуть Алгоритм Евклида(b, a % b)
```

Сложность алгоритма $O(\log(\min\{a, b\}))$.

Алгоритм 2 - расширенный алгоритм Евклида

Вход: целые числа a, b.

Выход: $d =$ НОД (a, b) и коэффициенты x, y.

Шаг 1. Положить $a_0 = a, a_1 = b, x_0 = 1, y_0 = 0, x_1 = 0, x_1 = 1, i = 1$.

Шаг 2. Найти остаток a_{i+1} от деления a_{i-1} на a_i.

Шаг 3. Найти $x_{i+1} = x_{i-1} - (\frac{a_{i-1}}{a_i} \cdot x_i)$.

Шаг 4. Найти $y_{i+1} = y_{i-1} - (\frac{a_{i-1}}{a_i} \cdot y_i)$.

Шаг 5. Если $a_{i+1} = 0$, то положить $d = a_i, x = x_i, y = y_i$. Иначе положить $i = i + 1$ и перейти к шагу 2.

Шаг 6. Результат: $d =$ НОД (a, b) и коэффициенты x, y.

Псевдокод:

```
Расширенный алгоритм Евклида(a, b):
если b = 0 то
    вернуть (a, 1, 0)
```

```
иначе
    (d, x, y) := Расширенный алгоритм Евклида(b, a % b)
    вернуть (d, y, x - (a / b) * y)
```

Сложность алгоритма $O(\log(\min\{a, b\}))$.

<u>Алгоритм 3 - бинарный алгоритм Евклида</u>

Вход: целые числа a, b.

Выход: $d =$ НОД (a, b).

<u>Шаг 1.</u> Положить $a_0 = a, a_1 = b$.

<u>Шаг 2.</u> Если $a = 0$, положить $d = b$.

<u>Шаг 3.</u> Если $b = 0$, положить $d = a$.

<u>Шаг 4.</u> Если $a = b$, положить $d = a$.

<u>Шаг 5.</u> Если $a = 1$ или $b = 1$, положить $d = 1$.

<u>Шаг 6.</u> Если a и b четные, положить $d = 2\cdot$ бинарный алгоритм Евклида $(a/2, b/2)$.

<u>Шаг 7.</u> Если a четное и b нечетное, положить $d =$ бинарный алгоритм Евклида $(a/2, b)$.

<u>Шаг 8.</u> Если a нечетное и b четное, положить $d =$ бинарный алгоритм Евклида $(a, b/2)$.

<u>Шаг 9.</u> Если a и b нечетные и $b > a$, положить $d =$ бинарный алгоритм Евклида $((b - a)/2, a)$.

<u>Шаг 10.</u> Если a и b нечетные и $a > b$, положить $d =$ бинарный алгоритм Евклида $((a - b)/2, b)$.

<u>Шаг 11.</u> Результат: d.

<u>Псевдокод:</u>

```
Бинарный алгоритм Евклида(a, b):
если a = b то
    вернуть a
если a = 0 то
    вернуть b
если b = 0 то
    вернуть a
если a чётное и b чётное то
    вернуть 2 * Бинарный алгоритм Евклида(a/2, b/2)
если a чётное и b нечётное то
    вернуть Бинарный алгоритм Евклида(a/2, b)
```

```
если a нечётное и b чётное то
    вернуть Бинарный алгоритм Евклида(a, b/2)
если a и b нечётные то
    если a > b то
        вернуть Бинарный алгоритм Евклида((a-b)/2, b)
    иначе
        вернуть Бинарный алгоритм Евклида((b-a)/2, a)
```

Сложность алгоритма $O(\log(\min\{a, b\})^2)$.

3.2 Описание алгоритмов решения систем сравнений

Алгоритм 4 - решение системы сравнений с помощью греко-китайской теоремы об остатках

Вход: целые числа $a_1, a_2, \ldots, a_n$, являющиеся коэффициентами системы линейных сравнений и $m_1, m_2, \ldots, m_n$ - сопростые числа, представляющие из себя модули этой системы.

Выход: целое число x — решение системы сравнений.

Шаг 1. Определить $M = \prod_{i=1}^{n} m_i$.

Шаг 2. Определить $c_1, \ldots, c_n$, где $c_i = \frac{M}{m_i}$.

Шаг 3. Определить $d_1, \ldots, d_n$, где $d_i = c_i^{-1}(mod\, m_i)$. c_i^{-1} находится с помощью расширенного алгоритма Евклида (алгоритм 2).

Шаг 4. Результат: $x = \sum_{i=1}^{n} c_i d_i a_i (mod\, M)$.

Псевдокод:

```
Китайская Теорема Об Остатках(a, m):
M := 1
ans := 0
для каждого (ai, mi) выполнить
    M := M * mi
для каждого (ai, mi) выполнить
    c := M / mi
    (d, x, y) := Расширенный Алгоритм Евклида(m, c)
    ans :+= ai * c * x
вернуть ans mod M
```

Сложность алгоритма $O(n^2 b^2)$, где b — число двоичных знаков, с помощью которых записываются числа $c_i d_i a_i$.

Алгоритм 5 - алгоритм Гарнера

Вход: целые числа $a_1, a_2, \ldots, a_n$ — коэффициенты системы линейных сравнений и сопростые числа $m_1, m_2, \ldots, m_n$ — из себя модули этой системы.

Выход: целое число x — решение системы сравнений.

Шаг 1. Определить $c_{11}, c_{12}, \ldots, c_{21}, c_{22}, \ldots, c_{nn} (\forall i \in \overline{0, n} \& i \neq j)$, где $c_{ij} = m_i^{-1} (mod\, m_j)$. Обратный элемент находится с помощью расширенного алгоритма Евклида (алгоритм 2).

Шаг 2. Определить последовательность Q, которая изначально состоит из одного элемента q_1 и имеет длину $l = 1$. Положить $i = 0$.

Шаг 3. Положить $i = i + 1$, $q = u_i$.

Шаг 4. Для $j = 1, \ldots, l$ выполнить $q = (q - q_j) \cdot c_{ji}$.

Шаг 5. Добавить $q(mod\, m_i)$ в Q и положить $l = l + 1$. Если $i \leq n$, перейти к шагу 3.

Шаг 6. Вернуть результат: $x = q_1 + \sum_{i=2}^{n} (q_i \prod_{j=1}^{i-1} m_i)$.

Псевдокод:

```
Алгоритм Гарнера(a, m):
n := размер массива a
c := [[0, 0, ..., 0], ..., [0, 0, ..., 0]] // Инициализация
 ↪  массива длинной n на n
q := [0, 0, ..., 0] // Инициализировать массив длинной n

для i от 1 до n-1 включительно выполнить
    q[i] := a[i]
    для j от 1 до i включительно выполнить
        q[i] := c[j][i] * (q[i] - q[j])
        q[i] := q[i] mod m[i]
m_iter := 1
x := 0
для i от 1 до n включительно выполнить
    x :+= q[i] * m_iter
    m_iter :*= m[i]

вернуть x
```

Трудоемкость алгоритма $O(n^2 b^2)$.

Алгоритм 6 - метод Гаусса решения систем линейных уравнений над конечными полями

Вход: Коэффициенты системы $a_{00}, \dots, a_{n-1,m-1}$, свободные члены системы $b_0, \dots, b_{n-1}$, размерность конечного поля p.

Выход: Если у системы есть решение, то выход: список $X = (\zeta_0, \dots, \zeta_{m-1})$, который является решением системы уравнений. Если решения у системы нет, выход: сообщение "Нет решения!".

Шаг 1. Положить $i = 0$.

Шаг 2. Если $a_{t0} = \dots = a_{t,m-1} = 0 (\forall t \in \overline{0,m})$, но $b_i \neq 0$, то вывести в качестве результата "Нет решения!".

Шаг 3. Положить $a_{ij} = a_{ij} * a_{ii}^{-1} (mod\, p) \& b_i = bi * a_{ii}^{-1} (mod\, p) (\forall j \in \overline{i,m})$.

Шаг 4. Положить $a_{tj} = (a_{tj} - a_{ti} * a_{ij}) (mod\, p) \& b_t = (bt - a_{ti} * bi) (mod\, p) (\forall j, t \in \overline{i,m} \& t \neq i)$.

Шаг 5. Положить $i + 1$и вернуться к шагу 2.

Шаг 6. Удалить строки, для которых выполняется $a_{i0} = \dots = a_{0,m-1} = 0 (\forall i \in \overline{0,m})$

Шаг 7. Положить $x_j = b_j$.

Шаг 8. Положить $k = j + 1$.

Шаг 9. Положить $x_j = (x_j - a_{jk} \cdot x_k) (mod\, p)$.

Шаг 10. Если $k < m$, положить $k = k + 1$ и перейти к шагу 9.

Шаг 11. Если $j > 0$, положить $j = j - 1$ и перейти к шагу 7.

Шаг 12. Вернуть X.

Псевдокод:

```
Метод Гаусса(A, B, p)
для i от 1 до n включительно выполнить
    если A[i][i] не равно 0 то
        A[i], B[i] :/= A[i][i] (mod p)
        для row от 1 до n включительно выполнить
            если row не равно i то
                A[row] :-= A[row][i] * A[i]
    A, B := найти и удалить тривиальные строку

для j от n-1 до 0 выполнить
    x[j] := B[j]
    Для k от j+1 до m выполнить
        x[j] := (x[j] - A[j][k] * x[k]) (mod p)

вернуть x
```

Трудоемкость алгоритма $O(n^2 \cdot m)$

3.3 Код программы, реализующей рассмотренные алгоритмы

```python
def extended_gcd(a, b):
    if a == 0:
        return (b, 0, 1)
    d, x1, y1 = extended_gcd(b % a, a)
    return (d, y1 - (b // a) * x1, x1)


def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)


def binary_gcd(a, b):
    if a == b or b == 0:
        return a
    if a == 0:
        return b
    if a % 2 == 0:
        if b % 2 == 0:
            return binary_gcd(a // 2, b // 2) * 2
        else:
            return binary_gcd(a // 2, b)
    if b % 2 == 0:
        return binary_gcd(a, b // 2)
    if a > b:
        return binary_gcd((a - b) // 2, b)
    return binary_gcd((b - a) // 2, a)


def find_comparison_solution(a, b, m):
    d = gcd(a, m)
    a = a // d
    b = b // d
    m = m // d
    x = extended_gcd(a, m)[1] * b % m
```

```python
    return x


def check_coprime(m):
    for i in range(len(m)):
        for j in range(len(m)):
            if i != j and gcd(m[i], m[j]) != 1:
                return False
    return True


def greco_chinese_theorem(u, m, M):
    x = 0
    for i in range(len(m)):
        c = M // m[i]
        d = find_comparison_solution(c, 1, m[i])
        x += c * d * u[i]
    return x % M


def garner_algorithm(u, m):
    c = [[0 for _ in m] for _ in m]
    for i in range(len(m)):
        for j in range(len(m)):
            c[i][j] = find_comparison_solution(m[i], 1, m[j])
    q = [0 for _ in m]

    for i in range(len(m)):
        q[i] = u[i]
        for j in range(i):
            q[i] = c[j][i] * (q[i] - q[j])
            q[i] = q[i] % m[i]
    m_iter = 1
    x = 0
    for i in range(len(m)):
        x += q[i] * m_iter
        m_iter *= m[i]
    return x, q


def swap_rows(A, B, row1, row2):
```

```python
    A[row1], A[row2] = A[row2], A[row1]
    B[row1], B[row2] = B[row2], B[row1]


def divide_row(A, B, row, divider, m):
    A[row] = [find_comparison_solution(divider, a, m) for a in A[row]]
    B[row] = find_comparison_solution(divider, B[row], m)


def combine_rows(A, B, row, source_row, weight, m):
    A[row] = [(a + (k * weight) % m) % m for a, k in zip(A[row],
    ↪ A[source_row])]
    B[row] = (B[row] + (B[source_row] * weight) % m) % m


def drop_trivial_row(A, B):
    A1, B1 = [], []
    for i in range(len(A)):
        f = False
        for j in range(len(A[i])):
            if A[i][j] != 0:
                f = True
        if f:
            A1.append(A[i])
            B1.append(B[i])
    return A1, B1


def gauss_algorithm(A, B, m):
    column = 0
    while column < len(B):
        current_row = None
        for r in range(column, len(A)):
            if current_row is None or abs(A[r][column]) >
            ↪ abs(A[current_row][column]):
                current_row = r
        if current_row is None:
            print("No solutions!")
            return None
        if current_row != column:
            swap_rows(A, B, current_row, column)
```

```python
        if A[column][column] != 0:
            divide_row(A, B, column, A[column][column], m)
        for row in range(len(A)):
            if row != column:
                combine_rows(A, B, row, column, -A[row][column], m)
        A, B = drop_trivial_row(A, B)
        column += 1
    return A, B


def check_solution_exist(B):
    solution = False
    for i in range(len(B) - 1):
        if B[i] != 0:
            solution = True
    return solution


def task1():
    a, b = map(int, input("Enter a and b : ").split(" "))
    print(f"GCD = {gcd(a, b)}")
    print(
        "Extended GCD:\n{res[1]} * {a} + {res[2]} * {b} = {res[0]}".format(
            res=extended_gcd(a, b), a=a, b=b
        )
    )
    print(f"Binary GCD = {binary_gcd(a, b)}")


def task2():
    n = int(input("Enter number of comparisons: "))
    u, m, M = [], [], 1
    comp = ""
    print("Enter comparisons (format u m)")
    for _ in range(n):
        u_i, m_i = map(int, input().split(" "))
        comp += f"x \u2261 {u_i} (mod {m_i})\n"
        M *= m_i
        u.append(u_i)
        m.append(m_i)
    print("Your system of comparisons")
```

```
    print(comp)
    if check_coprime(m):
        print(f"x = {greco_chinese_theorem(u, m, M)}")
        print("Garner's algorithm:")
        x, q = garner_algorithm(u, m)
        print(f"q : {q}\nx = {x}")
    else:
        print("The bases of the system are not relatively prime!")


def task3():
    rows = int(input("Enter number of rows: "))
    print("Enter matrix of odds")
    A = [list(map(int, (input(f"row {i+1}: ").split()))) for i in range(rows)]
    B = list(map(int, input("Enter free terms of the equation: ").split()))
    m = int(input("Enter field dimension: "))
    ans = gauss_algorithm(A, B, m)
    if ans:
        A, B = ans
        if check_solution_exist(B):
            for i in range(len(A)):
                ans_str = f"x_{i+1}="
                for j in range(len(A[i]) - len(A), len(A[i])):
                    ans_str += f"{-A[i][j] % m}*x_{j+1}+"
                ans_str += f"{B[i]}"
                print(ans_str)
        else:
            print("No solutions!")


def main_loop():
    while True:
        print('Tasks:\n1.GCD\n2.Solutions to comparison systems\n3.Solution for a system of linear equations\n4.Exit')
        task_num = int(input('Enter number of task: '))
        print()
        match task_num:
            case 1:
                task1()
            case 2:
                task2()
```

```
            case 3:
                task3()
            case 4:
                break
            case _:
                print("Wrong input!\n")

if __name__ == '__main__':
    main_loop()
```

3.4 Результаты тестирования программ

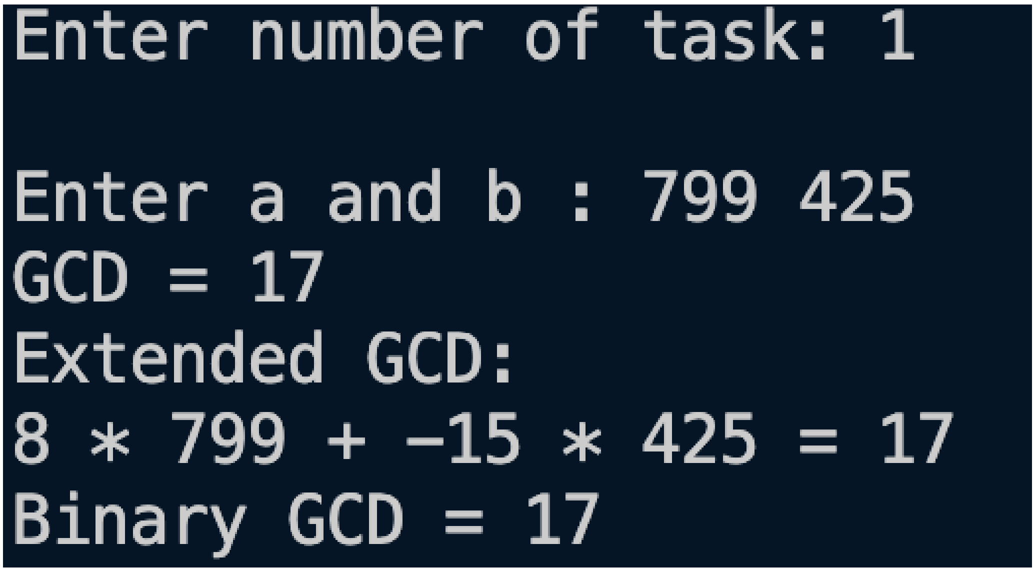

Рисунок 1 – Тест поиска НОД по алгоритмам Евклида

```
Enter number of task: 2

Enter number of comparisons: 3
Enter comparisons (format u m)
1 3
3 5
2 4
Your system of comparisons
x ≡ 1 (mod 3)
x ≡ 3 (mod 5)
x ≡ 2 (mod 4)

x = 58
Garner's algorithm:
q : [1, 4, 3]
x = 58
```

Рисунок 2 – Тест алгоритмов решения систем сравнений

```
Enter number of task: 3

Enter number of rows: 3
Enter matrix of odds
row 1: 1 0 -2 -4
row 2: 2 -2 0 -3
row 3: 0 2 -4 -5
Enter free terms of the equation: 2 1 3
Enter field dimension: 7
x_1=2*x_3+4*x_4+2
x_2=2*x_3+6*x_4+5
```

Рисунок 3 – Тест алгоритма решения СЛУ методом Гаусса

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были рассмотрены теоретические сведения об алгоритмах Евклида поиска НОД (обычного, расширенного и бинарного), греко-китайская теорема об остатках и алгоритм Гарнера для решения системы сравнений, а также метод Гаусса для решения линейных уравнений над конечными полями. На их основе были реализованны соответствующие алгоритмы на языке python. Далее была произведена оценка сложности и тестирование данных алгоритмов, результаты которого были прикреплены к отчету вместе с самим листингом программы.