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

Яхина Шамиля Илдусовича

Преподаватель

профессор, д.ф.-м.н. ___________ В. А. Молчанов

подпись, дата

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы - изучение основных операций 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.

1 Теория

1.1 Описание алгоритмов Евклида вычисления НОД целых чисел

Наибольшим общим делителем (НОД) целых чисел $a_1, ..., a_k$ называется такой положительный общий делитель этих чисел, который делится на любой другой делитель этих чисел. Если d — наибольший общий делитель для чисел a и b, то для него вводится обозначение $(a, b) = d$.

1.1.1 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}:

$$
\begin{gathered}
a_0 = a_1 q_1 + a_2, 0 \leq a_2 < a_1, \\
a_1 = a_2 q_2 + a_3, 0 \leq a_3 < a_2, \\
\dots\dots\dots\dots\dots\dots\dots\dots \\
a_{k-2} = a_{k-1} q_{k-1} + a_k, 0 \leq a_k < a_{k-1}, \\
a_{k-1} = a_k q_k.
\end{gathered}
$$

Так как остатки выполняемых делений образуют строго убывающую последовательность $a_1 > a_2 > a_3 > \cdots \geq 0$, то этот процесс обязательно остановится в результате получения нулевого остатка деления. Легко видеть, что НОД(a_0,a_1) = НОД(a_1,a_2) = ... = НОД(a_{k-1},a_k) = a_k. Значит, последний ненулевой остаток a_k = НОД(a,b).

Описание алгоритма.

Вход: целые числа a,b; $0 < b < a$

Выход: d = НОД(a,b).

Шаги алгоритма:

1. Положить a_0 = a, a_1 = b, i = 1.
2. Найти остаток a_{i+1} от деления a_{i-1} на a_i.
3. Если a_{i+1} = 0, то положить d = a_i. В противном случае положить i = $i + 1$ и вернуться на шаг 2.
4. Результат: d = НОД(a,b).

Временная сложность алгоритма Евклида $O(n^2)$.

1.1.2 Бинарный алгоритм Евклида

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

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

Описание алгоритма.

Вход: целые числа a,b; $0 < b < a$

Выход: d = НОД(a,b).

1. НОД(0, b) = b;
2. НОД(a, 0) = НОД(a, a) = a;
3. НОД(1, b) = НОД(a, 1) = 1;
4. Если a и b четные, то НОД(a, b) = 2 НОД($a/2$, $b/2$);
5. Если a четное и b нечетное, то НОД(a, b) = НОД($a/2$, b);
6. Если a нечетное и b четное, то НОД(a, b) = НОД(a, $b/2$);
7. Если a и b нечетные:
   а) при этом $b > a$, то НОД(a, b) = НОД($(b-a)/2$, a);
   б) при этом $b < a$, то НОД(a, b) = НОД($(a-b)/2$, b).

Временная сложность бинарного алгоритма Евклида $O(n^2)$.

1.1.3 Расширенный алгоритм Евклида

Пусть алгоритм Евклида на каждом шаге, кроме частного d_i и остатка r_i, вычисляет еще два значения u_i,v_i по правилу:

1. u_{-1} = 1, u_o = 0;
2. v_{-1} = 0, v_o = 1;
3. u_i = $u_{i-2} - d_i u_{i-1}$, $1 \leq i \leq k$;
4. v_i = $v_{i-2} - d_i v_{i-1}$, $1 \leq i \leq k$,

Такой алгоритм будем называть расширенным алгоритмом Евклида. В расширенном алгоритме Евклида для всех $i \in \{-1, 0, \dots, k\}$ выполняется равенство $u_i x_1 + v_i x_2 = r_i$. Значение расширенного алгоритма Евклида состоит в том, что он дает линейное разложение наибольшего общего делителя $u_k x_1 + v_k x_2 = r_k = (x_1, x_2)$, которое играет важнейшую роль в операциях модульной арифметики.

Описание алгоритма.

Вход:

Целые числа x_1 и x_2, где $x_2 \neq 0$.

Выход:

Наибольший общий делитель d, а также коэффициенты u_k и v_k такие, что $u_k x_1 + v_k x_2 = d$.

Шаги алгоритма:

1. Инициализация начальных значений:
   $u_{-1} = 1, u_0 = 0;$
   $v_{-1} = 0, v_0 = 1;$
   $a_0 = x_1, a_1 = x_2, i = 1.$
2. Вычисление d_i и остатка a_{i+1} от деления a_{i-1} на a_i:
   $d_i = \lfloor \frac{a_{i-1}}{a_i} \rfloor$
   $a_{i+1} = a_{i-1} - d_i \cdot a_i$
3. Вычисление новых коэффициенты:
   $u_i = u_{i-2} - d_i u_{i-1}$
   $v_i = v_{i-2} - d_i v_{i-1}$
4. Если $a_{i+1} = 0$, то $d = a_i$ и завершение алгоритма. Иначе увеличить i на 1 и вернуться к шагу 2.
5. Результат: d = НОД(x_1, x_2), и коэффициенты u_k и v_k такие, что $u_k x_1 + v_k x_2 = d$.

Временная сложность расширенного алгоритма Евклида $O(n^2)$.

1.2 Описание алгоритмов решения систем сравнений

1.2.1 Алгоритм решения системы линейных сравнений с помощью Греко-Китайской теоремы

Греко-китайская теорема об остатках. Пусть $m_1, m_2, ..., m_k$ - попарно взаимно простые целые числа и $M = m_1 m_2 \dots m_k$. Тогда система линейных сравнений

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

имеет единственное неотрицательное решение по модулю M.

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

Описание алгоритма.

Вход:

Система линейных сравнений, где $m_1, m_2, \dots, m_k$ - попарно взаимно простые целые числа.

Выход:

Решение системы линейных сравнений по модулю M, где $M = m_1 m_2 \dots m_k$.

Шаги алгоритма:

1. Вычисление $M = m_1 m_2 \dots m_k$;
2. Вычисление $M_j = M/m_j$ для каждого $1 \le j \le k$;
3. Решение сравнения $M_j x \equiv 1 \pmod{m_j}$ для каждого $1 \le j \le k$;
4. Поиск решения z_j для каждого $1 \le j \le k$;
5. Вычисление результата: $x = M_1 z_1 + M_2 z_2 + \dots + M_k z_k$.

Сложность вычислений φ в данном алгоритме равна $O(k^2 f_{div}(b) + k f_{inv}(b))$, где $b = max\{\log m_i : 1 \le i \le k\}$.

1.2.2 Алгоритм решения системы линейных сравнений алгоритмом Гарнера

Теорема. Пусть $M = \prod_{i=1}^{k} m_i$, числа $m_1, \dots, m_k$ попарно взаимно просты, и $c_{ij} \equiv m_i^{-1} \ (mod\, m_j)$, $i \ne j, i, j \in \{1, \dots, k\}$. Тогда решение системы

$$\begin{cases} u \equiv u_1 \ (mod\, m_1) \\ \dots \\ u \equiv u_k \ (mod\, m_k), \end{cases}$$

может быть представлено в виде

$$u = q_1 + q_2 m_1 + q_3 m_1 m_2 + \dots + q_k m_1 \dots m_{k-1},$$

где $0 \le q_i < m_i$, $i \in \{1, ..., k\}$, и числа q_i вычисляются по формулам

$$q_1 = u_1 \ mod\, m_1,$$
$$q_2 = (u_2 - q_1) c_{12} \ mod\, m_2,$$

$$...$$
$$q_k = (((u_k - q_1)c_{1k} - q_2)c_{2k} - ... - q_{k-1})c_{k-1k}\, mod\, m_k.$$

Описание алгоритма.

Вход:

Система линейных сравнений, где $m_1, m_2, \ldots, m_k$ - попарно взаимно простые целые числа.

Выход:

Решение u системы линейных сравнений по модулю M, где $M = m_1 m_2 \ldots m_k$.

Шаги алгоритма:

1. Вычисление $M = \prod_{i=1}^{k} m_i$.
2. Вычисление коэффициентов q_i:

   а) $q_1 = u_1 \bmod m_1$.

   б) Для каждого i от 2 до k:

   $q_i = ((u_i - q_1) \cdot c_{1i} \bmod m_i) \bmod m_i$, где $c_{1i} \equiv m_1^{-1} \pmod{m_i}$.
3. Вычисление решения u:

   $u = q_1 + q_2 \cdot m_1 + q_3 \cdot m_1 \cdot m_2 + \cdots + q_k \cdot m_1 \cdot m_2 \ldots m_{k-1}$.

Сложность вычислений φ в данном алгоритме равна $O(k^2 f_{div}(b) + k f_{inv}(b))$, где $b = max\{\log m_i : 1 \leq i \leq k\}$.

1.3 Описание алгоритмов решения систем линейных уравнений над конечными полями методом Гаусса

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

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

$$(s) \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1(1) \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2(2) \\ \ldots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m(m), \end{cases}$$

где (1),(2),...,(m) –линейные уравнения с неизвестными $x_1, \ldots, x_n$, коэффициентами $a_{11}, a_{12}, \ldots, a_{mn} \in P$ (первый индекс указывает номер уравнения, второй индекс - номер неизвестного) и свободными членами $b_1, \ldots, b_m \in P$

(индекс - номер уравнения). При этом числа $a_{11}, a_{12}, \ldots, a_{mn}$ называются также коэффициентами системы и $b_1, \ldots, b_m$ - свободными членами системы.

Система называется однородной, если $b_1 = \cdots = b_m = 0$. Система (s) кратко записывается в виде $\sum_{j=1}^{n} a_{ij}x_j = b_i(i = \overline{1,m})$.

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

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

Система (s) 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.

Решить систему линейных уравнений – значит найти множество всех ее решений.

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.

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 (здесь $k = \overline{1,m}, k \neq i$) соответствующие части нового i-го уравнения, умноженные на коэффициент $-a_{kj}$;

4. удаляем из системы тривиальные уравнения.

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

Конечной целью применения метода Гаусса к системе линейных уравнений (s) является преобразование с помощью Жордановых преобразований системы (s) в равносильную ей разрешенную систему (s') вида:

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

где $r \leq m$, переменные $x_1, \dots, x_r$ – разрешенные (или базисные) и переменные $x_{r+1}, \dots, x_n$ - свободные.

Матрица такой разрешенной системы (s') имеет вид:

$$\overline{(A')} = \begin{pmatrix} 1 & 0 & \dots & 0 & a'_{1,r+1} & \dots & a'_{1n} & b'_1 \\ 0 & 1 & \dots & 0 & a'_{2,r+1} & \dots & a'_{2n} & b'_2 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & a'_{r,r+1} & \dots & a'_{rn} & b'_r \end{pmatrix}$$

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.

В результате последовательных Жордановых преобразований матрицы $\overline{A}$ любой системы линейных уравнений (s) получаем один из следующих трех взаимоисключающих результатов:

1. либо получаем матрицу со строкой вида $(0, \dots, 0, b)$ с значением $b \neq 0$ и в этом случае система (s) не имеет решений;
2. либо получаем разрешенную матрицу $\overline{A'}$ без свободных столбцов, и в этом случае система (s) имеет единственное решение;
3. либо получаем разрешенную матрицу $\overline{A'}$ со свободными столбцами, и в этом случае система (s) имеет бесконечно много решений.

Описание алгоритма

Вход:

Система m линейных уравнений с n элементами в виде матрицы A размерности $m \times n$ с элементами из конечного поля P.

Выход:

Решение системы линейных уравнений, либо информация о том, что система не имеет решений.

Шаги алгоритма:

1. Преобразование системы к равносильной разрешенной форме методом Гаусса:
   а) Выбрать ненулевой элемент a_{ij} в матрице A.
   б) Умножить i-ое уравнение на элемент a_{ij}^{-1} (обратный элемент в поле P).
   в) Прибавить к остальным уравнениям k-ого столбца умноженное на $-a_{kj}$.
   г) Удалить тривиальные уравнения.
2. Проверка результатов:
   а) Если в результате преобразований в матрице A образуется строка вида $(0, \dots, 0, b)$, где $b \neq 0$, система не имеет решений.
   б) Если в полученной матрице нет свободных столбцов, то в этом случае система имеет единственное решение;
   в) Если в полученной матрице есть свободные столбцы, то в этом случае система имеет бесконечное количество решений;
   г) Удалить тривиальные уравнения.

Временная сложность алгоритма равна $O(\min(m, n) * nm)$

2 Псевдокоды и примеры работы программ

2.1 Алгоритм Евклида

Псевдокод функции algEuclidClassic(a, b):

Вход: числа a, b.

Выход: nod = НОД(a,b).

1. Если a < b, то swap(a,b)
2. Пока b не равно 0:
   а) b_dop := b
   б) b := a mod b
   в) a := b_dop
3. Вернуть a

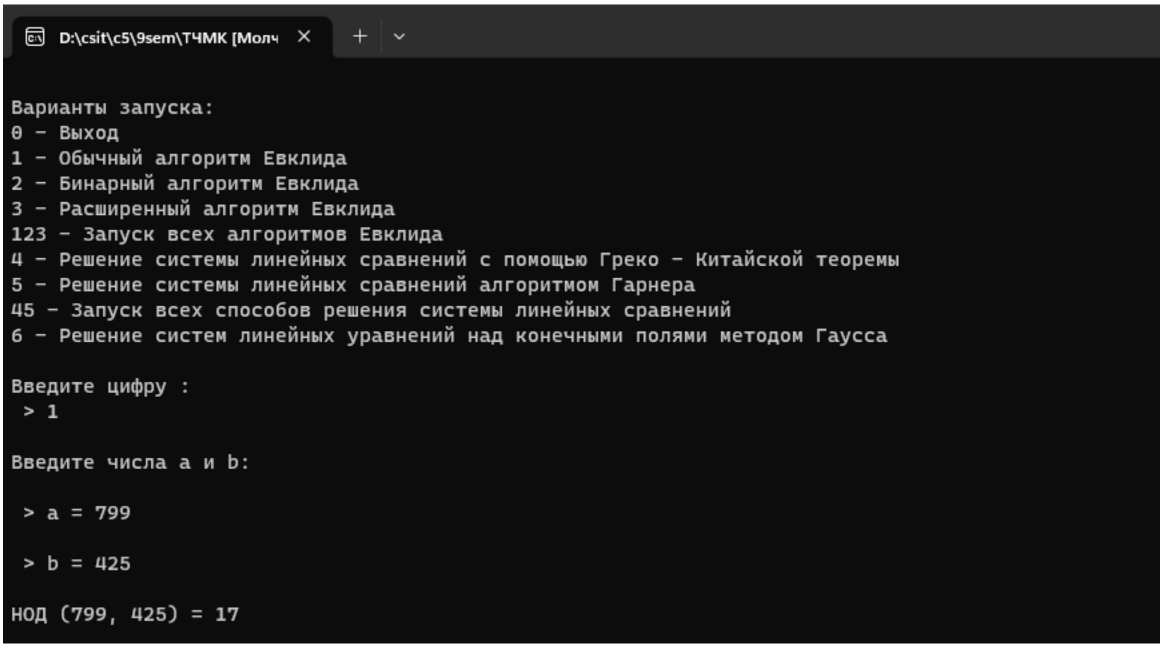


Рисунок 1 – Пример работы алгоритма Евклида

2.2 Бинарный алгоритм Евклида

Псевдокод функции algEuclidBinary(a, b):

Вход: числа a, b.

Выход: nod = НОД(a,b).

1. Если a равно 0, то вернуть b
2. Иначе если b равно 0 или a равно b, то вернуть a
3. Иначе если a равно 1 или b равно 1, то вернуть 1
4. Иначе если a и b чётные, то вернуть algEuclidBinary(a / 2, b / 2) умноженное на 2

5. Иначе если a чётное и b нечётное, то вернуть algEuclidBinary(a / 2, b)
6. Иначе если a нечётное и b чётное, то вернуть algEuclidBinary(a, b / 2)
7. Иначе если a и b нечётные, то
    а) Если b больше a, то вернуть algEuclidBinary((b - a) / 2, a)
    б) Иначе вернуть algEuclidBinary((a - b) / 2, b)

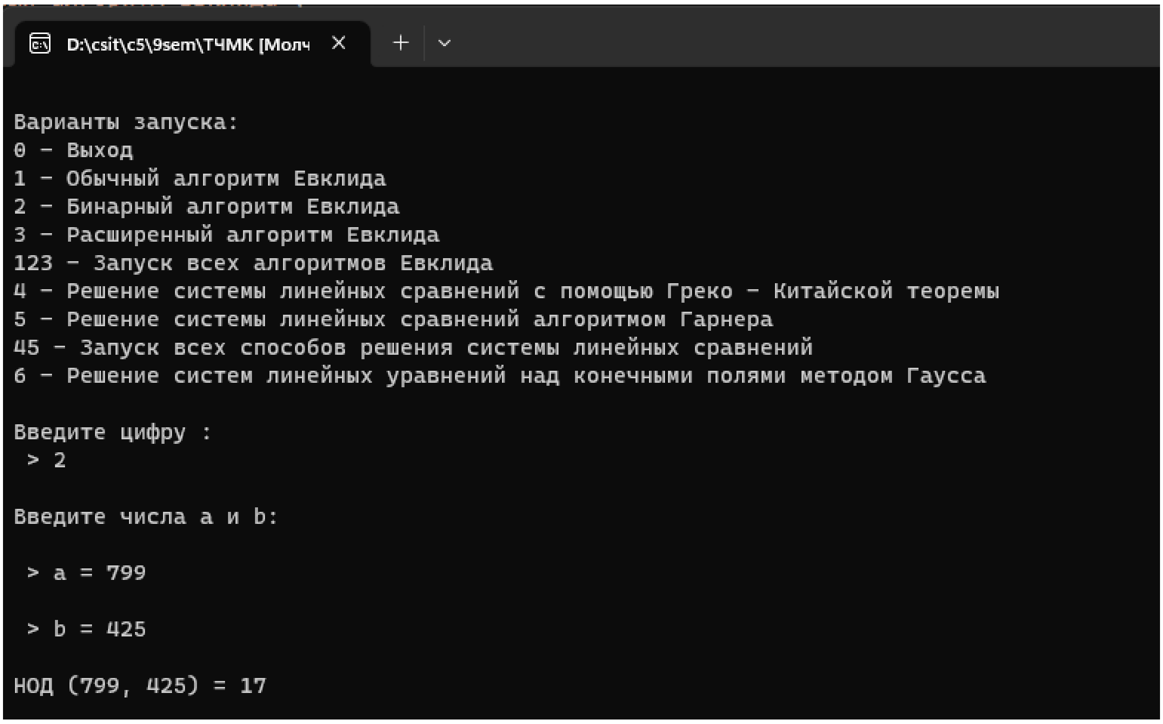


Рисунок 2 – Пример работы бинарного алгоритма Евклида

2.3 Расширенный алгоритм Евклида

<u>Псевдокод функции algEuclidExtended(a, b, &x, &y):</u>

Вход: числа a, b, x, y, где x и y передаются по ссылке.

Выход: nod = НОД(a,b), а также коэффициенты x и y.

1. Если a равно 0, то
    а) x := 0
    б) y := 1
    в) Вернуть b
2. Объявить переменные xi и yi
3. Объявить переменную nod и присвоить ей результат вызова algEuclidExtended (b mod a, a, xi, yi)
4. x := (yi - (b / a) * xi)
5. y := xi
6. Вернуть nod

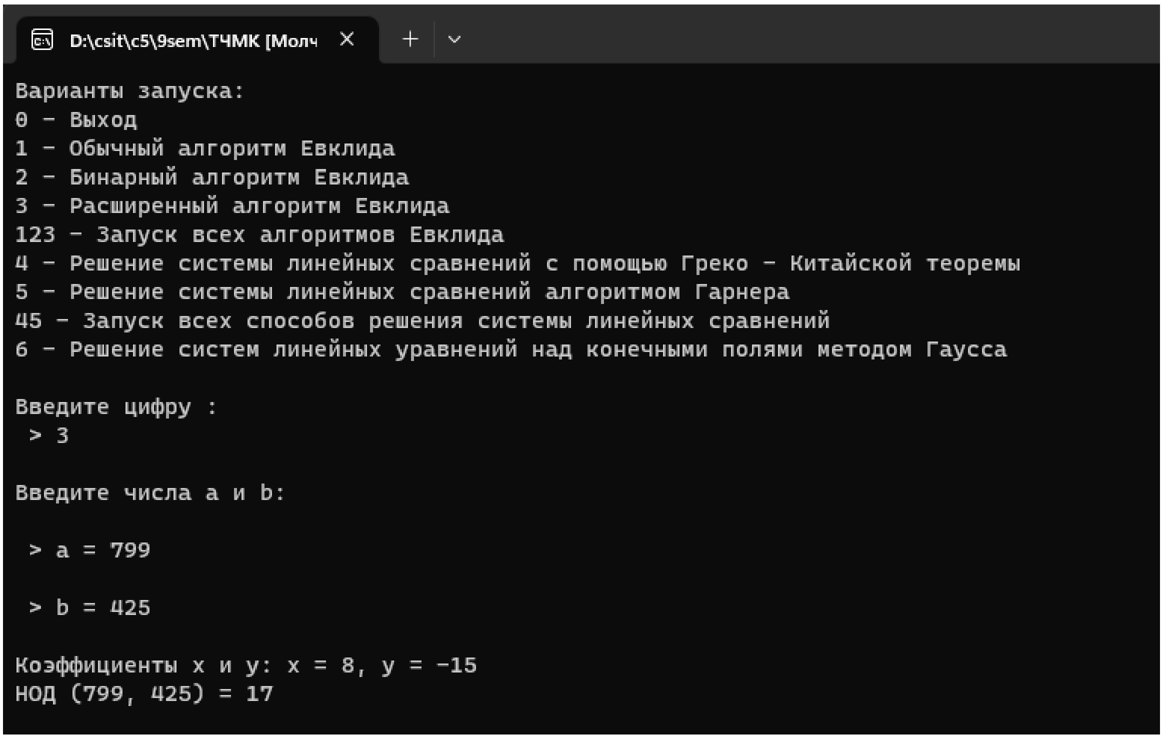


Рисунок 3 – Пример работы расширенного алгоритма Евклида

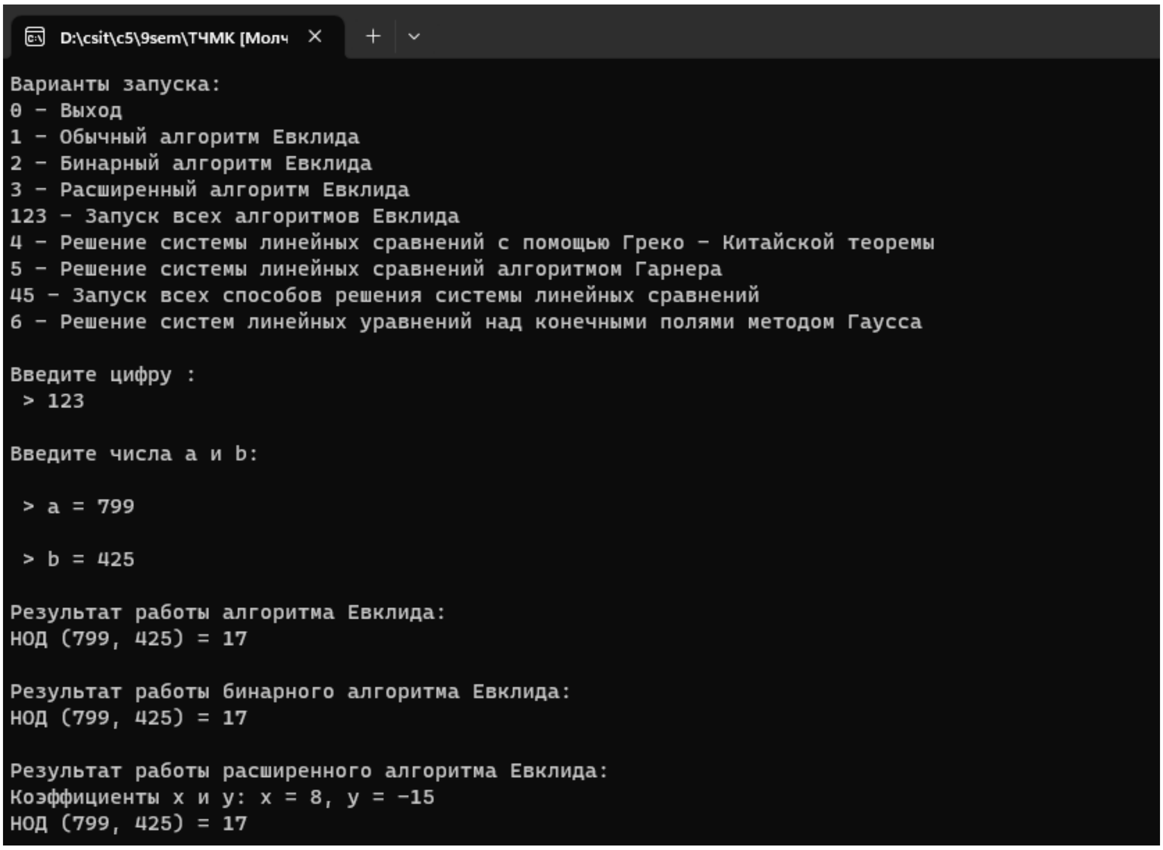


Рисунок 4 – Пример работы всех трех видов алгоритмов Евклида

2.4 Алгоритм решения системы линейных сравнений с помощью Греко-Китайской теоремы

Псевдокод функции congruencesFunc1(congruences, k):

Вход: congruences - массив пар, содержащий параметры введенных сравнений (первый элемент - a_i, второй элемент - m_i), k - количество сравнений.

Выход: число u, являющееся решением системы сравнений.

1. Объявить массив c_i размером k
2. Объявить массив d_i размером k
3. M := 1
4. Для i от 0 до k-1:
   а) m_i := congruences[i].second
   б) Умножить M на m_i
5. Для i от 0 до k-1:
   а) m_i := congruences[i].second
   б) c_i[i] := M / m_i
   в) Объявить переменные x и y
   г) Вызвать algEuclidExtended(c_i[i], m_i, x, y)
   д) Если x меньше 0, то прибавить m_i к x
   е) d_i[i] := x
6. u:= 0
7. Для i от 0 до k-1, делать:
   а) Умножить u на c_i[i] * d_i[i] * congruences[i].first
   б) Получить остаток от деления u на M
   в) Если u меньше 0, то прибавить M к u
8. Вернуть u

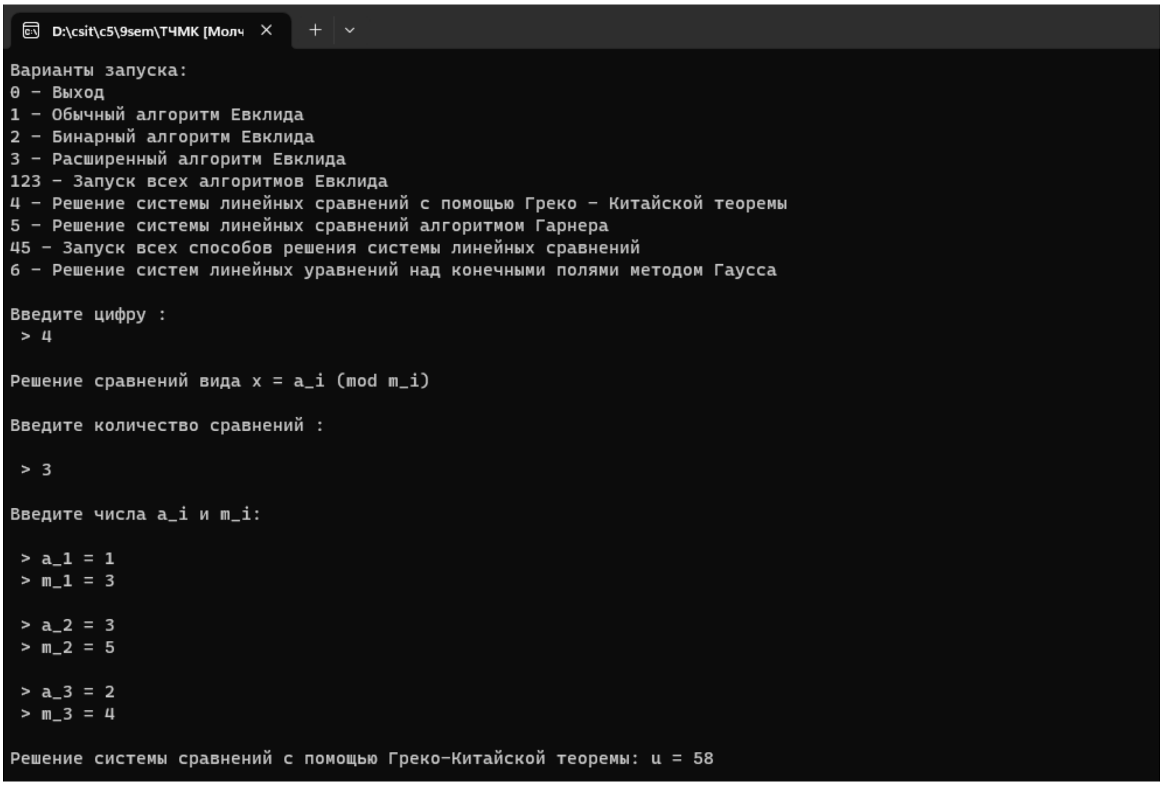


Рисунок 5 – Пример работы алгоритма решения системы линейных сравнений с помощью Греко-Китайской теоремы

2.5 Алгоритм решения системы линейных сравнений алгоритмом Гарнера

Псевдокод функции congruencesFuncGarner(congruences, k):

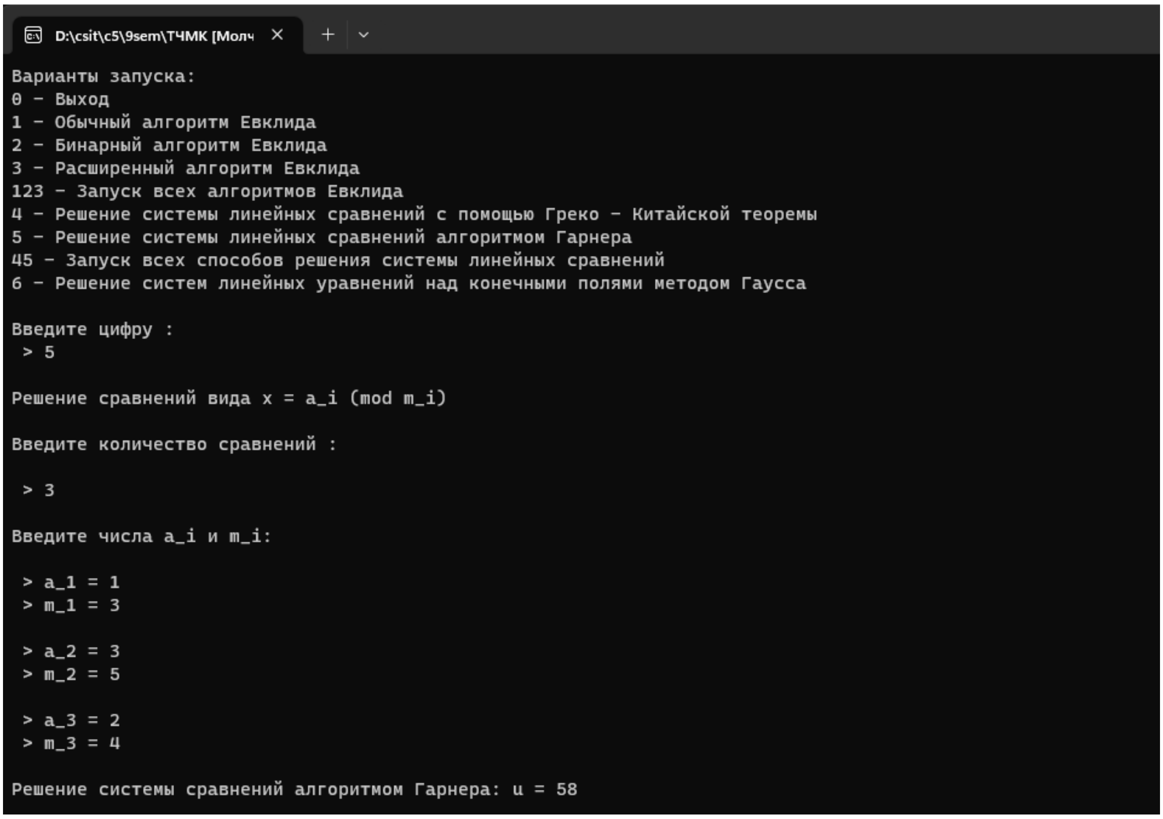
Вход: congruences - массив пар, содержащий параметры введенных сравнений (первый элемент - a_i, второй элемент - m_i), k - количество сравнений.

Выход: число u, являющееся решением системы сравнений.

1. Объявить массив c_i размером k
2. Объявить двумерный массив c_ij размером k x k и заполнить его значениями из c_i
3. M := 1
4. Для i от 0 до k-1:
   а) m_i := congruences[i].second
   б) Умножить M на m_i: M *= m_i
5. Для каждого i от 0 до k-1:
   а) m_i := congruences[i].second
   б) Для j от 0 до k-1:
      i. Если i не равно j, то
         A. Объявить переменные x и y

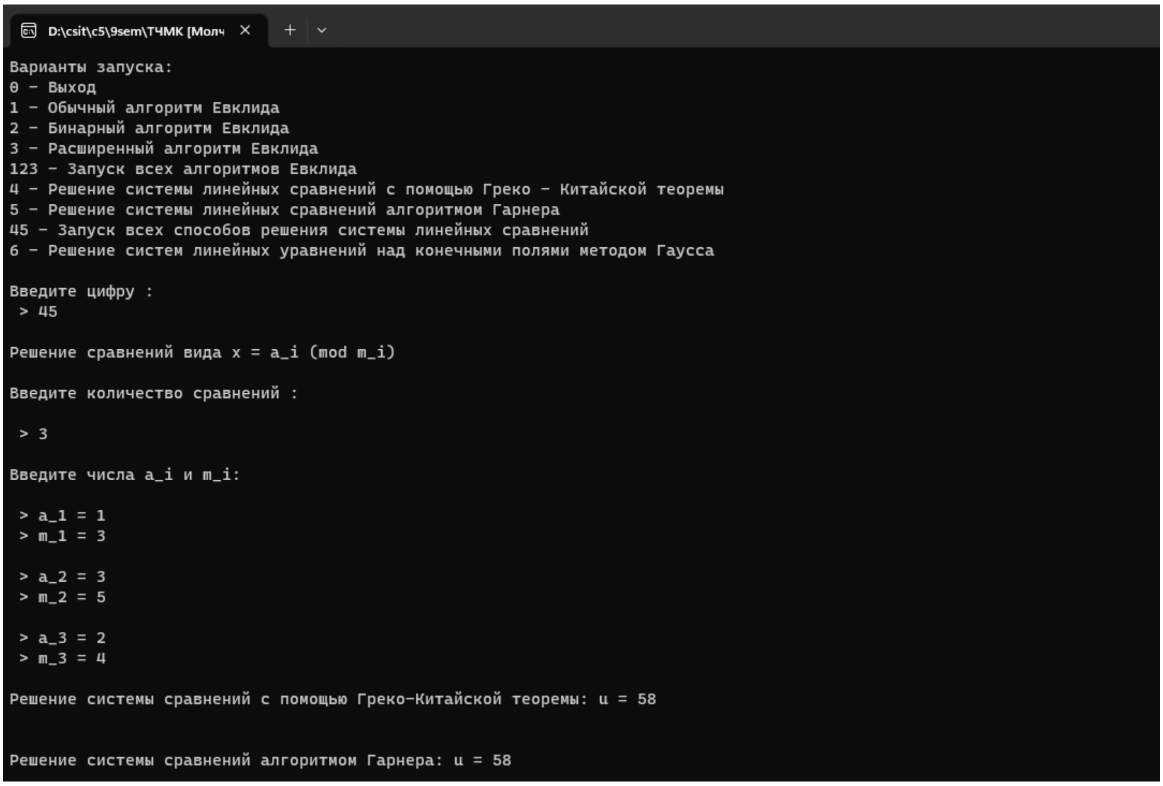
Б. Вызвать algEuclidExtended(m_i, congruences[j].second, x, y)
В. Если x меньше 0, то прибавить congruences[j].second к x
Г. Присвоить c_ij[i][j] := x

6. Объявить массив q_i размером k
7. q_i[0] := congruences[0].first mod congruences[0].second
8. Для i от 1 до k-1, делать:
   а) q_num := 0
   б) q_i[i] := congruences[i].first
   в) Пока q_num меньше i:
      i. Вычесть из q_i[i] := q_i[q_num]
      ii. Умножить q_i[i] на c_ij[q_num][i]
      iii. Увеличить q_num на 1
   г) Получить остаток от деления q_i[i] на congruences[i].second
   д) Если q_i[i] меньше 0, то прибавить congruences[i].second к q_i[i]
9. u := q_i[0]
10. Для i от 1 до k-1:
    а) mult_m := 1
    б) Для j от 0 до i-1:
       i. Умножить mult_m на congruences[j].second
    в) Умножить q_i[i] на mult_m
    г) Прибавить q_i[i] умноженное на mult_m к u
11. Вернуть u



```
D:\csit\c5\9sem\ТЧМК [Молч
Варианты запуска:
0 - Выход
1 - Обычный алгоритм Евклида
2 - Бинарный алгоритм Евклида
3 - Расширенный алгоритм Евклида
123 - Запуск всех алгоритмов Евклида
4 - Решение системы линейных сравнений с помощью Греко - Китайской теоремы
5 - Решение системы линейных сравнений алгоритмом Гарнера
45 - Запуск всех способов решения системы линейных сравнений
6 - Решение систем линейных уравнений над конечными полями методом Гаусса

Введите цифру :
 > 5

Решение сравнений вида x = a_i (mod m_i)

Введите количество сравнений :

 > 3

Введите числа a_i и m_i:

 > a_1 = 1
 > m_1 = 3

 > a_2 = 3
 > m_2 = 5

 > a_3 = 2
 > m_3 = 4

Решение системы сравнений алгоритмом Гарнера: u = 58
```

Рисунок 6 – Пример работы алгоритма решения системы линейных сравнений алгоритмом Гарнера



```
D:\csit\c5\9sem\ТЧМК [Молч
Варианты запуска:
0 - Выход
1 - Обычный алгоритм Евклида
2 - Бинарный алгоритм Евклида
3 - Расширенный алгоритм Евклида
123 - Запуск всех алгоритмов Евклида
4 - Решение системы линейных сравнений с помощью Греко - Китайской теоремы
5 - Решение системы линейных сравнений алгоритмом Гарнера
45 - Запуск всех способов решения системы линейных сравнений
6 - Решение систем линейных уравнений над конечными полями методом Гаусса

Введите цифру :
 > 45

Решение сравнений вида x = a_i (mod m_i)

Введите количество сравнений :

 > 3

Введите числа a_i и m_i:

 > a_1 = 1
 > m_1 = 3

 > a_2 = 3
 > m_2 = 5

 > a_3 = 2
 > m_3 = 4

Решение системы сравнений с помощью Греко-Китайской теоремы: u = 58


Решение системы сравнений алгоритмом Гарнера: u = 58
```

Рисунок 7 – Пример работы сразу двух алгоритмов решения системы линейных сравнений

2.6 Алгоритм решения систем линейных уравнений над конечными полями методом Гаусса

Псевдокод функции gaussFunc(n, m, mod, matr):

Данная функция находит общее решение системы линейных уравнений.

Вход: n и m - размерность матрицы системы, mod - модуль, matr - матрица системы.

Выход: matr - матрица общего решения системы.

1. Для i от 0 до n-1:
   а) jEl := 0
   б) Пока matr[i][jEl] равно 0:
      i. Если jEl не равно m-1, то увеличить jEl на 1
      ii. Иначе вернуть matr
   в) elem := matr[i][jEl]
   г) Объявить переменные x и y
   д) Вызвать algEuclidExtended(elem, mod, x, y)
   е) Если x меньше 0, то прибавить mod к x
   ж) obrElem := x
   з) Для j от jEl до m-1:
      i. Умножить matr[i][j] на obrElem
      ii. Получить остаток от деления matr[i][j] на mod
   и) Объявить переменную multElem
   к) Для imult от 0 до n-1, если imult не равно i, то:
      i. multElem := -1 * matr[imult][jEl]
      ii. Если multElem равно 0, то перейти на следующую итерацию цикла, иначе:
      iii. Объявить массив multVec размером m и заполнить его нулями
      iv. Для jDop от 0 до m-1: Умножить multVec[jDop] на multElem * matr[i][jDop]
      v. Для j1 от 0 до m-1:
         А. Прибавить multVec[j1] к matr[imult][j1]
         Б. Получить остаток от деления matr[imult][j1] на mod
         В. Если matr[imult][j1] меньше 0, то прибавить mod к matr[imult][j1]
2. Вернуть matr

Псевдокод функции checkGaussRes(n, m, matr):

Данная функция проверяет результата выполнения функции gaussFunc и выводит информации о решениях системы линейных уравнений.

Вход: n и m - размерность матрицы системы, matr - матрица системы.

Выход: информации о решениях системы линейных уравнений.

1. Для i от 0 до n-1:
   а) firstNotZero := -1
   б) Для j от 0 до m-1:
      i. Если matr[i][j] не равно 0, то:
         А. firstNotZero := j
         Б. Прервать цикл
   в) Если firstNotZero равно -1, то:
      i. Удалить строку i из матрицы matr
   г) Иначе если firstNotZero равно m-1, то:
      i. Вывести матрицу matr
      ii. Вернуть "Система не имеет решений"
2. Если размер матрицы matr равен m-1, то:
   а) Вывести матрицу matr
   б) Вернуть "Система имеет единственное решение"
3. Иначе:
   а) Вывести матрицу matr
   б) Вернуть "Система имеет бесконечное количество решений"
4. Вернуться

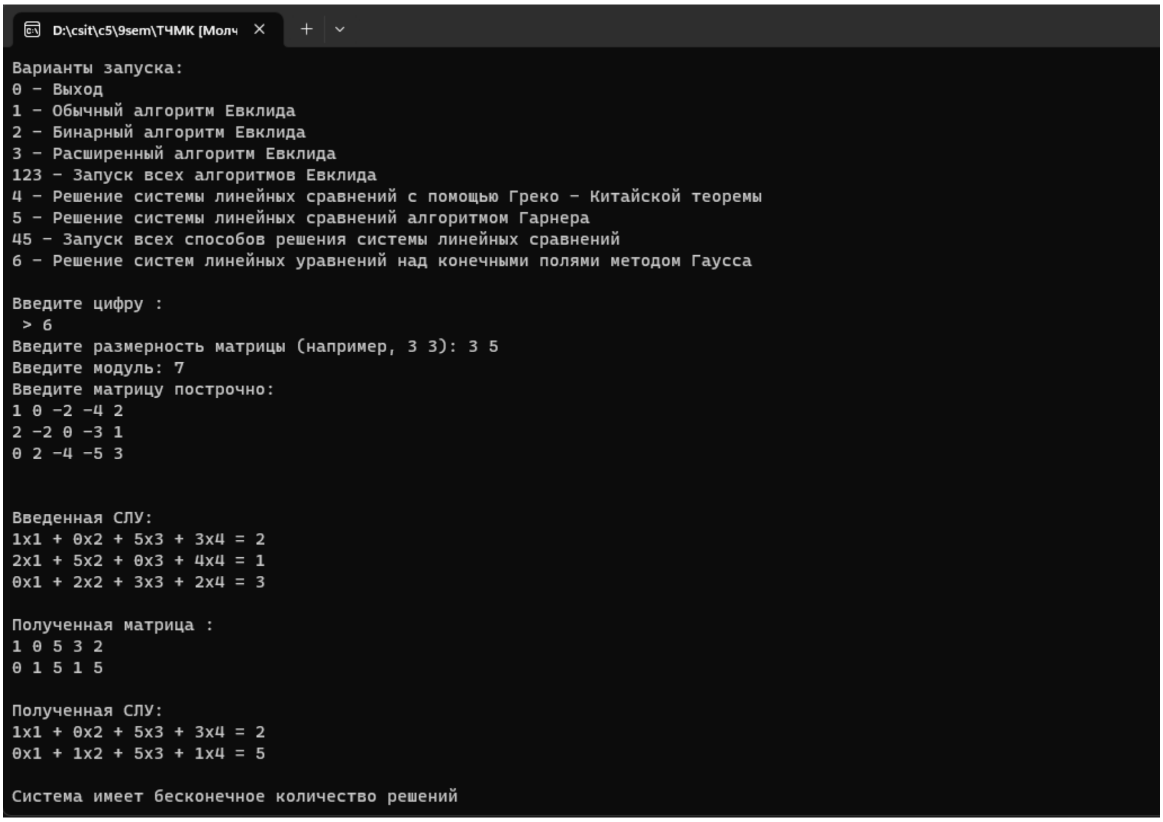


Рисунок 8 – Пример работы алгоритма решения систем линейных уравнений над конечными полями методом Гаусса

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы был рассмотрен алгоритм Евклида нахождения наибольшего общего делителя, а также его модификации: бинарный и расширенный алгоритмы Евклида. Также, в работе были рассмотрены алгоритмы решения систем сравнений с помощью Греко-Китайской теоремы и алгоритма Гарнера. Кроме этого, был разобран алгоритм решения систем линейных уравнений над конечными полями методом Гаусса.

В практической части лабораторной работы была написана программа на языке C++. Алгоритмы реализованы в виде функций, что позволяет тестировать корректность их работы, а также быстродействие.

ПРИЛОЖЕНИЕ А

Листинг программы

```
#include <iostream>
#include <vector>

using namespace std;

//Обычный алгоритм Евклида
int algEuclidClassic(int a, int b) {
    if (a < b) {
        int dop = a;
        a = b;
        b = dop;
    }
    while (b != 0) {
        int b_dop = b;
        b = a % b;
        a = b_dop;
    }
    return a;
}

//Бинарный алгоритм Евклида
int algEuclidBinary(int a, int b) {
    if (a == 0)
        return b;
    else if (b == 0 || a == b)
        return a;
    else if (a == 1 || b == 1)
        return 1;
    else if ((a & 1) == 0 && (b & 1) == 0) {
        return algEuclidBinary(a >> 1, b >> 1) << 1;
    }
    else if ((a & 1) == 0 && (b & 1) != 0) {
        return algEuclidBinary(a >> 1, b);
    }
    else if ((a & 1) != 0 && (b & 1) == 0) {
        return algEuclidBinary(a, b >> 1);
    }
    else if ((a & 1) != 0 && (b & 1) != 0) {
        if (b > a)
            return algEuclidBinary((b - a) >> 1, a);
        else
            return algEuclidBinary((a - b) >> 1, b);
    }
}

//Расширенный алгоритм Евклида
int algEuclidExtended(int a, int b, int& x, int& y) {
    if (a == 0) {
        x = 0;
        y = 1;
        return b;
```

```
    }
    int xi, yi;
    int nod = algEuclidExtended(b % a, a, xi, yi);
    x = yi - (b / a) * xi;
    y = xi;
    return nod;
}

//Решение системы линейных сравнений с помощью Греко-Китайской теоремы
int congruencesFunc1(vector <pair <int, int>> congruences, int k) {
    vector <int> c_i(k);
    vector <int> d_i(k);
    int M = 1;
    for (int i = 0; i < k; i++) {
        int m_i = congruences[i].second;
        M *= m_i;
    }
    for (int i = 0; i < k; i++) {
        int m_i = congruences[i].second;
        c_i[i] = M / m_i;
        int x, y;
        algEuclidExtended(c_i[i], m_i, x, y);
        if (x < 0)
            x += m_i;
        d_i[i] = x;
    }
    int u = 0;
    for (int i = 0; i < k; i++) {
        u += c_i[i] * d_i[i] * congruences[i].first;
        u %= M;
        if (u < 0)
            u += M;
    }
    return u;
}

//Решение системы линейных сравнений алгоритмом Гарнера
int congruencesFuncGarner(vector <pair <int, int>> congruences, int k) {
    vector <int> c_i(k);
    vector <vector <int>> c_ij(k, c_i);
    int M = 1;
    for (int i = 0; i < k; i++) {
        int m_i = congruences[i].second;
        M *= m_i;
    }
    for (int i = 0; i < k; i++) {
        int m_i = congruences[i].second;
        for (int j = 0; j < k; j++) {
            if (i != j) {
                int x, y;
                algEuclidExtended(m_i, congruences[j].second, x, y);
                if (x < 0)
                    x += congruences[j].second;
                c_ij[i][j] = x;
            }
        }
    }
```

```
    vector <int> q_i(k);
    q_i[0] = congruences[0].first % congruences[0].second;
    for (int i = 1; i < k; i++) {
        int q_num = 0;
        q_i[i] = congruences[i].first;
        while (q_num < i) {
            q_i[i] -= q_i[q_num];
            q_i[i] *= c_ij[q_num][i];
            q_num++;
        }
        q_i[i] %= congruences[i].second;
        if (q_i[i] < 0)
            q_i[i] += congruences[i].second;
    }
    int u = q_i[0];
    for (int i = 1; i < k; i++) {
        int mult_m = 1;
        for (int j = 0; j < i; j++) {
            mult_m *= congruences[j].second;
        }
        u = u + (q_i[i] * mult_m);
    }
    return u;
}

void printMatr(int n, int m, vector <vector<int>> matr) {
    cout << "\nПолученная матрица :\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout << matr[i][j] << " ";
        }
        cout << "\n";
    }
}
void printSLU(int n, int m, vector <vector<int>> matr) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (j == 0)
                cout << matr[i][j] << "x" << 1;
            else if (j == m - 1)
                cout << " = " << matr[i][j];
            else
                cout << " + " << matr[i][j] << "x" << j + 1;
        }
        cout << "\n";
    }
}
vector <vector <int>> gaussFunc(int n, int m, int mod, vector <vector
<int>> matr) {
    for (int i = 0; i < n; i++) {
        int jEl = 0;
        while (matr[i][jEl] == 0)
            if (jEl != m - 1)
                jEl++;
            else
                return matr;
        int elem = matr[i][jEl];
```

```
        int x, y;
        algEuclidExtended(elem, mod, x, y);
        if (x < 0)
            x += mod;
        int obrElem = x;
        for (int j = jEl; j < m; j++) {
            matr[i][j] *= obrElem;
            matr[i][j] %= mod;
        }
        int multElem;
        for (int imult = 0; imult < n; imult++) {
            if (imult != i) {
                multElem = -1 * matr[imult][jEl];
                if (multElem != 0) {
                    vector <int> multVec(m, 0);
                    for (int jDop = 0; jDop < m; jDop++) {
                        multVec[jDop] = multElem * matr[i][jDop];
                    }
                    for (int j1 = 0; j1 < m; j1++) {
                        matr[imult][j1] += multVec[j1];
                        matr[imult][j1] %= mod;
                        if (matr[imult][j1] < 0)
                            matr[imult][j1] += mod;
                    }
                }
            }
        }
    }
    return matr;
}

void checkGaussRes(int n, int m, vector <vector <int>> matr) {
    for (int i = 0; i < n; i++) {
        int firstNotZero = -1;
        for (int j = 0; j < m; j++) {
            if (matr[i][j] != 0) {
                firstNotZero = j;
                break;
            }
        }
        if (firstNotZero == -1)
            matr.erase(matr.begin() + i);
        else if (firstNotZero == m - 1) {
            printMatr(matr.size(), m, matr);
            cout << "\nПолученная СЛУ:\n";
            printSLU(matr.size(), m, matr);
            cout << "\nСистема не имеет решений\n\n";
            return;
        }
    }
    if (matr.size() == m - 1) {
        printMatr(matr.size(), m, matr);
        cout << "\nПолученная СЛУ:\n";
        printSLU(matr.size(), m, matr);
        cout << "\nСистема имеет единственное решение\n\n";
    }
    else {
```

```
        printMatr(matr.size(), m, matr);
        cout << "\nПолученная СЛУ:\n";
        printSLU(matr.size(), m, matr);
        cout << "\nСистема имеет бесконечное количество решений\n\n";
    }
    return;
}

int main()
{
    setlocale(LC_ALL, "rus");
    bool stopProg = 0;
    while (stopProg != 1)
    {
        cout << "\nВарианты запуска:\n0 - Выход";
        cout << "\n1 - Обычный алгоритм Евклида";
        cout << "\n2 - Бинарный алгоритм Евклида";
        cout << "\n3 - Расширенный алгоритм Евклида";
        cout << "\n123 - Запуск всех алгоритмов Евклида";
        cout << "\n4 - Решение системы линейных сравнений с помощью
Греко - Китайской теоремы";
        cout << "\n5 - Решение системы линейных сравнений алгоритмом
Гарнера";
        cout << "\n45 - Запуск всех способов решения системы линейных
сравнений";
        cout << "\n6 - Решение систем линейных уравнений над конечными
полями методом Гаусса";
        cout << "\n\nВведите цифру : \n > ";
        int userChoice;
        cin >> userChoice;
        if (userChoice == 0)
        {
            stopProg = 1;
            continue;
        }
        else if (userChoice == 1 || userChoice == 2 || userChoice == 3
|| userChoice == 123) {
            int a;
            int b;
            int nod = 0;
            cout << "\nВведите числа a и b:\n\n > a = ";
            cin >> a;
            cout << "\n > b = ";
            cin >> b;
            cout << "\n";
            if (a < 1 || b < 1) {
                cout << "Введите числа больше 0";
                continue;
            }
            if (a < b && userChoice != 1) {
                int dop = a;
                a = b;
                b = dop;
            }
            if (userChoice == 1)
                nod = algEuclidClassic(a, b);
            if (userChoice == 2)
```

```
                nod = algEuclidBinary(a, b);
            if (userChoice == 3) {
                int x, y;
                nod = algEuclidExtended(a, b, x, y);
                cout << "Коэффициенты x и y: x = " << x << ", y = " << y
<< "\n";
            }
            if (userChoice == 123) {
                int nod1 = algEuclidClassic(a, b);
                cout << "Результат работы алгоритма Евклида:\n";
                cout << "НОД (" << a << ", " << b << ") = " << nod1 <<
"\n\n";
                int nod2 = algEuclidBinary(a, b);
                cout << "Результат работы бинарного алгоритма
Евклида:\n";
                cout << "НОД (" << a << ", " << b << ") = " << nod2 <<
"\n\n";
                int x, y;
                int nod3 = algEuclidExtended(a, b, x, y);
                cout << "Результат работы расширенного алгоритма
Евклида:\n";
                cout << "Коэффициенты x и y: x = " << x << ", y = " << y
<< "\n";
                cout << "НОД (" << a << ", " << b << ") = " << nod3 <<
"\n";
                continue;
            }
            cout << "НОД (" << a << ", " << b << ") = " << nod << "\n";
        }
        else if (userChoice == 4 || userChoice == 5 || userChoice == 45)
{
            int congruencesCount;
            cout << "\nРешение сравнений вида x = a_i (mod m_i)\n";
            cout << "\nВведите количество сравнений :\n\n > ";
            cin >> congruencesCount;
            vector <pair <int, int>> congruences;
            cout << "\nВведите числа a_i и m_i:\n";
            for (int i = 1; i <= congruencesCount; i++)
            {
                int a_i;
                int m_i;
                cout << "\n > a_" << i << " = ";
                cin >> a_i;
                cout << " > m_" << i << " = ";
                cin >> m_i;
                congruences.push_back(make_pair(a_i, m_i));
            }
            bool stopNotPrime = 0;
            for (int i = 0; i < congruencesCount; i++) {
                for (int j = i + 1; j < congruencesCount; j++)
                    if (algEuclidClassic(congruences[i].second,
congruences[j].second) != 1) {
                        cout << "Не все числа попарно взаимно
простые\n";
                        stopNotPrime = 1;
                        break;
                    }
```

```
                if (stopNotPrime == 1)
                    break;
            }
            if (stopNotPrime == 1)
                continue;
            if (userChoice == 4) {
                int u = congruencesFunc1(congruences, congruencesCount);
                cout << "\nРешение системы сравнений с помощью Греко-
Китайской теоремы: u = " << u << "\n";
            }
            else if (userChoice == 5) {
                int u = congruencesFuncGarner(congruences,
congruencesCount);
                cout << "\nРешение системы сравнений алгоритмом Гарнера:
u = " << u << "\n";
            }
            else if (userChoice == 45) {
                int u1 = congruencesFunc1(congruences,
congruencesCount);
                cout << "\nРешение системы сравнений с помощью Греко-
Китайской теоремы: u = " << u1 << "\n\n";
                int u2 = congruencesFuncGarner(congruences,
congruencesCount);
                cout << "\nРешение системы сравнений алгоритмом Гарнера:
u = " << u2 << "\n";
            }
        }
        else if (userChoice == 6) {
            int n, m, mod;
            cout << "Введите размерность матрицы (например, 3 3): ";
            cin >> n >> m;
            cout << "Введите модуль: ";
            cin >> mod;
            vector <vector <int>> matr(n, vector <int> (m, 0));
            cout << "Введите матрицу построчно: \n";
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < m; j++) {
                    int elem;
                    cin >> elem;
                    if (elem < 0)
                        elem += mod;
                    matr[i][j] = elem;
                }
            }
            cout << "\n\nВведенная СЛУ:\n";
            printSLU(n, m, matr);
            vector <vector <int>> matrNew = gaussFunc(n, m, mod, matr);

            checkGaussRes(n, m, matrNew);

        }
    }
    return 0;
}
```