

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
Серебрякова Алексея Владимировича

Преподаватель
профессор, д.ф.-м.н.

_____ В. А. Молчанов
подпись, дата

Саратов 2023

1 Постановка задачи

Цель работы:

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

Задачи работы:

- изучить алгоритмы Евклида (обычный, бинарный и расширенный) вычисления НОД целых чисел и привести их программную реализацию;
- изучить алгоритм решения систем сравнений и привести его программную реализацию;
- изучить метод Гаусса решения систем линейных уравнений над конечными полями и привести его программную реализацию.

2 Теоретические сведения

2.1 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} :

$$a_0 = a_1 q_1 + a_2, 0 \leq a_2 < a_1,$$

$$a_1 = a_2 q_2 + a_3, 0 \leq a_3 < a_2,$$

...

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

$$a_{k-1} = a_k q_k$$

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

Сложность алгоритма Евклида: $BAE(n) = O(n^2)$.

Описание алгоритма Евклида:

Вход: целые числа $a, b; 0 < b < a$.

Выход: $d = \text{НОД}(a, b)$.

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: $d = \text{НОД}(a, b)$.

2.2 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 $\text{НОД}(a, b) = ax + by$ для некоторых $x, y \in \mathbb{Z}$. 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:

$$\begin{array}{ll}
 a_0 = a, & a_0 = ax_0 + by_0, \\
 a_1 = b, & a_1 = ax_1 + by_1, \\
 a_2 = a_0 - a_1q_1, & a_2 = ax_2 + by_2, \\
 & \dots \\
 a_i = a_{i-2} - a_{i-1}q_{i-1}, & a_i = ax_i + by_i, \\
 & \dots \\
 a_k = a_{k-2} - a_{k-1}q_{k-1}, & a_k = ax_k + by_k, \\
 0 = a_{k-1} - a_kq_k, & 0 = ax_{k+1} + by_{k+1}.
 \end{array}$$

В правом столбце все элементы $a_k, a_{k-1}, a_{k-2}, \dots, a_1, a_0$ 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 $\text{НОД}(a, b) = a_k = ax_k + by_k$.

Сложность расширенного алгоритма Евклида: $BAE(n) = O(n^2)$.

Описание расширенного алгоритма Евклида:

Вход: целые числа $a, b; 0 < b < a$.

Выход: d, x, y , такие, что $\text{НОД}(a, b) = d = ax + by$.

1. Положить $a_0 = a, a_1 = b, x_{-1} = 1, x_0 = 0, y_{-1} = 0, y_1 = 1, i = 1$.
2. Найти остаток a_{i+1} от деления a_{i-1} на a_i и частное d_i .
3. Положить $x_i = x_{i-2} - d_i x_{i-1}$ и $y_i = y_{i-2} - d_i y_{i-1}$.
4. Если $a_{i+1} = 0$, то положить $d = ax_k + by_k$. В противном случае положить $i = i + 1$ и вернуться на шаг 2.
5. Результат: $d = ax + by = \text{НОД}(a, b)$.

2.3 Бинарный алгоритм Евклида

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

$$\text{НОД}(2 * a, 2 * b) = 2 * \text{НОД}(a, b);$$

$$\text{НОД}(2 * a, 2 * b + 1) = \text{НОД}(a, 2 * b + 1);$$

$$\text{НОД}(-a, b) = \text{НОД}(a, b).$$

Сложность бинарного алгоритма Евклида: $BAE(n) = O(n^2)$.

Описание бинарного алгоритма Евклида:

Вход: целые числа a, b ; $0 < b < a$.

Выход: $d = \text{НОД}(a, b)$.

1. Положить $a_0 = a, b_0 = b, i = 1, k = 1$.
2. Если $a_i = 0$, то положить $d = b_i$ и перейти к шагу 9.
3. Если $b_i = 0$, то положить $d = a_i$ и перейти к шагу 9.
4. Если $a_i = 1$, то положить $d = 1$ и перейти к шагу 9.
5. Если a_i и b_i четные, то положить $a_{i+1} = \frac{a_i}{2}, b_{i+1} = \frac{b_i}{2}, k = k * 2$,
 $i = i + 1$ и перейти к шагу 2.
6. Если a_i четное и b_i нечетное, то положить $a_{i+1} = \frac{a_i}{2}, b_{i+1} = b_i, i = i + 1$ и перейти к шагу 2.
7. Если a_i нечетное и b_i четное, то положить $a_{i+1} = a_i, b_{i+1} = \frac{b_i}{2}, i = i + 1$ и перейти к шагу 2.
8. Если a_i и b_i нечетные:
9. Если $b > a$, то положить $a_{i+1} = \frac{b_i - a_i}{2}, b_{i+1} = a_i, i = i + 1$ и перейти к шагу 2.
10. Если $b < a$, то положить $a_{i+1} = \frac{a_i - b_i}{2}, b_{i+1} = b_i, i = i + 1$ и перейти к шагу 2.
11. Положить $d = d * k$.
12. Результат: $d = \text{НОД}(a, b)$.

2.4 Решение систем сравнений при помощи греко-китайской теоремы об остатках

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

$$\begin{cases} u \equiv u_1 \pmod{m_1} \\ \dots \\ u \equiv u_k \pmod{m_k} \end{cases}$$

имеет единственное неотрицательное решение по модулю M .

Пусть $c_i = \frac{M}{m_i}$ и $d_i \equiv c_i^{-1} \pmod{m_i}$. Тогда решение системы сравнений находится по формуле:

$$u \equiv \sum_{i=1}^k u_i c_i d_i \pmod{M}$$

Алгоритм:

Вход: целые попарно взаимно простые числа m_1, \dots, m_k и коэффициенты системы сравнений u_1, \dots, u_k .

Выход: решение системы u .

1. Положим $M = \prod_{i=1}^k m_i$.
2. Для всех $i = \overline{1, k}$ вычисляем $c_i = \frac{M}{m_i}$.
3. Для всех $i = \overline{1, k}$ вычисляем при помощи расширенного алгоритма Евклида $d_i \equiv c_i^{-1} \pmod{m_i}$.
4. Вычисляем решение системы сравнений $\sum_{i=1}^k u_i c_i d_i \pmod{M}$.
5. Результат: $u \equiv \sum_{i=1}^k u_i c_i d_i \pmod{M}$.

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

2.5 Решение системы сравнений при помощи алгоритма Гарнера

Существует и другой способ решения системы, носящий название алгоритма Гарнера.

Пусть $M = \prod_{i=1}^k m_i$, числа m_1, \dots, m_k попарно взаимно просты, и $c_{ij} \equiv m_i^{-1}(\text{mod } m_j)$, $i \neq j, i, j \in \{1, \dots, k\}$. Тогда решение системы 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 $0 \leq q_i < m_i$, $i \in \{1, \dots, k\}$, и числа q_i вычисляются по формулам

$$q_1 = u_1 \text{ mod } m_1,$$

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

...

$$q_k = \left((u_k - q_1) c_{1k} - q_2 c_{2k} - \dots - q_{k-1} c_{k-1k} \right) \text{ mod } m_k$$

Алгоритм:

Вход: целые попарно взаимно простые числа m_1, \dots, m_k и коэффициенты системы сравнений u_1, \dots, u_k .

Выход: решение системы u .

1. Положим $M = \prod_{i=1}^k m_i$.
2. Для всех $i = \overline{1, k}$ и $j = \overline{1, k}$ вычисляем при помощи расширенного алгоритма Евклида $c_{ij} \equiv m_i^{-1}(\text{mod } m_j)$.
3. Положить $q_1 = u_1 \text{ mod } m_1$ и $i = 2$.
4. Для всех $i = \overline{2, k}$ вычисляем $q_k = \left((u_k - q_1) c_{1k} - q_2 c_{2k} - \dots - q_{k-1} c_{k-1k} \right) \text{ mod } m_k$.
5. Вычисляем решение системы сравнений $q_1 + q_2 m_1 + q_3 m_1 m_2 + \dots + q_k m_1 \dots m_{k-1}$.
6. Результат: $u = q_1 + q_2 m_1 + q_3 m_1 m_2 + \dots + q_k m_1 \dots m_{k-1}$.

Сложность вычислений равна $O(k^2)$.

2.6 Решение систем линейных уравнений при помощи метода Гаусса

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

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

$$(s) \begin{cases} a_{11}x_1 + \dots + a_{1n}x_n = b_1(1) \\ \dots \\ a_{m1}x_1 + \dots + a_{mn}x_n = b_m(m) \end{cases}$$

, где $(1), \dots, (m)$ – линейные уравнения с неизвестными коэффициентами a_{11}, \dots, a_{mn} (первый индекс указывает номер уравнения, второй индекс — номер неизвестного) и свободными членами $b_1, \dots, b_m \in P$ (индекс — номер уравнения). При этом числа a_{11}, \dots, a_{mn} называются также коэффициентами системы и b_1, \dots, b_m – свободными членами системы.

Система называется однородной, если $b_1 = \dots = b_m = 0$.

Решением системы (s) называется такой упорядоченный набор $(\zeta_1, \dots, \zeta_n)$ n элементов $\zeta_1, \dots, \zeta_n \in P$, что при подстановке в уравнения $(1) - (m)$ значений $x_1 = \zeta_1, \dots, x_n = \zeta_n$ получается верные равенства $\sum_{j=1}^n a_{ij} \zeta_j = b_i$ ($i = 1 \dots m$).

Лемма. Следующие элементарные преобразования сохраняют множество решений любой системы линейных уравнений, т.е. являются равносильными:

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

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

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

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

Система (s') равносильна системе

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

которая называется *общим* решением исходной системы уравнений (s) .

Преобразование системы (s) в равносильную ей разрешенную систему (s') осуществляется по методу Гаусса с помощью последовательного выполнения *Жордановых преобразований*.

Матрицей системы (s) называется матрица

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} & b_1 \\ \dots & \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} & b_m \end{pmatrix}.$$

Алгоритм:

Вход: система линейных уравнений (s) с n неизвестными и m уравнениями.

Выход: общее решение системы (s'') .

1. Строим матрицу $A = \begin{pmatrix} a_{11} & \dots & a_{1n} & b_1 \\ \dots & \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} & b_m \end{pmatrix}$
2. Положим $i = 1$.
3. В i -ой строке выбирается ненулевой коэффициент a_{ij} , i -ая строка умножается на элемент $a_{ij} - 1$.
4. Прибавляем к остальным k -ым строкам ($k = \overline{1, m}$, $k \neq i$) новую i -ую строку, умноженную на коэффициент $-a_{kj}$.
5. Если $i = m + 1$, то переход к следующему шагу, иначе $i = i + 1$ и переход к шагу 3.
6. Удаляем из системы нулевые строки.
7. Если получили матрицу со строкой вида $(0, \dots, 0, b)$ со значением $b \neq 0$, то алгоритм закончен, система (s) не имеет решений.
8. Результат: строим общее решение (s'') системы по полученной матрице.

Сложность вычислений равна $O(\min(m, n) \cdot nm)$.

3 Результаты работы

3.1 Псевдокод алгоритма Евклида

```
Функция НОД (a, b)
    Пока b ≠ 0
        temp = b
        b = a % b
        a = temp
    Конец Пока
    Вернуть a
Конец Функции
```

3.2 Псевдокод бинарного алгоритма Евклида

```
Функция Бинарный_НОД(a, b)
    Если a == b
        Вернуть a
    Если a == 0
        Вернуть b
    Если b == 0
        Вернуть a
    Если a является четным числом и b является четным числом
        Вернуть 2 * Бинарный_НОД(a / 2, b / 2)
    Если a является четным числом и b является нечетным числом
        Вернуть Бинарный_НОД(a / 2, b)
    Если a является нечетным числом и b является четным числом
        Вернуть Бинарный_НОД(a, b / 2)
    Если a является нечетным числом и b является нечетным числом и a >
b
        Вернуть Бинарный_НОД((a - b) / 2, b)
    Если a является нечетным числом и b является нечетным числом и a <
b
        Вернуть Бинарный_НОД((b - a) / 2, a)
    Конец Функции
```

3.3 Псевдокод расширенного алгоритма Евклида

```
Функция Расширенный_НОД(a, b)
    u1 = 1;
    v1 = 0;
    u2 = 0;
    v2 = 1;
    Пока b != 0
        d = a / b;
        tmp = a % b;
        a = b;
        b = tmp;
        tmp = u1 - u2 * d;
        u1 = u2;
        u2 = tmp;
        tmp = v1 - v2 * d;
        v1 = v2;
        v2 = tmp;
    Конец Пока
    Вернуть (a, u1, v1)
Конец Функции
```

3.4 Псевдокод решения системы сравнений по греко-китайской теореме об остатках

```
Функция китайская_теорема(коэффициенты, модули)
    n = Длина(коэффициенты)
    M = 1
    x = 0
    Для i от 0 до n - 1
        M = M * модули[i]
    Для i от 0 до n - 1
        Mi = M / модули[i]
        yi = Расширенный_НОД(Mi, модули[i])[1]
        x = x + коэффициенты[i] * Mi * yi
    Вернуть x % M
Конец Функции
```

3.5 Псевдокод решения системы сравнений по алгоритму Гарнера

```
Функция Гарнера(коэффициенты, модули)
    Для i от 0 до Длина(модули)
        Для j от 0 до Длина(модули)
            Если (i == j)
                cij = 0
            Иначе
                cij = Расширенный_НОД(модули[i], модули[j])
    q = коэффициенты[0] mod модули[0]
    Для k от 0 до Длина(модули)
        q[i] = (коэффициенты[k] - q[0]) * c0k mod модули[k]
    Для i от 1 до k
        q[i] -= q[i]
        q[i] *= cik
        q[i] = q[i] mod m[k]
    mult = 1
    U = q[0]
    Для i от 0 до Длина(q)
        mult *= m[i - 1]
        U = U + q.get(i) * mult
        U %= M
    Вернуть U
Конец функции
```

3.6 Псевдокод решения системы линейных уравнений по методу Гаусса

```
Функция СЛУ(матрица A)
    вектор b = последний столбец A
    n = Размер(матрица A)[0] // Количество уравнений
    m = Размер(матрица A)[1] // Количество переменных
    Для i от 0 до n - 1
        max_row = i
        Для k от i + 1 до n
            Если Abs(A[k][i]) > Abs(A[max_row][i])
                max_row = k
            Конец Если
        Конец Для
        Если max_row ≠ i
            Обменять(матрица A[i], матрица A[max_row])
            Обменять(b[i], b[max_row])
```

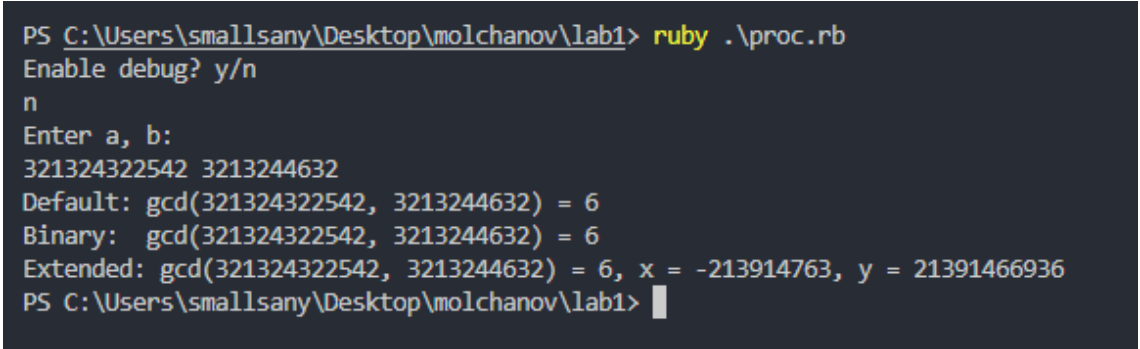
```

        Конеч Если
        pivot = A[i][i]
        Для j от i до m
            A[i][j] = A[i][j] / pivot
        Конеч Для
        b[i] = b[i] / pivot
        Для k от 0 до n
            Если k ≠ i
                factor = A[k][i]
                Для j от i до m
                    A[k][j] = A[k][j] - factor * A[i][j]
                Конеч Для
                b[k] = b[k] - factor * b[i]
            Конеч Если
        Конеч Для
    Конеч Для
    x = Создать_Вектор(m)
    Для i от n - 1 до 0 с шагом -1
        x[i] = b[i]
        Для j от i + 1 до m
            x[i] = x[i] - A[i][j] * x[j]
        Конеч Для
    Конеч Для
    Вернуть x
Конеч Функции

```

4 Тестирование программы

На рисунках 1-2 представлено тестирования работы программы для нахождения НОД



```

PS C:\Users\smallsany\Desktop\molchanov\lab1> ruby .\proc.rb
Enable debug? y/n
n
Enter a, b:
321324322542 3213244632
Default: gcd(321324322542, 3213244632) = 6
Binary: gcd(321324322542, 3213244632) = 6
Extended: gcd(321324322542, 3213244632) = 6, x = -213914763, y = 21391466936
PS C:\Users\smallsany\Desktop\molchanov\lab1>

```

Рисунок 1 - Нахождение НОД

```

PS C:\Users\smallsany\Desktop\molchanov\lab1> ruby .\proc.rb
Enable debug? y/n
y
Enter a, b:
136 24
using default gcd(136, 24)
gcd(24, 16)
gcd(16, 8)
gcd(8, 0)
Default: gcd(136, 24) = 8
using binary gcd(136, 24)
gcd(68, 12)
gcd(34, 6)
gcd(17, 3)
gcd(7, 3)
gcd(2, 3)
gcd(1, 3)
gcd(1, 1)
Binary: gcd(136, 24) = 8
using extended gcd(136, 24)
gcd(8, 16); koeff: (0, 1)
gcd(16, 24); koeff: (1, 0)
gcd(24, 136); koeff: (-1, 1)
gcd(136, 24); koeff: (6, -1)
Extended: gcd(136, 24) = 8, x = -1, y = 6
PS C:\Users\smallsany\Desktop\molchanov\lab1>

```

Рисунок 2 - Нахождение НОД (с отображением промежуточных шагов)

На рисунках 3 и 4 представлено тестирование работы программы для нахождения систем сравнений.

```

PS C:\Users\smallsany\Desktop\molchanov\lab1> ruby .\proc.rb
Enable debug? y/n
n
Solving comparison system!
Enter count of equations in system:
3
Enter equations:
6 8
5 13
3 15
Result: 798 (mod 1560)
PS C:\Users\smallsany\Desktop\molchanov\lab1>

```

Рисунок 3 - Решение системы сравнений

```

PS C:\Users\smallsany\Desktop\molchanov\lab1> ruby .\proc.rb
Enable debug? y/n
y
Solving comparison system!
Enter count of equations in system:
2
Enter equations:
36 47
15 9
equations:
x = 36 (mod 47)
x = 15 (mod 9)
coefficients: [36, 15]
modulus: [47, 9]
using chinese_reminder_theorem for [36, 15] in [47, 9]
using inverse of 9 in 47
using extended gcd(9, 47)
gcd(1, 2); koeff: (0, 1)
gcd(2, 9); koeff: (1, 0)
gcd(9, 47); koeff: (-4, 1)
gcd = 1, x = 21
using inverse of 47 in 9
using extended gcd(47, 9)
gcd(1, 2); koeff: (0, 1)
gcd(2, 9); koeff: (1, 0)
gcd(9, 47); koeff: (-4, 1)
gcd(47, 9); koeff: (21, -4)
gcd = 1, x = -4
Result: 177 (mod 423)
PS C:\Users\smallsany\Desktop\molchanov\lab1> 

```

Рисунок 4 - Решение системы сравнений (с отображением промежуточных шагов)

На рисунке 5 представлено тестирование работы программы для решения системы линейных уравнений над конечным полем.

```

PS C:\Users\smallsany\Desktop\molchanov\lab1> ruby .\proc.rb
Enable debug? y/n

Solving equation's system by Gauss method!
Enter count of equations in system:
3
Enter equations:
2 1 5 6
1 2 4 6
2 1 1 2
Enter module:
7
Result: [0, 1, 1]
PS C:\Users\smallsany\Desktop\molchanov\lab1> 

```

Рисунок 5 - Решение системы уравнений

ПРИЛОЖЕНИЕ А

Листинг программы methods.rb

```
class Methods
  def initialize(params = {})
    @debug_mode = params.dig(:debug_mode)
  end

  def gcd(a,b)
    puts "using default gcd(#{a}, #{b})" if @debug_mode
    while b != 0
      remainder = a % b
      a = b
      b = remainder
      #debug
      sleep 0.5 if @debug_mode
      puts "gcd(#{a}, #{b})" if @debug_mode
    end

    return a
  end

  def gcd_bin(a,b)
    puts "using binary gcd(#{a}, #{b})" if @debug_mode

    shift = 0

    while a != b
      if a % 2 == 0 && b % 2 == 0
        a = a / 2
        b = b / 2
        shift += 1
      elsif a % 2 == 0
        a = a / 2
      elsif b % 2 == 0
        b = b / 2
      elsif a > b
        a = (a-b)/2
      else
        b = (b-a)/2
      end
      #debug
      sleep 0.5 if @debug_mode
      puts "gcd(#{a}, #{b})" if @debug_mode
    end

    return a * (2 ** shift)
  end

  def gcd_ext(a, b, first = true)
    puts "using extended gcd(#{a}, #{b})" if @debug_mode && first

    if a == 0
      return b, 0, 1
    else
      res, x, y = gcd_ext(b%a, a, false)
      #debug
      sleep 0.5 if @debug_mode
      puts "gcd(#{a}, #{b}); koeff: (#{x}, #{y})" if @debug_mode
      return res, y - (b / a) * x, x
    end
  end
end
```

```

def inverse(a, md)
  puts %{using inverse of #{a} in #{md}} if @debug_mode
  gcd, x, _ = gcd_ext(a, md)
  puts %{gcd = #{gcd}, x = #{x}} if @debug_mode
  if gcd != 1
    raise "\nNo inverse element exists\n"
  else
    return x % md
  end
end

def chinese_remainder_theorem(coefficients = [], modulus = [])

  if coefficients.empty? || modulus.empty?
    raise "Not enough data!"
  end

  puts %{using chinese_remainder_theorem for #{coefficients} in
#{modulus}} if @debug_mode
  x = 0
  fact = modulus.reduce(:*)

  coefficients.zip(modulus).each do |a, m|
    ci = fact / m
    ci_inv = inverse(ci, m)
    x += a * ci * ci_inv
  end

  x %= fact

  return {x:x, md: fact}
end

def gauss(matrix = [], field_dimension = 0)

  if @debug_mode
    puts %{using gaussian_elimination for matrix: #{matrix}}
  end

  if matrix.empty? || field_dimension == 0
    raise "Not enough data!"
  end

  n = matrix.length

  (0..n - 1).each do |i|
    max_row = i
    (i + 1..n - 1).each do |k|
      # find max element in the column
      max_row = k if (matrix[k][i] > matrix[max_row][i])
    end

    # swap rows
    matrix[i], matrix[max_row] = matrix[max_row], matrix[i]

    if matrix[i][i] == 0
      # main element == 0 => no solution
      raise "Equation's system has no solution!"
    end

    (i + 1..n - 1).each do |k|
      factor = matrix[k][i] * inverse(matrix[i][i], field_dimension) %
field_dimension
      (i..n).each do |j|

```



```

        matrix[k][j] = (matrix[k][j] - matrix[i][j] * factor %
field_dimension + field_dimension) % field_dimension
    end
end

    puts %{step #{i} matrix: #{matrix}} if @debug_mode
end

    result = Array.new(n, 0)

    (n - 1).downto(0).each do |i|
        result[i] = (matrix[i][n] * inverse(matrix[i][i], field_dimension)
% field_dimension + field_dimension) % field_dimension
        (0..i - 1).each do |k|
            matrix[k][n] = (matrix[k][n] - matrix[k][i] * result[i] %
field_dimension + field_dimension) % field_dimension
            matrix[k][i] = 0
        end
        puts %{reverse step #{i} matrix: #{matrix}} if @debug_mode
    end

    return result
end

end

```

ПРИЛОЖЕНИЕ Б

Листинг программы proc.rb

```
require './methods.rb'

puts %{Enable debug? y/n}
@debug_mode = gets.strip == 'y'
@methods = Methods.new({debug_mode: @debug_mode})

def many_gcds
  puts %{Enter a, b:}
  a, b = gets.strip.split.map(&:to_i)
  puts %{Default: gcd(#{a}, #{b}) = #{@methods.gcd(a, b)}}
  puts %{Binary: gcd(#{a}, #{b}) = #{@methods.gcd_bin(a, b)}}
  res = @methods.gcd_ext(a, b)
  puts %{Extended: gcd(#{a}, #{b}) = #{res[0]}, x = #{res[1]}, y =
#{res[2]}}
end

def solve_comparation_system
  puts %{Solving comparation system!}

  equations = []
  coefficients = []
  modulus = []

  puts %{Enter count of equations in system:}
  n = gets.strip.to_i

  puts %{Enter equations:}
  n.times do
    equation = gets.strip.split.map(&:to_i)
    a, b = equation[0], equation[1]
    equations.append(%{x = #{a} (mod #{b})})
    coefficients.append(a)
    modulus.append(b)
  end

  #debug
  if @debug_mode
    puts %{equations:}
    equations.map {|eq| puts eq}
    puts %{coefficients: #{coefficients}}
    puts %{modulus: #{modulus}}
  end

  res = @methods.chineese_reminder_theorem(coefficients, modulus)

  puts %{Result: #{res[:x]} (mod #{res[:md]})}
end

def solve_gauss_system
  puts %{Solving equation's system by Gauss method!}

  matrix = []

  puts %{Enter count of equations in system:}
  n = gets.strip.to_i

  puts %{Enter equations:}
  n.times do
    matrix << gets.split.map(&:to_i)
```

```
end

puts %{Enter module:}
field_dimension = gets.to_i

puts %{Result: #{@methods.gauss(matrix, field_dimension)}}

end

#1
many_gcds

#2
solve_comparation_system

#3
solve_gauss_system
```