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

Алексеева Александра Александровича

Преподаватель

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

подпись, дата

Саратов 2023

СОДЕРЖАНИЕ

1 Цель работы и порядок её выполнения

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

Порядок выполненных 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 Теория

2.1 Обычный 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}:

$$a_0 = a_1 q_1 + a_2,\ 0 \le a_2 \le a_1,$$

$$a_1 = a_2 q_2 + a_3,\ 0 \le a_3 \le a_2,$$

$$\dots$$

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

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

Так как остатки выполняемых делений образуют строго убывающую последовательность $a_1 > a_2 > a_3 > \dots \ge 0$, 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).

Псевдокод обычного алгоритма Евклида

Вход. Целые числа a, b; $0 < b < a$.

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 $d = a_i$. В противном случае положить $i = i + 1$ и вернуться на шаг 2.

Шаг 4. Результат: d = НОД(a, b).

2.2 Бинарный алгоритм Евклида

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

• НОД($2a$, $2b$) = 2НОД(a, b);

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

• НОД($-a$, b) = НОД(a, b).

Описание алгоритма:

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 чётные, то НОД(a, b) = 2НОД(a/2, b/2);
5. Если a чётное и b нечётное, то НОД(a, b) = НОД(a/2, b);
6. Если a нечётное и b чётное, то НОД(a, b) = НОД(a, b/2);
7. Если a и b нечётные:
   - при этом $b > a$, то НОД(a, b) = НОД((b-a)/2, a);
   - при этом $b < a$, то НОД(a, b) = НОД((a-b)/2, b).

При возможности использовать битовые операции:

• Умножение на два – битовое смещение: x • 2 эквивалентно смещению на один бит влево;

• Деление на два – битовое смещение: x / 2 эквивалентно смещению на бит вправо;

• Проверка на чётность: x % 2 == 0 эквивалентно (x & 1) == 0 (у чётного числа последний бит равен нулю);

• Соответственно, проверка на нечётность: x % 2 != 0 эквивалентно (x & 1) == 1 (у нечётного числа последний бит равен единице).

Псевдокод бинарного алгоритма Евклида

Вход. Целые числа a, b; $0 < b < a$.

Выход: d = НОД(a, b).

Шаг 1. Проверка условий:

• если a = 0, то результат: d = НОД(a, b) = b;

• если b = 0 или a = b, то результат d = НОД(a, b) = a;

• если a = 1 или b = 1, то d = НОД(a, b) = 1;

• если (a & 1) = 0 и (b & 1) = 0, то результат: бинарный алгоритм Евклида (a >> 1, b >> 1) << 1;

• если (a & 1) = 0 и (b & 1) = 1, то результат: бинарный алгоритм Евклида (a >> 1, b);

• если (a & 1) = 1 и (b & 1) = 0, то результат: бинарный алгоритм Евклида (a, b >> 2);

• иначе если $b > a$, то результат: бинарный алгоритм Евклида (($b - a$) >> 1, a). В противном случае результат: бинарный алгоритм Евклида (($a - b$) >> 1, b).

2.3 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 ∈ $\mathbb{Z}$. Значения x, y находятся в результате обратного прохода этапов алгоритма Евклида, в каждом из которых уравнение разрешается относительно остатка a_i, который представляет в форме $a_i = ax_i + by_i$ для некоторых x_i, $y_i \in \mathbb{Z}$.

В результате получается следующая последовательность вычислений:

$$
\begin{array}{ll}
a_0 = a, & a_0 = ax_0 + by_0, \\
a_1 = b, & a_1 = ax_1 + by_1, \\
a_2 = a_0 - a_1 q_1, & a_2 = ax_2 + by_2, \\
a_3 = a_1 - a_2 q_2, & a_3 = ax_3 + by_3, \\
\quad \ldots\ldots\ldots\ldots & \quad \ldots\ldots\ldots\ldots\ldots \\
a_i = a_{i-2} - a_{i-1} q_{i-1}, & a_i = ax_i + by_i, \\
\quad \ldots\ldots\ldots\ldots & \quad \ldots\ldots\ldots\ldots\ldots \\
a_k = a_{k-2} - a_{k-1} q_{k-1}, & a_k = ax_k + by_k, \\
0 = a_{\kappa-1} - a_k q_k, & 0 = ax_{k+1} + by_{k+1}.
\end{array}
$$

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}, …, a_1, a_0 и, в частности, представление НОД(a, b) = $a_k = ax_k + by_k$.

Псевдокод расширенного алгоритма Евклида

Вход. Целые числа a, b; $0 < b$ и $0 < a$.

Выход: Целые числа x, y : d = НОД(a, b) = $a \cdot x + b \cdot y$.

Шаг 1. Инициализируем q, $aPrev = a$, $aCur = b$, $aNext = -1$, $xPrev = 1$, $xCur = 0$, $xNext$, $yPrev = 0$, $yCur = 1$, $yNext$.

Шаг 2. Цикл пока $aNext$!= 0:

Шаг 2.1. $q = aPrev / aCur$.

Шаг 2.2. Вычисляем $aNext$ = остаток от деления $aPrev$ на $aCur$. Присваиваем $aPrev = aCur$, $aCur = aNext$.

Шаг 2.3. Вычисляем $xNext = xPrev - (xCur \cdot q)$. Присваиваем $xPrev = xCur$, $xCur = xNext$.

Шаг 2.4. Вычисляем $yNext = yPrev - (yCur \cdot q)$. Присваиваем $yPrev = yCur$, $yCur = yNext$.

Шаг 3. Результат: Целые числа $x, y : d =$ НОД$(a, b) = a \cdot x + b \cdot y$.

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

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

имеет единственное неотрицательное решение по модулю M.

При этом, если для каждого $1 \le j \le k$ число $M_j = {}^{M}/_{m_j}$ 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 $x = M_1 z_1 + M_2 z_2 + \ldots + M_1 z_1$.

Псевдокод греко-китайской теоремы об остатках

Вход. Система линейных сравнений *raws*: $\forall\ i = \overline{1, k}\ \ x \equiv a_i (mod\ m_i)$

Выход. Числа x, M: x – наименьшее положительное целочисленное решение, M – произведение модулей m_1, m_2, …, m_k системы линейных сравнений.

Шаг 1. Вычисляем $M = m_1 m_2 \ldots m_k$.

Шаг 2. Инициализируем $res = 0$.

Шаг 3. Цикл по i от 1 до *raws*.size():

Шаг 3.1. Вычисляем $M_i = m_1m_2\ldots m_{i-1}m_{i+1}\ldots m_k$.

Шаг 3.2. Находим j : остаток от деления ($M_i \bullet j$) на m_i равен a_i.

Шаг 3.3. прибавляем к res значение $M_i \bullet j$;

Шаг 4. Результат: остаток от деления res на M, M.

2.5 Алгоритм Гарнера

На входе алгоритма заданы числа $a_1, \ldots, a_k, m_1, \ldots, m_k$ $(m_i, m_j) = 1$ при $i \neq j$, и число $M = m_1\ldots m_k$. На выходе получается решение x, $0 \leq x < M$.

1 шаг. Для $i = 2, \ldots, k$ выполнить пункты 1) и 2):

1) $c_i = 1$;

2) для $j = 1, \ldots, i - 1$ выполнить: $u = m_j^{-1}$ (mod m_i) (нахождение обратного элемента можно делать с помощью обобщённого бинарного алгоритма Евклида), $c_i = uc_i$ (mod m_i).

2 шаг. $u = a_1$, $x = u$.

3 шаг. Для $i = 2, \ldots, k$ вычислить $u = (a_i - x)c_i$ (mod m_i) – наименьший неотрицательный вычет по модулю m_i,

$$x = x + u\prod_{j=1}^{i-1} m_j.$$

Полученное значение x является искомым решением.

Псевдокод алгоритма Гарнера

Вход. Система линейных сравнений $raws$: $\forall\ i = \overline{1,k}\ \ x\ \equiv a_i(mod\ m_i)$

Выход. Числа x, M: x – наименьшее положительное целочисленное решение, M – произведение модулей $m_1, m_2, \ldots, m_k$ системы линейных сравнений.

Шаг 1. Вычисляем $M = m_1m_2\ldots m_k$.

Шаг 2. Инициализируем массив c размером $raws$.size(), каждый элемент которого хранит значение 1.

Шаг 3. Цикл по i от 1 до $raws$.size():

Шаг 3.1. Цикл по j от 1 до i:

Шаг 3.1.1. Вычисляем u = НОД(m_i, m_j).

Шаг 3.1.2. Вычисляем $c[i]$ = остаток от деления ($u \bullet c[i]$) на m_i.

Шаг 4. Присваиваем $u = a_1$, $x = u$, $mAcc = 1$.

Шаг 5. Цикл по i от 1 до $raws$.size():

Шаг 5.1. Вычисляем u = остаток от деления (($a_i - x$) • $c[i]$) на m_i.

Шаг 5.2. Вычисляем $mAcc = mAcc \bullet m_{i\text{-}1}$.

Шаг 5.3. Вычисляем $x = x + u * mAcc$.

Шаг 6. Результат: числа x, M.

2.6 Решение систем линейных уравнений над конечными полями методом Гаусса

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

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

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

Решением системы (s) называется такой упорядоченный набор ($\xi_1, \ldots, \xi_n$) n элементов $\xi_1, \ldots, \xi_n \in$ P, что при подстановке в уравнения (1)-(m) значений $x_1 = \xi_1$, …, $x_1 = \xi_1$ получаются верные равенства $\sum_{j=1}^{n} a_{ij}\xi_j = b_i$ (i = $\overline{1, m}$).

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

a) удаление из системы тривиальных уравнений,

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

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

Метод решения системы (s) заключается в равносильном преобразовании её в систему линейных уравнений с противоречивым уравнением или в разрешённую систему линейных уравнений вида:

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

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

Система (s') равносильна системе

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

которая называется общим решением исходной системы уравнений (s).

Преобразование системы (s) в равносильную её разрешённую системы (s') 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-го уравнения, умноженные на коэффициент $-a_{kj}$;

4) удаляем из системы тривиальные уравнения.

Псевдокод метода Гаусса

Вход: матрица системы линейных уравнений *matrix*, поле *field*

Выход: общее решение исходной системы

Шаг 1. Цикл по i от 1 до *matrix*.size():

Шаг 1.1. Вычисляем обратный элемент *revEl* для *matrix*[*i*][*i*] в поле *field*.

Шаг 1.2. Умножаем строку *matrix*[*i*] на элемент *revEl*.

Шаг 1.3. Цикл по j от 1 до *matrix*.size():

Шаг 1.3.1. Если $j = i$, то переход к следующей итерации. В противном случае *matrix*[*j*] = *matrix*[*j*] + (-*matrix*[*j*][*i*] • *matrix*[*i*]).

Шаг 2. Удаляем тривиальные уравнения из *matrix*.

Шаг 3. Результат: получившаяся матрица СЛУ *matrix* является общем решением исходной системы уравнений.

3 Результаты работы

3.1 <u>Оценки сложности рассмотренных алгоритмов</u>

Обычный алгоритм Евклида – $O(n^2)$;

Бинарный алгоритм Евклида – $O(n^2)$;

Расширенный алгоритм Евклида – $O(n^2)$;

Греко-китайская теорема об остатках – $O(k^2 f_{div}(b) + k f_{inv}(b))$, где $b = \max\{\log m_i : 1 \le i \le k\}$;

Алгоритм Гарнера – $O(k^2 f_{div}(b) + k f_{inv}(b))$, где $b = \max\{\log m_i : 1 \le i \le k\}$;

Метод Гаусса – $O(\min(m, n) \cdot mn)$.

3.2 <u>Результаты тестирования программ</u>

```
1 - Алгоритм Евклида
2 - Решение систем сравнений
3 - Метод Гаусса
4 - Выход
1

1 - Обычный алгоритм Евклида
2 - Бинарный алгоритм Евклида
3 - Расширенный алгоритм Евклида
4 - Назад
1

        Обычный алгоритм Евклида нахождения НОД
a = 799
b = 425
НОД(799, 425) = 17
```

```
        Бинарный алгоритм Евклида нахождения НОД
a = 799
b = 425
НОД(799, 425) = 17

1 - Обычный алгоритм Евклида
2 - Бинарный алгоритм Евклида
3 - Расширенный алгоритм Евклида
4 - Назад
```

```
        Расширенный алгоритм Евклида нахождения НОД
a = 799
b = 425
НОД(799, 425) = 799*8 + 425*-15 = 17

1 - Обычный алгоритм Евклида
2 - Бинарный алгоритм Евклида
3 - Расширенный алгоритм Евклида
4 - Назад
```

```
1 - Алгоритм Евклида
2 - Решение систем сравнений
3 - Метод Гаусса
4 - Выход
2

1 - Греко-китайская теорема об остатках
2 - Алгоритм Гарнера
3 - Назад
1

Количество сравнений: 3
Введите a1, m1: 1 3
Введите a2, m2: 3 5
Введите a3, m3: 2 4

Система линейных сравнений:
x = 1(mod 3)
x = 3(mod 5)
x = 2(mod 4)
Решение данной системы линейных сравнений: x = 58 (mod 60)
```

```
1 - Греко-китайская теорема об остатках
2 - Алгоритм Гарнера
3 - Назад
2

Количество сравнений: 3
Введите a1, m1: 1 3
Введите a2, m2: 3 5
Введите a3, m3: 2 4

Система линейных сравнений:
x = 1(mod 3)
x = 3(mod 5)
x = 2(mod 4)
Решение данной системы линейных сравнений: x = 58 (mod 60)
```

```
1 - Алгоритм Евклида
2 - Решение систем сравнений
3 - Метод Гаусса
4 - Выход
3

        Решение системы линейных уравнений методом Гаусса
Поле: 7
Количество строк и столбцов матрицы системы: 3 5
Матрица системы:
1 0 -2 -4 2
2 -2 0 -3 1
0 2 -4 -5 3

Общее решение исходной системы:
x1 = 2x3 + 4x4 + 2
x2 = 2x3 + 6x4 + 5

Свободные неизвестные x3, x4: 0 0
Частное решение: (2, 5, 0, 0)
```

3.3 Код программы

```
#include "iostream"
#include "vector"

using namespace std;
//////////////////////////////////////////////////////////////////////АЛГОРИТМ
ЕВКЛИДА//////////////////////////////////////////////////////////////////////
/
//Обычный алгоритм Евклида
int usualEuclid(int a, int b) {
	if (a < b)
		swap(a, b);
	else if (a < 0 || b < 0)
		throw string{ "Выполнение невозможно: a < 0 или b < 0" };
	else if (b == 0)
		return a;

	int r = a % b;
	usualEuclid(b, r);
}


//Бинарный алгоритм Евклида
int binaryEuclid(int a, int b) {
	if (a < 0 || b < 0)
		throw string{ "Выполнение невозможно: a < 0 или b < 0" };

	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 binaryEuclid(a >> 1, b >> 1) << 1;
	else if ((a & 1) == 0 && (b & 1) == 1)
		return binaryEuclid(a >> 1, b);
```

```
        else if ((a & 1) == 1 && (b & 1) == 0)
                return binaryEuclid(a, b >> 2);
        else {
                if (b > a)
                        return binaryEuclid((b - a) >> 1, a);
                else
                        return binaryEuclid((a - b) >> 1, b);
        }
}


//Расширенный алгоритм Евклида
pair <int, int> advancedEuclid (int a, int b) {
        if (a < 0 || b < 0)
                throw string{ "Выполнение невозможно: a < 0 или b < 0" };

        int q, aPrev = a, aCur = b, aNext = -1;
        int xPrev = 1, xCur = 0, xNext;
        int yPrev = 0, yCur = 1, yNext;
        while (aNext != 0) {
                q = aPrev / aCur;
                aNext = aPrev % aCur;
                aPrev = aCur; aCur = aNext;

                xNext = xPrev - (xCur * q);
                xPrev = xCur; xCur = xNext;

                yNext = yPrev - (yCur * q);
                yPrev = yCur; yCur = yNext;
        }

        return make_pair(xPrev, yPrev);
}


//Выбор алгоритма Евклида
void mainEuclid() {
        for (;;) {
                cout << "\n1 - Обычный алгоритм Евклида \n2 - Бинарный алгоритм 
Евклида \n3 - Расширенный алгоритм Евклида \n4 - Назад \n";
                int x;
                cin >> x;
                int a, b;
                switch (x) {
                case 1:
                        cout << "\n\tОбычный алгоритм Евклида нахождения НОД \n";
                        cout << "a = ";
                        cin >> a;
                        cout << "b = ";
                        cin >> b;

                        try {
                                cout << "НОД(" << a << ", " << b << ") = " << 
usualEuclid(a, b);
                        }
                        catch (string& error_message) {
                                cout << error_message;
                        }
                        cout << endl;
                        break;
                case 2:
                        cout << "\n\tБинарный алгоритм Евклида нахождения НОД 
\n";
```

```
			cout << "a = ";
			cin >> a;
			cout << "b = ";
			cin >> b;

			cout << "НОД(" << a << ", " << b << ") = " <<
binaryEuclid(a, b);
			cout << endl;
			break;
		case 3:
			cout << "\n\tРасширенный алгоритм Евклида нахождения НОД
\n";
			cout << "a = ";
			cin >> a;
			cout << "b = ";
			cin >> b;

			try {
				pair <int, int> xy = advancedEuclid(a, b);
				cout << "НОД(" << a << ", " << b << ") = " << a
<< "*" << xy.first << " + " << b << "*" <<
					xy.second << " = " << a * xy.first + b
* xy.second;
			}
			catch (string& error_message) {
				cout << error_message;
			}
			cout << endl;
			break;
		case 4:
			return;
		default:
			cout << "Incorrect. Try again \n\n";
		}
	}
}


/////////////////////////////////////////////////////////////////РЕШЕНИЕ СИСТЕМЫ
ЛИНЕЙНЫХ
СРАВНЕНИЙ/////////////////////////////////////////////////////////////////
//Провека взаимной простоты m
bool checkMutualSimplicity(vector <pair <int, int>> raws) {
	for (int i = 0; i < raws.size(); i++)
		for (int j = i + 1; j < raws.size(); j++) {
			if (raws[i].second > raws[j].second) {
				if (binaryEuclid(raws[i].second, raws[j].second)
!= 1)
					return false;
			}
			else
				if (binaryEuclid(raws[j].second, raws[i].second)
!= 1)
					return false;
		}
	return true;
}


//Греко-китайская теорема об остатках
pair <int, int> gcTheorem(vector <pair <int, int>> raws) {
	if (!checkMutualSimplicity)
		throw string{ "Модули m не являются попарно взаимно простыми!" };
```

```
        int M = 1;
        for (int i = 0; i < raws.size(); i++)
                M *= raws[i].second;

        int res = 0;
        for (int i = 0; i < raws.size(); i++) {
                int Mi = 1;
                for (int j = 0; j < raws.size(); j++) {
                        if (i == j)
                                continue;
                        Mi *= raws[j].second;
                }
                for (int j = 1;; j++)
                        if ((Mi * j) % raws[i].second == raws[i].first) {
                                res += Mi * j;
                                break;
                        }
        }

        return make_pair(res % M, M);
}


//Алгоритм Гарнера
pair <int, int> garner(vector <pair <int, int>> raws) {
        if (!checkMutualSimplicity)
                throw string{ "Модули m не являются попарно взаимно простыми!" };

        int M = 1;
        for (int i = 0; i < raws.size(); i++)
                M *= raws[i].second;

        vector <int> c(raws.size(), 1);
        int u, x;
        for (int i = 1; i < raws.size(); i++) {
                for (int j = 0; j < i; j++) {
                        u = advancedEuclid(raws[j].second, raws[i].second).first;
                        c[i] = (u * c[i]) % raws[i].second;
                }
        }

        u = raws[0].first;
        x = u;
        int mAcc = 1;
        for (int i = 1; i < raws.size(); i++) {
                u = ((raws[i].first - x) * c[i]) % raws[i].second;
                mAcc *= raws[i - 1].second;
                x = x + u * mAcc;
        }

        if (x < 0)
                x += M;
        return make_pair(x, M);
}


//Ввод системы линейных сравнений
void mainSLC() {
        for (;;) {
                cout << "\n1 - Греко-китайская теорема об остатках \n2 - Алгоритм
Гарнера \n3 - Назад \n";
                int x;
```

```
			cin >> x;
			if (x == 3)
				return;

			vector <pair <int, int>> raws;
			cout << "\nКоличество сравнений: ";
			int k, a, m;
			cin >> k;
			for (int i = 0; i < k; i++) {
				cout << "Введите a" << i + 1 << ", m" << i + 1 << ": ";
				cin >> a >> m;
				raws.push_back(make_pair(a % m, m));
			}

			cout << "\nСистема линейных сравнений: \n";
			for (int i = 0; i < raws.size(); i++)
				cout << "x = " << raws[i].first << "(mod " <<
raws[i].second << ") \n";

			switch (x) {
			case 1:
				try {
					pair <int, int> res = gcTheorem(raws);
					cout << "Решение данной системы линейных
сравнений: x = " << res.first << " (mod " << res.second << ")" << endl;
				}
				catch (string& error_message) {
					cout << error_message;
				}
				cout << endl;
				break;
			case 2:
				try {
					pair <int, int> res = garner(raws);
					cout << "Решение данной системы линейных
сравнений: x = " << res.first << " (mod " << res.second << ")" << endl;
				}
				catch (string& error_message) {
					cout << error_message;
				}
				cout << endl;
				break;
			default:
				cout << "Incorrect. Try again \n\n";
			}
		}
}


///////////////////////////////////////////////////////////////////////////МЕТОД
ГАУССА//////////////////////////////////////////////////////////////////////////////
//Проверка, имеет ли решения СЛУ
bool haveSolution(vector <vector <int>> matrix) {
		for (int i = 0; i < matrix.size(); i++) {
			bool flag = true;
			for (int j = 0; j < matrix[i].size(); j++) {
				if (j == matrix[i].size() - 1 && matrix[i][j] == 1 &&
flag)
					return false;
				else if (matrix[i][j] != 0)
					break;
			}
		}
```

```
        return true;
}


//Умножение строки на число
vector <int> multRawtoNum(vector <int> raw, int num, int field) {
        vector <int> res;
        for (int i = 0; i < raw.size(); i++) {
                int x = (raw[i] * num) % field;
                res.push_back(x >= 0 ? x : x + field);
        }
        return res;
}


//Сложение строк
vector <int> addRaws(vector <int> a, vector <int> b, int field) {
        vector <int> res;
        for (int i = 0; i < a.size(); i++) {
                int x = (a[i] + b[i]) % field;
                res.push_back(x >= 0 ? x : x + field);
        }
        return res;
}


//Удаление нулевых строк
void delZeroRaws(vector <vector <int>>& matrix) {
        for (int i = 0; i < matrix.size(); i++) {
                bool flag = true;
                for (int j = 0; j < matrix[i].size(); j++)
                        if (matrix[i][j] != 0) {
                                flag = false;
                                break;
                        }
                if (flag) {
                        matrix.erase(matrix.begin() + i);
                        i--;
                }
        }
}


//Метод Гаусса
void gauss() {
        cout << "\n\tРешение системы линейных уравнений методом Гаусса \n";
        cout << "Поле: ";
        int field;
        cin >> field;
        cout << "Количество строк и столбцов матрицы системы: ";
        int raws, cols;
        cin >> raws >> cols;
        vector <vector <int>> matrix(raws);
        cout << "Матрица системы: \n";
        for (int i = 0; i < raws; i++)
                for (int j = 0; j < cols; j++) {
                        int x;
                        cin >> x;
                        matrix[i].push_back(x);
                }

        for (int i = 0; i < matrix.size(); i++) {
                int revEl = advancedEuclid(matrix[i][i], field).first;
```

```
			matrix[i] = multRawtoNum(matrix[i], revEl, field);
			for (int j = 0; j < matrix.size(); j++) {
				if (i == j)
					continue;
				matrix[j] = addRaws(matrix[j], multRawtoNum(matrix[i], -
matrix[j][i], field), field);
			}
		}
		delZeroRaws(matrix);

		if (!haveSolution) {
			cout << "Решений нет! \n";
			return;
		}

		cout << "\nОбщее решение исходной системы: \n";
		for (int i = 0; i < matrix.size(); i++) {
			cout << "x" << i + 1 << " = ";
			for (int j = i + 1; j < matrix[i].size() - 1; j++) {
				if (matrix[i][j] == 0)
					continue;
				matrix[i][j] = -matrix[i][j] + field;
				cout << matrix[i][j] << "x" << j + 1 << " + ";
			}
			cout << matrix[i].back() << endl;
		}

		cout << "\nСвободные неизвестные ";
		raws = matrix.size();
		int numVals = 0;
		for (int i = raws; i < matrix[0].size() - 1; i++) {
			numVals++;
			if (i == matrix[0].size() - 2)
				cout << "x" << i + 1 << ": ";
			else
				cout << "x" << i + 1 << ", ";
		}
		vector <int> vals;
		for (int i = 0; i < numVals; i++) {
			int x;
			cin >> x;
			vals.push_back(x);
		}

		vector <int> res;
		for (int i = 0; i < matrix.size(); i++) {
			int ans = 0;
			for (int j = raws; j < matrix[i].size() - 1; j++)
				ans += vals[j - raws] * matrix[i][j];
			ans += matrix[i].back();
			res.push_back(ans);
		}
		res.insert(res.end(), vals.begin(), vals.end());

		cout << "Частное решение: (";
		for (int i = 0; i < res.size(); i++) {
			if (i == res.size() - 1)
				cout << res[i] << ")";
			else
				cout << res[i] << ", ";
		}
		cout << endl;
	}
```

```
int main() {
	setlocale(LC_ALL, "ru");
	for (;;) {
		cout << "1 - Алгоритм Евклида \n2 - Решение систем сравнений \n3 - Метод Гаусса \n4 - Выход \n";
		int x;
		cin >> x;
		switch (x) {
		case 1:
			mainEuclid();
			cout << endl;
			break;
		case 2:
			mainSLC();
			cout << endl;
			break;
		case 3:
			gauss();
			cout << endl;
			break;
		case 4:
			return 0;
		default:
			cout << "Incorrect. Try again \n\n";
		}
	}
}
```

ЗАКЛЮЧЕНИЕ

В ходе данной работы были изучены и реализованы арифметические операции в числовых полях. Были рассмотрены обычный, бинарный и расширенный алгоритмы Евклида для нахождения НОД, греко-китайская теорема об остатках и алгоритм Гарнера для решения систем сравнений и метод Гаусса для решения систем линейных уравнений.