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
Сенокосова Владислава Владимировича

Преподаватель, профессор _____ В.А. Молчанов
подпись, дата

Саратов 2024

Содержание

1 Цель работы и порядок выполнения

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

Порядок выполнения работы

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

Вход: Два целых числа a и b

Выход: НОД(a, b)

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

Вход: Множество параметров u_{i}, множество попарно простых модулей m_{i}

Выход: Найденное решение системы сравнений

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

Вход: Матрица коэффициентов системы

Выход: Найденные решения системы линейных уравнений

2 Алгоритм Евклида нахождения НОД двух чисел

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}:

$$a_0 = a_1 q_1 + a_2, 0 \le a_2 < a_1,$$
$$a_1 = a_2 q_2 + a_3, 0 \le a_3 < a_2,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_{k-2} = a_{k-1} q_{k-1} + a_k, 0 \le a_k < a_{k-1},$$
$$a_{k-1} = a_k q_k$$

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

Сложность алгоритма Евклида. Теорема (Ламе). Для любого натурального числа $N > 0$ число делений в алгоритме Евклида для нахождения наибольшего общего делителя чисел A и B, $0 < B < A \le N$ не превосходит $1 + log_R N$. Где R – это золотое сечение равное $\frac{1+\sqrt{5}}{2}$ (корень квадратного уравнения $x^2 - x - 1 = 0$)

Алгоритм Евклида:

Вход: целые числа $a, b;\ 0 < b < a.$

Выход: $d = gcd(a, b)$

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**:** $d = gcd(a, b)$

Сложность алгоритма: $O(\log^2 a)$

Псевдокод алгоритма Евклида:

Начало алгоритма

Функция $GCD(a, b)$:

Пока $b\ != 0$:

$t = b$

$b = a\ mod\ b$

$a = t$

Вернуть a

Конец алгоритма

Сложность реализованного алгоритма: $O(log(min(a, b)))$

Тестирование программной реализации алгоритма Евклида для поиска НОД двух чисел a, b. В качестве примера будут введены значения $a = 3456$ и $b = 45$. Результат выполнения программы представлен на рисунке 1.

```
Укажите один из режимов работы программы:

1 - НОД двух чисел (Алгоритм Евклида)
2 - НОД двух чисел (Бинарный алгоритм Евклида)
3 - НОД двух чисел + коэффициенты Безу (Расширенный алгоритм Евклида)
4 - Решение системы сравнений первой степени (Китайская теорема об остатках)
5 - Решение системы сравнений первой степени (Алгоритм Гарнера)
6 - Решение системы линейных уравнений над полем целых чисел (Алгоритм Гаусса)

:>1
Алгоритм Евклида нахождения НОД двух чисел
Укажите два числа a и b:

a = 3456
b = 45
gcd(a, b) = 9

:>
```

Рисунок 1 – Нахождение НОД чисел a и b алгоритмом Евклида

3 Бинарный алгоритм Евклида нахождения НОД двух чисел

Бинарный алгоритм Евклида — метод нахождения наибольшего общего делителя двух целых чисел. Данный алгоритм быстрее обычного алгоритма Евклида, т.к. вместо медленных операций деления и умножения используются сдвиги. Он основан на использовании следующих свойств НОД:

1. НОД($2m$, $2n$) = 2 НОД(m, n);
2. НОД($2m$, $2n+1$) = НОД(m, $2n+1$);
3. НОД($-m$, n) = НОД(m, n).

Бинарный алгоритм Евклида:

Вход: целые числа a, b; $0 < b < a$.

Выход: $d = gcd(a, b)$

1. $gcd(0, b) = b$;
2. $gcd(a, 0) = gcd(a, a) = a$;
3. $gcd(1, b) = gcd(a, 1) = 1$;
4. Если a и b четные, то $gcd(a, b) = 2 \cdot gcd(a/2, b/2)$;
5. Если a четное и b нечетное, то $gcd(a, b) = gcd(a/2, b)$;
6. Если a нечетное и b четное, то $gcd(a, b) = gcd(a, b/2)$;
7. Если a и b нечетные:

   при этом $b > a$, то $gcd(a, b) = gcd((b - a)/2, a)$;

   при этом $b < a$, то $gcd(a, b) = gcd((a - b)/2, b)$.

Сложность бинарного алгоритма Евклида: $O(\log^2 a)$

Псевдокод бинарного алгоритма Евклида:

```
Начало алгоритма
    Функция GCD_bin(a, b):
        Если a < b:
            Поменять местами a и b
```

Пока $b\ !=\ 0$:

$e\ =\ int(log2(a\ /\ b))$

$t\ =\ min(2\hat{}(e\ +\ 1)\ *\ b\ -\ a, a\ -\ 2\hat{}e\ *\ b)$

Если $t\ <=\ b$:

$a\ =\ b$

$b\ =\ t$

Иначе:

$a\ =\ t$

$b\ =\ b$

Вернуть a

Конец алгоритма

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

Тестирование программной реализации бинарного алгоритма Евклида для поиска НОД двух чисел a, b. В качестве примера будут введены значения $a\ =\ 3456$ и $b\ =\ 45$. Результат выполнения программы представлен на рисунке 2.

```
Укажите один из режимов работы программы:

1 - НОД двух чисел (Алгоритм Евклида)
2 - НОД двух чисел (Бинарный алгоритм Евклида)
3 - НОД двух чисел + коэффициенты Безу (Расширенный алгоритм Евклида)
4 - Решение системы сравнений первой степени (Китайская теорема об остатках)
5 - Решение системы сравнений первой степени (Алгоритм Гарнера)
6 - Решение системы линейных уравнений над полем целых чисел (Алгоритм Гаусса)

:>2
Бинарный алгоритм Евклида нахождения НОД двух чисел
Укажите два числа a и b:

a = 3456
b = 45
gcd(a, b) = 9
:>
```

Рисунок 2 – Нахождение НОД чисел a и b бинарным алгоритмом Евклида

4 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 $x, y \in \mathbb{Z}$. Значения x, y находятся в результате обратного прохода этапов алгоритма Евклида, и называются коэффициентами Безу. В каждом из которых уравнение разрешается относительно остатка a_i, который представляется в форме $a_i = a_{x_i} + b_{y_i}$ для некоторых $x_i , y_i \in \mathbb{Z}$:

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

Расширенный алгоритм Евклида:

Вход: целые числа a, b

Выход: коэффициенты Безу и наибольший общий делитель d, такие что $d = gcd(a, b) = s * a + t * b$

Функция $gcd\_xt(a, b)$:

1. $s_0 = 1, t_0 = 0$
2. $s_1 = 0, t_1 = 1$
3. Если $b != 0$ тогда присваиваем

$$q = int(a / b)$$

$$a, b = b, a \bmod b$$

$$s_0, s_1 = s_1, s_0 - q * s_1$$

$$t_0, t_1 = t_1, t_0 - q * t_1$$

4. Вернуть s_0, t_0, a

Сложность расширенного алгоритма $O(log^2 N)$ где $N - max\{a, b\}$

Псевдокод расширенного алгоритма Евклида:

Функция $gcd\_xt(a, b)$:

Инициализировать $s_0 = 1,\ t_0 = 0$

Инициализировать $s_1 = 0, t_1 = 1$

Пока $b != 0$:

$$q = a // b$$

$$a, b = b, a \bmod b$$

$$s_0, s_1 = s_1, s_0 - q * s_1$$

$$t_0, t_1 = t_1, t_0 - q * t_1$$

Вернуть s_0, t_0, a

Сложность реализованного алгоритма $O(log^2 N)$.

Тестирование программной реализации расширенного алгоритма Евклида для поиска НОД двух чисел a, b. В качестве примера будут введены значения $a = 3456$ и $b = 45$. Также после выполнения мы функции мы получим коэффициенты Безу – x, y, для которых $x * a + y * b = gcd(a, b)$ Результат выполнения программы представлен на рисунке 3.

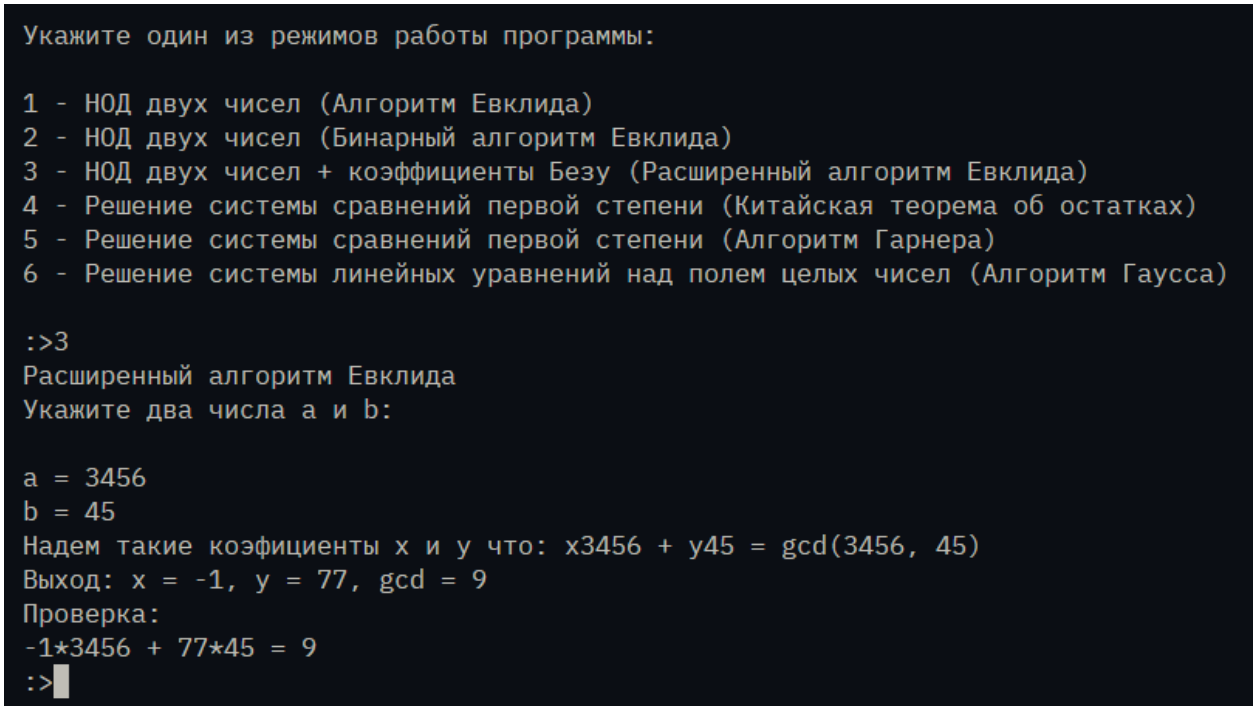

Рисунок 3 – Нахождение НОД чисел a и b и коэффициентов Безу – x и y расширенным алгоритмом Евклида

Таким образом в результате тестирования трех алгоритмов, можно заметить, что на одинаковых входных параметрах выходные значения совпадают, что подтверждает корректность работы данных алгоритмов.

5 Решение системы сравнений первой степени

5.1 Алгоритм Греко-китайской теоремы об остатках

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

$$(s) = \begin{cases} x \equiv a_1 (mod\, m_1)) \\ ... \\ x \equiv a_k (mod\, m_k) \end{cases}$$

имеет единственное неотрицательное решение по модулю M. При этом, если для каждого $1 \leq j \leq n$ 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 $x = M_1 z_1 + \ldots + M_k z_k$.

Алгоритм решения ГКТ:

1. Вычисляем $\prod_{j=1}^{k} m_j$
2. Для всех $i \in \{1, 2, \ldots, k\}$ находим $M_i = \frac{M}{a_i}$.
3. Находим $M_i^{-1} = \frac{1}{M_i} \ (mod\ m_i)$
4. Вычисляем искомое значение по формуле $\sum_{i=1}^{k} a_i M_i M_i^{-1}$.

Сложность алгоритма $O(klog^2 M)$

Псевдокод представленного алгоритма:

Функция $china\_theorem(params, moduls)$:

M = произведение всех чисел в $moduls$

Инициализировать $u = 0$

k − количетсво элементов в $moduls$

Для каждого i, m в перечислении модулей $moduls$:

$$c = M // m\ d, \_, \_ = gcd\_xt(c, m)$$

$$u = (u + c * d * params[i])\ mod\ M$$

Вернуть $u\ mod\ M$

Сложность реализованного алгоритма: $O(klog^2 M)$

Протестируем алгоритм на следующей системе сравнений:

$$(s) = \begin{cases} u \equiv 2\ (mod\ 3) \\ u \equiv 45(mod\ 17) \\ u \equiv 64(mod\ 31) \\ u \equiv 12(mod\ 71) \\ u \equiv 23\ (mod\ 23) \\ u \equiv 5(mod\ 77) \end{cases}$$

Ответом данной системы сравнения будет $u = 79895744$. Результат и тестирование программной реализации алгоритма отображено на рисунке 4.

```
Укажите один из режимов работы программы:

1 - НОД двух чисел (Алгоритм Евклида)
2 - НОД двух чисел (Бинарный алгоритм Евклида)
3 - НОД двух чисел + коэффициенты Безу (Расширенный алгоритм Евклида)
4 - Решение системы сравнений первой степени (Китайская теорема об остатках)
5 - Решение системы сравнений первой степени (Алгоритм Гарнера)
6 - Решение системы линейных уравнений над полем целых чисел (Алгоритм Гаусса)

:>4
Решение системы сравнений первой степени (Китайская теорема об остатках)
Укажите множество параметров U через пробел:
2 45 64 12 23 5
Укажите множество взаимопростых параметров m через пробел:
3 17 31 71 23 77
В результате имеем систему сравнений следующего вида:
u ≡ 2 (mod 3)
u ≡ 45 (mod 17)
u ≡ 64 (mod 31)
u ≡ 12 (mod 71)
u ≡ 23 (mod 23)
u ≡ 5 (mod 77)
Ответ: u = 79895744
Осуществим проверку
2 == 2
11 == 11
2 == 2
12 == 12
0 == 0
5 == 5
:>
```

Рисунок 4 – Решение системы сравнений (s) через программную реализацию ГКТ

5.2 Решение системы сравнений с помощью алгоритма Гарнера

Из китайской теоремы об остатках следует, что можно заменять операции над числами операциями над кортежами. Это может найти широкое применение на практике (помимо непосредственного применения для восстановления числа по его остаткам по различным модулям), поскольку мы таким образом можем заменять операции в длинной арифметике операциями с массивом "коротких" чисел:

Алгоритм Гарнера:

1. Определить $i = 2$, $b = 1$, $s = a_1 \pmod{m_1}$.
2. Пока $i \leq k$ выполнять
   2.1. $b = bm_{i-1}$, $d \equiv b^{-1} \pmod{m_i}$,
   2.2. $x \equiv da_i - s \pmod{m_i}$,
   2.3. $s = s + xb$ и положить $i = i + 1$.
3. Вернуть значение $s \; mod \; M$.

Сложность алгоритма $O(k^2 log^2 M)$

Псевдокод реализованного алгоритма:

Функция $GarnerAlg(r, m)$:

$n = \text{длина}(m)$

$res = 0$

$c = \text{массив из } n \text{ нулей}$

Для i от 0 до $n - 1$:

$c[i] = r[i]$

Для j от 0 до $i - 1$:

$d = \text{обратный элемент } m[j] \text{ по модулю } m[i]$

$c[i] = \big((c[i] - c[j]) * d\big) \; mod \; m[i]$

$factor = 1$

Для i от 0 до $n - 1$:

$res \mathrel{+}= c[i] * factor$

$factor \mathrel{*}= m[i]$

Вернуть $res\ mod\ M$

Сложность реализованного алгоритма $O(n^2 logM)$

Протестируем алгоритм на следующей системе сравнений:

$$(s) = \begin{cases} u \equiv 2\ (mod\ 3) \\ u \equiv 45 (mod\ 17) \\ u \equiv 64 (mod\ 31) \\ u \equiv 12 (mod\ 71) \\ u \equiv 23\ (mod\ 23) \\ u \equiv 5 (mod\ 77) \end{cases}$$

Получим решение системы сравнений $u = 79895744$. Результат и тестирование программной реализации алгоритма отображено на рисунке 5.

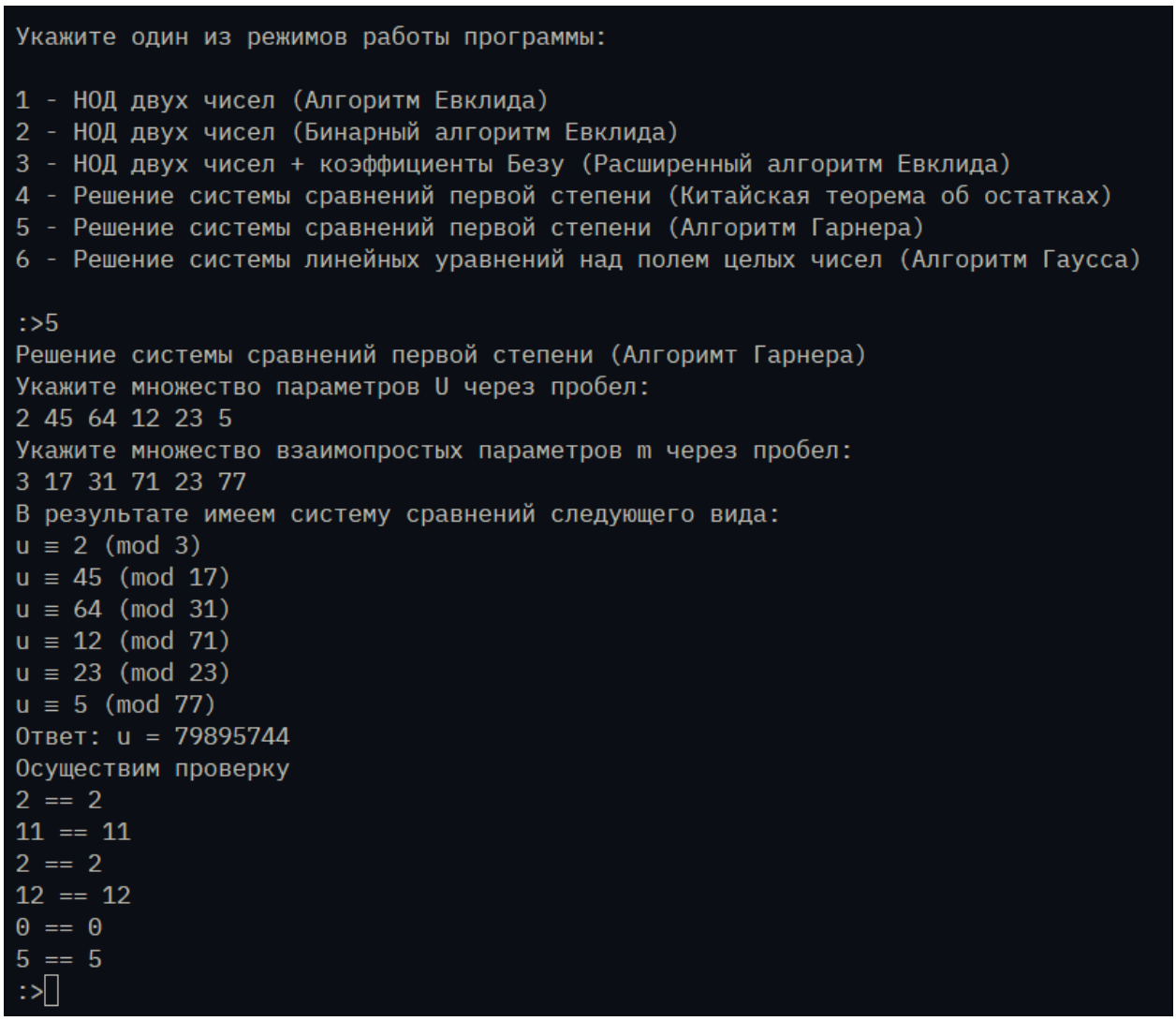

Рисунок 5 - Решение системы сравнений (s) через программную реализацию алгоритма Гарнера

В результате тестирования двух алгоритмов мы получили значения $u = 79895744$, которые являются решениями данной системы сравнений и при этом является единственным.

6 Решение системы линейных уравнений методом Гаусса в кольце целых чисел

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

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

система линейных уравнения с неизвестными x_1, …, x_n, коэффициентами a_{11}, …, a_{mn} (первый индекс указывает номер уравнения, второй индекс – номер неизвестного) и свободными членами b_1, …, b_m (индекс – номер уравнения). При этом числа a_{11}, …, a_{mn} называются также коэффициентами системы и b_1, …, b_m – свободными членами системы. Решением системы (s) называется такой упорядоченный набор $(\zeta_1, \ldots, \zeta_n)$ n вещественных чисел ζ_1, …, ζ_n, что при подстановке в уравнения значений $x_1=\zeta_1, \ldots, x_n=\zeta_n$ получаются верные равенства.

Системы линейных уравнений с одинаковыми множествами решений называются равносильными. 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.

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

Алгоритм Гаусса над конечным полем

Пусть дана система линейных уравнений:

$$\begin{cases} a_{11}x_1 + \ldots + a_{1n}x_n = b_1, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + \ldots + a_{mn}x_n = b_m, \end{cases}$$

Где a_{ij}, $b_i \in Z$. Опишем алгоритм нахождения всех решений в целых числах. Составим матрицу A размера $m \times n$ и матрицу B размера $(m + n) \times (n + 1)$, в которой под матрицей A стоит единичная матрица размера $n \times n$:

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}, \quad B = \begin{pmatrix} & & -b_1 \\ & A & \dots \\ & & -b_m \\ 1 & \dots & 0 \quad 0 \\ \dots & \dots & \dots \\ 0 & \dots & 1 \quad 0 \end{pmatrix}$$

Теперь необходимо привести матрицу к верхней треугольной с помощью операций, определенных над строками и столбцами матрицы.

С первыми n столбцами матрицы B можно производить следующие действия:

1) переставлять их

2) вычитать из одного столбца другой, умноженный на целое число. Также можно переставлять какие-либо из первых m строк матрицы B. С помощью этих действий мы преобразуем матрицу B к частично треугольному виду:

$$B_1 = \begin{pmatrix} u_{11} & 0 & \dots & 0 & 0 & \dots & 0 & -b'_1 \\ u_{21} & u_{22} & \dots & 0 & 0 & \dots & 0 & -b'_2 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ u_{k1} & u_{k2} & \dots & u_{kk} & 0 & \dots & 0 & -b'_k \\ u_{k+1,1} & u_{k+1,2} & \dots & u_{k+1,k} & 0 & \dots & 0 & -b'_{k+1} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ u_{m1} & u_{m2} & \dots & u_{mk} & 0 & \dots & 0 & -b'_m \\ c_{11} & c_{12} & \dots & c_{1k} & c_{1,k+1} & \dots & c_{1n} & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & c_{nk} & c_{n,k+1} & \dots & c_{nn} & 0 \end{pmatrix}$$

После получения матрицы B_1 необходимо получить матрицу B_2 следующего вида:

$$B_2 = \begin{pmatrix} u_{11} & 0 & \dots & 0 & 0 & \dots & 0 \\ u_{21} & u_{22} & \dots & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ u_{k1} & u_{k2} & \dots & u_{kk} & 0 & \dots & 0 \\ u_{k+1,1} & u_{k+1,2} & \dots & u_{k+1,k} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ u_{m1} & u_{m2} & \dots & u_{mm} & 0 & \dots & 0 \\ c_{11} & c_{12} & \dots & c_{1k} & c_{1,k+1} & \dots & f_{1n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ c_{n1} & c_{n2} & \dots & c_{nk} & c_{n,k+1} & \dots & f_{nn} \end{pmatrix}.$$

Для этого мы сначала мы прибавляем первый столбец B_1, умноженный на обратное число к u_{11} и вычитаем из последнего столбца B_1 и получаем ноль в первом элементе $(n + 1)$ - го столбца. Затем с помощью второго столбца и элемента u_{22} мы получаем ноль во втором элементе $(n + 1)$ - го столбцами так далее.

Если такой переход от матрицы B_1 к матрице B_2 указанным способом невозможен, то система не имеет решений в целых числах. Если же мы нашли матрицу B_2 из B_1, то общее решение системы в целых числах имеет вид:

$$\begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} = \begin{pmatrix} f_1 \\ \dots \\ f_n \end{pmatrix} + t_1 \begin{pmatrix} c_{1,k+1} \\ \dots \\ c_{n,k+1} \end{pmatrix} + \dots + t_{n-k} \begin{pmatrix} c_{1n} \\ \dots \\ c_{nn} \end{pmatrix}$$

где $t_1, \dots, t_{n-k}$ пробегают все множество целых чисел (из нашего кольца) $k \leq \min(n, m)$.

Временная сложность алгоритма: $O(min(n, m) * n * m)$

Псевдокод реализованного алгоритма:

Вся реализация представлена в виде отдельных функций, поэтому опишем каждую по-отдельности.

Функция *triangular_matrix*

Шаг 1: Преобразование матрицы в треугольную форму

Для каждого элемента на диагонали (от 0 до $min(n, m)$):

1.1 Если элемент на позиции (k, k) равен 0: - Найти строку ниже с ненулевым элементом в этом столбце. - Если найдена строка, поменять местами строки.

1.2 Найти минимальный ненулевой элемент в строке от позиции (k, k) до конца строки: - Если найден минимальный элемент (min_el) в позиции (k, j), где $j \neq k$: - Поменять местами столбцы k и j.

1.3 Найти обратный элемент для $matrix[k][k]$ по модулю p.

1.4 Для каждого столбца справа (от $k + 1$ до n): Вычислить множитель $(factor)$ для сокращения этого столбца. Вычесть кратный k-й столбец из текущего столбца с учетом модуля p.

Шаг 2: Возврат преобразованной треугольной матрицы.

Функция $find\_roots$

Шаг 1: Нормализация свободных членов

Для каждого элемента на диагонали (от 0 до $min(m, n)$):

1.1 Найти обратный элемент для $matrix[k][k]$ по модулю p.

1.2 Умножить свободный член на этот обратный элемент и вычесть его из всех других строк.

Шаг 2: Проверка на отсутствие решений

Если в какой-либо строке результат свободного члена не равен 0, вернуть "Не имеет решения в целых числах".

Шаг 3: Получение возможных решений

Перебрать все возможные комбинации решений для свободных переменных (с учетом модуля p).

3.1 Для каждой комбинации:

Построить линейную комбинацию с матрицей. Добавить результат в список решений.

Шаг 4: Возврат всех найденных решений.

Функция $gauss\_mod$

Шаг 1: Получение расширенной матрицы системы

Разделить систему на матрицу коэффициентов и вектор свободных членов.

Шаг 2: Приведение матрицы к треугольному виду

Вызвать функцию $triangular\_matrix$ для приведения матрицы к треугольной форме.

Шаг 3: Поиск решений

Вызвать функцию $find\_roots$ для нахождения всех возможных решений.

Шаг 4: Возврат решений системы.

Временная сложность реализованного алгоритма: $O(min(n, m) * n * m)$

Тестирование и проверку правильности будем осуществлять на следующей системе по модулю 2.

1 0 1 1 1 0 0

1 1 1 1 0 1 1

1 0 1 1 0 1 1

0 1 0 0 1 1 1

0 1 1 1 0 1 1

В результате работы программы получаем следующие решения системы. Решение является общим, поэтому рассмотрены все возможные комбинации корней. Информация о тестировании и полученных корнях отображена на рисунке 6.

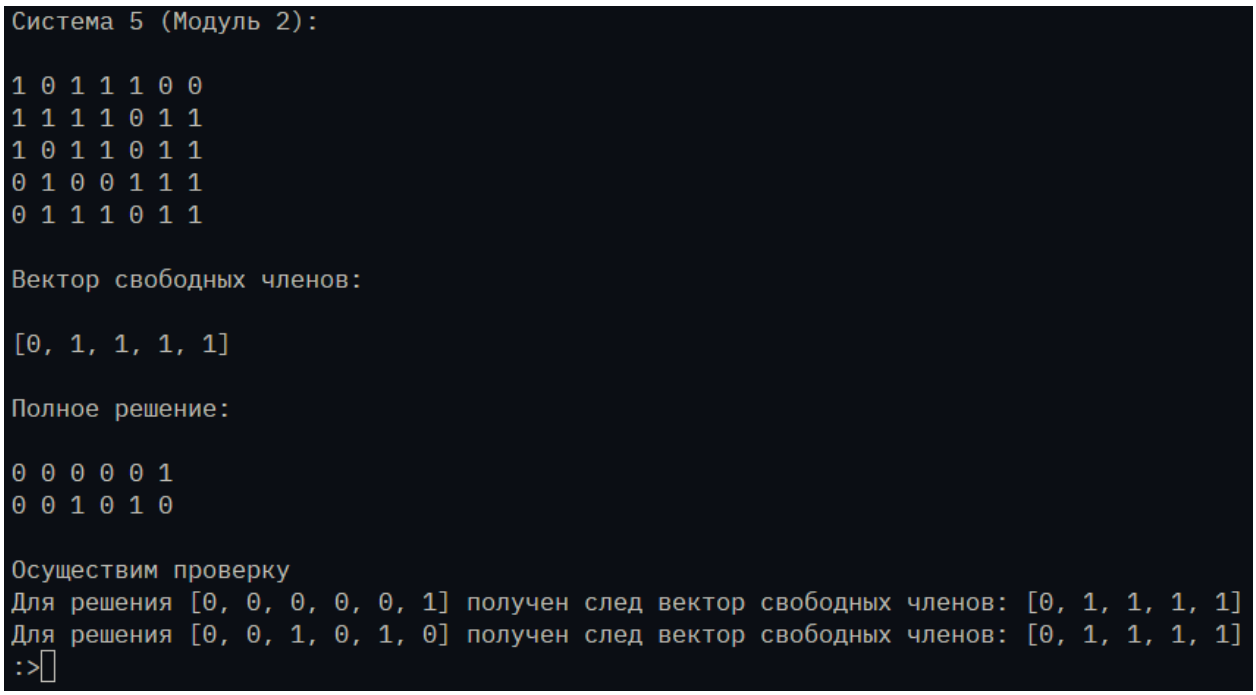

Рисунок 6 – Решение системы линейных уравнений над конечным полем

В данном случае в качестве ответа выдается множество последовательностей неизвестных $x_1, x_2, \ldots . x_n$ в порядке их следования. То есть в приведенном выше примере имеется два решения системы: $(x_1, x_2, x_3\ x_4, x_{5,} x_6)$ = (0, 0, 0, 0, 0, 1), $(x_1, x_2, x_3\ x_4, x_{5,} x_6)$ = (0, 0, 1, 0, 1, 0). Как можно убедиться при подстановке полученных корней, мы получаем свободные члены изначальной системы, что говорит о том, что корни найденные являются верными.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были изучены и реализованы основные операции в числовых полях, включая вычисление наибольшего общего делителя (НОД) с помощью различных версий алгоритма Евклида (обычный, бинарный и расширенный), а также методы решения систем сравнений и систем линейных уравнений в конечных полях методом Гаусса.

Все алгоритмы были успешно реализованы на языке программирования Python. Программные реализации были протестированы, и результаты тестирования подтвердили корректность работы алгоритмов. Для каждого алгоритма были проанализированы и оценены их временные сложности, что позволило глубже понять их эффективность.

В результате работы были достигнуты следующие цели:

- Освоены и программно реализованы алгоритмы Евклида для вычисления НОД;
- Изучены и реализованы алгоритмы решения систем сравнений;
- Программно реализован метод Гаусса для решения систем линейных уравнений над конечными полями.

Таким образом, данная работа позволила не только изучить теоретические основы числовых операций и алгоритмов, но и применить их на практике через программную реализацию и тестирование.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Глухов М. М. и др. Введение в теоретико-числовые методы криптографии: учеб. пособие - Москва : Лань, 2011.

2. Маховенко Е.Б. Теоретико-числовые методы в криптографии. М.: Гелиос АРВ, 2006.

3. Черемушкин, А. В. Лекции по арифметическим алгоритмам в криптографии. - Москва : МЦНМО, 2002.

4. Панкратова И.А. Теоретико-числовые методы в криптографии. Томск, 2009.

5. Василенко О.Н. Теоретико-числовые алгоритмы в криптографии. М.:МЦНМО, 2003.

6. Венбо Мао. Современная криптография: теория и практика. М.:Вильямс, 2005.

ПРИЛОЖЕНИЕ А

Лабораторная работа с реализованными алгоритмами

```
from math import log2, gcd
from copy import deepcopy
import itertools
import interface

# Обычный алгоритм Евклида (НОД двух чисел)
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

# Бинарный алгоритм Евклида
def gcd_bin(a, b):
    if a < b:
        a, b = b, a
    while b != 0:
        e = int(log2(a / b))
        t = min(pow(2, e + 1) * b - a, a - pow(2, e) * b)
        if t <= b:
            a, b = b, t
        else:
            a, b = t, b
    return a

# Расширенный алгоритм Евклида
def gcd_xt(a, b):
    s0, t0 = 1, 0
    s1, t1 = 0, 1
    while b != 0:
        q = a // b
```

```
        a, b = b, a % b
        s0, s1 = s1, s0 - q * s1
        t0, t1 = t1, t0 - q * t1
    return s0, t0, a

# Нахождение произведения элеметов в массиве
def prod(list):
    res = 1
    for val in list: res *= val
    return res

# Китайсткая теорема об остатках (система сравнений)
def china_theorem(params, moduls):
    M = prod(moduls)
    u = 0
    for i, m in enumerate(moduls):
        c = M // m
        d, _, _ = gcd_xt(c, m)
        u += c * d * params[i] % M
    return u % M

# Алгоритм Гарнера (система сравнений)
def garner_alg(params, moduls):
    k = len(moduls)
    u, q = 0, [0] * k
    for i in range(k):
        q[i] = params[i]
        for j in range(i):
            if i != j:
                d, _, _ = gcd_xt(moduls[j], moduls[i])
                q[i] = ((q[i] - q[j]) * d) % moduls[i]
    factor = 1
    for i in range(k):
        u += q[i] * factor
        factor *= moduls[i]
```

```
    return u % pord(moduls)

# Вывод матрицы в терминал
def print_matrix(matrix):
    for row in matrix:
        print(*row)
    print()

# Получение матрицы коэфициентов системы и вектора свободных
членов
def get_matrix_A_and_vector(system):
    A, vec = [], []
    for row in system:
        A.append(row[:-1])
        vec.append(row[-1])
    return A, vec

# Умножение строки матрицы на число в кольце по модулю p
def mul_row_on_num(vec, num, p):
    return list(map(lambda x: x * num % p, vec))

# Сложение строк матрицы по модулю p
def add_rows_mod(row1, row2, p):
    res = []
    for i in range(len(row1)):
        res.append((row1[i] + row2[i]) % p)
    return res

# Получение новой матрицы B из матрицы коэфициаентов A,
# вектора свобоных членов и единичной матрицы
def get_matrix_B(A, vec, p):
    B, n, m = [], len(A[0]), len(A)
    vec = mul_row_on_num(vec, -1, p)
    for i in range(m):
        new_row = A[i] + [vec[i]]
```

```
        B.append(new_row)
    for i in range(n):
        buff = []
        for j in range(n + 1):
            if i == j:
                buff.append(1)
            else:
                buff.append(0)
        B.append(buff)
    return B

# Перестановка строк в матрице
def swap_row(matrix, row1, row2):
    matrix[row1], matrix[row2] = matrix[row2], matrix[row1]

# Перестановка столбцов в матрице
def swap_col(matrix, index1, index2):
    for row in matrix:
        row[index1], row[index2] = row[index2], row[index1]

# Вычитание столбцов по модулю
def sub_colums(matrix, index1, index2, factor, m):
    for row in matrix:
        row[index1] -= factor * row[index2]
        row[index1] %= m

# Сложение столбцов по модулю
def add_columns_mod(matrix, col1, col2, m):
    num_rows = len(matrix)
    for i in range(num_rows):
        matrix[i][col1] = (matrix[i][col1] + matrix[i][col2]) %
m

# Умножение столбца на число по модулю
def mul_column_on_num(matrix, col, multiplier, m):
```

```
    num_rows = len(matrix)
    for i in range(num_rows):
        matrix[i][col] = (matrix[i][col] * multiplier) % m

# Поиск ненулевого элемента в столбце начиная со строки start
def find_not_zero_elem_in_col(matrix, start, end, col):
    for i in range(start, end):
        if matrix[i][col] != 0:
            return i
    return None

# Преобразование столбца в строку
def col_to_vec(matrix, col, row):
    res = []
    for i in range(row, len(matrix)):
        res.append(matrix[i][col])
    return res

# Приведение матрицы к верхней треугольной
def triangular_matrix(matrix, m, n, p):

    for k in range(min(n , m)):
        if matrix[k][k] == 0:
            i_new_row = find_not_zero_elem_in_col(matrix, k, m,
k)
            if i_new_row is not None:
                swap_row(matrix, k, i_new_row)

        min_el = float('inf')
        min_index = None

        for j in range(k, n):
            if matrix[k][j] != 0 and abs(matrix[k][j]) <
abs(min_el):
                min_el = matrix[k][j]
```

```
                min_index = j

        if min_index is not None and k != min_index:
            swap_col(matrix, k, min_index)

        inv, _, _ = gcd_xt(matrix[k][k], p)
        inv %= p

        for j in range(k + 1, n):
            if matrix[k][j] != 0 and j != k:
                factor = (matrix[k][j] * inv) % p
                sub_colums(matrix, j, k, factor, p)
    return matrix

# Поиск корней системы уравнений в кольце целых чисел
def find_roots(matrix, p, n, m):

    # Обнуляем свободные члены
    for k in range(min(m, n)):
        inv, _, _ = gcd_xt(matrix[k][k], p)
        inv %= p
        factor = (matrix[k][-1] * inv) % p
        sub_colums(matrix, n, k, factor, p)
    for i in range(m):
        if matrix[i][-1] != 0:
            return "Не имеет решения в целых числах"
    roots = []

    # Получаем линейную комбинацию
    for comb in itertools.product(range(p), repeat=(n - min(m,
n))):
        start_i = -1
        f = col_to_vec(matrix, start_i, m)
        for i in range(n - min(m, n)):
            start_i -= 1
```

```
            new_row = col_to_vec(matrix, start_i, m)
            f = add_rows_mod(f, mul_row_on_num(new_row, comb[i],
p), p)
        roots.append(f)

    return roots

# Функция Гаусса решения системы уравнений в кольце
def gauss_mod(system, p):
    A, vec = get_matrix_A_and_vector(system)
    B = get_matrix_B(A, vec, p)
    m, n = len(A), len(A[0])
    triangular = triangular_matrix(B, m, n, p)
    return find_roots(triangular, p, n, m)

if __name__ == "__main__":
    print(interface.start_massage)
    param = None
    while param not in ["1", "2", "3", "4", "5", "6"]:
        param = input(":>").strip()
        match param:
            case "1":
                print("Алгоритм Евклида нахождения НОД двух
чисел")
                print("Укажите два числа a и b: \n")
                a = input("a = ")
                b = input("b = ")
                print(f"gcd(a, b) = {gcd(int(a), int(b))} \n")
                param = None
            case "2":
                print("Бинарный алгоритм Евклида нахождения НОД
двух чисел")
                print("Укажите два числа a и b: \n")
                a = input("a = ")
                b = input("b = ")
```

```
                print(f"gcd(a, b) = {gcd_bin(int(a), int(b))}")
                param = None
            case "3":
                print("Расширенный алгоритм Евклида")
                print("Укажите два числа a и b: \n")
                a = input("a = ")
                b = input("b = ")
                print(f"Надем такие коэфициенты x и y что: x{a}
+ y{b} = gcd({a}, {b})")
                x, y, gcd_ = gcd_xt(int(a), int(b))
                print(f"Выход: x = {x}, y = {y}, gcd = {gcd_}")
                print("Проверка:")
                if y < 0:
                    print(f"{x}*{a} - {abs(y)}*{b} = {gcd_}")
                else:
                    print(f"{x}*{a} + {y}*{b} = {gcd_}")
                param = None
            case "4":
                print("Решение системы сравнений первой степени
(Китайская теорема об остатках)")
                flag = False
                while not flag:
                    u = input("Укажите множество параметров U
через пробел: \n").split()
                    m = input("Укажите множество взаимопростых
параметров m через пробел: \n").split()
                    if interface.check_params(u, m):
                        flag = True
                    else:
                        print("Введены некорректные данные!!!")
                        flag = False
                u = list(map(lambda x: int(x), u))
                m = list(map(lambda x: int(x), m))
                print("В результате имеем систему сравнений
следующего вида:")
```

```
                for i in range(len(u)):
                    print(f"u ≡ {u[i]} (mod {m[i]})")
                res = china_theorem(u, m)
                print(f"Ответ: u = {res}")
                print("Осуществим проверку")
                for i in range(len(u)):
                    print(f"{res % m[i]} == {u[i] % m[i]}")
                param = None
            case "5":
                print("Решение системы сравнений первой степени
(Алгоримт Гарнера)")
                flag = False
                while not flag:
                    u = input("Укажите множество параметров U
через пробел: \n").split()
                    m = input("Укажите множество взаимопростых
параметров m через пробел: \n").split()
                    if interface.check_params(u, m):
                        flag = True
                    else:
                        print("Введены некорректные данные!!!")
                        flag = False
                u = list(map(lambda x: int(x), u))
                m = list(map(lambda x: int(x), m))
                print("В результате имеем систему сравнений
следующего вида:")
                for i in range(len(u)):
                    print(f"u ≡ {u[i]} (mod {m[i]})")
                res = garner_alg(u, m)
                print(f"Ответ: u = {res}")
                print("Осуществим проверку")
                for i in range(len(u)):
                    print(f"{res % m[i]} == {u[i] % m[i]}")
                param = None
            case "6":
```

```
                print("Решение системы уравнений первой степени
в кольце целых чисел (Алгоритм Гаусса)")
                file = input("Укажите файл с записанной системой
и модулем: ")
                data = interface.parse_file(file)
                for i, pair in enumerate(data):
                    m = pair[0]
                    system = pair[1]
                    cop_system = deepcopy(system)
                    print(f"Система {i + 1} (Модуль {m}): \n")
                    print_matrix(system)
                    roots = gauss_mod(system, m)
                    roots.sort()
                    print(f"Вектор свободных членов: \n")
                    vec = []
                    for i in range(len(system)):
                        vec.append(system[i][-1])
                    print(vec, "\n")
                    print(f"Полное решение: \n")
                    print_matrix(roots)
                    print(f"Осуществим проверку")
                    for root in roots:
                        res = []
                        for j in range(len(cop_system)):
                            sum_ = 0
                            for i in range(len(cop_system[j]) -
1):
                                sum_ += cop_system[j][i] *
root[i]
                            res.append(sum_ % m)
                        print(f"Для решения {root} получен след
вектор свободных членов: {res}")
                param = None
```

ПРИЛОЖЕНИЕ В
Интерфейс программы

```
start_massage = """
Укажите один из режимов работы программы:

1 - НОД двух чисел (Алгоритм Евклида)
2 - НОД двух чисел (Бинарный алгоритм Евклида)
3 - НОД двух чисел + коэффициенты Безу (Расширенный алгоритм
Евклида)
4 - Решение системы сравнений первой степени (Китайская теорема
об остатках)
5 - Решение системы сравнений первой степени (Алгоритм Гарнера)
6 - Решение системы линейных уравнений над полем целых чисел
(Алгоритм Гаусса)
"""

def check_params(params, moduls):
    if len(params) != len(moduls):
        return False
    for i in range(len(params)):
        if not (params[i].isdigit() and moduls[i].isdigit()):
            return False
    return True

def parse_file(filepath):
    with open(filepath, "r") as f:
        res = []
        m, system = None, []
        lines = f.readlines()
        for i, line in enumerate(lines):
            new_l = line.strip().split()
            if len(new_l) == 1 and i == 0:
                m = int(new_l[0])
                system = []
            if len(new_l) == 1 and i != 0:
```

```
                m = int(new_l[0])
                system = []
                res.append([m, system])
            else:
                system.append(list(map(lambda x: int(x),
new_l)))
    return res
```