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

Мызникова Сергея Анатольевича

|  |  |  |
| --- | --- | --- |
| Преподаватель, профессор | \_\_\_\_\_\_\_\_\_\_ | В.А. Молчанов |
|  | подпись, дата |  |

Саратов 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc177814196)

[2 Теоретические сведения 4](#_Toc177814197)

[2.1 Алгоритм Евклида 4](#_Toc177814198)

[2.2 Расширенный алгоритм Евклида 4](#_Toc177814199)

[2.3 Бинарный алгоритм Евклида 5](#_Toc177814200)

[2.4 Греко-китайская теорема об остатках 5](#_Toc177814201)

[2.5 Алгоритм Гарнера 5](#_Toc177814202)

[2.6 Метод Гаусса 6](#_Toc177814203)

[3 Результаты работы 8](#_Toc177814204)

[3.1 Алгоритмы: описание, псевдокод и временная сложность 8](#_Toc177814205)

[3.1.1 Обычный алгоритм Евклида 8](#_Toc177814206)

[3.1.2 Расширенный алгоритм Евклида 8](#_Toc177814207)

[3.1.3 Бинарный алгоритм Евклида 9](#_Toc177814208)

[3.1.4 Греко-китайская теорема об остатках 11](#_Toc177814209)

[3.1.5 Алгоритм Гарнера 12](#_Toc177814210)

[3.1.6 Метод Гаусса 14](#_Toc177814211)

[3.2 Тестирование алгоритмов 16](#_Toc177814212)

[ЗАКЛЮЧЕНИЕ 19](#_Toc177814213)

[ПРИЛОЖЕНИЕ А 20](#_Toc177814214)

# **1 Постановка задачи**

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.

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

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

# **2 Теоретические сведения**

## **2.1 Алгоритм Евклида**

Пусть даны целые числа x1 > x2 > 0. Для вычисления наибольшего общего делителя (x1, x2) существует алгоритм Евклида

r-1 = x1;

r0 = x2;

ri-2 = di \* ri-1 + ri, 0 < ri < ri-1, i = 1, …, k;

rk-1 = dk+1 \* rk.

Тогда НОД(x1, x2) = rk и для его нахождения требуется выполнить k + 1 делений с остатком. Так как остатки выполняемых делений образуют строго убывающую последовательность, то этот процесс обязательно остановится в результате получения нулевого остатка деления.

## **2.2 Расширенный алгоритм Евклида**

Если в алгоритме Евклида на каждом шаге вычислять кроме частного di и остатка ri еще два значения ui и vi по правилу:

u-1 = 1, u0 = 0;

v-1 = 0, v0 = 1;

ui = ui-2 - di \* ui-1, i = 1, …, k;

vi = vi-2 – di \* vi-1, i = 1, …, k.

то такой алгоритм будет называться расширенным алгоритмом Евклида. Его суть состоит в том, чтобы дать линейное разложение наибольшего общего делителя uk \* x1 + vk \* x2 = rk = НОД(x1, x2). Данное уравнение играет важную роль в операциях модульной арифметики.

## **2.3 Бинарный алгоритм Евклида**

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

• НОД(2⋅a, 2⋅b) = 2⋅НОД(a, b);

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

• НОД(-a, b) = НОД(a, b).

## **2.4 Греко-китайская теорема об остатках**

Пусть m1, m2, …, mk - попарно взаимно простые целые числа и M = m1m2…mk. Тогда система линейных сравнений

имеет единственное неотрицательное решение по модулю M.

При этом, если для каждого 1 ≤ j ≤ k число Mj = и сравнение Mj \* x = aj (mod mj) имеет решение zj, то решением системы линейных сравнений является остаток по модулю M числа x = M1z1 + M2z2 + … + Mkzk.

## **2.5 Алгоритм Гарнера**

Пусть все mi > 1, m = m1\*…\*mt. Тогда любое число 0 ≤ x < m может быть однозначным образом представлено в виде

x = x0 + x1m1 + x2m1m2 + ... + xt-1m1m2\*...\*mt-1,

где 0 ≤ xi < mi+1, i = 0, 1, …, t-1.

Для xi верно соотношение

Таким образом, xi могут быть вычислены один за другим. Получившийся алгоритм носит имя Гарнера.

## **2.6 Метод Гаусса**

Системой m линейных уравнений с n неизвестными x1, …, xn называется выражение вида

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:

a) удаление из системы тривиальных уравнений;

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

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

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

где r ≤ m, так как в процессе элементарных преобразований исходной системы удаляются тривиальные уравнения. В этом случае неизвестные x1, …, xr называются разрешенными (или базисными) и xr+1, …, xn – свободными.

Преобразование системы в равносильную ей разрешенную систему осуществляется по методу Гаусса с помощью последовательного выполнения следующих Жордановых преобразований:

1. выбираем один из коэффициентов системы 0;
2. умножаем i-ое уравнение системы на элемент ;
3. прибавляем к обеим частям остальных k-ых уравнений системы соответствующие части нового i-ого уравнения, умноженные на коэффициент –
4. удаляем из системы тривиальные уравнения.

При этом выбранный ненулевой элемент называется разрешающим, строка и столбец, содержащие элемент также называются разрешающими.

# **3 Результаты работы**

## **3.1 Алгоритмы: описание, псевдокод и временная сложность**

### **3.1.1 Обычный алгоритм Евклида**

Вход: два целых числа a, b.

Выход: НОД(a, b)

Шаг 1: пока a не равно 0 и b не равно 0 выполнить:

Шаг 1.1: если a > b, то положить a = a mod b и вернуться к шагу 1

Шаг 1.2: иначе положить b = b mod a и вернуться к шагу 1

Шаг 2: вывести a + b

**Псевдокод**:

Начало алгоритма

Пока a не равно 0 и b не равно 0 выполнить:

Если a больше b выполнить:

a = a mod b

Иначе выполнить:

b = b mod a

Вернуть a + b

Конец алгоритма

**Временная сложность алгоритма**. O(L(a)\*L(b))

### **3.1.2 Расширенный алгоритм Евклида**

Вход: два целых числа a, b.

Выход: НОД(a, b)

Шаг 1: положить x, xx, y, yy равными 1, 0, 0, 1

Шаг 2: пока b не равен 0 выполнить:

Шаг 2.1: положить q равным a // b

Шаг 2.2: положить a и b равными b и a mod b соответственно

Шаг 2.3: положить x и xx равными xx, x – xx\*q

Шаг 2.4: положить y, yy равными yy, y – yy\*q и вернуться к шагу 2

Шаг 3: вернуть a, x, y

**Псевдокод:**

Начало алгоритма

Инициализировать x = 1, xx = 0, y = 0, yy = 1

Пока b не равно 0 выполнить:

q = целая часть от a / b

a = b

b = a mod b

x = xx

xx = x - xx \* q

y = yy

yy = y - yy \* q

Вернуть a, x, y

Конец алгоритма

**Временная сложность алгоритма.** O(L(a)\*L(b))

### **3.1.3 Бинарный алгоритм Евклида**

Вход: два целых числа a, b.

Выход: НОД(a, b)

Шаг 1: если a = 0, вернуть b;

Шаг 2: если a = b или b = 0, вернуть a;

Шаг 3: если a = 1 или b = 1, вернуть 1;

Шаг 4: Если a и b четные, то вернуть 2\*НОД(a/2, b/2);

Шаг 5: Если a четное и b нечетное, то вернуть НОД(a/2, b);

Шаг 6: Если a нечетное и b четное, то вернуть НОД(a, b/2);

Шаг 7: Если a и b нечетные, то выполнить:

Шаг 7.1: если b > a, то вернуть НОД((b-a)/2, a);

Шаг 7.2: иначе вернуть НОД((a-b)/2, b).

**Псевдокод:**

Начало алгоритма

Функция binary\_Euclid(a, b)

Если a равно 0 выполнить:

Вернуть b

Иначе если b равно 0 или a равно b выполнить:

Вернуть a

Иначе если a равно 1 или b равно 1 выполнить:

Вернуть 1

Иначе если a и b четные выполнить:

Вернуть 2 \* binary\_Euclid (a / 2, b / 2)

Иначе если a четное и b нечетное выполнить:

Вернуть binary\_Euclid (a / 2, b)

Иначе если a нечетное и b четное выполнить:

Вернуть binary\_Euclid (a, b / 2)

Иначе если a и b нечетные выполнить:

Если b больше a выполнить:

Вернуть binary\_Euclid ((b - a) / 2, a)

Иначе выполнить:

Вернуть binary\_Euclid ((a - b) / 2, b)

Конец функции

Конец алгоритма

**Временная сложность алгоритма**. O(L(a)\*L(b))

### **3.1.4 Греко-китайская теорема об остатках**

Вход: список коэффициентов и список модулей

Выход: решение системы

Шаг 1: найти произведение всех модулей и записать в M

Шаг 2: создать список с элементами M / m[i]

Шаг 3: найти все решения zj сравнений Mj \* x = aj (mod mj) и добавить в список z

Шаг 4: вычислить x = M1z1 + M2z2 + … + Mkzk

**Псевдокод:**

Начало алгоритма

Инициализировать M = 1

Для каждого числа num в m:

M = M \* num

Инициализировать M\_j как пустой список

Для каждого i от 0 до длины m:

Добавить M / m[i] в M\_j

Инициализировать z как пустой список

Для каждого i от 0 до длины M\_j:

Вычислить обратное по модулю mod\_inverse(M\_j[i], m[i])

Добавить (mod\_inverse(M\_j[i], m[i]) \* a[i]) % m[i] в z

Инициализировать x = 0

Для каждого i от 0 до длины z:

x = x + M\_j[i] \* z[i]

Вернуть x % M

Конец алгоритма

**Временная сложность алгоритма.** O(k2\*log2(mi) + k\*log2(mi)), k – количество mi

### **3.1.5 Алгоритм Гарнера**

Вход: список коэффициентов и список модулей

Выход: решение системы

Шаг 1: для i = 2…k выполнить следующие действия

Шаг 1.1: установить ci = 1

Шаг 1.2: для j = 1…i-1 выполнить

Шаг 1.2.1: установить u = (mod mi)

Шаг 1.2.2: установить ci = uci (mod mi)

Шаг 2: установить u = a1

Шаг 3: установить x = u

Шаг 4: для i = 2…k вычислить u = (ai – x)ci и x = x + u

**Псевдокод:**

Начало алгоритма

Инициализировать список c длиной m, заполненный нулями

Для каждого i от 1 до длины m:

c[i] = 1

Для каждого j от 0 до i:

Вычислить u как обратное по модулю mod\_inverse(m[j], m[i])

c[i] = (u \* c[i]) % m[i]

Инициализировать u как a[0]

Инициализировать x как u

Для каждого i от 1 до длины m:

Вычислить u = (a[i] - x) \* c[i]

Если u меньше 0:

u = u + m[i]

u = u % m[i]

Инициализировать m\_j = 1

Для каждого j от 0 до i:

m\_j = m\_j \* m[j]

x = x + u \* m\_j

Вернуть x

Конец алгоритма

**Временная сложность алгоритма.** O(m2\*L(mi)\*L(mj) + m2\*L(mi)\*L(mj))

### **3.1.6 Метод Гаусса**

Вход: матрица вместе со столбцом свободных членов, модуль, недостающие корни, если нужно

Выход: частное решение

Шаг 1: найти строку с ненулевым ведущим элементом

Шаг 2: поменять строки местами так, чтобы ведущая строка была выше

Шаг 3: привести ведущий элемент в строке к 1

Шаг 4: для каждой строки найти определить, насколько нужно умножить найденный ведущий элемент, чтобы обнулить в остальных строках

Шаг 5: обнулить все элементы в текущем столбце (помимо ведущего)

Шаг 6: удалить нулевые строки, если имеются

Шаг 7: увеличить индекс строки для следующей итерации

Шаг 8: обновить количество строк

**Псевдокод:**

Начало алгоритма

n = количество строк в matrix

m = количество столбцов в matrix - 1

row\_idx = 0

Для каждого col\_idx от 0 до m:

pivot\_row = None

Для каждого i от row\_idx до n:

Если элемент matrix[i][col\_idx] не равен 0:

pivot\_row = i

Прервать цикл

Если pivot\_row равен None:

Перейти к следующей итерации цикла

Если pivot\_row не равен row\_idx:

Поменять местами строки matrix[row\_idx] и matrix[pivot\_row]

inv = обратное по модулю значение matrix[row\_idx][col\_idx] относительно mod

Для каждого k от 0 до m + 1:

matrix[row\_idx][k] = (matrix[row\_idx][k] \* inv) % mod

Для каждого i от 0 до n:

Если i не равен row\_idx и элемент matrix[i][col\_idx] не равен 0:

factor = matrix[i][col\_idx]

Для каждого k от 0 до m + 1:

matrix[i][k] = (matrix[i][k] - factor \* matrix[row\_idx][k]) % mod

matrix = список строк из matrix, где хотя бы один элемент в строке не равен 0

Увеличить row\_idx на 1

n = количество строк в обновленной matrix

Вернуть matrix

Конец алгоритма

**Временная сложность алгоритма.** O(m2\*n\*log(mod))

## **3.2 Тестирование алгоритмов**

Обычный алгоритма Евклида

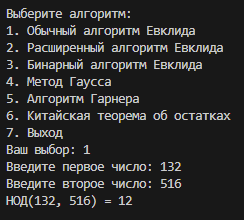


Рисунок 1. Тестирование обычного алгоритма Евклида

Расширенный алгоритм Евклида

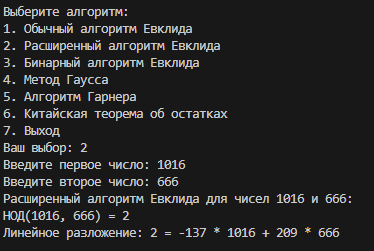


Рисунок 2. Тестирование расширенного алгоритма Евклида

Бинарный алгоритм Евклида

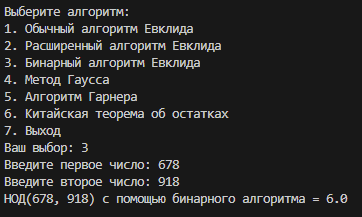
****

Рисунок 3. Тестирование бинарного алгоритма Евклида

Метод Гаусса

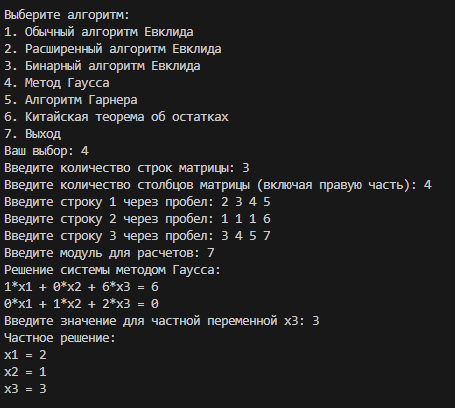


Рисунок 4. Тестирование метода Гаусса

Алгоритм Гарнера

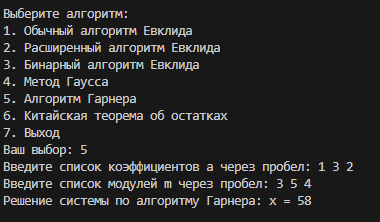


Рисунок 5. Тестирование алгоритма Гарнера

Греко-китайская теорема об остатках

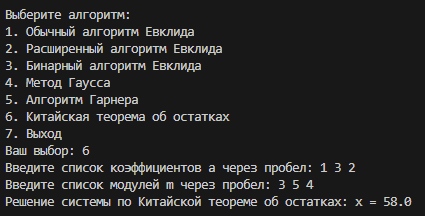


Рисунок 6. Тестирование греко-китайской теоремы об остатках

# **ЗАКЛЮЧЕНИЕ**

В ходе работы били рассмотрены алгоритмы Евклида (обычный, расширенный, бинарный), некоторые из алгоритмов для решения систем сравнений, а именно, греко-китайская теорема об остатках и алгоритм Гарнера. Также был рассмотрен алгоритм Гаусса для решения систем линейных уравнений над конечным полем.

Для проверки работоспособности этих алгоритмов была реализована программа на языке Python. Программы были протестированы и была подтверждена правильность их работы. Также была произведена оценка временной сложности алгоритмов, приведены их псевдокоды.

# **ПРИЛОЖЕНИЕ А**

**Программа с реализованными алгоритмами**

# Обычный алгоритм Евклида

def normal\_Euclid(a, b):

while a !=0 and b != 0:

if a > b:

a = a % b

else:

b = b % a

return a + b

# Расширенный алгоритм Евклида

def extended\_Euclid(a, b):

x, xx, y, yy = 1, 0, 0, 1

while b:

q = a // b

a, b = b, a % b

x, xx = xx, x - xx\*q

y, yy = yy, y - yy\*q

return a, x, y

# Бинарный алгоритм Евклида

def binary\_Euclid(a, b):

if a == 0:

return b

elif b == 0 or a == b:

return a

elif a == 1 or b == 1:

return 1

elif a % 2 == 0 and b % 2 == 0:

return 2 \* binary\_Euclid(a / 2, b / 2)

elif a % 2 == 0 and b % 2 != 0:

return binary\_Euclid(a / 2, b)

elif a % 2 != 0 and b % 2 == 0:

return binary\_Euclid(a, b / 2)

elif a % 2 != 0 and b % 2 != 0:

if b > a:

return binary\_Euclid((b - a) / 2, a)

else:

return binary\_Euclid((a - b) / 2, b)

# Поиск обратного элемента

def mod\_inverse(z, m):

gcd, x, \_ = extended\_Euclid(z, m)

return inv\_elem(x, m)

def inv\_elem(x, m):

if x < 0:

x = x + m

return x % m

inv\_elem(-5, 4)

# Китайская теорема об остатках

def chinese\_theorem(a, m):

M = 1

for num in m:

M \*= num

M\_j = []

for i in range(len(m)):

M\_j.append(M / m[i])

z = []

for i in range(len(M\_j)):

z.append((mod\_inverse(M\_j[i], m[i]) \* a[i]) % m[i])

x = 0

for i in range(len(z)):

x += M\_j[i] \* z[i]

return x % M

# Алгоритм Гарнера

def Garner\_algorithm(a, m):

c = [0] \* len(m)

for i in range(1, len(m)):

c[i] = 1

for j in range(i):

u = mod\_inverse(m[j], m[i])

c[i] = (u \* c[i]) % m[i]

u = a[0]

x = u

for i in range(1, len(m)):

u = (a[i] - x) \* c[i]

if u < 0:

u += m[i]

u %= m[i]

m\_j = 1

for j in range(i):

m\_j \*= m[j]

x += u \* m\_j

return x

# Метод Гаусса

def gauss\_jordan\_modul(matrix, mod):

n = len(matrix)

m = len(matrix[0]) - 1

row\_idx = 0

for col\_idx in range(m):

# Найдем строку с ненулевым ведущим элементом

pivot\_row = None

for i in range(row\_idx, n):

if matrix[i][col\_idx] != 0:

pivot\_row = i

break

if pivot\_row is None:

continue

# Меняем строки местами, если нужно

if pivot\_row != row\_idx:

matrix[row\_idx], matrix[pivot\_row] = matrix[pivot\_row], matrix[row\_idx]

# Приведение ведущего элемента к единице

inv = mod\_inverse(matrix[row\_idx][col\_idx], mod)

for k in range(m + 1):

matrix[row\_idx][k] = (matrix[row\_idx][k] \* inv) % mod

# Обнуление всех элементов в текущем столбце, кроме ведущего

for i in range(n):

if i != row\_idx and matrix[i][col\_idx] != 0:

factor = matrix[i][col\_idx]

for k in range(m + 1):

matrix[i][k] = (matrix[i][k] - factor \* matrix[row\_idx][k]) % mod

# Удаление нулевых строк

matrix = [row for row in matrix if any(row[:-1])]

# Увеличиваем индекс строки для следующей итерации

row\_idx += 1

# Обновляем количество строк n, так как могли удалить нулевые строки

n = len(matrix)

return matrix

# Приведение матрицы к виду для поля

def reverse\_matrix(matrix, m):

for i in range(len(matrix)):

for j in range(len(matrix[i])):

matrix[i][j] = inv\_elem(matrix[i][j], m)

return matrix

def algorithm\_interface():

def input\_numbers():

a = int(input("Введите первое число: "))

b = int(input("Введите второе число: "))

return a, b

def input\_chinese\_theorem():

a = list(map(int, input("Введите список коэффициентов a через пробел: ").split()))

m = list(map(int, input("Введите список модулей m через пробел: ").split()))

return a, m

def input\_matrix():

rows = int(input("Введите количество строк матрицы: "))

cols = int(input("Введите количество столбцов матрицы (включая правую часть): "))

matrix = []

for i in range(rows):

row = list(map(int, input(f"Введите строку {i + 1} через пробел: ").split()))

matrix.append(row)

mod = int(input("Введите модуль для расчетов: "))

return matrix, mod

def explain\_extended\_Euclid(a, b, gcd, x, y):

explanation = f"Расширенный алгоритм Евклида для чисел {a} и {b}:\nНОД({a}, {b}) = {gcd}\n"

explanation += f"Линейное разложение: {gcd} = {x} \* {a} + {y} \* {b}\n"

return explanation

def explain\_gauss(matrix, mod):

print("Решение системы методом Гаусса:")

num\_vars = len(matrix[0]) - 1

num\_rows = len(matrix)

for row in matrix:

print(" + ".join(f"{el}\*x{i + 1}" for i, el in enumerate(row[:-1])) + f" = {row[-1]}")

partial\_solution = {}

for i in range(num\_rows, num\_vars):

value = int(input(f"Введите значение для частной переменной x{i + 1}: "))

partial\_solution[f'x{i + 1}'] = value

complete\_solution = [0] \* num\_vars

for row in reversed(matrix):

lead\_idx = None

for i in range(len(row) - 1):

if row[i] != 0:

lead\_idx = i

break

if lead\_idx is None:

continue

result = row[-1]

for i in range(lead\_idx + 1, len(row) - 1):

if f'x{i + 1}' in partial\_solution:

result -= row[i] \* partial\_solution[f'x{i + 1}']

else:

result -= row[i] \* complete\_solution[i]

if mod != 0:

result = (result % mod + mod) % mod

if row[lead\_idx] != 0:

complete\_solution[lead\_idx] = result // row[lead\_idx] if mod == 0 else (result \* mod\_inverse(row[lead\_idx], mod)) % mod

for i in range(num\_rows, num\_vars):

complete\_solution[i] = partial\_solution[f'x{i + 1}']

print("Частное решение:")

for i in range(num\_vars):

print(f"x{i + 1} = {complete\_solution[i]}")

while True:

print("\nВыберите алгоритм:")

print("1. Обычный алгоритм Евклида")

print("2. Расширенный алгоритм Евклида")

print("3. Бинарный алгоритм Евклида")

print("4. Метод Гаусса")

print("5. Алгоритм Гарнера")

print("6. Китайская теорема об остатках")

print("7. Выход")

choice = int(input("Ваш выбор: "))

if choice == 1:

a, b = input\_numbers()

result = normal\_Euclid(a, b)

print(f"НОД({a}, {b}) = {result}")

elif choice == 2:

a, b = input\_numbers()

gcd, x, y = extended\_Euclid(a, b)

explanation = explain\_extended\_Euclid(a, b, gcd, x, y)

print(explanation)

elif choice == 3:

a, b = input\_numbers()

result = binary\_Euclid(a, b)

print(f"НОД({a}, {b}) с помощью бинарного алгоритма = {result}")

elif choice == 4:

matrix, mod = input\_matrix()

matrix = reverse\_matrix(matrix, mod)

result\_matrix = gauss\_jordan\_modul(matrix, mod)

explain\_gauss(result\_matrix, mod)

elif choice == 5:

a, m = input\_chinese\_theorem()

result = Garner\_algorithm(a, m)

print(f"Решение системы по алгоритму Гарнера: x = {result}")

elif choice == 6:

a, m = input\_chinese\_theorem()

result = chinese\_theorem(a, m)

print(f"Решение системы по Китайской теореме об остатках: x = {result}")

elif choice == 7:

print("Выход.")

break

else:

print("Некорректный выбор, попробуйте снова.")

algorithm\_interface()