Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №1**

**Дисциплина: Криптографические протоколы**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_М.С. Прозоров

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Крамаренко

**Цель работы:** Реализовать программный продукт решения сравнений первой степени двумя способами с указанием всех промежуточных шагов вычисления (текущее значение коэффициентов расширенном алгоритме Евклида и текущее значение степеней в формуле Эйлера), программный продукт так же должен реализовывать возможность того, что сравнение не имеет решений или имеет больше одного решения. В первом случае сообщать пользователю с пояснением, во втором строить все возможные решения.

**Ход работы:**

1) Напишем реализацию функции Эйлера по формулам:

def prime\_factors(n):

    i = 2

    factors = set()

    while i \* i <= n:

        if n % i:

            i += 1

        else:

            n //= i

            factors.add(i)

    if n > 1:

        factors.add(n)

    return factors

def euler\_phi(m):

    factors = prime\_factors(m)

    result = m

    for factor in factors:

        result \*= (1 - 1 / factor)

    return int(result)

2) Напишем реализацию расширенного алгоритма Евклида

def gcd\_extended(a, b):

if a == 0:

print(f"Текущие значения: gcd={b}, x=0, y=1, a={a}, b={b}")

return b, 0, 1

else:

gcd, x1, y1 = gcd\_extended(b % a, a)

x = y1 - (b // a) \* x1

y = x1

print(f"Текущие значения: gcd={gcd}, x={x}, y={y}, a={a}, b={b}")

return gcd, x, y

3) Напишем реализацию решения сравнений методом Эйлера:

def euler\_method(a, b, m):

g, \_, \_ = gcd\_extended(a, m)

if b % g != 0 and g != 1:

print("Сравнение не может быть решено.")

return None

phi\_m = euler\_phi(m)

print(f"Функция Эйлера (phi) для m = {m} равна {phi\_m}")

inv\_a = pow(a, phi\_m - 1, m)

x0 = (inv\_a \* b) % m

print(f"Обратный элемент a по модулю m с использованием функции Эйлера: {inv\_a}")

return x0

4) Напишем реализацию решения сравнений методом Эйлера:

def euler\_method(a, b, m):

g, \_, \_ = gcd\_extended(a, m)

if b % g != 0 and g != 1:

print("Сравнение не может быть решено.")

return None

phi\_m = euler\_phi(m)

print(f"Функция Эйлера (phi) для m = {m} равна {phi\_m}")

inv\_a = pow(a, phi\_m - 1, m)

x0 = (inv\_a \* b) % m

print(f"Обратный элемент a по модулю m с использованием функции Эйлера: {inv\_a}")

return x0

5) Напишем реализацию решения сравнений с помощью расширенного алгоритма Евклида:

def gcd\_extended\_method(a, b, m):

g, x, y = gcd\_extended(a, m)

solutions = []

if b % g != 0:

print("Сравнение не может быть решено.")

return None

x0 = (x \* (b // g)) % m

for i in range(g):

solution = (x0 + i \* (m // g)) % m

print(f"Решение: x = {solution} (для i = {i})")

solutions.append(solution)

return solutions

6) Протестируем

def modular\_linear\_equation\_solver(a, b, m):

"""Решение сравнения ax ≡ b (mod m)"""

print(f"Функция Эйлера: {euler\_method(a, b, m)}")

print(f"Расширенный алгоритма Евклида: {gcd\_extended\_method(a, b, m)}")

modular\_linear\_equation\_solver(3, 19, 34)

В результате в итоге получился программный продукт для решения сравнений первой степени двумя способами с указанием всех промежуточных шагов вычисления (текущее значение коэффициентов расширенном алгоритме Евклида и текущее значение степеней в формуле Эйлера), который так же реализовывает возможность того, что сравнение не имеет решений или имеет больше одного решения.

**Листинг программы**

def prime\_factors(n):

i = 2

factors = set()

while i \* i <= n:

if n % i:

i += 1

else:

n //= i

factors.add(i)

if n > 1:

factors.add(n)

return factors

def euler\_phi(m):

factors = prime\_factors(m)

result = m

for factor in factors:

result \*= (1 - 1 / factor)

return int(result)

def gcd\_extended(a, b):

if a == 0:

print(f"Текущие значения: gcd={b}, x=0, y=1, a={a}, b={b}")

return b, 0, 1

else:

gcd, x1, y1 = gcd\_extended(b % a, a)

x = y1 - (b // a) \* x1

y = x1

print(f"Текущие значения: gcd={gcd}, x={x}, y={y}, a={a}, b={b}")

return gcd, x, y

def euler\_method(a, b, m):

g, \_, \_ = gcd\_extended(a, m)

if b % g != 0 and g != 1:

print("Сравнение не может быть решено.")

return None

phi\_m = euler\_phi(m)

print(f"Функция Эйлера (phi) для m = {m} равна {phi\_m}")

inv\_a = pow(a, phi\_m - 1, m)

x0 = (inv\_a \* b) % m

print(f"Обратный элемент a по модулю m с использованием функции Эйлера: {inv\_a}")

return x0

def gcd\_extended\_method(a, b, m):

g, x, y = gcd\_extended(a, m)

solutions = []

if b % g != 0:

print("Сравнение не может быть решено.")

return None

x0 = (x \* (b // g)) % m

for i in range(g):

solution = (x0 + i \* (m // g)) % m

print(f"Решение: x = {solution} (для i = {i})")

solutions.append(solution)

return solutions

def modular\_linear\_equation\_solver(a, b, m):

"""Решение сравнения ax ≡ b (mod m)"""

print(f"Функция Эйлера: {euler\_method(a, b, m)}")

print(f"Расширенный алгоритма Евклида: {gcd\_extended\_method(a, b, m)}")

modular\_linear\_equation\_solver(3, 19, 34)