

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Кибербезопасность информационных систем»

Лабораторная работа №2

по дисциплине: «Теоретико-числовые методы в криптографии»

на тему: «Остаток от деления»

Выполнил обучающийся гр. ВКБ43

Ковалев Данил Петрович

          Проверила:

Ст. пр. Артамонова Е.А.

Ростов-на-Дону

2025

**Цель:** изучить всевозможные вариации алгоритма Евклида.

**Задание 1.** Найти остаток от деления, используя теорему Эйлера или малую теорему Ферма. Написать программу, реализующую нахождения остатка.

**Вариант №11**

Ручная реализация:

Изображение выглядит как текст, рукописный текст, бумага, блокнот

Содержимое, созданное искусственным интеллектом, может быть неверным.

Программная реализация:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Листинг программы.**

import math  
from typing import Iterable, Final  
  
from app.core.registry import LogRegistry  
from app.exceptions.models.remainder\_division import TheoremsCannotBeApplied  
from app.models.remainder\_division.base import IRemainderDivision  
  
  
class EulerFermatModel(IRemainderDivision):  
 def \_\_init\_\_(self) -> None:  
 self.\_registry: Final[LogRegistry] = LogRegistry()  
  
 def solve(self, a: int, exponent: int, modulus: int) -> int | None:  
 *"""  
 Вычисляет a^exponent % modulus с использованием теоремы Эйлера или Ферма.  
 Возвращает результат и лог вычислений.  
 """* # Проверка, что a и modulus взаимно просты  
 if math.gcd(a, modulus) != 1:  
 raise TheoremsCannotBeApplied("Ошибка: a и модуль не взаимно просты. Теоремы не применимы.")  
  
 # Проверка, является ли модуль простым (для малой теоремы Ферма)  
 is\_prime = self.\_is\_prime(modulus)  
 self.\_registry.add\_log(f"Модуль {modulus} {'простой' if is\_prime else 'составной'}")  
  
 if is\_prime:  
 # Малая теорема Ферма: a^(p-1) ≡ 1 mod p  
 reduced\_exponent = exponent % (modulus - 1)  
 self.\_registry.add\_log(f"Малая теорема Ферма: уменьшаем степень {exponent} до {reduced\_exponent}")  
 self.\_registry.add\_log(f"Малая теорема Ферма: формула - {a}^{reduced\_exponent} = 1 mod {modulus}")  
 else:  
 # Теорема Эйлера: a^φ(n) ≡ 1 mod n  
 phi = self.\_euler\_phi(modulus)  
 reduced\_exponent = exponent % phi  
 self.\_registry.add\_log(f"Теорема Эйлера: уменьшаем степень {exponent} до {reduced\_exponent}")  
 self.\_registry.add\_log(f"Теорема Эйлера: формула - {a}^{phi} = 1 mod {modulus}")  
  
 # Вычисляем результат  
 result = self.\_modular\_pow(a, reduced\_exponent, modulus)  
 self.\_registry.add\_log(f"Итоговое значение: {result}")  
  
 return result  
  
 @staticmethod  
 def \_is\_prime(n: int) -> bool:  
 *"""Проверяет, является ли число простым"""* if n < 2:  
 return False  
  
 for i in range(2, int(n \*\* 0.5) + 1):  
 if n % i == 0:  
 return False  
  
 return True  
  
 def \_euler\_phi(self, n: int) -> int:  
 *"""Вычисляет функцию Эйлера φ(n)"""* result = n  
 i = 2  
 temp\_n = n  
 self.\_registry.add\_log(f"φ({n}) = {n}")  
  
 while i \* i <= temp\_n:  
 if temp\_n % i == 0:  
 self.\_registry.add\_log(f"Найден множитель: {i}")  
 while temp\_n % i == 0:  
 temp\_n //= i  
 result -= result // i  
 self.\_registry.add\_log(f"φ(n) = {result}")  
 i += 1  
  
 if temp\_n > 1:  
 self.\_registry.add\_log(f"Остаточный множитель: {temp\_n}")  
 result -= result // temp\_n  
 self.\_registry.add\_log(f"φ(n) = {result}")  
  
 return result  
  
 def \_modular\_pow(self, base: int, exponent: int, modulus: int) -> int:  
 *"""Быстрое возведение в степень по модулю с логированием"""* result = 1  
 base = base % modulus  
 self.\_registry.add\_log(f"Начальные значения: base={base}, exponent={exponent}, modulus={modulus}")  
  
 while exponent > 0:  
 if exponent % 2 == 1:  
 result = (result \* base) % modulus  
 self.\_registry.add\_log(f"Нечетный показатель -> result = {result}")  
 base = (base \* base) % modulus  
 exponent = exponent // 2  
 self.\_registry.add\_log(f"exponent = {exponent}, base = {base}")  
  
 return result  
  
 def get\_logs(self) -> Iterable[str]:  
 *"""  
 Возвращает логи  
 """* return self.\_registry.logs

**Вывод по работе:** в ходе выполнения лабораторной работы были изучены лекционный материал и алгоритм поиска остатка от деления.