

# Лабораторная работа №5

Вычисление наибольшего общего делителя

Каймакджыоглу Мериц Дорук

2025-11-02

# Содержание I

1. Информация

2. Вводная часть

3. Теоретические сведения & Реализация

## Раздел 1

### 1. Информация

## 1.1 Докладчик

► **Каймакджыоглу Мериц Дорук**

## 1.1 Докладчик

- ▶ **Каймакджыоглу Мериш Дорук**
- ▶ Студент, кафедра Математического моделирования и искусственного интеллекта (ММиИИ)

## 1.1 Докладчик

- ▶ **Каймакджыоглу Мериш Дорук**
- ▶ Студент, кафедра Математического моделирования и искусственного интеллекта (ММиИИ)
- ▶ Российский университет дружбы народов

## 1.1 Докладчик

- ▶ **Каймакджыоглу Мериш Дорук**
- ▶ Студент, кафедра Математического моделирования и искусственного интеллекта (ММиИИ)
- ▶ Российский университет дружбы народов
- ▶ 1032245391@pfur.ru

## 1.1 Докладчик

- ▶ **Каймакджыоглу Мериш Дорук**
- ▶ Студент, кафедра Математического моделирования и искусственного интеллекта (ММиИИ)
- ▶ Российский университет дружбы народов
- ▶ 1032245391@pfur.ru
- ▶ <https://github.com/dorukme123>

## Раздел 2

### 2. Вводная часть

## 2.1 Актуальность

- ▶ Вероятностные тесты простоты являются фундаментальным инструментом в современной криптографии.

## 2.1 Актуальность

- ▶ Вероятностные тесты простоты являются фундаментальным инструментом в современной криптографии.
- ▶ Асимметричные криптосистемы, такие как **RSA**, требуют генерации очень больших простых чисел ( $p$  и  $q$ ).

## 2.1 Актуальность

- ▶ Вероятностные тесты простоты являются фундаментальным инструментом в современной криптографии.
- ▶ Асимметричные криптосистемы, такие как **RSA**, требуют генерации очень больших простых чисел ( $p$  и  $q$ ).
- ▶ Детерминированная проверка на простоту для чисел такого размера вычислительно невозможна.

## 2.1 Актуальность

- ▶ Вероятностные тесты простоты являются фундаментальным инструментом в современной криптографии.
- ▶ Асимметричные криптосистемы, такие как **RSA**, требуют генерации очень больших простых чисел ( $p$  и  $q$ ).
- ▶ Детерминированная проверка на простоту для чисел такого размера вычислительно невозможна.
- ▶ Быстрые вероятностные тесты (Ферма, Миллера-Рабина) — это единственный практически применимый метод для **генерации ключей RSA**, что делает их критически важными для информационной безопасности.

## 2.2 Объект и предмет исследования

- **Объект:** Вероятностные алгоритмы в вычислительной теории чисел, применяемые в криптографии.

## 2.2 Объект и предмет исследования

- ▶ **Объект:** Вероятностные алгоритмы в вычислительной теории чисел, применяемые в криптографии.
- ▶ **Предмет:** Тест Ферма, тест Соловья-Штрассена и тест Миллера-Рабина для проверки чисел на простоту.

## 2.3 Цели и задачи

- ▶ **Цель:** Изучить и программно реализовать три основных вероятностных алгоритма для проверки чисел на простоту.

## 2.3 Цели и задачи

- ▶ **Цель:** Изучить и программно реализовать три основных вероятностных алгоритма для проверки чисел на простоту.
- ▶ **Задачи:**

## 2.3 Цели и задачи

- ▶ **Цель:** Изучить и программно реализовать три основных вероятностных алгоритма для проверки чисел на простоту.
- ▶ **Задачи:**
  - ▶ Реализовать тест Ферма, основанный на Малой теореме Ферма.

## 2.3 Цели и задачи

- ▶ **Цель:** Изучить и программно реализовать три основных вероятностных алгоритма для проверки чисел на простоту.
- ▶ **Задачи:**
  - ▶ Реализовать тест Ферма, основанный на Малой теореме Ферма.
  - ▶ Реализовать алгоритм вычисления символа Якоби.

## 2.3 Цели и задачи

- ▶ **Цель:** Изучить и программно реализовать три основных вероятностных алгоритма для проверки чисел на простоту.
- ▶ **Задачи:**
  - ▶ Реализовать тест Ферма, основанный на Малой теореме Ферма.
  - ▶ Реализовать алгоритм вычисления символа Якоби.
  - ▶ Реализовать тест Соловья-Штрассена, основанный на критерии Эйлера.

## 2.3 Цели и задачи

- ▶ **Цель:** Изучить и программно реализовать три основных вероятностных алгоритма для проверки чисел на простоту.
- ▶ **Задачи:**
  - ▶ Реализовать тест Ферма, основанный на Малой теореме Ферма.
  - ▶ Реализовать алгоритм вычисления символа Якоби.
  - ▶ Реализовать тест Соловья-Штрассена, основанный на критерии Эйлера.
  - ▶ Реализовать тест Миллера-Рабина, являющийся промышленным стандартом.

## 2.3 Цели и задачи

- ▶ **Цель:** Изучить и программно реализовать три основных вероятностных алгоритма для проверки чисел на простоту.
- ▶ **Задачи:**
  - ▶ Реализовать тест Ферма, основанный на Малой теореме Ферма.
  - ▶ Реализовать алгоритм вычисления символа Якоби.
  - ▶ Реализовать тест Соловья-Штрассена, основанный на критерии Эйлера.
  - ▶ Реализовать тест Миллера-Рабина, являющийся промышленным стандартом.
  - ▶ Продемонстрировать работу алгоритмов на примерах.

## 2.4 Материалы и методы

► **Язык программирования:** Python (с использованием модульной структуры).

## 2.4 Материалы и методы

- ▶ **Язык программирования:** Python (с использованием модульной структуры).
- ▶ **Алгоритмы:** Тест Ферма, Алгоритм Якоби, Тест Соловья-Штрассена, Тест Миллера-Рабина.

## 2.4 Материалы и методы

- ▶ **Язык программирования:** Python (с использованием модульной структуры).
- ▶ **Алгоритмы:** Тест Ферма, Алгоритм Якоби, Тест Соловья-Штрассена, Тест Миллера-Рабина.
- ▶ **Математический аппарат:** Модульная арифметика, Малая теорема Ферма, критерий Эйлера, символ Якоби, свойства нетривиальных корней из 1.

## Раздел 3

### 3. Теоретические сведения & Реализация

## 3.1 Ключевые алгоритмы

### ► 1. Тест Ферма:

## 3.1 Ключевые алгоритмы

### ▶ 1. Тест Ферма:

▶ Принцип:  $a^{p-1} \equiv 1 \pmod{p}$ .

## 3.1 Ключевые алгоритмы

### ► 1. Тест Ферма:

► **Принцип:**  $a^{p-1} \equiv 1 \pmod{p}$ .

► **Проблема:** Простой, но существуют составные «числа Кармайкла», которые проходят этот тест.

## 3.1 Ключевые алгоритмы

### ▶ 1. Тест Ферма:

▶ **Принцип:**  $a^{p-1} \equiv 1 \pmod{p}$ .

▶ **Проблема:** Простой, но существуют составные «числа Кармайкла», которые проходят этот тест.

### ▶ 2. Тест Соловья-Штрассена:

## 3.1 Ключевые алгоритмы

### ▶ 1. Тест Ферма:

▶ **Принцип:**  $a^{p-1} \equiv 1 \pmod{p}$ .

▶ **Проблема:** Простой, но существуют составные «числа Кармайкла», которые проходят этот тест.

### ▶ 2. Тест Соловья-Штрассена:

▶ **Принцип:**  $a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n}$ .

## 3.1 Ключевые алгоритмы

### ▶ 1. Тест Ферма:

▶ **Принцип:**  $a^{p-1} \equiv 1 \pmod{p}$ .

▶ **Проблема:** Простой, но существуют составные «числа Кармайкла», которые проходят этот тест.

### ▶ 2. Тест Соловья-Штрассена:

▶ **Принцип:**  $a^{\frac{n-1}{2}} \equiv (\frac{a}{n}) \pmod{n}$ .

▶ **Преимущество:** Более надежен, чем тест Ферма; нет аналогов числам Кармайкла. Требуется вычисления символа Якоби  $(\frac{a}{n})$ .

## 3.1 Ключевые алгоритмы

### ▶ 1. Тест Ферма:

▶ **Принцип:**  $a^{p-1} \equiv 1 \pmod{p}$ .

▶ **Проблема:** Простой, но существуют составные «числа Кармайкла», которые проходят этот тест.

### ▶ 2. Тест Соловья-Штрассена:

▶ **Принцип:**  $a^{\frac{n-1}{2}} \equiv (\frac{a}{n}) \pmod{n}$ .

▶ **Преимущество:** Более надежен, чем тест Ферма; нет аналогов числам Кармайкла. Требуется вычисления символа Якоби  $(\frac{a}{n})$ .

### ▶ 3. Тест Миллера-Рабина:

## 3.1 Ключевые алгоритмы

### ► 1. Тест Ферма:

► **Принцип:**  $a^{p-1} \equiv 1 \pmod{p}$ .

► **Проблема:** Простой, но существуют составные «числа Кармайкла», которые проходят этот тест.

### ► 2. Тест Соловья-Штрассена:

► **Принцип:**  $a^{\frac{n-1}{2}} \equiv (\frac{a}{n}) \pmod{n}$ .

► **Преимущество:** Более надежен, чем тест Ферма; нет аналогов числам Кармайкла. Требуется вычисления символа Якоби  $(\frac{a}{n})$ .

### ► 3. Тест Миллера-Рабина:

► **Принцип:** Представляет  $n - 1 = 2^s r$  и проверяет последовательность  $a^r, a^{2r}, \dots, a^{2^{s-1}r} \pmod{n}$ .

## 3.1 Ключевые алгоритмы

### ► 1. Тест Ферма:

► **Принцип:**  $a^{p-1} \equiv 1 \pmod{p}$ .

► **Проблема:** Простой, но существуют составные «числа Кармайкла», которые проходят этот тест.

### ► 2. Тест Соловья-Штрассена:

► **Принцип:**  $a^{\frac{n-1}{2}} \equiv (\frac{a}{n}) \pmod{n}$ .

► **Преимущество:** Более надежен, чем тест Ферма; нет аналогов числам Кармайкла. Требуется вычисления символа Якоби  $(\frac{a}{n})$ .

### ► 3. Тест Миллера-Рабина:

► **Принцип:** Представляет  $n - 1 = 2^s r$  и проверяет последовательность  $a^r, a^{2r}, \dots, a^{2^{s-1}r} \pmod{n}$ .

► **Преимущество:** Самый надежный из трех. Если  $n$  — составное, тест обнаружит это с вероятностью  $\geq 3/4$  за одну итерацию. Является промышленным стандартом.

## 3.2 Демонстрация

### 1. Алгоритм, реализующий тест Ферма (fermat\_test.py)

```
import random

def fermat_test_single(n: int) -> str:
    a = random.randint(2, n-2)
    r = pow(a, n - 1, n)

    if r == 1:
        return "probably prime"
    else:
        return "composite"

def run_feramn_test(n: int, t: int) -> str:
    if n < 5:
```

## 3.2 Демонстрация

### 1. Алгоритм, реализующий тест Ферма (fermat\_test.py)

```
import random

def fermat_test_single(n: int) -> str:
    a = random.randint(2, n-2)
    r = pow(a, n - 1, n)

    if r == 1:
        return "probably prime"
    else:
        return "composite"

def run_feramn_test(n: int, t: int) -> str:
    if n < 5:
```

## 3.2 Демонстрация

### 1. Алгоритм, реализующий тест Ферма (fermat\_test.py)

```
import random

def fermat_test_single(n: int) -> str:
    a = random.randint(2, n-2)
    r = pow(a, n - 1, n)

    if r == 1:
        return "probably prime"
    else:
        return "composite"

def run_feramn_test(n: int, t: int) -> str:
    if n < 5:
```

## 3.2 Демонстрация

### 1. Алгоритм, реализующий тест Ферма (fermat\_test.py)

```
import random

def fermat_test_single(n: int) -> str:
    a = random.randint(2, n-2)
    r = pow(a, n - 1, n)

    if r == 1:
        return "probably prime"
    else:
        return "composite"

def run_feramn_test(n: int, t: int) -> str:
    if n < 5:
```

## 3.2 Демонстрация

### 1. Алгоритм, реализующий тест Ферма (fermat\_test.py)

```
import random

def fermat_test_single(n: int) -> str:
    a = random.randint(2, n-2)
    r = pow(a, n - 1, n)

    if r == 1:
        return "probably prime"
    else:
        return "composite"

def run_feramn_test(n: int, t: int) -> str:
    if n < 5:
```

## 3.3 Выводы

Задачи выполнены: Все три вероятностных теста (Ферма, Соловья-Штрассена, Миллера-Рабина) и алгоритм вычисления символа Якоби были успешно реализованы на Python.

Ключевой вывод: Практическая демонстрация (на 561) подтвердила теоретическую слабость теста Ферма и показала надежность тестов Соловья-Штрассена и Миллера-Рабина.

Навыки: Получен практический опыт реализации фундаментальных алгоритмов, лежащих в основе генерации ключей для современных асимметричных криптосистем.