

# **Лабораторная работа 6**

**Разложение чисел на множители**

Баранов Иван

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Описание задачи</b>	<b>5</b>
2.1	Каноническое разложение . . . . .	5
2.2	Задача нахождения сомножителей . . . . .	5
<b>3</b>	<b>Описание алгоритма</b>	<b>6</b>
<b>4</b>	<b>Реализация алгоритма</b>	<b>7</b>
<b>5</b>	<b>Программный код</b>	<b>8</b>
<b>6</b>	<b>Выводы</b>	<b>9</b>

# List of Figures

3.1	р-Метод Полларда . . . . .	6
4.1	Реализация алгоритма . . . . .	7

# 1 Цель работы

- Ознакомиться с задачей разложения простого числа на множители
- Рассмотреть каноническое представление числа
- Реализовать метод нахождения нетривиальных сомножителей

## 2 Описание задачи

### 2.1 Каноническое разложение

Задача разложения на множители - одна из первых задач, использованных для построения криптосистем с открытым ключом.

Задача разложения составного числа на множители формулируется следующим образом:

для данного положительного целого числа  $n$  найти его каноническое разложение  $n = p_1^{a_1} p_2^{a_2} \dots p_s^{a_s}$ , где  $p_i$  - попарно различные простые числа,  $a_i \geq 1$

### 2.2 Задача нахождения сомножителей

На практике не обязательно находить каноническое разложение числа  $n$ .

Достаточно найти его разложение на два нетривиальных сомножителя:  $n = pq$ ,  $1 \leq p \leq q < n$ .

Далее будем понимать задачу разложения именно в этом смысле. Для её решения воспользуемся р-Методом Полларда (рис. 3.1) и реализуем его посредством Python (рис. 4.1)

### 3 Описание алгоритма

*p-Метод Полларда.* Пусть  $n$  – нечетное составное число,  $S = \{0, 1, \dots, n - 1\}$  и  $f: S \rightarrow S$  – случайное отображение, обладающее сжимающими свойствами, например  $f(x) \equiv x^2 + 1 \pmod{n}$ . Основная идея метода состоит в следующем. Выбираем случайный элемент  $x_0 \in S$  и строим последовательность  $x_0, x_1, x_2, \dots$ , определяемую рекуррентным соотношением

$$x_{i+1} = f(x_i),$$

где  $i \geq 0$ , до тех пор, пока не найдем такие числа  $i, j$ , что  $i < j$  и  $x_i = x_j$ . Поскольку множество  $S$  конечно, такие индексы  $i, j$  существуют (последовательность «заиклиивается»). Последовательность  $\{x_i\}$  будет состоять из «хвоста»  $x_0, x_1, \dots, x_{i-1}$  длины  $O\left(\sqrt{\frac{\pi n}{8}}\right)$  и цикла  $x_i = x_j, x_{i+1}, \dots, x_{j-1}$  той же

Figure 3.1: p-Метод Полларда

## 4 Реализация алгоритма

```
1  import math
2  from random import randint
3
4  def pollard(n: int) -> int:
5      f = lambda x: (x**2 + 1) % n
6      c = randint(0, n-1)
7      a = c
8      b = c
9      while True:
10         a = f(a)
11         b = f(f(b))
12         d = math.gcd(a-b, n)
13
14         if 1 < d < n:
15             return d
16         elif d == n:
17             return None
18
19  for i in range(10000, 10100):
20      p = pollard(i)
21      if p: print(i, p, i//p)
22
```

Figure 4.1: Реализация алгоритма

## 5 Программный код

```
import math
from random import randint
```

```
def pollard(n: int) -> int:
    f = lambda x: (x**2 + 1) % n
    c = randint(0,n-1)
    a = c
    b = c
    while True:
        a = f(a)
        b = f(f(b))
        d = math.gcd(a-b, n)

        if 1 < d < n:
            return d
        elif d == n:
            return None
```

```
for i in range(10000, 10100):
    p = pollard(i)
    if p: print(i, p, i//p)
```



## 6 Выводы

- Ознакомились с задачей разложения простого числа на множители
- Рассмотрели каноническое представление числа
- Реализовали метод нахождения нетривиальных сомножителей