

# **Отчёт по лабораторной работе №6**

**Разложение чисел на множители**

Тимофей Сергеев

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Теоретические сведения</b>	<b>5</b>
2.1	р-алгоритм Поллрада . . . . .	6
<b>3</b>	<b>Выполнение работы</b>	<b>7</b>
3.1	Реализация алгоритма на языке Python . . . . .	7
3.2	Контрольный пример . . . . .	9
<b>4</b>	<b>Выводы</b>	<b>10</b>
	<b>Список литературы</b>	<b>11</b>

# List of Figures

3.1 Работа алгоритма . . . . . 9

# 1 Цель работы

Изучение задачи разложения на множители, изучение р-алгоритма Поллрада.

## 2 Теоретические сведения

Разложение на множители — предмет непрерывного исследования в прошлом; и такие же исследования, вероятно, продолжатся в будущем. Разложение на множители играет очень важную роль в безопасности некоторых криптосистем с открытым ключом.

Согласно Основной теореме арифметики любое положительное целое число больше единицы может быть уникально записано в следующей главной форме разложения на множители, где  $p_1, p_2, \dots, p_k$  — простые числа и  $e_1, e_2, \dots, e_k$  — положительные целые числа.

$$n = p_1^{e_1} * p_2^{e_2} * \dots * p_k^{e_k}$$

Поиск эффективных алгоритмов для разложения на множители больших составных чисел ведется давно. К сожалению, совершенный алгоритм для этого пока не найден. Хотя есть несколько алгоритмов, которые могут разложить число на множители, ни один не способен провести разложение достаточно больших чисел в разумное время. Позже мы увидим, что это хорошо для криптографии, потому что современные криптографические системы полагаются на этот факт. В этой секции мы даем несколько простых алгоритмов, которые проводят разложение составного числа. Цель состоит в том, чтобы сделать процесс разложения на множители менее трудоёмким.

В 1974 г. Джон Поллард разработал метод, который находит разложение числа  $p$  на простые числа. Метод основан на условии, что  $p-1$  не имеет сомножителя, большего, чем заранее определенное значение  $B$ , называемое границей. Алго-

ритм Полларда показывает, что в этом случае

$$p = GCD(2^{B!} - 1, n)$$

Сложность. Заметим, что этот метод требует сделать  $B-1$  операций возведения в степень  $a = a^e \bmod n$ . Есть быстрый алгоритм возведения в степень, который выполняет это за  $2 * \log_2 B$  операций. Метод также использует вычисления НОД, который требует  $n^3$  операций. Мы можем сказать, что сложность — так или иначе больше, чем  $O(B)$  или  $O(2^n)$ , где  $n_b$  — число битов в  $B$ . Другая проблема — этот алгоритм может заканчиваться сигналом об ошибке. Вероятность успеха очень мала, если  $B$  имеет значение, не очень близкое к величине  $\sqrt{n}$ .

## 2.1 р-алгоритм Полларда

- Вход. Число  $n$ , начальное значение  $c$ , функция  $f$ , обладающая сжимающими свойствами.
- Выход. Нетривиальный делитель числа  $n$ .

1. Положить  $a = c, b = c$
2. Вычислить  $a = f(a)(\bmod n), b = f(b)(\bmod n)$
3. Найти  $d = GCD(a - b, n)$
4. Если  $1 < d < n$ , то положить  $p = d$  и результат:  $p$ . При  $d = n$  результат: ДЕЛИТЕЛЬ НЕ НАЙДЕН. При  $d = 1$  вернуться на шаг 2.

## 3 Выполнение работы

### 3.1 Реализация алгоритма на языке Python

```
from math import gcd

def f(x, n):
    return (x*x+5)%n

def fu(n, a, b, d):
    a = f(a, n)
    b = f(f(b, n), n)
    d = gcd(a-b, n)
    if 1<d<n:
        print(d)
        exit()
    if d == n:
        print("not found")
    if d == 1:
        fu(n, a, b, d)

def main():
    n = 1359331
    c = 1
```

```
a = f(c, n)
b = f(a, n)
d = gcd(a-b, n)
if 1 < d < n:
    print(d)
    exit()
if d == n:
    pass
if d == 1:
    fu(n, a, b, d)
```



## 3.2 Контрольный пример

```
In [1]: 1 from math import gcd
2
3 def f(x, n):
4     return (x*x+5)%n
5
6 def fu(n, a, b, d):
7     a = f(a, n)
8     b = f(f(b, n), n)
9     d = gcd(a-b, n)
10    if 1<d<n:
11        print(d)
12        exit()
13    if d == n:
14        print("not found")
15    if d == 1:
16        fu(n, a, b, d)
17
18 def main():
19     n = 1359331
20     c = 1
21     a = f(c, n)
22     b = f(a, n)
23     d = gcd(a-b, n)
24     if 1< d < n:
25         print(d)
26         exit()
27     if d == n:
28         pass
29     if d == 1:
30         fu(n, a, b, d)
```

```
In [2]: 1 main()
```

1181

```
In [ ]: 1
```

Figure 3.1: Работа алгоритма

## 4 Выводы

Изучили задачу разложения на множители и р-алгоритм Поллрада.

# Список литературы

1. Алгоритмы тестирования на простоту и факторизации
2. Р-метод Полларда