

# **Отчёт по лабораторной работе №6**

**Разложение чисел на множители**

Милёхин Александр НПИМд-02-21

# 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 Полларда . . . . .	6
<b>3</b>	<b>Выполнение работы</b>	<b>7</b>
3.1	Реализация алгоритмов 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	Пример работы алгоритма . . . . .	9
-----	-----------------------------------	---

# 1 Цель работы

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

## 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 p-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 алгоритмов на языке Python

```
from math import gcd
```

```
ag = 1
```

```
bg = 1
```

```
def f(x, n):
```

```
    return (x*x+5)%n
```

```
def method(n, a, b, d):
```

```
    a = f(a, n)%n
```

```
    b = f(f(b,n), n)%n
```

```
    d = gcd(a-b, n)
```

```
    if 1 < d < n:
```

```
        p = d
```

```
        print(p)
```

```
        exit()
```

```
    if d == n:
```

```
        print("Делитель не найден")
```

```
    if d == 1:
```

```
        global ag
```

```

        ag = b
        method(n, a, b, d)

def main():
    n = 1359331
    c = 1
    a = c
    b = c
    a = f(a, n)%n
    b = f(a, n)%n
    d = gcd(a-b, n)
    if 1 < d < n:
        p = d
        print(p)
        exit()
    if d == n:
        pass
    if d == 1:
        method(n, a, b, d)

main()

```



## 3.2 Контрольный пример

```
print("Делитель не найден")
if d == 1:
    global ag
    ag = b
    method(n, a, b, d)

def main():
    n = 1359331
    c = 1
    a = c
    b = c
    a = f(a, n)%n
    b = f(a, n)%n
    d = gcd(a-b, n)
    if 1 < d < n:
        p = d
        print(p)
        exit()
    if d == n:
        pass
    if d == 1:
        method(n, a, b, d)
```

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

```
1181
```

Figure 3.1: Пример работы алгоритма

Таким образом, число 1181 является нетривиальным делителем числа 1359331.

## 4 Выводы

Я изучил задачу разложения на множители и р-алгоритм Полларда, а также реализовал данный алгоритм программно на языке Python.

# Список литературы

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