Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

**Лабораторная работа №5**

студента 5 курса 531 группы

специальности 10.05.01 «Компьютерная безопасность»

факультета компьютерных наук и информационных технологий

Енца Михаила Владимировича

|  |  |  |
| --- | --- | --- |
| Преподаватель  профессор | \_\_\_\_\_\_\_\_\_\_ | В. А.  Молчанов |
|  | подпись, дата |  |
| Заведующий кафедрой  д.ф.-м.н., доцент | \_\_\_\_\_\_\_\_\_\_ | М. Б. Абросимов |
|  | подпись, дата |  |

Саратов 2018

**1. Постановка задачи.**

Изучение основных методов факторизации целых чисел и их программная реализация.

**2. Теоретические сведения по рассмотренным темам с их обоснованием.**

*Определение.* Задача разложения составного числа на множители формулируется так: для данного положительного целого числа найти его каноническое разложение , где – попарно различные простые числа, .

-Метод Полларда

Пусть – нечетное составное число, и –случайное отображение, обладающее сжимающими свойствами, например . Основная идея метода состоит в следующем. Выбираем случайный элемент и строим последовательность определяемую рекуррентным соотношением , где , до тех пор, пока не найдем такие числа , что и . Поскольку множество конечно, такие индексы существуют (последовательность «зацикливается»). Последовательность будет состоять из «хвоста» длины и цикла той же длины.

Если – простой делитель числа и , то разность делится на и на . Число нам нужно найти, поэтому все вычисления в алгоритме будем проводить по модулю и на каждом шаге вычислять . Нетривиальный наибольший общий делитель (когда числа и принадлежат одному классу вычетов по модулю , но разным классам вычетов по модулю ) как раз и будет искомым делителем числа . Случай имеет место с пренебрежимо малой вероятностью.

Псевдокод -Метод Полларда.

*Вход.* Число , начальное значение , функция , обладающая сжимающими свойствами.

*Выход.* Нетривиальный делитель числа .

1. Положить .

2. Вычислить

3. Найти

4. Если , то положить и результат: . При результат: «Делитель не найден»; при вернуться на шаг 2.

(p-1)-Метод Полларда.

Пусть – нечетное составное число и – его нетривиальный делитель. (p-1)-Метод особенно эффективен при разложении таких чисел , для которых число сильно составное.

Определение.

Пусть – множество различных простых чисел. Назовем множество базой разложения. Целое число назовем -гладким, если все его простые делители являются элементами множества .

Пусть и пусть каноническое разложение числа имеет вид . Найдем максимальные показатели , для которых . Прологарифмируем обе части этого неравенства: , откуда , тогда для некоторого целого числа .

Согласно малой теореме Ферма, выполняется сравнение для любого целого , взаимно простого с . Возводя обе части этого сравнения в степень , получаем

Обозначим Если то число должно делиться на , поскольку разность делится на и число делится на .

Псевдокод (p-1)-Метод Полларда.

*Вход*. Составное число .

*Выход*. Нетривиальный делитель числа .

1. Выбрать базу разложения .

2. Выбрать случайное целое , и вычислить . При положить и результат: .

3. Для выполнить следующие действия.

3.1. Вычислить .

3.2. Положить

4. Вычислить

5. при или результат: «Делитель не найден». В противном случае положить и результат: .

Метод непрерывных дробей

Метод факторизации с помощью непрерывных дробей является оптимизацией метода Диксона, который в свою очередь основывается на методе квадратов.

Теорема Ферма о разложении: − нечетное существует взаимно однозначное соответствие между множеством делителей числа 𝑛, не меньших , и множеством пар {𝑠,𝑡} таких неотрицательных целых чисел, что .

Если , где числа 𝑝 и 𝑞 достаточно близки друг к другу, то число 𝑡 мало, а значит, 𝑠 намного больше, чем . В этом случае можно найти 𝑝 и 𝑞, последовательно перебирая числа до тех пор, пока не найдется такое 𝑠, что разность является полным квадратом, то есть равна .

Далее будем считать, что число 𝑛 не является полным квадратом.

На практике для разложения числа 𝑛 достаточно найти такие целые числа 𝑠, 𝑡, что , то есть . Если , то число , но не делит ни один из сомножителей. Значит, один делитель числа 𝑛 делит разность , а другой делитель делит сумму .

Метод Диксона

*Вход*: составное число 𝑛.

*Выход*: нетривиальный делитель 𝑝 числа 𝑛.

1. Построить базу разложения , где и - попарно различные простые числа.

2. Найти целых чисел для каждого из которых абсолютно наименьший вычет является 𝐵-гладким: ,

где , и каждому числу сопоставить вектор показателей .

3. Найти (например, методом гауссова исключения) такое непустое множество , что, для .

4. Положить .

.

5. Если , то положить и результат: 𝑝. В противном случае вернуться на шаг 3 и поменять множество 𝐾 (на практике обычно есть несколько вариантов выбора множества 𝐾 и при одной и той же базе разложения 𝐵. Если все возможности исчерпаны, то следует увеличить базу разложения).

Метод непрерывных дробей является оптимизацией метода Диксона (опубликован Дж. Брилхартом и обычно данный алгоритм называют его именем). В качестве чисел выбираются числители подходящих дробей к обыкновенной непрерывной дроби, выражающей число . Малый размер чисел обеспечивается тем, что для всех справедливо , где – 𝑘-ая подходящая дробь к числу .

Таким образом, в алгоритме Диксона в качестве можно брать числители подходящих дробей к . Кроме того, из базы 𝐵 можно исключить те простые , по модулю которых 𝑛 является квадратичным невычетом.

Формула для непрерывной дроби :

1. Положить (действительное число), .

2. Для всех имеем:

, (целочисленное деление), .

**Коды программ, реализующей рассмотренные алгоритмы**

Программа реализована на языке Python (версия интерпретатора 3.6).

*factorization.py*

import sys

import time

from math import log, sqrt, exp, gcd

import random

import functools

import sympy

import utils

import gaussian

from chain\_fractions import \*

B\_DEFAULT = (2, 3, 5)

N\_DEFAULT = 1728239

def compressor(x):

return x \* x + 1

def compressor2(x):

return x \* x + 31

def ppollard(n, c=1, f=compressor2):

a = c

b = c

while True:

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

d = gcd(a - b, n)

if 1 < d < n:

return d

elif d == n:

return

def p1pollard(n):

b = p1pollard\_base(n)

a = random.randint(2, n - 2)

d = gcd(a, n)

if d >= 2:

return d

for pi in b:

li = int(log(n) / log(pi))

a = pow(a, pow(pi, li), n)

d = gcd(a - 1, n)

if 1 < d < n:

return d

def p1pollard\_base(n):

base\_size = int(n \*\* (1 / 6))

return utils.generate\_base(min(base\_size, utils.MAX\_B\_SIZE))

def is\_b\_smooth(p, b):

alpha = []

for bi in b:

k = 0

while p % bi == 0 and not bi < 0 < p and p != 1:

p //= bi

k += 1

alpha.append(k)

return p == 1, alpha, [al % 2 for al in alpha]

def dixon(n):

base = dixon\_base(n)

base = [-1] + list(filter(lambda bi: utils.legendre(n, bi) == 1, base))

h = len(base) - 1

ps = []

alphas = []

es = []

convergent = gen\_convergent(gen\_square\_chain\_fraction(n))

while len(ps) < h + 2:

try:

pi, qi = next(convergent)

except ValueError:

return None

pi2 = pi \*\* 2 % n

if n - pi2 < pi2:

pi2 = -(n - pi2)

smooth, alpha, e = is\_b\_smooth(pi2, base)

if smooth:

ps.append(pi)

alphas.append(alpha)

es.append(e)

for ks in gaussian.gen\_gaussian(es):

s = 1

for k in ks:

s = (s \* ps[k]) % n

t = 1

for b\_idx, b in enumerate(base):

t = (t \* pow(b, functools.reduce(int.\_\_add\_\_, (alphas[k][b\_idx] for k in ks)) // 2, n)) % n

# проверка, что ks - не решение системы

assert pow(s, 2, n) == pow(t, 2, n)

if s != t and s != n - t:

p = gcd((s - t) % n, n)

return p

def dixon\_base(n):

base\_size = int(sqrt(exp(2 / 3 \* sqrt(log(n) \* log(log(n))))))

return utils.generate\_base(min(base\_size, utils.MAX\_B\_SIZE))

FUNCS = [('p-метод Полларда', ppollard),

('(p-1)-метод Полларда', p1pollard),

('Метод непрерывных дробей', dixon)]

def test\_accuracy(n):

print('Тест корректности числа {}'.format(n))

for (func\_name, func) in FUNCS:

divisor = func(n)

print(func\_name + ':', 'Делитель не найден' if divisor is None else divisor)

def test\_speed(n):

print('Тест скорости работы алгоритмов (10000 запусков) на числе {}:'.format(n))

divisors = [str(func(n)) for (func\_name, func) in FUNCS]

times = []

for (func\_name, func) in FUNCS:

start = time.time()

for \_idx in range(10000):

func(n)

times.append(time.time() - start)

print(' ' \* 10, 'p-метод Полларда ', '(p-1)-метод Полларда ', 'Метод непрерывных дробей')

print('{:>9} {:>15} {:>20} {:>23}'.format('Делитель', \*divisors))

print('{:>9} {:>13.3f} c {:>18.3f} c {:>21.3f} c'.format('Время', \*times))

if \_\_name\_\_ == '\_\_main\_\_':

arg = sys.argv[1]

test\_n, k = utils.fac2k(int(sys.argv[2]))

print(test\_n)

if sympy.isprime(test\_n):

print('{} - простое'.format(test\_n))

exit(0)

if arg == '-a':

test\_accuracy(test\_n)

elif arg == '-s':

test\_speed(test\_n)

import math

def gen\_chain\_fraction(p, q):

a = int(p / q)

yield a

while p != q:

p, q = q, p - q \* a

a = int(p / q)

yield a

def gen\_square\_chain\_fraction(n):

a0 = math.sqrt(n)

r0 = int(a0)

yield r0

ratio0 = 1

numenator0 = 0

while True:

numenator1 = r0 \* ratio0 - numenator0

ratio1 = (n - numenator1 \* numenator1) // ratio0

if ratio1 == 0:

raise ValueError('Число является полным квадратом')

r1 = int((a0 + numenator1) / ratio1)

yield r1

r0, ratio0, numenator0 = r1, ratio1, numenator1

def gen\_convergent(generator):

p0, p1 = 0, 1

q0, q1 = 1, 0

while True:

ai = next(generator)

pi = ai \* p1 + p0

qi = ai \* q1 + q0

yield pi, qi

p0, p1, q0, q1 = p1, pi, q1, qi

if \_\_name\_\_ == '\_\_main\_\_':

pass

**Оценки сложности рассмотренных алгоритмов**

Сложность алгоритма -Метод Полларда равна .

Сложность алгоритма -Метод Полларда равна .

Временная сложность алгоритма Брилхарта, равна .

**Результаты тестирования программ**

В ходе работы были рассмотрены следующие методы разложения целых числе на множители: метод непрерывных дробей, -метод Полларда, -метод Полларда. Так как в общем случае проверка числа на простоту является менее трудоемкой, чем любой из алгоритмов разложения числа на множители, то имеет смысл каждое число перед разложением на множители проверить на простоту.

Пример работы алгоритмов (анализ корректности)

|  |  |  |  |
| --- | --- | --- | --- |
|  | Делитель | | |
| Число | -Метод Полларда | -Метод Полларда | Метод непрерывных дробей |
| 63169 | 181 | 181 | Делитель не найден |
| 1001 | Число прошло тест на простоту! | | |
| 25414271907951613 | 179425457 | 141642509 | 179425457 |

Примеры работы алгоритмов (анализ эффективности)

Для сравнительно анализа алгоритмов Полларда целесообразно воспользоваться тестом, аналогичным одному из тестов модификаций алгоритма Евклида.

Для фиксированного числа оцениваем время работы алгоритмов (на 1000 запусков). При такой оценки многое зависит как от вида числа, так и от свойств выбранного отображения. В данном случае оно представляло собой функцию Эту функцию достаточно легко вычислить. Очевидно, такая реализация p-метода Полларда будет выигрывать по сравнению с реализацией -метода хотя бы потому, что последняя требует (помимо операций умножения и деления) вычисления экспоненты и трехкратного вычисления логарифма на каждой итерации. Действительно, -метод Полларда превосходит -метод более чем в 10 раз для числа любого вида.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | -метод Полларда | | -Метод Полларда | |
| Число | Делитель | Время | Делитель | Время |
| 11021 | 103 | 0.264 | Не найден | 0.523 |
| 63169 | 181 | 0.782 | 181 | 1.103 |

Несмотря на эффективность, -метод Полларда имеет существенный недостаток: если случайное отображение фиксировано, то для некоторых чисел он не работает. В этом случае требуется поменять отображение или воспользоваться иным методом разложения на множители.