МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

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

ФАКТОРИЗАЦИЯ ЦЕЛЫХ ЧИСЕЛ

ЛАБОРАТОРНАЯ РАБОТА

студента 5 курса 531 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Стаина Романа Игоревича

Проверил
профессор ______________ В. А. Молчанов

Саратов 2023

1 Постановка задачи

Цель работы – изучение основных методов факторизации целых чисел и их программная реализация.

Порядок выполнения работы

1. Рассмотреть ρ-метод Полларда разложения целых чисел на множители и привести его программную реализацию.
2. Рассмотреть $(p-1)$-метод Полларда разложения целых чисел на множители и привести его программную реализацию
3. Рассмотреть метод цепных дробей разложения целых чисел на множители и привести его программную реализацию.

2 Методы факторизации целых чисел

2.1 ρ-метод Полларда

Это вероятностный алгоритм факторизации целых чисел. С помощью сжимающего отображения $f : Z_n \to Z_n$ (например, многочлена) строится реккурентная последовательность $x_{i+1} = f(x_i) \pmod n$ со случайным начальным условием $x_0 \in Z_n$ и проверяется

$$1 < \text{НОД}(x_i - x_k, n) < n.$$

Так как составное число n имеет простой делитель $p < \sqrt{n}$, то последовательность $\{x_i\}$ имеет период $\leq n$ и последовательность $\{x_i \pmod p\}$ имеет период $\leq p$. Значит, с большой вероятностью найдутся такие значения последовательности x_i, x_k, для которых

$$x_i \equiv x_k \pmod p, x_i \not\equiv x_k \pmod n$$

и, значит, $1 < \text{НОД}(x_i - x_k, n) < n$.

2.1.1 Теорема («парадокс дней рождения»)

Пусть $\lambda > 0$ и $k = \left\lceil \sqrt{2\lambda n} \right\rceil$. Для случайной выборки объёма $k + 1$ из n элементов вероятность $P_{n,k}$ того, что все элементы выборки попарно различны, удовлетворяет условию $P_{n,k} < e^{-\lambda}$.

Значит, для собственного делителя $p < \sqrt{n}$, $\lambda = \ln \frac{1}{\varepsilon}$ и значения $k = \left\lceil \sqrt{2p \ln \frac{1}{\varepsilon}} \right\rceil$ в последовательности $x_i \pmod p, 1 \leq i \leq k + 1$ с вероятностью не менее $1 - e^{-\lambda} = 1 - \varepsilon$ найдутся одинаковые члены.

Таким образом, число шагов алгоритма можно ограничить значением $T = \left\lceil \sqrt{2\sqrt{n} \ln \frac{1}{\varepsilon}} \right\rceil + 1$ и получаем экспоненциальную общую сложность вычислений

$$O(k^2 \log^2 n) = O(\sqrt{n} \ln \frac{1}{\varepsilon} \log^2 n).$$

2.1.2 Замечание

Алгоритм значительно ускоряется, если для $2^h \leq i \leq 2^{h+1}$ вычислять $d_k = \text{НОД}(x_{i+1} - x_k, n)$ для $k = 2^{h-1}$. Получаем экспоненциальную общую сложность вычислений $O(\sqrt[4]{n}\ln\frac{1}{\varepsilon}\log^2 n)$.

2.1.3 Алгоритм факторизации целых чисел ρ-методом Полларда

Вход. Составное число n и значение $0 < \varepsilon < 1$.

Выход. Нетривиальный делитель d числа n, $1 < d < n$ с вероятностью не менее $1 - \varepsilon$.

1. Вычислить $T = \left\lceil \sqrt{2\sqrt{n}\ln\frac{1}{\varepsilon}} \right\rceil + 1$
2. Случайно выбрать $x_0 \in Z_n$ и, вычисляя последовательно элементы $x_{i+1} \equiv f(x_i) \pmod{n}$, где $f(x) = x^2 + 1$, $0 \le i \le T$, выполнить тест на шаге 3.
3. Если $2^h \le i \le 2^{h+1}$, то для $k = 2^h - 1$ найти $d = \text{НОД}(x_i - x_k, n)$.
   — Если $1 < d < n$, то найден искомый делитель.
   — Если $d = n$, то выбрать новое случайное $x_0 \in Z_n$.
   — Если вычислено T членов последовательности, то делитель не найден, алгоритм останавливается.

2.2 $(p-1)$-метод Полларда

Пусть n – составное число. Фиксируется параметр метода – число $B > 0$.

Будем называеть B-гладкими числами, у которых все простые множители не превосходят B.

Рассматривается множество простых чисел $\{q_1, \ldots, q_{\pi(B)}\}$ – факторная база и значения

$$k_i = \left[\frac{\ln n}{\ln q_i}\right] (\text{ чтобы } q_i^{k_i} \le n), T = \prod_{i=1}^{\pi(B)} q_i^{k_i}.$$

2.2.1 Алгоритм факторизации целых чисел $(p-1)$-методом Полларда

Вход. Составное число n, число $B > 0$ и значение $T = \prod_{i=1}^{\pi(B)} q_i^{k_i}$.

Выход. Нетривиальный делитель d числа n.

1. Случайно выбрать $a \in Z_n$ и вычислить $d = \text{НОД}(a, n)$. Если $1 < d < n$, то найден нетривиальный делитель d числа n. Если $d = 1$, то вычислить $b \equiv a^T - 1 \pmod{n}$.
2. Вычислить $n_1 = \text{НОД}(b, n)$.
   — Если $1 < n_1 < n$, то найден искомый делитель.

— Если $n_1 = 1$, то увеличить B.

— Если $n_1 = n$, то перейти к шагу 1 и выбрать новое $a \in Z_n$. Если для нескольких значений a выполняется $n_1 = n$, то уменьшить B.

В реализации данной работы число B выбирается случайно из интервала $(\left\lfloor \frac{\sqrt{n}}{10} + 1 \right\rfloor, \lfloor\sqrt{n}\rfloor)$. B уменьшается или увеличивается в 2 раза. Уменьшается B, если для $\lfloor \log_2 B \rfloor$ значений a выполняется $n_1 = n$.

2.2.2 Обоснование алгоритма

Если p – простой делитель числа n, то условие $p|\text{НОД}(b, n)$ равносильно $a^T \equiv 1 \pmod p$, и, значит

$$(p-1)|T,\ p-1 = \prod_{i=1}^{\pi(B)} q_i^{l_i}$$

для некоторых $l_i \leq k_i$, что равносильно B-гладкости числа $p-1$.

Поэтому в случае, когда для всех простых делителей p числа n число $p-1$ не является B-гладким, для любого $a \in Z_n$ выполняется НОД(b, n) = 1 и необходимо увеличить B. Если же для всех простых делителей p числа n число $p-1$ является B-гладким, то для любого $a \in Z_n$ может получиться $\text{НОД}(b, n) = n$ и необходимо уменьшить B. Значит, в случае, когда среди простых числа n есть как делители p с значением $p-1$ B-гладким, так и делители p с значением $p-1$ не B-гладким, алгоритм найдёт нетривиальный делитель числа n.

Сложность вычисления $a^T \equiv 1 \pmod n$ равна $O(\log T) = O(\pi(B) \log n)$, сложность вычисления $\text{НОД}(b, n)$ равна $O(\log^2 n)$ и общая сложность алгоритма равна $O(\pi(B) \log^3 n)$. Сложность алгоритма при малых B полиномиальна и при $B \approx \sqrt{n}$ экспоненциальная.

2.3 Субэкспоненциальные алгоритмы факторизации

Пусть n – составное число, которое не имеет небольших простых делителей.

Общая идея Лагранжа: найти решения сравнения $x^2 \equiv y^2 \pmod n$, удовлетворяющее условию $x \not\equiv \pm y \pmod n$, и, значит,

$$(x-y)(x+y) \equiv 0 \pmod n$$

влечёт, что один делитель p числа n делит $x-y$ и другой делитель q числа n делит $x+y$. Для этого проверяются два условия $1 <$НОД$(x-y,n)$, $1 <$НОД$(x+y,n)$.

2.4 Алгоритм Диксона

Пусть $0 < a < 1$ – некоторый параметр и B – факторная база всех простых чисел, не превосходящих L^a, $k = \pi(L^a)$.

$Q(m) \equiv m^2 \pmod n$ – наименьший неотрицательный вычет числа m^2.

1. Случайным выбором ищем $k+1$ чисел $m_1, \ldots, m_{k+1}$ для которых $Q(m_i) = p_1^{\alpha_{i1}} \ldots p_k^{\alpha_{ik}}$, обозначаем $\overline{v_l} = (\alpha_{i1}, \ldots, \alpha_{i2})$.
2. Найти ненулевое решение $(x_1, \ldots, x_{k+1}) \in \{0,1\}^{k+1}$ системы k линейных уравнений с $k+1$ неизвестными

$$x_1\overline{v_1} + \cdots + x_{k+1}\overline{v_{k+1}} = \overline{0} \pmod 2$$

3. Положить

$$X \equiv m_1^{x_1} \ldots m_{k+1}^{x_{k+1}} \pmod n, \; Y \equiv \prod_{j=1}^{k} p_j^{\frac{\sum x_i \alpha_{ij}}{2}} \pmod n,$$

для которых

$$X^2 \equiv p_1^{\sum_{i=1}^{k+1} x_1 \alpha_{i1}} \ldots p_k^{\sum_{i=1}^{k+1} x_1 \alpha_{ik}} \equiv Y^2 \pmod n$$

Проверить условие $1 <$НОД$(X \pm Y, n) < n$. Если выполняется, то получаем собственный делитель числа n (с вероятностью успеха $P_0 \geq \frac{1}{2}$). В противном случае возвращаемся на шаг 1 и выбираем другие значения $m_1, \ldots, m_{k+1}$.

Сложность алгоритма минимальна при $a = \frac{1}{\sqrt{2}}$ и равна $L_n\left[\frac{1}{2}, \sqrt{2}\right] = O(e^{\sqrt{2\log n \log\log n}})$.

2.5 Метод Моррисона и Бриллхарта

Отличается от алгоритма Диксона только способом выбора значений $m_1, \ldots, m_{k+1}$ на шаге 1: случайный выбор заменяется детерминированным определением этих значений с помощью подходящих дробей для представления числа $\sqrt{n}$ цепной дробью.

2.5.1 Теорема

Пусть $n \in \mathbb{N}, n > 16, \sqrt{n} \notin \mathbb{N}$ и $\frac{P_i}{Q_i}$ – подходящая дробь для представления числа $\sqrt{n}$ цепной дробью. Тогда абсолютно наименьший вычет числа $P_i^2 \pmod{n}$ равен значению $P_i^2 - nQ_i^2$ и выполняется

$$|P_i^2 - nQ_i^2| < 2\sqrt{n}.$$

Разложение числа $\sqrt{n}$ в цепную дробь с помощью только операции с целыми числами и нахождения целой части чисел вида $\frac{\sqrt{D-u}}{v}$ быть найдено по следующей теореме.

2.5.2 Теорема

Пусть α – квадратичная иррациональность вида $\alpha = \frac{\sqrt{D-u}}{v}$, где $D \in \mathbb{N}, \sqrt{D} \notin \mathbb{N}, v \in \mathbb{N}, u \in \mathbb{N}, v|(D-u^2)$. Тогда для любого $k \geq 0$ справедливо разложение в бесконечную цепную дробь $\alpha = [a_0, a_1, \ldots, a_k, a_{k+1}, \ldots]$, где $a_0 \in \mathbb{Z}, a_1, \ldots, a_k, \ldots inN$. При этом справедливы соотношения

$$a_0 = [\alpha], v_0 = v, u_0 = u + a_0 v$$

и при $k \geq 0$

$$a_{k+1} = [\alpha_{k+1}], \text{ где } v_{k+1} = \frac{D - u_k^2}{v_k} \in \mathbb{Z}, v_{k+1} \neq 0,$$

$$a_{k+1} = \frac{\sqrt{D} + u_k}{v_{k+1}} > 1$$

и числа u_k получаются с помощью реккурентной формулы $u_{k+1} = a_{k+1} v_{k+1} - u_k$.

Таким образом, в алгоритме Диксона возможен выбор

$$m_i = P_i, Q(m_i) = m_i^2 = P_i^2 \equiv P_i^2 - nQ_i^2 \pmod{n}, Q(m_i) = P_i^2 - nQ_i^2$$

и факторная база сужается

$$B = \{p_0 = -1\} \cup \{p \text{ – простое число: } p \leq L^a \text{ и } n \in QR_p\},$$

так как $p|Q(m_i) = P_i^2 - nQ_i^2$ влечёт

$$P_i^2 - nQ_i^2 \equiv 0 \pmod p, P_i^2 \equiv nQ_i^2 \pmod p$$

и в силу НОД$(P_i, Q_i) = 1$ выполняется: $p \nmid P_i, p \nmid Q_i$, НОД$(p, Q_i) = 1$, существует Q_i^{-1} в группе $\mathbb{Z}_p^*$ и $n \equiv (P_i Q_i^{-1})^2 \pmod p$, $\left(\frac{n}{p}\right) = 1$, т.е. $n \in QR_p$.

При этом $|Q(m_i)| = |P_i^2 - nQ_i^2| < 2\sqrt{n}$ – повышает вероятность B-гладкости значения $Q(m_i)$.

2.5.3 Алгоритм Бриллхарта-Моррисона

Вход. Составное число n.

Выход. Нетривиальный делитель d числа n.

1. Построить базу разложения $B = \{p_0, p_1, \ldots, p_k\}$, где $p_0 = -1$ и $p_1, \ldots, p_k$ – попарно различные простые числа, по модулю которых n является квадратичным вычетом.
2. Берутся целые числа m_i, являющиеся числителями подходящих дробей к обыкновенной непрерывной дроби, выражающей число $\sqrt{n}$. Из этих чисел выбираются $k+1$, для которых абсолютно наименьшие вычеты m_i^2 по модулю n являются B-гладкими:

$$m_i^2 \mod n = \prod_{j=0}^{k} p_j^{\alpha_{ij}} = v_i,$$

где $\alpha_{ij} \geq 0$.

3. Найти ненулевое решение $(x_1, \ldots, x_{k+1}) \in \{0,1\}^{k+1}$ системы k линейных уравнений с $k+1$ неизвестными

$$x_1\overline{v_1} + \cdots + x_{k+1}\overline{v_{k+1}} = \overline{0} \pmod 2$$

4. Положить

$$X \equiv m_1^{x_1} \ldots m_{k+1}^{x_{k+1}} \pmod n,\; Y \equiv \prod_{j=1}^{k} p_j^{\frac{\sum x_i \alpha_{ij}}{2}} \pmod n,$$

для которых

$$X^2 \equiv p_1^{\sum_{i=1}^{k+1} x_1 \alpha_{i1}} \ldots p_k^{\sum_{i=1}^{k+1} x_1 \alpha_{ik}} \equiv Y^2 \pmod n$$

5. Если $X \neq Y \pmod{n}$, то $d = $ НОД$(X - Y, n)$ и результат d.
   В противном случае вернуть на шаг 3 и поменять множество $(x_1, \ldots, x_{k+1})$.
   Если все возможности этого выбора исчерпаны, то увеличить размер факторной базы B.
   Сложность алгоритма $O(e^{\sqrt{2 \log n \log \log n}})$.

3 Тестирование

```
Введите число 35
Факторизация rho-методом Полларда [5, 7]
0.0 секунд.

Факторизация (p-1)-методом Полларда [5, 7]
0.0 секунд.

Факторизация алгоритмом Бриллхарта-Моррисона [5, 7]
0.0 секунд.
```

Рисунок 1

```
Введите число 1557697
Факторизация rho-методом Полларда [1201, 1297]
0.0 секунд.

Факторизация (p-1)-методом Полларда [1297, 1201]
0.004986524581909l8 секунд.

Факторизация алгоритмом Бриллхарта-Моррисона [1297, 1201]
0.057845354080200195 секунд.
```

Рисунок 2

```
Введите число 21299881
Факторизация rho-методом Полларда [3851, 5531]
0.000997781753540039 секунд.

Факторизация (p-1)-методом Полларда [3851, 5531]
0.006003618240356445 секунд.

Факторизация алгоритмом Бриллхарта-Моррисона [5531, 3851]
0.003989696502685547 секунд.
```

Рисунок 3

```
Введите число 3865489
Факторизация rho-методом Полларда [1907, 2027]
0.0 секунд.

Факторизация (p-1)-методом Полларда [1907, 2027]
0.007975101470947266 секунд.

Факторизация алгоритмом Бриллхарта-Моррисона [2027, 1907]
0.03293943405151367 секунд.
```

Рисунок 4

```
Введите число 32880121853
Факторизация rho-методом Полларда [104999, 313147]
0.0019941329956054688 секунд.

Факторизация (p-1)-методом Полларда [104999, 313147]
1.1389813423156738 секунд.

Факторизация алгоритмом Бриллхарта-Моррисона [313147, 104999]
0.02191305160522461 секунд.
```

Рисунок 5

4 Выводы по работе

В ходе работы были рассмотрены теоретические сведения о таких алгоритмах факторизации целых чисел, как ρ и $(p-1)$ методы Полларда, алгоритм Диксона и алгоритм Бриллхарта-Моррисона (или метод непрерывных дробей). Была разработана программа на языке Python, в которой были реализованы ρ и $(p-1)$ методы Полларда и алгоритм Бриллхарта-Моррисона.

ПРИЛОЖЕНИЕ А

Листинг main.py

```
from random import randint
from math import gcd, log, floor, sqrt, ceil
from sympy import isprime, sieve, legendre_symbol
from itertools import combinations
from numpy import sum as npsum
import time


def pollards_rho(n, eps):
    if n & 1 == 0:
        return 2
    t = 4 * floor(sqrt(2 * sqrt(n) * log(1 / eps))) + 1

    def f(x):
        return (x * x + 1) % n

    xi, xk = randint(1, n - 1), 1
    i, k = 0, 2
    for _ in range(t):
        if gcd(n, abs(xi - xk)) == n:
            xi, xk = randint(1, n - 1), 1
            i, k = 0, 2
        if i == k:
            xk = xi
            k *= 2
        xi = f(xi)
        i += 1
        if gcd(n, abs(xi - xk)) != 1:
            return gcd(n, abs(xi - xk))

    return n


def p_rho_factorization(n, eps):
    primes = [2, 3, 5, 7, 11, 13, 17, 19, 23]
    factors = []
    for p in primes:
        if n % p == 0:
            n //= p
```

```
                factors.append(p)
        old_n = n
        while n != 1:
            d = pollards_rho(n, eps)
            if d == old_n:
                print(f'Число {n}, вероятно, простое.')
                return [1, n]
            n //= d
            factors.append(d)
        return factors


    def sieve_of_erato(n):
        a = [i for i in range(n + 1)]

        a[1] = 0
        i = 2
        while i * i <= n:
            if a[i] != 0:
                j = i ** 2
                while j <= n:
                    a[j] = 0
                    j = j + i
            i += 1

        a = [i for i in a if i != 0]
        return a


    def gcd_extended(num1, num2):
        if num1 == 0:
            return num2, 0, 1
        else:
            div, x, y = gcd_extended(num2 % num1, num1)
        return div, y - (num2 // num1) * x, x


    def congruence_relation(a, b, m):
        d, u, v = gcd_extended(a, m)
        if d > 1:
            if b % d != 0:
```

```
            print('Нет решений сравнения.')
            return None
        else:
            b //= d
            m //= d

    x = int(u * b % m)
    return x


def pollards_p1(n, t, big_b):
    for _ in range(floor(log(big_b))):
        a = randint(2, n - 2)
        d = gcd(a, n)
        if d == 1:
            b = congruence_relation(1, pow(a, t, n) - 1, n)
        else:
            return 1, d
        n1 = gcd(b, n)
        if n1 == n:
            continue
        if n1 > 1:
            return 1, n1
        if n1 == 1:
            return -1, None
    return 0, None


def call_pollards_p1(n):
    big_b = randint(floor(sqrt(n)) // 10 + 1, floor(sqrt(n)))
    qs, t = list(sieve.primerange(big_b)), 1

    for qi in qs:
        t *= pow(qi, int(log(n) // log(qi)))
    result, d = pollards_p1(n, t, big_b)

    while d is None:
        if result == 0:
            big_b //= 2
        if result == -1:
            big_b *= 2
```

```
        qs, t = list(sieve.primerange(big_b)), 1
        for qi in qs:
            t *= pow(qi, int(log(n) // log(qi)))
        result, d = pollards_p1(n, t, big_b)

    return d


def pp1_factorization(n):
    primes = [2, 3, 5, 7, 11, 13, 17, 19, 23]
    factors = []
    for p in primes:
        if n % p == 0:
            n //= p
            factors.append(p)

    while n != 1:
        d = call_pollards_p1(n)
        n //= d
        factors.append(d)
        if isprime(n):
            factors.append(n)
            return factors
    return factors


def successive_convergents_sqrt(n):
    qs = [0, 0] + continued_fraction_sqrt(n)[0:25]
    big_ps = [0, 1]
    big_qs = [1, 0]
    for i in range(2, len(qs)):
        big_ps.append(big_ps[i - 1] * qs[i] + big_ps[i - 2])
        big_qs.append(big_qs[i - 1] * qs[i] + big_qs[i - 2])
    return big_ps[2:], big_qs[2:]


def continued_fraction_sqrt(n):
    a0 = int(sqrt(n))
    coefficients = [a0]
    if a0 * a0 == n:
```

```
        return coefficients
    m = 0
    d = 1
    a = a0
    while a != 2 * a0:
        m = d * a - m
        d = (n - m * m) // d
        a = (a0 + m) // d
        coefficients.append(a)
    return coefficients


def combs(matr):
    arr = []
    for k in range(2, len(matr) + 1):
        rows = combinations(matr, k)
        indices = list(combinations(range(len(matr)), k))
        for i, j in enumerate(rows):
            if sum(npsum(j, axis=0) % 2) == 0:
                arr.append(indices[i])
    return arr


def call_continued_fraction_factorization(n):
    sieve = 20
    while True:
        primes = sieve_of_erato(sieve)[1:]
        b = [-1, 2]
        for p in primes:
            if legendre_symbol(n, p) == 1:
                b.append(p)
        k = len(b)
        ps, qs = successive_convergents_sqrt(n)
        vs, ms = [], []

        for i in range(len(ps)):
            p = ps[i] * ps[i] - n * qs[i] * qs[i]
            p2 = p
            vi = [0] * k
            if abs(p2) // p2 == -1:
                vi[0] = 1
```

```
                p2 = -p2
            for j in range(1, k):
                while p2 % b[j] == 0:
                    vi[j] += 1
                    p2 //= b[j]
            if p2 <= 19:
                vs.append(vi)
                ms.append((ps[i], p))

        vs_len = len(vs)
        if not vs_len:
            sieve *= 2
            continue
        # print('vs', vs)
        # print('ms', ms)
        for c in combs(vs):
            x = [0] * vs_len
            for i in c:
                x[i] = 1

            big_x, big_y = 1, 1
            for i in range(vs_len):
                if x[i]:
                    big_x *= ms[i][0] % n

            big_x %= n
            ds = [0] * k
            for i in range(0, vs_len):
                if x[i]:
                    ds = list(map(sum, zip(ds, vs[i])))
            for i in range(1, k):
                big_y *= pow(b[i], ds[i] // 2, n)
            big_y %= n

            xy = (big_x - big_y) % n
            if xy != 0:
                d = gcd(xy, n)
                if d > 1:
                    return d
        sieve *= 2
        # return n
```

```


def cf_factorization(n):
    primes = [2, 3, 5, 7, 11, 13, 17, 19, 23]
    factors = []
    for p in primes:
        if n % p == 0:
            n //= p
            factors.append(p)

    while n != 1:
        d = call_continued_fraction_factorization(n)
        n //= d
        factors.append(d)
        if isprime(n):
            factors.append(n)
            return factors
    return factors


# 1557697, 21299881, 3865489, 32880121853, 15053151547, 991258173307575289
if __name__ == '__main__':
    n = int(input('Введите число '))

    time0 = time.time()
    print('Факторизация rho-методом Полларда', p_rho_factorization(n, 0.01))
    print(f'{time.time() - time0} секунд.\n')

    time0 = time.time()
    print('Факторизация (p-1)-методом Полларда', pp1_factorization(n))
    print(f'{time.time() - time0} секунд.\n')

    time0 = time.time()
    print('Факторизация алгоритмом Бриллхарта-Моррисона', cf_factorization(n))
    print(f'{time.time() - time0} секунд.\n')


```