МИНОБРНАУКИ РОССИИ
ФГБОУ ВО «СГУ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

ФАКТОРИЗАЦИЯ ЦЕЛЫХ ЧИСЕЛ

ЛАБОРАТОРНАЯ РАБОТА

студента 5 курса 531 группы
направления 100501 — Компьютерная безопасность
факультета КНиИТ
Улитина Ивана Владимировича

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 $(\rho - 1)$-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 Теоретические сведения

2.1 ρ-метод Полларда

Это вероятностный алгоритм факторизации целых чисел, с помощью которого разложено число $F_8 = 2^{2^8} + 1$.

С помощью случайного сжимающего отображения $f : Z_n \to Z_n$ (например, многочлена) строится рекуррентная 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

$$x_i \equiv x_k \pmod p,\ x_i \neq x_k \pmod n$$

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

Графически члены последовательности $\{x_i\}$ изображаются так, что сначала образуется конечный "хвост а затем - цикл конечной длины $\leq p$. Из-за такой фигуры метод называется ρ-методом.

Теорема ("парадокс дней рождения"). Пусть $\lambda > 0$ и $k = \left\lceil \sqrt{2\lambda n} \right\rceil$. Для случайной выборки объема $k+1$ 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 $T = \left[\sqrt{2\sqrt{n}\ln\frac{1}{\varepsilon}} \right] + 1$ и получаем экспоненциальную общую сложность вычислений

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

2.2 $(\rho - 1)$-метод Полларда

Пусть n - составное число. Фиксируется параметр метода - число $B > 0$, (для больших чисел n, как правило, $10^5 < B \leq \sqrt{n}$).

Будем называть B-гладкими числа, у которых все простые множители не превосходят B.

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

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

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 $p-1$. Действительно, если число $p-1$ B-гладкое, то $p-1 = \prod_{i=1}^{\pi(B)} q_i^{l_i}$ и в силу $p-1 < n$ для любого $i = 1, \ldots, \pi(B)$ выполняется

$$q_i^{l_i} \leq p-1 < n < q_i^{k_i+1}, l_i \leq k_i.$$

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 $\text{НОД}(b, n) = 1$ 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 найдет нетривиальный делитель числа n.

2.3 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.

Шаг 1. Случайным выбором ищем $k+1$ чисел $m_1, \ldots, m_{k+1}$, для которых

$Q(m_i) = p_1^{\alpha_{i1}} \dots p_k^{\alpha_{ik}}$, обозначаем $\overline{v_l} = (\alpha_{i1}, \dots, \alpha_{ik})$.
Шаг 2. Найти ненулевое решение $(x_1, \dots, x_{k+1}) \in \{0,1\}^{k+1}$ системы k линейных уравнений с $k+1$ неизвестными

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

Шаг 3. Положить

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

для которых

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

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 $m_1, \dots, m_{k+1}$.

2.4 Алгоритм Бриллхарта-Моррисона

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

Теорема. Пусть $n \in N, n > 16, \sqrt{n} \notin N$ и $\frac{P_i}{Q_i}$ - подходящая дробь ждя 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 $\frac{\sqrt{D}-u}{v}$ может быть найдено по следующей теореме.

Теорема. Пусть α - квадратичная иррациональность вида $\alpha = \frac{\sqrt{D}-u}{v}$, где $D \in N, \sqrt{D} \notin N, v, u \in N, v|(D-u^2)$. 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 $a_0 \in Z, a_1, \ldots, a_k, \dots \in N$. При этом справедливы соотношения

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

и при $k \geq 0$.

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

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

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

Таким образом, в алгоритме возможен выбор $m_i = P_i, Q(m_i) \equiv m_i^2 = P_i^2 \equiv P_i^2 - nQ_i^2 \pmod{n},\ Q(m_i) = P_i^2 - nQ_i^2$ 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 $p|Q(m_i) = P_i^2 - nQ_i^2$ влечет

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

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 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 $Q(m_i)$.

3 Результаты работы

3.1 Описание и псевдокод алгоритмов факторизации целых чисел

Алгоритм 1 - ρ-метод Полларда разложения целых чисел

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

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

Шаг 1. Вычислить $T = \left[\sqrt{2\sqrt{n}\ln\frac{1}{\varepsilon}}\right] + 1$ и выбрать случайный многочлен $f \in Z_n[x]$ (например, $f(x) = x^2 + 1$).

Шаг 2. Случайно выбрать $x_0 \in Z_n$ и, последовательно вычисляя значения $x_{i+1} = f(x_i) \pmod n, 0 \le i \le T$, проверять тест на шаге 3.

Шаг 3. Для каждого $0 \le k \le i$ вычислить $d_k =$ НОД$(x_{i+1} - x_k, n)$ и проверить условие $1 < d_k < n$. Если это выполняется, то найден нетривиальный делитель d_k числа n. Если же $d_k = 1$ для всех $0 \le k \le i$, то перейти к выбору следующего значения последовательности на шаге 2. Если найдется $d_k = n$ для некоторого $0 \le k \le i$, то перейти к выбору нового значения $x_0 \in Z_n$ на шаге 2.

Псевдокод:

```
Функция Полларда_Ро(число, эпсилон):
    Если n == 1:
        Вернуть 1
    Если n четное:
        Вернуть 2

    Пусть T = вычислить_T_(эпсилон)
    Пусть rng - генератор случайных чисел
    Пусть x = случайное_Число_Из_Диапазона(2..n)
    Пусть y = x
    Пусть d = 1

    Функция f(x):
        Вернуть (x * x + 1) % n

    Пока d == 1 и количество_повторений < T:
        x = f(x)
        y = f(f(y))
        Если x > y:
            d = НОД(x - y, n)
```

```
        Иначе:
            d = НОД(y - x, n)
        T = T - 1

    Вернуть d
```

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

Алгоритм 2 - $(\rho - 1)$-метод Полларда разложения целых чисел

Вход: Составное число n, число $B > 0$.

Выход: Разложение числа n на нетривиальные делители.

Шаг 1. Случайно выбрать $a \in Z_n$ и вычислить $d =$ НОД(a, n). Если $1 < d < n$, то найден нетривиальный делитель d числа n. Если $d = 1$, то вычислить $b \equiv a^B - 1 \pmod{n}$.

Шаг 2. Вычислить $n_1 =$ НОД(b, n). Если $1 < n_1 < n$, то найден нетривиальный делитель n_1 числа n. Если $n_1 = 1$, то увеличить B. Если $n_1 = n$, то перейти к шагу 1 и выбрать новое значение $a \in Z_n$. Если для нескольких значений $a \in Z_n$ выполняется $n_1 = n$, то уменьшить B.

Псевдокод:

```
функция Полларда_Ро_минус_1(число, база):
    Пусть rng - генератор случайных чисел
    Пусть a = случайное_Число_Из_Диапазона(2..n)

    Пусть power = база
    Пусть d = 1
    Пусть верхняя_Граница = 18446744073709551615 / 1000

    Пока d == 1:
        power *= 2
        Пусть a_powered = в_степень_по_модулю(a, power, n)
        d = НОД(a_powered - 1, n)

        Если power > верхняяГраница:
            Если база == 2:
                Вернуть n
            Иначе:
                база = база - 1
                power = база
```

```
        Если d != 1:
            Прервать цикл

    Вернуть d
```

Трудоемкость алгоритма $O(\pi(B)\log^3 n)$.

Алгоритм 3 - метод цепных дробей разложения целых чисел

Вход: Составное число m.

Выход: Нетривиальный делитель p числа m.

Шаг 1. Построить базу разложения $B = \{p_0, p_1, \ldots, p_h\}$, где $p_0 = -1$ и $p_1, \ldots, p_h$ - попарно различные простые числа, по модулю которых m является квадратичным вычетом.

Шаг 2. Берутся целые числа u_i, являющиеся числителями подходящих дробей к обыкновенной непрерывной дроби, выражающей число $\sqrt{m}$. Из этих числителей выбираются $h+2$ чисел, для которых абсолютно наименьшие вычеты u_i^2 по модулю m являются B-гладкими:

$$u_i^2 \pmod m = \prod_{j=0}^{h} p_j^{\alpha_{ij}} = \upsilon_i,$$

где $\alpha_{ij} \geqslant 0$. Также каждому числу u_i сопоставляется вектор показателей $(\alpha_{i0}, \alpha_{i1}, \ldots, \alpha_{ih})$.

Шаг 3. Найти такое непустое множество $K \subsetneqq \{1, 2, \ldots, h+1\}$, что $\oplus_{i \in K} \mathbf{e}_i = 0$, где $\oplus$ - операция исключающее ИЛИ, $\mathbf{e}_i = (e_{i1}, e_{i2}, \ldots, e_{ih})$, $e_{ij} \equiv \alpha_{ij} \pmod 2$, $0 \leqslant j \leqslant h$.

Шаг 4. Положить

$$x \leftarrow \prod_{i \in K} u_i \pmod m, \quad y \leftarrow \prod_{j=1}^{h} p_j^{\frac{1}{2}\sum_{i \in K} \alpha_{ij}} \pmod m.$$

Тогда $x^2 \equiv y^2 \pmod m$.

Шаг 5. Если $x \not\equiv y \pmod m$, то положить $p \leftarrow$ НОД$(x-y, m)$ и выдать результат: p.

Псевдокод:

```
Функция Бриллхарт_Моррисон_метод(n):
    Пусть max_iterations = 100
    Пусть result = Пустой_Список
    Пусть cf_expansion = непрерывная_дробь_кв_корня(n)
    Для каждого i в диапазоне от 0 до max_iterations:
        Пусть a_i = cf_expansion.удалить_первый_элемент()

        Пусть gcd_result = НОД(a_i, n)
        Если gcd_result не равно 1 и gcd_result не равно n:
            result.добавить(gcd_result)

            // Если число разложено полностью
            Если n == gcd_result:
                Вернуть result

            Пусть factor2 = n / gcd_result
            result.добавить(factor2)
            Вернуть result
    Вернуть result
```

Трудоемкость алгоритма $L_n[\frac{1}{2}, \sqrt{2}]$ при $a = \frac{1}{\sqrt{2}}$.

3.2 Код программы, реализующей рассмотренные алгоритмы

```rust
use std::io;
use std::num;
use rand::Rng;
use gcd::Gcd;
use std::f64;
// use quadratic::jacobi;
use rand::distributions::Uniform;
use std::collections::HashSet;

fn read_integer() -> i128 {
    let mut n = String::new();
    io::stdin()
        .read_line(&mut n)
        .expect("failed to read input.");
    let n: i128 = n.trim().parse().expect("invalid input");
    n
}
```

```


fn read_float() -> f32 {
    let mut num:f32=0.0;
    let mut input = String::new();
    io::stdin().read_line(&mut input).expect("Not a valid string");
    num = input.trim().parse().expect("Not a valid number");
    num
}


fn euclid_gcd_extended(a: i128, b: i128, _x: i128, _y: i128) -> (i128, i128,
  ↪  i128, i128) {
    if a == 0 {
        return (a, b, 0, 1);
    }
    else {
        let (_, d, x1, y1) = euclid_gcd_extended(b % a, a, 0, 0);
        let division = b / a;
        let x = y1 - division * x1;
        let y = x1;
        return (0, d, x, y)
    }
}


fn continued_fraction(mut a: i128, mut b: i128) -> Vec<i128> {
    let mut fraction: Vec<i128> = Vec::new();
    while b != 0 {
        fraction.push(a / b);
        let c = a;
        a = b;
        b = c % b;
    }
    fraction
}


fn print_array(arr: Vec<i128>) {
    print!("[");
    for i in 0..arr.len() {
```

```
        print!("{}", arr[i]);
        if i < arr.len() - 1 {
            print!(", ");
        }
    }
    println!("]");
}



fn mod_inverse(a: i128, n: i128) -> i128 {
    let fraction: Vec<i128> = Vec::new();
    let (_, g, x, _) = euclid_gcd_extended(a, n, 0, 0);
    if g != 1 {
        return -1;
    } else {
        let result = (x % n + n) % n;
        return result;
    }
}


fn is_prime(n: i128) -> bool {
    if n <= 1 {
        return false;
    }
    if n == 2 || n == 3 {
        return true;
    }
    if n % 2 == 0 || n % 3 == 0 {
        return false;
    }

    let mut i = 5;
    while i * i <= n {
        if n % i == 0 || n % (i + 2) == 0 {
            return false;
        }
        i += 6;
    }

```

```
    true
}


fn power_mod(base: u64, exponent: u64, modulus: u64) -> u64 {
    if modulus == 1 {
        return 0;
    }
    let mut result = 1;
    let mut base = base.rem_euclid(modulus);
    let mut exp = exponent;

    while exp > 0 {
        if exp % 2 == 1 {
            result = (result * base).rem_euclid(modulus);
        }
        exp = exp >> 1;
        base = (base * base).rem_euclid(modulus);
    }

    result.rem_euclid(modulus)
}


fn jacobi_symbol(mut a: i128, mut n: i128) -> i128 {
    let mut t = 1;
    while a != 0 {
        while a % 2 == 0 {
            a /= 2;
            let n_mod_8 = n % 8;
            if n_mod_8 == 3 || n_mod_8 == 5 {
                t = -t;
            }
        }

        std::mem::swap(&mut a, &mut n);
        if a % 4 == 3 && n % 4 == 3 {
            t = -t;
        }

        a %= n;
```

```
    }

    if n == 1 {
        return t;
    } else {
        return 0;
    }
}


fn rho_method(n: u128) -> u128 {
    if n == 1 {
        return 1;
    }

    if n % 2 == 0 {
        return 2;
    }

    let mut rng = rand::thread_rng();
    let mut x = rng.gen_range(2..n);
    let mut y = x;
    let mut d: u128 = 1;

    let mut f = |x: u128| -> u128 { (x * x + 1) % n };

    while d == 1 {
        x = f(x);
        y = f(f(y));
        if x > y {
            d = (x - y).gcd(n);
        }
        else
        {
            d = (y - x).gcd(n);
        }
    }

    d
}

```

```
fn factorize_rho_method() -> () {
    println!("Введите число: ");
    let mut n = read_integer() as u128;
    println!("Введите эпсилон: ");
    read_float();
    let kekw = n;
    let mut factors = Vec::new();
    while n > 1 {
        let factor = rho_method(n);
        factors.push(factor);
        n /= factor;
    }

    println!("Разложение числа {}: {:?}", kekw, factors)
}

fn rho_minus_1_method(n: u64, mut base: u64) -> u64 {
    let mut rng = rand::thread_rng();
    let a = rng.gen_range(2..n);

    let mut power = base;
    let mut d = 1;
    let upper_bound = 18446744073709551615 / 1000;
    while d == 1 {
        power *= 2;
        let a_powered = power_mod(a, power, n);
        d = (a_powered - 1).gcd(n);
        if power > upper_bound {
            if base == 2 {
                return n;
            }
            else {
                base = base - 1;
                power = base
            }
        }
        if d != 1 {
            break;
        }
    }
```

```
    d
}


fn factorize_rho_minus_1_method() {
    println!("Введите число: ");
    let mut n = read_integer() as u64;
    println!("Введите базу B: ");
    let base = read_integer() as u64;
    let mut factors = Vec::new();
    let kekw = n;
    while n > 1 {
        let factor = rho_minus_1_method(n, base);
        factors.push(factor);
        n /= factor;
    }
    println!("Разложение числа {}: {:?}", kekw, factors)
}


fn continued_fraction_sqrt(n: u64) -> Vec<u64> {
    let mut a = (n as f64).sqrt() as u64;
    let mut m = 0;
    let mut d = 1;
    let mut num1 = a;
    let mut num2 = 1;
    let mut result = vec![a];

    while num1 * num1 != n {
        m = d * a - m;
        d = (n - m * m) / d;
        a = (a + m) / d;

        result.push(a);
        let num3 = num1;
        num1 = a * num1 + num2;
        num2 = num3;
    }
    println!("suka");

    result
```

```
}


fn eratho_sieve(n: usize) -> Vec<usize> {
    let mut is_prime = vec![true; n + 1];
    is_prime[0] = false;
    is_prime[1] = false;

    for i in 2..=((n as f64).sqrt() as usize) {
        if is_prime[i] {
            let mut multiple = i * i;
            while multiple <= n {
                is_prime[multiple] = false;
                multiple += i;
            }
        }
    }
    let primes: Vec<usize> = (0..=n).filter(|&x| is_prime[x]).collect();
    primes
}

fn get_factor_base(n: u64) -> Vec<usize> {
    let m = f64::sqrt(((f64::exp(f64::sqrt(f64::ln(n as f64) * f64::ln(f64::ln(n as f64)))) as u64) as f64) as usize;
    let factor_base = eratho_sieve(m);
    factor_base
}

fn brillhart_morrison_method(n: u64) -> Vec<u64> {
    let factor_base = get_factor_base(n);
    println!("Фактор база: {:?}", factor_base);

    let mut smooth_b = Vec::new();
    for b in (f64::sqrt(n as f64) as u64)..n {
        let a = b.pow(2) % n;
        for &fact in &factor_base {
            let f = fact.pow(2) % n as usize;
            if a == f as u64 {
                smooth_b.push((b, fact));
            }
        }
```

```
    }
    let mut factors = HashSet::new();
    for (a, fact) in smooth_b {
        let factor = (a - fact as u64).gcd(n);
        if factor != 1 {
            factors.insert(factor);
        }
    }
    let mut result: Vec<u64> = factors.into_iter().collect();
    result.sort();
    result
}

// 7839991
// a =
fn factorize_brillhart_morrison_method() {
    println!("Введите число: ");
    let mut n = read_integer() as u64;
    let mut factors = brillhart_morrison_method(n);
    let kekw = n;
    println!("Разложение числа {}: {:?}", kekw, factors)
}


fn main() {
    println!("Выберите метод разложения:");
    println!("1. p-метод Полларда");
    println!("2. (p-1)-метод Полларда");
    println!("3. Метод цепных дробей (Бриллхарта-Моррисона)");

    let chosen_method = read_integer();
    match chosen_method {
        1 => factorize_rho_method(),
        2 => factorize_rho_minus_1_method(),
        3 => factorize_brillhart_morrison_method(),
        _ => println!("Введено неверное число!"),
    }

}
```

3.3 Результаты тестирования программ

```
Выберите метод разложения:
1. p-метод Полларда
2. (p-1)-метод Полларда
3. Метод цепных дробей (Бриллхарта-Моррисона)
1
Введите число:
8051
Введите эпсилон:
0.3
Разложение числа 8051: [83, 97]
```

Рисунок 1 – Тест первого алгоритма факторизации чисел

```
Выберите метод разложения:
1. p-метод Полларда
2. (p-1)-метод Полларда
3. Метод цепных дробей (Бриллхарта-Моррисона)
2
Введите число:
75361
Введите базу B:
4
Разложение числа 75361: [221, 11, 31]
```

Рисунок 2 – Тест второго алгоритма факторизации чисел

```
Выберите метод разложения:
1. p-метод Полларда
2. (p-1)-метод Полларда
3. Метод цепных дробей (Бриллхарта-Моррисона)
3
Введите число:
5083
Разложение числа 5083: [17, 13, 23]
```

Рисунок 3 – Тест третьего алгоритма факторизации чисел

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были изучены теоретические сведения о методах факторизации чисел (ρ-метод Полларда, $(\rho - 1)$-метод Полларда, алгоритм Бриллхарта-Моррисона). На их основе были рассмотрены соответствующие алгоритмы. Была произведена оценка сложности созданных алгоритмов. Они послужили фундаментом для программной реализации, которая впоследствии успешно прошла тестирование, результаты которого были прикреплены к отчету вместе с листингом программы, написанной на языке Rust с использованием стандартных библиотек языка.