МИНОБРНАУКИ РОССИИ

ФГБОУ ВО «СГУ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

ПРОВЕРКА ЧИСЕЛ НА ПРОСТОТУ

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 проверки простоты чисел и их программная реализация.

Порядок выполнения работы:

1. Рассмотреть тест Ферма проверки чисел на простоту и привести его программную реализацию;
2. Рассмотреть тест Соловея-Штрассена проверки чисел на простоту и привести его программную реализацию;
3. Рассмотреть тест Миллера-Рабина и привести его программную реализацию;

2 Теоретические сведения

2.1 Детерминированные алгоритмы проверки чисел на простоту

Решето Эратосфена. Построение простых чисел, не превосходящих заданного числа N.

Критерий Вильсона. Для любого $n \in \mathbb{N}$ следующие условия эквивалентны:

1. n — простое,
2. $(n-1)! \equiv -1 \pmod{n}$.

2.2 Вероятностные алгоритмы проверки чисел на простоту

Вероятностный алгоритм проверки числа n на простоту использует необходимое условие простоты $P(a)$:

1. выбирается случайным образом $1 < a < n$ и проверяется выполнимость теста $P(a)$ — некоторого условия алгоритма,
2. если тест не проходит, т.е. $P(a)$ не выполняется, то вывод ”число составное”,
3. если тест проходит, т.е. $P(a)$ выполняется, то вывод ”число, вероятно, простое”.

Если событие A — ”число n простое” имеет вероятность $P(A) > \frac{1}{2}$, то вероятность ошибки — получить для составного числа n вывод ”число n возможно простое” $P(\overline{A}) < \frac{1}{2}$ и при t повторах теста вероятность ошибки $P(\overline{A}^t) < \frac{1}{2^t} \approx 0$.

2.2.1 Тест Ферма

Малая теорема Ферма. Если p — простое число, то для любого $a \in Z_p^*$ выполняется свойство $F_p(a) = (a^{p-1} \equiv 1 \pmod{p})$.

p — простое число $\Longrightarrow F_p^+ = Z_n^*$,

где $F_p^+ = \{a \in Z_p^* : F_p(a)\}$ — множество истинности предиката $F_p(a)$.

Определение. Число n называется псевдопростым по основанию $a \in Z_n^*$, если выполняется $F_n(a)$. Здесь $F_n^+ = \{a \in Z_n^* | F_n(a)\}$.

Лемма 1. Для нечетного числа n справедливы утверждения:

1. F_n^+ — подгруппа Z_n^*;
2. если $F_n^+ \neq Z_n^*$, то по теореме Лагранжа

$$|Z_n^*| = |F_n^+| \cdot |Z_n^*/F_n^+| \geq 2 \cdot |F_n^+|, \ |F_n^+| \leq \frac{|Z_n^*|}{2}.$$

Вероятность успеха — вероятность получить "Число n составное" для составного числа n равна $P_0 = 1 - \frac{|F_n^+|}{n-1}$.

Возможны три случая:

1. число n простое и тест всегда дает ответ "Число n, вероятно, простое";
2. число n составное и $F_n^+ \neq Z_n^*$, тогда тест дает ответ "Число n составное" с вероятностью успеха
$$P_0 = 1 - \frac{|F_n^+|}{n-1} \geq 1 - \frac{|F_n^+|}{|Z_n^*|} \geq 1 - \frac{1}{2} = \frac{1}{2};$$
3. число n составное и $F_n^+ = Z_n^*$, тогда тест дает ответ "Число n составное" с вероятностью успеха $P_0 = 1 - \frac{\varphi(n)}{n-1}$.

Во втором случае при k повторах теста вероятность успеха

$$P_0^{(k)} = 1 - (1 - P_0)^k \geq 1 - \frac{1}{2^k} \approx 1.$$

2.2.2 Числа Кармайкла

Определение. Нечетное составное число n называется числом Кармайкла, если $F_n^+ = Z_n^*$ (и, значит, вероятность успеха теста простоты на основе малой теоремы Ферма будет $P_0 = 1 - \frac{\varphi(n)}{n-1}$).

Лемма 2. Для любого числа Кармайкла n справедливы утверждения:

1. $n = p_1 p_2 \ldots p_k$ для $k \geq 3$ простых различных чисел $p_1 p_2 \ldots p_k$;
2. ($\forall p$ — простое) $p | n \Rightarrow p - 1 | n - 1$.

Количество чисел Кармайкла $k \leq n : C(n) > n^{\frac{2}{7}}$.

2.2.3 Тест Соловея-Штрассена

Критерий Эйлера. Нечетное число n является простым тогда и только тогда, когда для любого $a \in Z_n^*$ выполняется свойство

$$E_n(a) = \left(a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n} \right) \pmod{n} \right).$$

n — простое число $\Longleftrightarrow F_n^+ = Z_n^*$, где

$$E_n^+ = \{a \in Z_n^* \mid E_n(a)\}.$$

Определение. Число n называется эйлеровым псевдопростым по основанию $a \in Z_n^*$, если выполнения $E_n(a)$.

Лемма 1. Для нечетного числа n справедливы утверждения:

1. E_n^+ — подгруппа Z_n^*;
2. Если n — составное число, то $|E_n^+| \le \frac{|Z_n^*|}{2}$

Вероятность успеха тест простоты Соловея-Штрассена на основе критерия Эйлера для составного числа n равна $P_0 = 1 - \frac{|E_n^+|}{n-1} \ge \frac{1}{2}$.

Возможны два случая:

1. число n простое и тест всегда дает ответ "Число n, вероятно, простое";
2. число n составное и тест дает ответ "Число n составное" с вероятностью успеха $P_0 \ge \frac{1}{2}$.

Во втором случае при k повторах теста вероятность успеха

$$P_0^{(k)} = 1 - (1 - P_0)^k \ge 1 - \frac{1}{2^k} \approx 1.$$

2.2.4 Тест Миллера-Рабина

Теорема (Критерий Миллера). Пусть n — нечетное число и $n - 1 = 2^s t$ для нечетного t. Тогда n является простым в том и только том случае, если для любого $a \in Z_n^*$ выполняется свойство

$$M_n(a) = \left(a^t \equiv 1 \pmod{n} \vee (\exists 0 \le k < s)(a^{2^k t}) \equiv -1 \pmod{n}\right).$$

n — просто число $\Longleftrightarrow M_n^+ = Z_n^*$, где

$$M_n^+ = \{a \in Z_n^* \mid M_n(a)\}.$$

Определение. Число n, псевдопростое по основанию $a \in Z_n^*$, называется сильно псевдопростым по этому основанию $a \in Z_n^*$, если выполняется $M_n(a)$, т.е. выполняется одно из условий:

1. либо $a^t \equiv 1 \pmod{n}$;
2. либо $a^{2^k t} \equiv -1 \pmod{n}$ для некоторого $0 \le k < s$.

Для составного числа n выполняется $|M_n^+| \le \frac{|Z_n^*|}{4}$ и, значит, вероятность успеха теста простоты Миллера-Рабина на основе критерия Миллера для составного числа n равна $P_0 = 1 - \frac{|M_n^+|}{n-1} \ge \frac{3}{4}$. При k повторах теста вероятность успеха

$$P_0^{(k)} = 1 - (1 - P_0)^k \geq 1 - \frac{1}{4^k} \approx 1.$$

2.2.5 Сравнение тестов простоты чисел

$$M_n(a) \Longrightarrow E_n(a) \Longrightarrow F_n(a) \text{ и, значит, } M_n^+ \subset E_m^+ \subset F_n^+.$$

3 Результаты работы

3.1 Описание алгоритма разложения чисел в цепную дробь и алгоритмов приложений цепных дробей

Алгоритм 1 - тест простоты на основе малой теоремы Ферма

Вход: Нечетное число $n > 5$.

Выход: ”Число n, вероятно, простое” или ”Число n составное”.

Шаг 1. Выбрать случайно $a \in \{1, 2, \ldots, n-1\}$ и вычислить $d = \text{НОД}(a, n)$. Если $d > 1$, то ответ ”Число n составное”.

Шаг 2. Если $d = 1$, то проверить условие

$$F_n(a) = (a^{n-1} \equiv 1 \pmod n).$$

Если оно не выполнено, то ответ ”Число n составное”. В противном случае ответ ”Число n, вероятно, простое”.

Псевдокод:

```
функция Тест_Ферма(число):
    a = случайное_число_в_диапазоне(1, число - 1)

    если НОД(a, число) == 1 то
        powered_a = в_степень_по_модулю(a, число - 1, число)
        если powered_a == 1 то
            Вывести "Число", число, "вероятно, простое."
        иначе
            Вывести "Число", число, "составное."
        конец если
    иначе
        Вывести "Число", число, "составное."
    конец если
```

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

Алгоритм 2 - тест простоты Соловея-Штрассена

Вход: Нечетное число $n > 5$.

Выход: ”Число n, вероятно, простое” или ”Число n составное”.

Шаг 1. Выбрать случайно $a \in \{1, 2, \ldots, n-1\}$ и вычислить $d = \text{НОД}(a, n)$. Если $d > 1$, то ответ ”Число n составное”.

Шаг 2. Если $d = 1$, то проверить условие $E_n(a)$. Если оно не выполнено, то ответ "Число n составное". В противном случае ответ "Число n, вероятно, простое".

Псевдокод:

```
функция Тест_Соловея_Штрассена(число):
    a = случайное_число_в_диапазоне(1, число - 1)

    если НОД(a, число) == 1 то
        powered_a = в_степень_по_модулю(a, (число - 1) / 2, число)
        a_n = символ_Якоби(a, число)
        если powered_a == a_n.взято_по_модулю(число) то
            Вывести "Число", число, "вероятно, простое."
        иначе
            Вывести "Число", число, "составное."
        конец если
    иначе
        Вывести "Число", число, "составное."
    конец если
```

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

Алгоритм 3 - тест простоты Миллера-Рабина

Вход: Нечетное число $n > 5$.

Выход: "Число n, вероятно, простое" или "Число n составное".

Шаг 1. Выбрать случайно $a \in \{1, 2, \ldots, n-1\}$ и вычислить $d =$ НОД(a, n). Если $d > 1$, то ответ "Число n составное".

Шаг 2. Если $d = 1$, то вычислить $r_k = a^{2^k t}$ для значений $k \in \{0, 1, 2, \ldots, s-1\}$. Если $r_0 \equiv 1 \pmod n$ или $r_k \equiv -1 \pmod n$ для некоторого $0 \leq k < s$, то ответ "Число n, вероятно, простое". В противном случае ответ "Число n составное".

Псевдокод:

```
функция Тест_Миллера_Рабина(число):
    a = случайное_число_в_диапазоне(1, число - 1)

    если НОД(a, число) == 1 то
        t = число - 1
        s = 0
        пока t % 2 == 0 делать
```

```
            t = t / 2
            s = s + 1
        конец пока

        r_0 = в_степень_по_модулю(a, t, число)
        если r_0 != 1 и r_0 != (число - 1) то
            Вывести "Число", число, "составное."
            вернуть
        конец если

        для k от 1 до (s - 1) делать
            r_k = в_степень_по_модулю(a, 2^k * t, число)
            если r_k != (число - 1) то
                Вывести "Число", число, "составное."
                вернуть
            конец если
        конец для

        Вывести "Число", число, "вероятно, простое."
    иначе
        Вывести "Число", число, "составное."
    конец если
```

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

3.2 Код программы, реализующей рассмотренные алгоритмы

```rust
use std::io;
use rand::Rng;
// use quadratic::jacobi;
use rand::distributions::Uniform;

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 euclid_gcd(a: i128, b: i128) -> i128 {
    if b == 0 {
        return a;
    };
    let r = a % b;
    return euclid_gcd(b, r);
}


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 check_coprime(modules: Vec<i128>, n: i128) -> bool {
    for i in 0..n {
        for j in 0..n {
            let m_i = modules[i as usize];
            let m_j = modules[j as usize];
            if euclid_gcd(m_i, m_j) != 1 && i != j {
                return false;
            }
        }
    }
    return true;
}


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: i128, exponent: i128, modulus: i128) -> i128 {
    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 fermat_test(number: i128) -> () {
    let mut rng = rand::thread_rng();
    let a = rng.gen_range(1..=number - 1);

    if euclid_gcd(a, number) == 1 {
```

```
        let powered_a = power_mod(a, number - 1, number);
        if powered_a == 1 {
            println!("Число {}, вероятно, простое.", number);
        }
        else {
            println!("Число {} составное.", number);
        }
    }
    else {
        println!("Число {} составное.", number);
    }
}


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 solovei_strassen_test(number: i128) -> () {
    let mut rng = rand::thread_rng();
    let a = rng.gen_range(1..=number - 1);

    if euclid_gcd(a, number) == 1 {
        let powered_a = power_mod(a, (number - 1) / 2, number);
        let jacobi_a_n = jacobi_symbol(a, number);
        if powered_a == jacobi_a_n.rem_euclid(number) {
            println!("Число {}, вероятно, простое.", number);
        }
        else {
            println!("Число {} составное.", number);
        }
    }
    else {
        println!("Число {} составное.", number);
    }
}


fn miller_rabin_test(number: i128) -> () {
    let mut rng = rand::thread_rng();
    let a = rng.gen_range(1..=number - 1);

    if euclid_gcd(a, number) == 1 {
        let mut t = number - 1;
        let mut s = 0;
        while t % 2 == 0 {
            t = t / 2;
            s = s + 1;
        }

        let r_0 = power_mod(a, t, number);
        if r_0 != 1 && r_0 != (number - 1) {
            println!("Число {} составное.", number);
            return ();
        }

        for k in 1..(s - 1) {
            let r_k = power_mod(a, 2_i128.pow(k) * t, number);
            if r_k != (number - 1) {
```

```
                println!("Число {} составное.", number);
                return ();
            }
        }

        println!("Число {}, вероятно, простое.", number);
    }
    else {
        println!("Число {} составное.", number);
    }
}


fn main() {
    println!("Введите число:");
    let number = read_integer();
    println!("");

    println!("Тест Ферма:");
    fermat_test(number);
    println!("");

    println!("Тест Соловея-Штрассена:");
    solovei_strassen_test(number);
    println!("");

    println!("Тест Миллера-Рабина:");
    miller_rabin_test(number);
    println!("");

}
```

3.3 Результаты тестирования программ

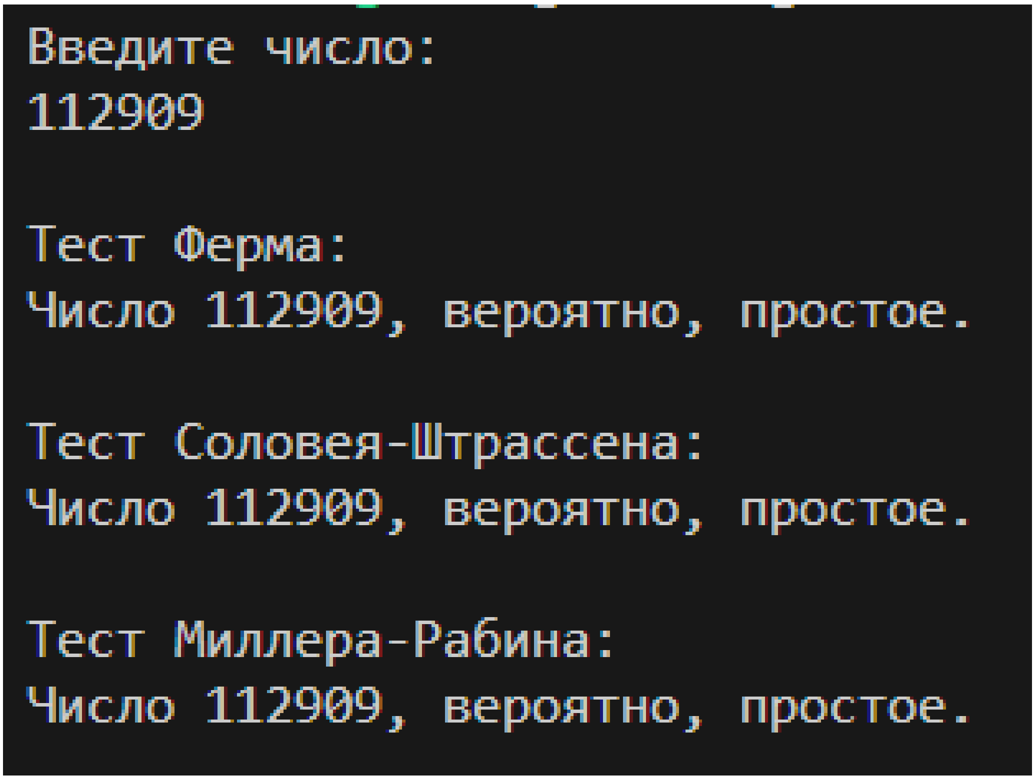

Рисунок 1 – Первый тест алгоритмов проверки чисел на простоту

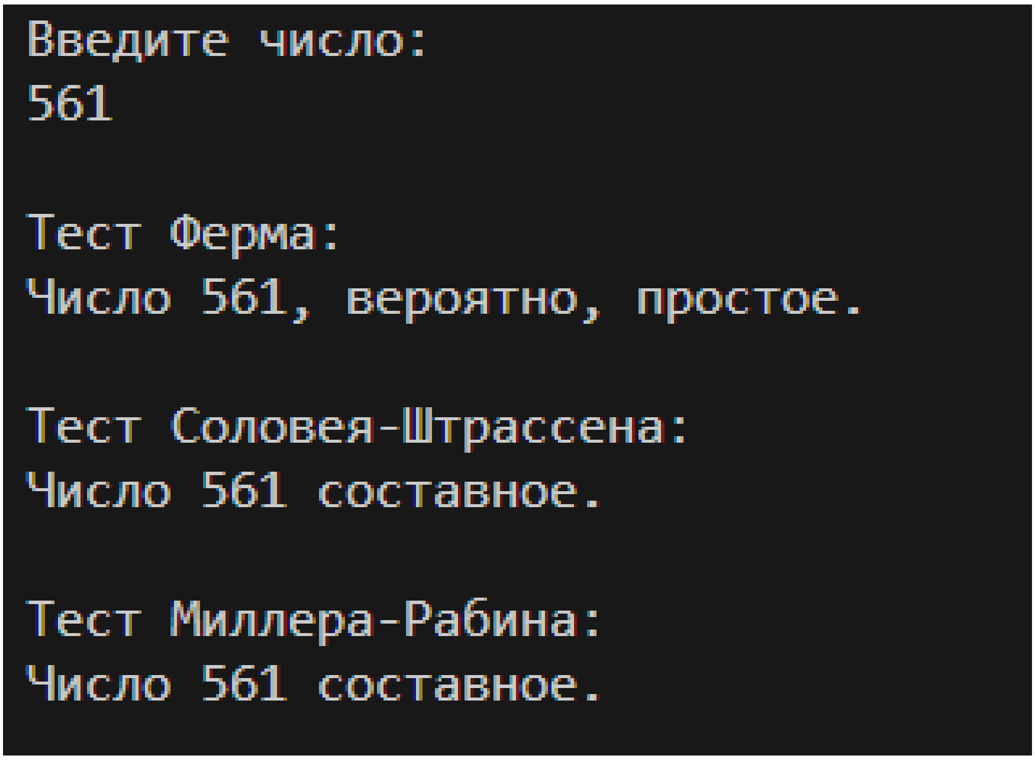

Рисунок 2 – Второй тест алгоритмов проверки чисел на простоту

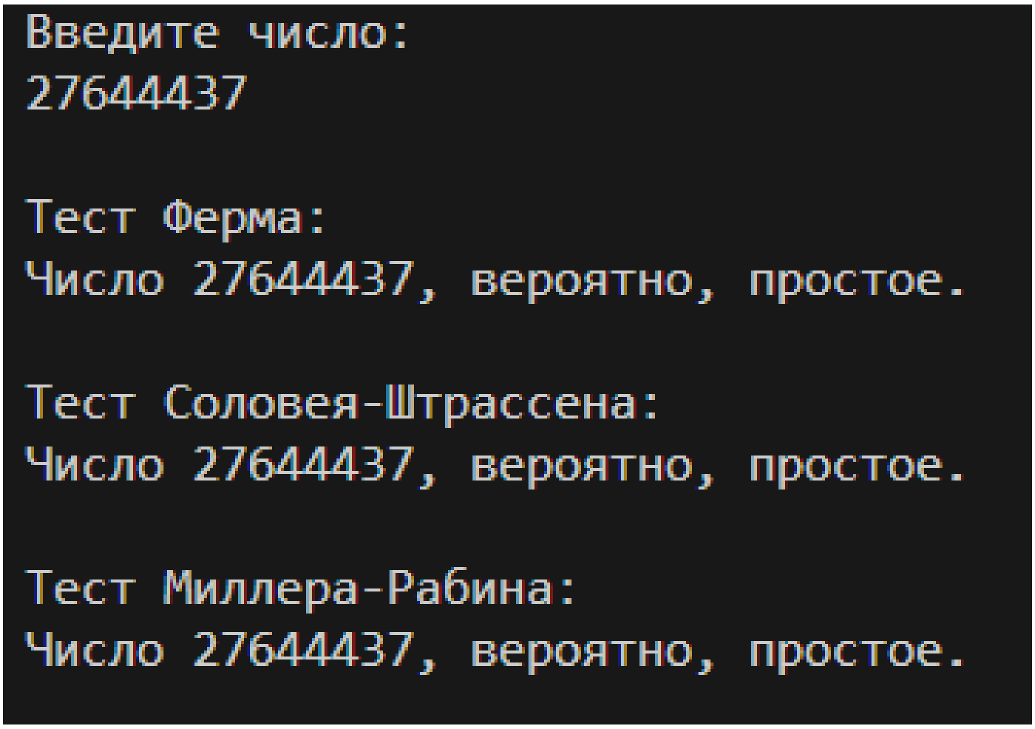

Рисунок 3 – Третий тест алгоритмов проверки чисел на простоту

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были изучены теоретические сведения о методах проверки простоты чисел (тест Ферма, Соловея-Штрассена, Миллера-Рабина). 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.