

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

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" $P(\bar{A}) < \frac{1}{2}$ и при t повторях теста вероятность ошибки $P(\bar{A}^t) < \frac{1}{2^t} \approx 0$.

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 $\implies F_p^+ = Z_n^*$,

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

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

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

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 $\iff F_n^+ = Z_n^*$, 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 $|E_n^+| \leq \frac{|Z_n^*|}{2}$

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

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 $P_0 \geq \frac{1}{2}$.

Во втором случае при k повторях теста вероятность успеха

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

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

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

n — просто число $\iff M_n^+ = Z_n^*$, 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 $0 \leq k < s$.

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

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

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 $a \in \{1, 2, \dots, n-1\}$ 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):  
    а = случайное_число_в_диапазоне(1, число - 1)  
  
    если НОД(а, число) == 1 то  
        powered_a = в_степень_по_модулю(а, 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 $a \in \{1, 2, \dots, n-1\}$ 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):  
    а = случайное_число_в_диапазоне(1, число - 1)  
  
    если НОД(а, число) == 1 то  
        powered_a = в_степень_по_модулю(а, (число - 1) / 2, число)  
        а_n = символ_Якоби(а, число)  
        если powered_a == а_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 $a \in \{1, 2, \dots, n-1\}$ и вычислить $d = \text{НОД}(a, n)$. Если $d > 1$, то ответ "Число n составное".

Шаг 2. Если $d = 1$, то вычислить $r_k = a^{2^k t}$ для значений $k \in \{0, 1, 2, \dots, s-1\}$. 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):  
    а = случайное_число_в_диапазоне(1, число - 1)  
  
    если НОД(а, 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

```

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

3.2 Код программы, реализующей рассмотренные алгоритмы

```

1  use std::io;
2  use rand::Rng;
3  // use quadratic::jacobi;
4  use rand::distributions::Uniform;
5
6  fn read_integer() -> i128 {
7      let mut n = String::new();
8      io::stdin()
9          .read_line(&mut n)
10         .expect("failed to read input.");
11     let n: i128 = n.trim().parse().expect("invalid input");
12     n
13 }
14

```

```

15
16 fn euclid_gcd(a: i128, b: i128) -> i128 {
17     if b == 0 {
18         return a;
19     };
20     let r = a % b;
21     return euclid_gcd(b, r);
22 }
23
24
25 fn euclid_gcd_extended(a: i128, b: i128, _x: i128, _y: i128) -> (i128, i128,
    ↪ i128, i128) {
26     if a == 0 {
27         return (a, b, 0, 1);
28     }
29     else {
30         let (_, d, x1, y1) = euclid_gcd_extended(b % a, a, 0, 0);
31         let division = b / a;
32         let x = y1 - division * x1;
33         let y = x1;
34         return (0, d, x, y)
35     }
36 }
37
38
39 fn continued_fraction(mut a: i128, mut b: i128) -> Vec<i128> {
40     let mut fraction: Vec<i128> = Vec::new();
41     while b != 0 {
42         fraction.push(a / b);
43         let c = a;
44         a = b;
45         b = c % b;
46     }
47     fraction
48 }
49
50
51 fn print_array(arr: Vec<i128>) {
52     print!("{}",);
53     for i in 0..arr.len() {
54         print!("{}", arr[i]);

```

```

55         if i < arr.len() - 1 {
56             print!("{}", " ");
57         }
58     }
59     println!("{}", "");
60 }
61
62
63 fn check_coprime(modules: Vec<i128>, n: i128) -> bool {
64     for i in 0..n {
65         for j in 0..n {
66             let m_i = modules[i as usize];
67             let m_j = modules[j as usize];
68             if euclid_gcd(m_i, m_j) != 1 && i != j {
69                 return false;
70             }
71         }
72     }
73     return true;
74 }
75
76
77 fn mod_inverse(a: i128, n: i128) -> i128 {
78     let fraction: Vec<i128> = Vec::new();
79     let (_, g, x, _) = euclid_gcd_extended(a, n, 0, 0);
80     if g != 1 {
81         return -1;
82     } else {
83         let result = (x % n + n) % n;
84         return result;
85     }
86 }
87
88
89 fn is_prime(n: i128) -> bool {
90     if n <= 1 {
91         return false;
92     }
93     if n == 2 || n == 3 {
94         return true;
95     }

```

```

96     if n % 2 == 0 || n % 3 == 0 {
97         return false;
98     }
99
100    let mut i = 5;
101    while i * i <= n {
102        if n % i == 0 || n % (i + 2) == 0 {
103            return false;
104        }
105        i += 6;
106    }
107
108    true
109 }
110
111
112 fn power_mod(base: i128, exponent: i128, modulus: i128) -> i128 {
113     if modulus == 1 {
114         return 0;
115     }
116     let mut result = 1;
117     let mut base = base.rem_euclid(modulus);
118     let mut exp = exponent;
119
120     while exp > 0 {
121         if exp % 2 == 1 {
122             result = (result * base).rem_euclid(modulus);
123         }
124         exp = exp >> 1;
125         base = (base * base).rem_euclid(modulus);
126     }
127
128     result.rem_euclid(modulus)
129 }
130
131
132 fn fermat_test(number: i128) -> i32 {
133     let mut rng = rand::thread_rng();
134     let a = rng.gen_range(1..=number - 1);
135
136     if euclid_gcd(a, number) == 1 {

```

```

137     let powered_a = power_mod(a, number - 1, number);
138     if powered_a == 1 {
139         // println!("Число {}, вероятно, простое.", number);
140         return 1;
141     }
142     else {
143         // println!("Число {} составное.", number);
144         return 0;
145     }
146 }
147 else {
148     // println!("Число {} составное.", number);
149     return 0;
150 }
151 }
152
153
154 fn jacobi_symbol(mut a: i128, mut n: i128) -> i128 {
155     let mut t = 1;
156     while a != 0 {
157         while a % 2 == 0 {
158             a /= 2;
159             let n_mod_8 = n % 8;
160             if n_mod_8 == 3 || n_mod_8 == 5 {
161                 t = -t;
162             }
163         }
164
165         std::mem::swap(&mut a, &mut n);
166         if a % 4 == 3 && n % 4 == 3 {
167             t = -t;
168         }
169
170         a %= n;
171     }
172
173     if n == 1 {
174         return t;
175     } else {
176         return 0;
177     }

```

```

178 }
179
180
181 fn solovei_strassen_test(number: i128) -> i32 {
182     let mut rng = rand::thread_rng();
183     let a = rng.gen_range(1..=number - 1);
184
185     if euclid_gcd(a, number) == 1 {
186         let powered_a = power_mod(a, (number - 1) / 2, number);
187         let jacobi_a_n = jacobi_symbol(a, number);
188         if powered_a == jacobi_a_n.rem_euclid(number) {
189             // println!("Число {}, вероятно, простое.", number);
190             return 1;
191         }
192     } else {
193         // println!("Число {} составное.", number);
194         return 0;
195     }
196 }
197
198 // println!("Число {} составное.", number);
199 return 0;
200 }
201 }
202
203
204 fn miller_rabin_test(number: i128) -> i32 {
205     let mut rng = rand::thread_rng();
206     let a = rng.gen_range(1..=number - 1);
207
208     if euclid_gcd(a, number) == 1 {
209         let mut t = number - 1;
210         let mut s = 0;
211         while t % 2 == 0 {
212             t = t / 2;
213             s = s + 1;
214         }
215
216         let r_0 = power_mod(a, t, number);
217         if r_0 != 1 && r_0 != (number - 1) {
218             // println!("Число {} составное.", number);

```

```

219         return 0;
220     }
221
222     for k in 1..(s - 1) {
223         let r_k = power_mod(a, 2_i128.pow(k) * t, number);
224         if r_k != (number - 1) {
225             // println!("Число {} составное.", number);
226             return 0;
227         }
228     }
229
230     return 1;
231     // println!("Число {}, вероятно, простое.", number);
232 }
233 else {
234     return 0;
235     // println!("Число {} составное.", number);
236 }
237 }
238
239
240 fn main() {
241     println!("Введите число:");
242     let number = read_integer();
243     println!("");
244
245     println!("Введите количество раундов:");
246     let rounds = read_integer();
247     println!("");
248
249     let mut is_prime_fermat = 0;
250     let mut is_prime_ss = 0;
251     let mut is_prime_mr = 0;
252
253     let mut went_rounds = 0;
254     while went_rounds != rounds {
255         is_prime_fermat = is_prime_fermat + fermat_test(number);
256         is_prime_ss = is_prime_ss + solovei_strassen_test(number);
257         is_prime_mr = is_prime_mr + miller_rabin_test(number);
258         went_rounds = went_rounds + 1;
259     }

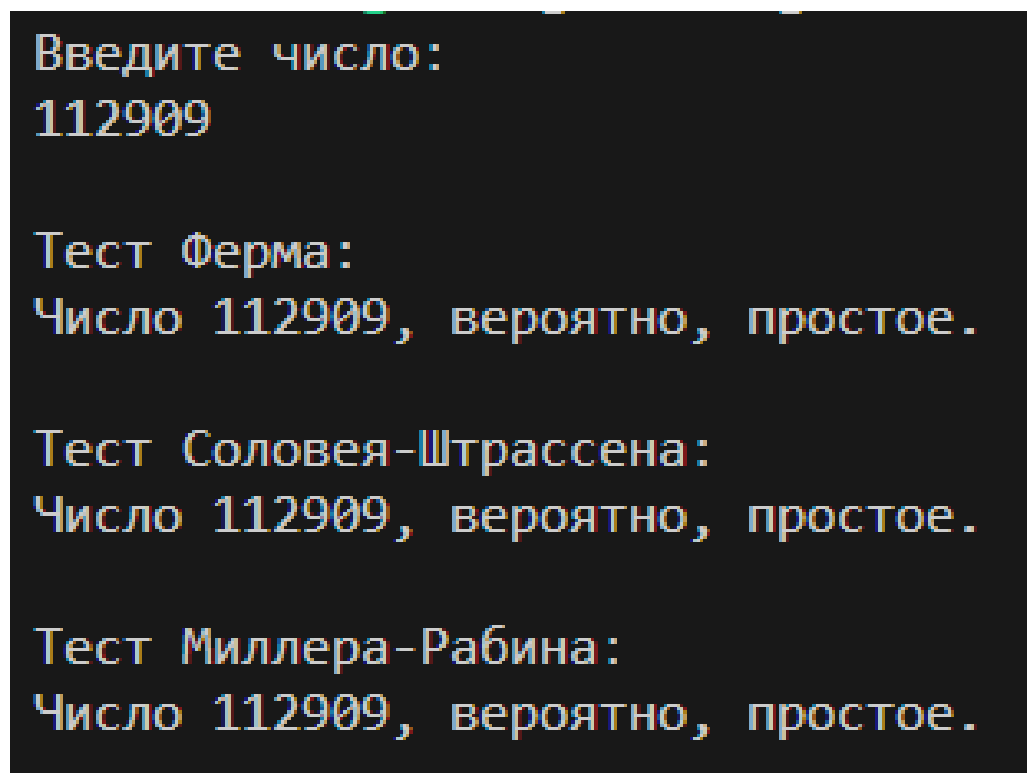
```

```

260
261     println!("Тест Ферма:");
262     if (i128::from(is_prime_fermat) > rounds / 2) {
263         println!("Число {}, вероятно, простое.", number);
264     }
265     else {
266         println!("Число {} составное.", number);
267     }
268     // fermat_test(number);
269     println!("");
270
271     println!("Тест Соловея-Штрассена:");
272     // solovei_strassen_test(number);
273     if (i128::from(is_prime_ss) > rounds / 2) {
274         println!("Число {}, вероятно, простое.", number);
275     }
276     else {
277         println!("Число {} составное.", number);
278     }
279     println!("");
280
281     println!("Тест Миллера-Рабина:");
282     // miller_rabin_test(number);
283     if (i128::from(is_prime_mr) > rounds / 2) {
284         println!("Число {}, вероятно, простое.", number);
285     }
286     else {
287         println!("Число {} составное.", number);
288     }
289     println!("");
290
291 }

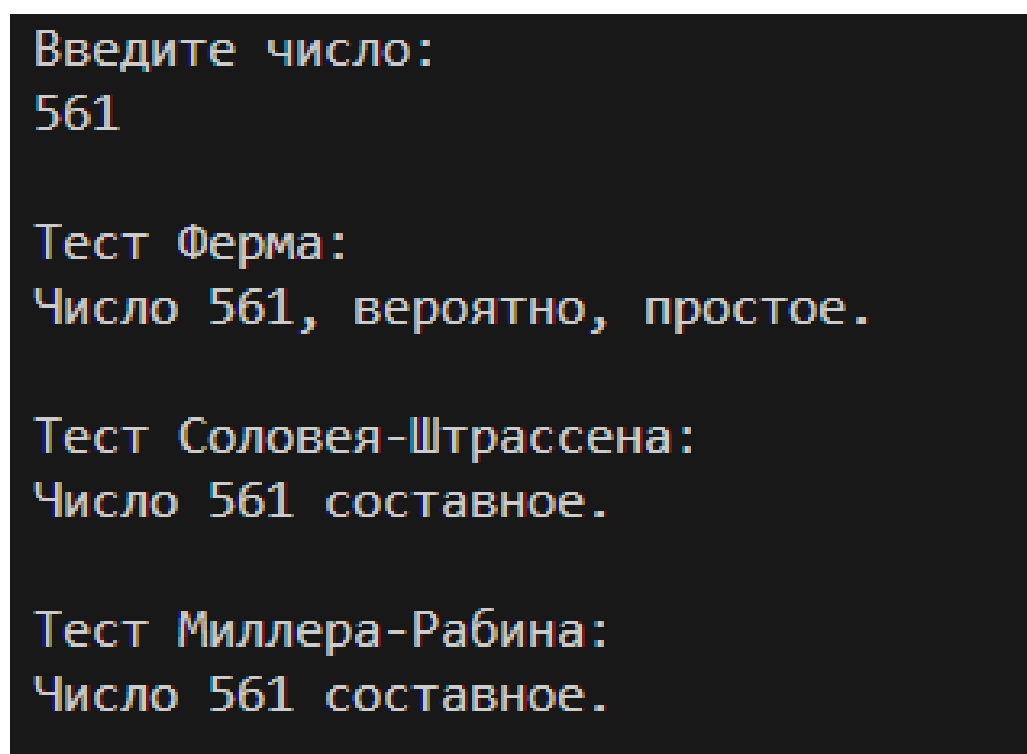
```


3.3 Результаты тестирования программ



```
Введите число:  
112909  
  
Тест Ферма:  
Число 112909, вероятно, простое.  
  
Тест Соловея-Штрассена:  
Число 112909, вероятно, простое.  
  
Тест Миллера-Рабина:  
Число 112909, вероятно, простое.
```

Рисунок 1 – Первый тест алгоритмов проверки чисел на простоту



```
Введите число:  
561  
  
Тест Ферма:  
Число 561, вероятно, простое.  
  
Тест Соловея-Штрассена:  
Число 561 составное.  
  
Тест Миллера-Рабина:  
Число 561 составное.
```

Рисунок 2 – Второй тест алгоритмов проверки чисел на простоту

```
Введите число:
27644437

Тест Ферма:
Число 27644437, вероятно, простое.

Тест Соловея-Штрассена:
Число 27644437, вероятно, простое.

Тест Миллера-Рабина:
Число 27644437, вероятно, простое.
```

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.