МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Проверка чисел на простоту

ОТЧЁТ
ПО ДИСЦИПЛИНЕ
«ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ»

студента 5 курса 531 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Арбузова Матвея Александровича

Преподаватель
профессор, д.ф.-м.н. ___________ В. А. Молчанов
подпись, дата

Саратов 2023

1 Постановка задачи

Целью данной лабораторной работы является 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 по рассмотренным темам

Тесты простоты

Для проверки простоты числа n достаточно найти его каноническое разложение. Однако для больших n это потребует значительных вычислений, поскольку задача факторизации целых чисел является вычислительно сложной. Поэтому для проверки простоты чисел разработаны другие эффективные алгоритмы, называемые *тестами простоты*. Согласно [ДМЭ] под тестом простоты понимается «детерминированный или вероятностный алгоритм, позволяющий для любого целого $n > 1$, не находя его канонического разложения, определять, является ли число n простым или составным».

В основе любого теста простоты чаще всего лежит некоторый критерий простоты числа n, состоящий из конечной серии условий простоты. Проверка всех этих условий приводит к точному ответу на вопрос: «Является ли число n простым?»

Вероятностные тесты на простоту

Однако полная проверка всех условий может оказаться слишком трудоемкой. В связи с этим на практике иногда ограничиваются проверкой лишь части условий. Тогда возможны две ситуации: либо нашлось не выполняющееся условие (и мы получаем точный ответ — «число n составное»), либо все проверенные условия выполнены (и мы можем говорить о простоте числа n лишь с некоторой вероятностью). Алгоритмы, основанные на проверке части условий критерия простоты, принято называть *вероятностными тестами простоты*. Вероятностные тесты простоты обычно довольно просты в обосновании и реализации, их временная сложность выражается полиномом от $log n$.

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

1) выбирается случайный образом $1 < a < n$ и проверяется выполнимость теста $P(a)$ – некоторого условия алгоритма;

2) если тест не проходит, то есть $P(a)$ не выполняется, то вывод «число n составное»;

3) если тест проходит, то есть $P(a)$ выполняется, то вывод «число n, 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, вероятно, простое» $P(\bar{A}) < \frac{1}{2}$ и при t повторах теста вероятность ошибки $P(\bar{A}^t) < \frac{1}{2^t} \approx 0$.

Тест Ферма

Данный тест стоится на малой теореме Ферма.

Малая теорема Ферма. Если n – простое число, то для любого $a \in \boldsymbol{Z}_n^*$ выполняется свойство $F_n(a) = (a^{n-1} \equiv 1 (\operatorname{mod} n))$.

n – простое число $\Longrightarrow F_n^+ = \boldsymbol{Z}_n^*$, где $F_n^+ = \{a \in \boldsymbol{Z}_n^* : F_n(a)\}$ – множество истинности предиката $F_n(a)$.

Опр. Число n называется *псевдопростым по основанию* $a \in \boldsymbol{Z}_n^*$, если выполняется $F_n(a)$.

Здесь $F_n^+ = \{a \in \boldsymbol{Z}_n^* | F_n(a)\}$.

Опр. Вероятность получить «Число n составное» для составного числа n называется *вероятностью успеха* и 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 $F_n^+ \neq \boldsymbol{Z}_n^*$, тогда тест даёт ответ «Число n составное» с вероятностью успеха

$$P_0 = 1 - \frac{|F_n^+|}{n-1} \geq 1 - \frac{|F_n^+|}{|\boldsymbol{Z}_n^*|} \geq 1 - \frac{1}{2} = \frac{1}{2};$$

3) число n составное и $F_n^+ = \boldsymbol{Z}_n^*$, тогда тест даёт ответ «Число n составное» с вероятностью успеха $P_0 = 1 - \frac{\varphi(n)}{n-1}$.

В случае 2) при k повторах теста вероятность успеха $P_0^{(k)} = 1 - (1 - P_0)^k \geq 1 - \frac{1}{2^k} \approx 1$.

<u>Опр.</u> Нечётное составное число n называется *числом Кармайкла*, если $F_n^+ = \boldsymbol{Z}_n^*$.

<u>Лемма.</u> Для любого числа Кармайкла 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) ($\forall p$ – простое) $p|n \implies p - 1|\, n - 1$.

Тест Соловея-Штрассена

Данный тест строится на критерии Эйлера.

<u>Критерий Эйлера</u>. Нечетное число n является простым тогда и только тогда, когда для любого целого числа $a \in \boldsymbol{Z}_n^*$ выполняется свойство $E_n(a) = (a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) (mod\, n))$.

n – простое число $\leftrightarrow E_n^+ = \boldsymbol{Z}_n^*$, где $E_n^+ = \{a \in \boldsymbol{Z}_n^* | E_n(a)\}$.

<u>Опр.</u> Число n называется *эйлеровым пресвдопростым по основанию* $a \in \boldsymbol{Z}_n^*$, если выполняется $E_n(a)$.

Вероятность успеха 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 выдает ответ «Число n, вероятно, простое»;

2) число n составное и тест даёт ответ «Число n составное» с вероятностью успеха $P_0 \geq \frac{1}{2}$.

В случае 2) при k повторах теста вероятность успеха $P_0^{(k)} = 1 - (1 - P_0)^k \geq 1 - \frac{1}{2^k} \approx 1$.

Тест Миллера-Рабина

Данный тест основывается на критерии Миллера.

<u>Теорема (Критерий Миллера)</u> Пусть n – нечётное число и $n - 1 = 2^s t$, для нечётного t. Тогда n является простым в том и только в том случае, если

для любого $a \in \boldsymbol{Z}_n^*$, взаимно выполняется свойство $M_n(a) = (a^t \equiv 1(mod\ n) \vee (\exists\ 0 \leq k < s)(a^{2^k t} \equiv -1(mod n)))$.

n – простое число $\leftrightarrow M_n^+ = \boldsymbol{Z}_n^*$, где $M_n^+ = \{a \in \boldsymbol{Z}_n^* | M_n(a)\}$.

Необходимость. Для простого n выполняется:

$$a^{n-1} \equiv 1(mod\ n),\ a^{n-1} - 1 \equiv 0(mod\ n)$$

$$a^{2^s t} - 1 \equiv ((a^t)^{2^{s-1}})^2 - 1 \equiv (a^t - 1)(a^t + 1) \ldots \left((a^t)^{2^{s-1}} + 1\right) \equiv 0(mod\ n) \blacksquare$$

Опр. Число n, псевдопростое по основанию $a \in \boldsymbol{Z}_n^*$, называется *сильно псевдопростым по этому основанию* a, если выполняется одно из условий:

1) $a^t \equiv 1(mod\ n)$;

2) $a^{2^k t} \equiv -1(mod\ n)$ для некоторого $0 \leq k < s$.

Для составного числа n выполняется $|M_n^+| \leq \frac{|\boldsymbol{Z}_n^*|}{4}$ и, значит, вероятность успеха для составного числа n равна $P_0 = 1 - \frac{|M_n^+|}{n-1} \geq \frac{3}{4}$.

При k повторах теста вероятность успеха $P_0^{(k)} = 1 - (1 - P_0)^k \geq 1 - \frac{1}{4^k} \approx 1$.

Сравнение тестов простоты чисел

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

3 Практическая реализация

3.1 Описание и оценка сложности алгоритмов

Алгоритм теста простоты на основе малой теоремы Ферма

Вход: Нечетное целое число $n \geq 5$.

Выход: «Число n, вероятно, простое» или «Число n составное».

Шаг 1. Выбрать случайное целое число $a \in \{1,2, \dots, n-1\}$ и вычислить $d = \text{НОД}(a, n)$. Если $d > 1$, то «Число n составное»;

Шаг 2. Если $d = 1$, то проверить условие $F_n(a) = (a^{n-1} \equiv 1(\operatorname{mod} n))$. Если оно не выполнено, то ответ «Число n составное». В противном случае ответ «Число n, вероятно, простое».

Временная сложность алгоритма $O(\log^3 n)$.

Алгоритм теста простоты Соловея-Штрассена на основе критерия Эйлера

Вход: Нечетное целое число $n \geq 5$.

Выход: «Число n, вероятно, простое» или «Число n составное».

Шаг 1. Выбрать случайное целое число $a \in \{1,2, \dots, n-1\}$ и вычислить $d = \text{НОД}(a, n)$. Если $d > 1$, то «Число n составное»;

Шаг 2. Если $d = 1$, то проверить условие $E_n(a) = (a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) (mod\, n))$. Если оно не выполнено, то ответ «Число n составное». В противном случае ответ «Число n, вероятно, простое».

Временная сложность алгоритма $O(\log^3 n)$.

Алгоритм теста простоты Миллера-Рабина на основе критерия Миллера

Вход: Нечетное целое число $n \geq 5$.

Выход: «Число n, вероятно, простое» или «Число n составное».

Шаг 1. Выбрать случайное целое число $a \in \{1,2, \dots, n-1\}$ и вычислить $d = \text{НОД}(a, n)$. Если $d > 1$, то «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 $r_0 \equiv 1 (mod\, n)$ или $r_k \equiv -1 (mod\, n)$ для некоторого $0 \le k < s$, то ответ «Число n, вероятно, простое». В противном случае ответ «Число n составное».

Временная сложность алгоритма $O(\log^3 n)$.

3.2 Псевдокоды рассмотренных алгоритмов

Псевдокод алгоритма теста Ферма

```
Ввод n,k, где n - число, которое нужно проверить, k - число тестов
Если (a < 5)
      вывести «Число должно быть больше 4»
Если (k < 1)
      вывести «Число должно быть больше 0»
Вывести результат функции Ferma(n,k)

Функция Ferma(n,k):
      В цикле по i от 1 до k
            a = случайное число от 2 до n − 1;
            Если (НОД(a,n) > 1)
                  Вывести «Число n составное»
            Если (a^(n−1)(mod n) ≠ 1)
                  Вывести «Число n составное»
      Вывести «Число n, вероятно, простое»
```

Псевдокод алгоритма теста Соловея-Штрассена

```
Ввод n,k, где n - число, которое нужно проверить, k - число тестов
Если (a < 5)
      вывести «Число должно быть больше 4»
Если (k < 1)
      вывести «Число должно быть больше 0»
Вывести результат функции SolovSht(n,k)

Функция SolovSht(n,k):
      В цикле по i от 1 до k
            a = случайное число от 2 до n − 1;
            Если (НОД(a,n) > 1)
                  Вывести «Число n составное»
            newa = a^((n−1)/2)(mod n)
            Если (newa ≠ 1) и (newa ≠ n − 1)
                  Вывести «Число n составное»
            l = (a/n)(mod n)
            Если (newa ≠ l)
                  Вывести «Число n составное»
      Вывести «Число n, вероятно, простое»
```

Псевдокод алгоритма теста Миллера-Рабина

```
Ввод n,k, где n – число, которое нужно проверить, k – число тестов
Если (a < 5)
      вывести «Число должно быть больше 4»
Если (k < 1)
      вывести «Число должно быть больше 0»
Вывести результат функции MilRab(n,k)

Функция MilRab(n,k):
      t = n – 1
      s = 0
      Пока (t чётное)
            t = t/2
            s = s + 1
      В цикле по i от 1 до k
            a = случайное число от 2 до n − 1;
            Если (НОД(a,n) > 1)
                  Вывести «Число n составное»
            x = a^t (mod n)
            Если (x = 1) или (x = n – 1)
                  Переход к следующей итерации цикла
            В цикле по g от 1 до s
                  x = x^2 (mod n)
                  Если (x = 1)
                        Вывести «Число n составное»
                  Если (x = n − 1)
                        Выйти из цикла по g
            Если (x ≠ n − 1)
                  Вывести «Число n составное»
      Вывести «Число n, вероятно, простое»
```

3.3 Результаты тестирования программы

Тестирование всех алгоритмов происходит одновременно. Тесты представлены на рисунках 1-3.

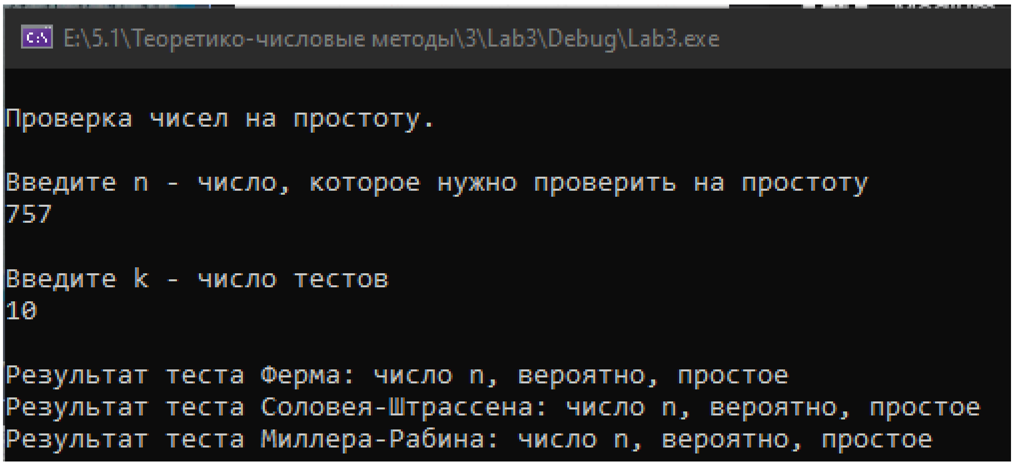
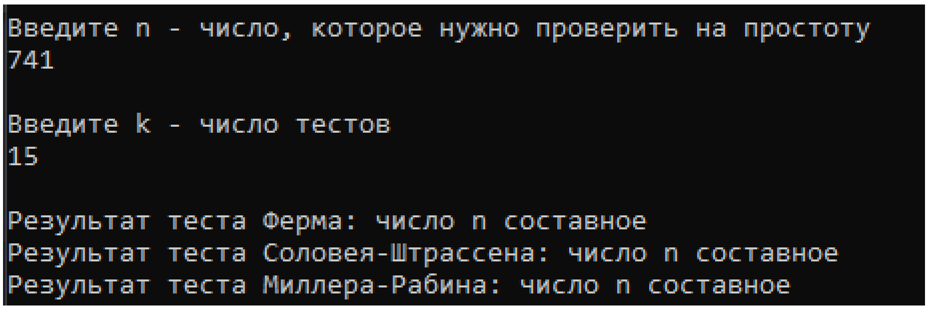


Рисунок 1 – Тест простого числа.

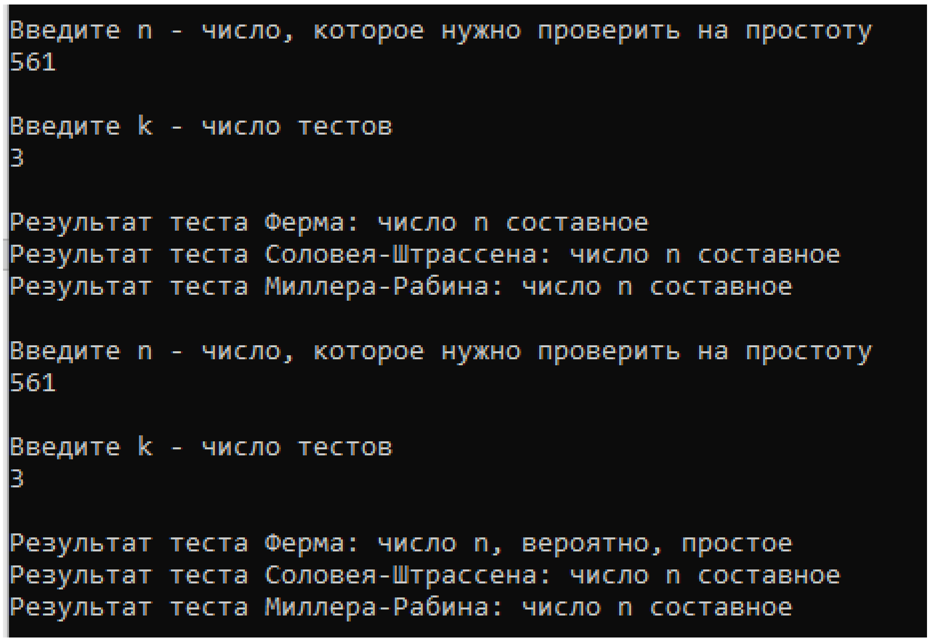


```
Введите n - число, которое нужно проверить на простоту
741

Введите k - число тестов
15

Результат теста Ферма: число n составное
Результат теста Соловея-Штрассена: число n составное
Результат теста Миллера-Рабина: число n составное
```

Рисунок 2 – Тест составного числа



```
Введите n - число, которое нужно проверить на простоту
561

Введите k - число тестов
3

Результат теста Ферма: число n составное
Результат теста Соловея-Штрассена: число n составное
Результат теста Миллера-Рабина: число n составное

Введите n - число, которое нужно проверить на простоту
561

Введите k - число тестов
3

Результат теста Ферма: число n, вероятно, простое
Результат теста Соловея-Штрассена: число n составное
Результат теста Миллера-Рабина: число n составное
```

Рисунок 3 – Тест числа Кармайкла

4 Выводы по работе

В ходе выполнения лабораторной работы были рассмотрены следующие вероятностные тесты проверки числа на простоту: тест Ферма, основанный на малой теореме Ферма, тест Соловея-Штрассена, основанный на критерии Эйлера, и тест Миллера-Рабина, основанный на критерии Миллера.

В практической части лабораторной работы была написана программа на языке C++, содержащая в себе реализацию всех описанных в теоретической части алгоритмов, работоспособность которых продемонстрирована на рисунках 1-3.

5 Код программы

```
#include <iostream>
#include <vector>
#include <boost/multiprecision/cpp_int.hpp>

using namespace std;
using namespace boost::multiprecision;


cpp_int NegativeMod(cpp_int a, cpp_int m) {
    while (a < 0)
        a = a + m;
    return a % m;
}


cpp_int StandartEuclid(cpp_int a, cpp_int b) {
    if (b == 0)
        return a;
    else
        return StandartEuclid(b, a % b);
}


cpp_int Exponentiation(cpp_int x, cpp_int n, cpp_int m) {
    cpp_int N = n, Y = 1, Z = x % m;
    while (N != 0) {
        cpp_int lastN = N % 2;
        N = N / 2;
        if (lastN == 0) {
            Z = (Z * Z) % m;
            continue;
        }
        Y = (Y * Z) % m;
        if (N == 0)
            break;
        Z = (Z * Z) % m;
    }
    Y = Y % m;
    return Y;
}


cpp_int Jac(cpp_int a, cpp_int b) {
    if (StandartEuclid(a, b) != 1)
        return 0;
    else {
        cpp_int r = 1;
        while (a != 0) {
            cpp_int t = 0;
            while (a % 2 == 0) {
                t = t + 1;
                a = a / 2;
            }
            if (t % 2 != 0)
```

```
                if (Exponentiation(b, 1, 8) == 3 || Exponentiation(b, 1, 8) ==
5)
                    r = r * (-1);
            if (Exponentiation(a, 1, 4) == 3 && Exponentiation(b, 1, 4) == 3)
                r = r * (-1);
            cpp_int c = a;
            if (c != 0)
                a = Exponentiation(b, 1, c);
            b = c;
        }
        return r;
    }
}


bool Ferma(cpp_int n, cpp_int k) {
    cpp_int a, r;
    for (int i = 1; i <= k; i++) {
        a = rand() % (n - 3) + 2;
        if (StandartEuclid(a, n) > 1)
            return false;
        r = Exponentiation(a, n - 1, n);
        if (r != 1)
            return false;
    }
    return true;
}


bool SolovSht(cpp_int n, cpp_int k) {
    if (n > 0 && n < 4)
        return true;
    if (n % 2 == 0)
        return false;
    cpp_int a, newa, l;
    for (int i = 1; i <= k; i++) {
        a = rand() % (n - 3) + 2;
        if (StandartEuclid(a, n) > 1)
            return false;
        newa = Exponentiation(a, (n - 1) / 2, n);
        if (newa != 1 && newa != n - 1)
            return false;
        l = NegativeMod(Jac(a, n), n);
        if (newa != l)
            return false;
    }
    return true;
}


bool MilRab(cpp_int n, cpp_int k) {
    if (n == 1 || n == 2 || n == 3)
        return true;
    if (n % 2 == 0)
        return false;
    cpp_int t = n - 1;
```

```
    cpp_int s = 0;
    while (t % 2 == 0) {
        t = t / 2;
        s++;
    }
    for (cpp_int i = 0; i < k; i++) {
        cpp_int a =  rand() % (n - 3) + 2;
        if (StandartEuclid(a, n) > 1)
            return false;
        cpp_int x = Exponentiation(a, t, n);
        if (x == 1 || x == n - 1)
            continue;
        for (cpp_int g = 1; g < s; g++) {
            x = x * x % n;
            if (x == 1)
                return false;
            if (x == n - 1)
                break;
        }
        if (x != n - 1)
            return false;
    }
    return true;
}


int main() {
    setlocale(LC_ALL, "Russian");
    cpp_int c = 100, k;
    cout << "\nПроверка чисел на простоту.";
    cpp_int n;
    while (c != 0) {
        cout << "\n\nВведите n - число, которое нужно проверить на простоту\n";
        cin >> n;
        if (n < 5) {
            cerr << "\nn должно быть больше 4\n";
            continue;
        }
        cout << "\nВведите k - число тестов\n";
        cin >> k;
        if (k < 1) {
            cerr << "\nk должно быть больше 0\n";
            continue;
        }
        cout << "\nРезультат теста Ферма: число n";
        srand(time(0));
        if (Ferma(n, k))
            cout << ", вероятно, простое";
        else
            cout << " составное";
        cout << "\nРезультат теста Соловея-Штрассена: число n";
        if (SolovSht(n, k))
            cout << ", вероятно, простое";
        else
            cout << " составное";
        cout << "\nРезультат теста Миллера-Рабина: число n";
```

```
        if (MilRab(n, k))
            cout << ", вероятно, простое";
        else
            cout << " составное";
    }
    return 0;
}
```