

МИНОБРАЗОВАНИЯ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

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

УТВЕРЖДАЮ

Зав.кафедрой,
доцент, к. ф.-м. н.

_____ Л. Б. Тяпаев

ОТЧЕТ О ПРАКТИКЕ

студента 2 курса 221 группы факультета КНиИТ
Устюшина Богдана Антоновича

вид практики: учебная практика, рассредоточенная
кафедра: дискретной математики и информационных технологий
курс: 2

семестр: 4

продолжительность: 18 нед., с 08.02.2023 г. по 12.06.2023 г.

Руководитель практики от университета,

доцент, к. ф.-м. н.

В. А. Молчанов

Тема практики: «Проверка чисел на простоту»

СОДЕРЖАНИЕ

1	Введение	4
2	Теоретические сведения из теории чисел	6
2.1	Используемые понятия	6
2.2	Проверка числа на простоту	7
2.3	Тест и теорема Ферма	8
2.4	Тест Соловея-Штрассена	9
2.5	Тест Миллера-Рабина	11
3	Результаты работы	13
3.1	Псевдокоды алгоритмов	13
3.2	Результаты тестирования	15
	ЗАКЛЮЧЕНИЕ	16
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	17
	Приложение А Приложение А	19
	Приложение Б Приложение Б	26

1 Введение

Теория чисел является интенсивно развивающейся наукой с большим количеством приложений. Сегодня данная область математики как никогда востребована: она является фундаментом для всей криптографии, что позволяет говорить о важности и необходимости изучения методов и теоретических основ данной дисциплины.

В современном мире криптография используется повсеместно. Например, она применяется для безопасной отправки паролей по сетям при совершении покупок по интернету. Серверы банков и почтовые клиенты сохраняют пароли пользователей с помощью методов шифрования. Они обеспечивают безопасную передачу всей информации в интернете вещей для аутентификации людей и устройств, а также при установлении связи между только устройствами.

Если бы все криптографические функции перестали осуществляться, современная жизнь прекратилась бы в том ее виде, который нам известен. Перестали бы проходить банковские транзакции, прекратился бы интернет-трафик, а сотовые телефоны не смогли бы работать. Вся наша важная информация стала бы общедоступной, что нанесло бы непоправимый вред, попади она в руки злоумышленников.

Криптография позволяет предотвратить такие угрозы, обеспечивая безопасность информации и связи с помощью правил, которые позволяют получать данные и обрабатывать их только тем, кто имеет соответствующий доступ. Для реализации высокоуровневой защиты систем, выполняющих миллиарды транзакций, применяются современные криптографические методы, представляющие собой неотъемлемую часть безопасной, но удобной связи, которой мы пользуемся в повседневной жизни [1, 2].

Для целей криптографии (как для практической реализации и обоснования стойкости криптографических средств, так и для разработки методов их вскрытия) необходимо в первую очередь повышать эффективность следующих методов и алгоритмов:

1. Алгоритмы проверки простоты целых чисел
2. Методы факторизации (т. е. методы поиска разложения целых чисел на множители);
3. Вычисления, использующие эллиптические кривые над конечными по-

лями;

4. Методы разложения многочленов на множители над конечными полями и над полем рациональных чисел;
5. Алгоритмы дискретного логарифмирования;
6. Способы решения систем линейных уравнений над конечными полями;
7. Алгоритмы для выполнения арифметических операций с большими целыми числами;
8. Алгоритмы полиномиальной арифметики

Среди большого количества разнообразных задач теории чисел и связанных с ними криптографических в данном реферате будет рассмотрена задача определения простоты числа, для которой в ходе развития математики открывалось и разрабатывалось несколько алгоритмов.

Задачами данного реферата будет:

1. Показать теоретическую основу трёх алгоритмов: теста Ферма, теста Соловея-Штрассена и теста Миллера-Рабина проверки чисел на простоту, а также произвести исходя из этого оценку сложности алгоритмов
2. Раскрыть сами алгоритмы
3. Показать их реализацию на языке псевдокода
4. Реализовать и протестировать алгоритмы на языке программирования C#

2 Теоретические сведения из теории чисел

2.1 Используемые понятия

В данной работе для освещения алгоритмов будут использоваться некоторые понятия из теории чисел, которые могут быть незнакомы или не до конца понятны. Здесь будут описаны лишь базовые понятия, не соотносящиеся с темой доклада: остальные понятия будут описаны и раскрыты ниже.

Определение 1. *Сравнение двух целых чисел по модулю натурального числа $p \in \mathbb{N}$ — математическая операция, позволяющая ответить на вопрос о том, дают ли два выбранных целых числа при делении на p один и тот же остаток.*

Любое целое число при делении на p дает один из p возможных остатков: число от 0 до $p - 1$; это значит, что все целые числа можно разделить на p групп, каждая из которых отвечает определённому остатку от деления на p . Эти группы называются классами вычетов по модулю p , а содержащиеся в них целые числа — вычетами по модулю p [3].

Для фиксированного натурального числа m отношение сравнимости по модулю p обладает следующими свойствами:

1. свойством рефлексивности: для любого целого a справедливо $a \equiv a \pmod{p}$;
2. свойством симметричности: если $a \equiv b \pmod{p}$, то $b \equiv a \pmod{p}$;
3. свойством транзитивности: если $a \equiv b \pmod{p}$ и $b \equiv c \pmod{p}$, то $a \equiv c \pmod{p}$.

Таким образом, отношение сравнимости по модулю m является отношением эквивалентности на множестве целых чисел [4].

Определение 2. *Функция Эйлера $\varphi(n)$ — мультипликативная арифметическая функция, значение которой равно количеству натуральных чисел, меньших либо равных $n \in \mathbb{N}$ и взаимно простых с ним.*

Например, для числа 36 существует 12 меньших его и взаимно простых с ним чисел (1, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35), поэтому $\varphi(36) = 12$ [5].

Определение 3. *Целое число a называется квадратичным вычетом по модулю $m \in \mathbb{N}$, если разрешимо сравнение:*

$$x^2 \equiv a \pmod{p}.$$

При этом числа a и p взаимно просты.

Если указанное сравнение не разрешимо (т.е. такого x^2 , который бы при делении на m давал остаток a нет), то число a называется **квадратичным невычетом** по модулю m . Решение приведенного выше сравнения означает извлечение квадратного корня в кольце классов вычетов [6].

Определение 4. *Символ Лежандра – функция со следующими значениями:*

- $\left(\frac{a}{p}\right) = 0$, если a делится на p ;
- $\left(\frac{a}{p}\right) = 1$, если a является квадратичным вычетом по модулю p
- $\left(\frac{a}{p}\right) = -1$, если a является квадратичным невычетом по модулю p .

2.2 Проверка числа на простоту

Вопрос определения того, является ли натуральное число N простым, известен как проблема простоты.

Определение 5. *Тестом простоты (или проверкой простоты) называется алгоритм, который, приняв на входе число N , позволяет либо не подтверждать предположение о составности числа, либо точно утверждать его простоту. Во втором случае он называется истинным тестом простоты.*

Таким образом, тест простоты представляет собой только гипотезу о том, что если алгоритм не подтвердил предположение о составности числа N , то это число может являться простым с определённой вероятностью. Это определение подразумевает меньшую уверенность в соответствии результата проверки истинному положению вещей, нежели истинное испытание на простоту, которое даёт математически подтверждённый результат.

Существующие алгоритмы проверки числа на простоту могут быть разделены на две категории:

1. Истинные тесты простоты
2. Вероятностные тесты простоты

Истинные тесты результатом вычислений всегда выдают факт простоты либо составности числа, вероятностный тест даёт ответ о составности числа либо его несоставности с некоторой вероятностью ε . Если сказать проще,

то вероятностный алгоритм говорит, что число скорее всего не является составным, однако в итоге оно может оказаться как простым, так и составным.

Числа, удовлетворяющие вероятностному тесту простоты, но являющиеся составными, называются псевдопростыми. Одним из примеров таких чисел являются числа Кармайкла. Также можно назвать числа Эйлера-Якоби для теста Соловея-Штрассена и псевдопростые числа Люка [7].

Очевидно, что самым важным показателем для истинностных тестов является скорость, а для вероятностных – соотношение скорости и вероятности. Мы могли бы довольствоваться обычным истинностным тестом на простоту со скоростью $O(\sqrt{n})$, однако продолжаем искать его альтернативы. В данной работе будет рассмотрено три таких.

2.3 Тест и теорема Ферма

Первым рассмотрим тест Ферма: он основан на малой теореме Ферма.

Перед доказательством теоремы примем без доказательства следующую лемму:

Лемма 1. *Для любого простого числа p и целого числа k , не кратного p , произведения k и чисел $1, 2, 3, \dots, p-1$ при делении на p в остатке дают те же самые числа $1, 2, 3, \dots, p-1$, возможно, записанные в некотором другом порядке.*

Теорема 1. Малая Теорема Ферма. *Если p – простое число, и $a \in \mathbb{Z}$ и не делится на p , то $a^{p-1} - 1$ делится на p . Иными словами, $a^{p-1} \equiv 1 \pmod{p}$*

Доказательство. Поскольку согласно вышеприведённой лемме остатки от деления чисел $a, 2a, 3a, \dots, (p-1)a$ – это с точностью до перестановки числа $a \cdot 2a \cdot 3a \dots (p-1)a \equiv 1 \cdot 2 \cdot 3 \dots (p-1) \pmod{p}$. Отсюда $a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}$. Последнее соотношение можно сократить на $(p-1)!$, поскольку все сомножители являются числами, взаимно простыми с основанием p , в результате получаем требуемое утверждение $a^{p-1} \equiv 1 \pmod{p}$ \square

По Теореме Ферма, если p – простое число, тогда для любого a справедливо следующее равенство $a^{p-1} \equiv 1 \pmod{p}$. Отсюда мы можем вывести правило теста Ферма на проверку простоты числа: возьмем случайное $a \in \{1, \dots, p-1\}$ и проверим будет ли соблюдаться равенство $a^{p-1} \equiv 1 \pmod{p}$. Если равенство не соблюдается, значит p составное.

Однако существуют составные числа, для которых сравнение $a^{p-1} \equiv 1 \pmod{p}$ выполняется для одного из a или всех a , взаимно простых с p . В первом случае число p – псевдопростое по отношению к a . Во втором случае — это числа Кармайкла (псевдопростые по отношению ко всем числам, также так называемые лжецы Ферма). Чисел Кармайкла — бесконечное множество, наименьшее число Кармайкла — 561. Тем не менее, тест Ферма довольно эффективен для обнаружения составных чисел [8].

Алгоритм Ферма:

1. Выбрать случайное целое число $a, 2 \leq a \leq p - 2$
2. Вычислить $r = a^{p-1} \pmod{p}$
3. Если $r = 1$, то ответ «Число, вероятно, простое», иначе ответ «Число составное».

Сложность теста Ферма равна $O(\log^3 n)$ при обычном умножении «в столбик» и $O(t \cdot \log^2 n \cdot \log \log n)$ при умножении алгоритмом Шенхаге-Штрассена, которые выполняется не за n^2 , а за $n^{\log_2 7}$ [9].

2.4 Тест Соловея-Штрассена

Мы можем усовершенствовать тест, сказав, что p — простое тогда и только тогда, когда решениями $x^2 \equiv 1 \pmod{p}$ являются $x = \pm 1$

Таким образом, если p проходит тест Ферма, то есть $a^{p-1} \equiv 1$, тогда мы проверяем еще чтобы $a^{(p-1)/2} \equiv \pm 1$, поскольку $a^{(p-1)/2}$ — это квадратный корень 1.

В основе теста Соловея-Штрассена лежит следующая теорема, называемая критерием Эйлера:

Теорема 2. Критерий Эйлера Пусть $p > 2, p \in \mathbb{P}$ (то есть множеству простых чисел). Число a , взаимнопростое с p , является квадратичным вычетом по модулю p тогда и только тогда, когда $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$, и является квадратичным невычетом по модулю p тогда и только тогда, когда $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$. То есть:

$$a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right) \pmod{p}$$

К сожалению, такие числа, как, например 1729 — третье число Кармайкла — до сих пор могут обмануть этот улучшенный тест. Однако мы можем продолжать проводить итерации. Это значит, что пока это будет возможно,

мы будем уменьшать экспоненту вдвое, до тех пор пока не дойдем до какого-либо числа, помимо 1. Если мы получим в итоге что-то, кроме -1, тогда p будет составным [10].

В 2005 году на Международной конференции «Informational Technologies» А. А. Балабанов, А. Ф. Агафонов, В. А. Рыку предложили модернизированный тест Соловея-Штрассена. Тест Соловея-Штрассена основан на вычислении символа Лежандра, что занимает время. Идея улучшения состоит в том, чтобы, в соответствии с теоремой квадратичной взаимности Гаусса, перейти к вычислению величины, являющейся обратной символу Лежандра. Это является более простой процедурой [11].

Теорема 3. Квадратичный закон взаимности – ряд утверждений, касающихся разрешимости квадратичного сравнения по модулю. Согласно этому закону, если p, q – нечётные простые числа и хотя бы одно из них имеет вид $4k + 1$, то два сравнения

$$x^2 \equiv q \pmod{p},$$

$$x^2 \equiv p \pmod{q}$$

либо оба имеют решения для x , либо оба не имеют. Поэтому в названии закона используется слово «взаимность». Если же p, q оба имеют вид $4k + 3$, то решение имеет одно и только одно из указанных сравнений [12].

На основании теоремы 2 и по аналогии с алгоритмом Ферма получаем алгоритм проверки числа тестом Соловея-Штрассена:

Тест Соловея-Штрассена:

1. Выбрать случайное целое число a , что $2 \leq a \leq p - 2$
2. Вычислить $r = a^{\frac{p-1}{2}} \pmod{p}$
3. Если $r \neq 1$ и $r \neq p - 1$, то «Число p составное».
4. Вычислить символ Лежандра $\left(\frac{a}{p}\right)$
5. При $r \neq s \pmod{p}$ результат «Число p составное». Иначе «Число p , вероятно, простое».

Сложность теста Соловея-Штрассена определяется сложностью вычисления символа Лежандра (так как это самая содержательная часть алгоритма) и равна $O(\log^3 n)$ [9].

2.5 Тест Миллера-Рабина

Алгоритм Миллера-Рабина является модификацией алгоритма Миллера, разработанного Гари Миллером в 1976 году. Алгоритм Миллера является детерминированным, но его корректность опирается на недоказанную расширенную гипотезу Римана. Майкл Рабин модифицировал его в 1980 году. Алгоритм Миллера — Рабина не зависит от справедливости гипотезы, но является вероятностным.

Как и тесты Ферма и Соловея — Штрассена, тест Миллера — Рабина опирается на проверку ряда равенств, которые выполняются для простых чисел. Если хотя бы одно такое равенство не выполняется, это доказывает что число составное. Для теста Миллера — Рабина используется следующее утверждение:

Теорема 4. Пусть p — простое число и $p - 1 = 2^s d$, где d — нечётно. Тогда для любого a из \mathbb{Z}_n выполняется хотя бы одно из условий:

- $a^d \equiv 1 \pmod{p}$
- Существует целое число $r < s$ такое, что $a^{2^r d} \equiv -1 \pmod{p}$

На основании данного утверждения и строится весь алгоритм определения простоты числа.

Также существует **Теорема Рабина**, определяющая верхнюю границу числа свидетелей простоты [13].

Теорема 5. Теорема Рабина. Пусть p — составное нечётное число. Тогда оно имеет не более $\frac{\varphi(p)}{4}$ различных свидетелей простоты.

При случайно выбранном a вероятность ошибочно принять составное число за простое составляет 25%, но её можно уменьшить, выполнив проверки для других a [14].

У нечётных составных чисел n существует, согласно теореме Рабина, не более $\varphi(n)/4$ свидетелей простоты, таким образом вероятность того, что случайно выбранное число a окажется свидетелем простоты, меньше $1/4$. Идея теста заключается в том, чтобы проверять для случайно выбранных чисел $a < n$, являются ли они свидетелями простоты числа n . Если найдётся свидетель того, что число составное, то число действительно является составным. Если было проверено k чисел, и все они оказались свидетелями

простоты, то число считается простым. Для такого алгоритма вероятность принять составное число за простое будет меньше $(1/4)^k$.

Алгоритм Миллера-Рабина параметризуется количеством раундов r . Рекомендуется брать r порядка величины $\log_2(n)$, где n — проверяемое число.

В силу вышеописанного, получаем **алгоритм Миллера-Рабина**:

1. Для данного p находятся такие целое число s и целое нечетное число t , что $p - 1 = 2^s t$.
2. Выбирается случайное число a , $1 < a < p$
3. — Если a не является свидетелем простоты числа p , то выдаётся ответ « p — составное», алгоритм завершается
— Иначе идём к шагу 2.
4. При нахождении r свидетелей простоты получаем ответ « p — вероятно простое», конец.

Считая, что время умножения логарифмическое, используя быстрое умножение по модулю, сложность работы алгоритма $O(k \log^3 n)$, где k — количество раундов. Таким образом, время работы алгоритма полиномиально.

Также время работы алгоритма можно сократить, используя быстрое преобразование Фурье [15].

3 Результаты работы

3.1 Псевдокоды алгоритмов

В псевдокоде и реализации программ на языке C# также присутствует дополнительная проверка на взаимную простоту чисел a и p , что усиливает стойкость алгоритма, незначительно увеличивая время его работы.

Для окончательной реализации вышеописанных алгоритмов сначала опишем их псевдокоды, затем перенесём их на наш язык программирования (C#).

Алгоритм Ферма: [16]

Вход:

$p \in \mathbb{Z}$, проверяемое на простоту

$a \leftarrow \{1, 2, \dots, p-1\}$ (параметр)

k – количество итераций тестов с разными параметрами

Выход:

0 – число составное, 1 – число вероятно простое

Алгоритм:

Проверка тривиальных случаев, не охватываемых алгоритмом: $p = 1, 2$

for k **do**

$a \leftarrow \{1, 2, \dots, p-1\}$ ▷ (без повторений a)

if $(a, p) \neq 1$ **then return** 0

end if

if $a^{p-1} \not\equiv 1 \pmod{p}$ **then return** 0

end if

end for

return 1

Алгоритм Соловея-Штрассена: [17, 18]

Вход:

$p \in \mathbb{Z}$, проверяемое на простоту

$a \leftarrow \{1, 2, \dots, p-1\}$ (параметр)

k – количество итераций тестов с разными параметрами

Выход:

0 – число составное, 1 – число вероятно простое

Алгоритм:

Проверка тривиальных случаев, не охватываемых алгоритмом: $p = 1, 2$

```

for A:  $k$  раз do
     $a \leftarrow \{1, 2, \dots, p-1\}$  ▷ (без повторений  $a$ )
    if  $(a, p) \neq 1$  then
        return 0
    end if
    if  $a^{\frac{p-1}{2}} \not\equiv \left(\frac{a}{p}\right) \pmod{p}$  then
        return 0
    end if
    return 1

```

end for

Алгоритм Миллера-Рабина: [19]

Вход:

$p \in \mathbb{Z}$, проверяемое на простоту

$a \leftarrow \{1, 2, \dots, p-1\}$ (параметр)

k – количество итераций тестов с разными параметрами

Выход:

0 – число составное, 1 – число вероятно простое

Алгоритм:

Проверка тривиальных случаев, не охватываемых алгоритмом: $p = 1, 2$

Представить $p-1$ в виде $2^s \cdot t$, последовательно деля на 2:

while t нечётное **do**

$t = t/2$

$s++$

end while

for A: k раз **do**

if $(a, p) \neq 1$ **then**

return 0

end if

$a \leftarrow \{1, 2, \dots, p-1\}$

▷ (без повторений a)

$x \leftarrow a^t \pmod{p}$

if $x \equiv 1$ **or** $x \equiv p-1$ **then**

Перейти к следующей итерации A

end if

for B: $s-1$ раз **do** $x \leftarrow x^2 \pmod{p}$

```

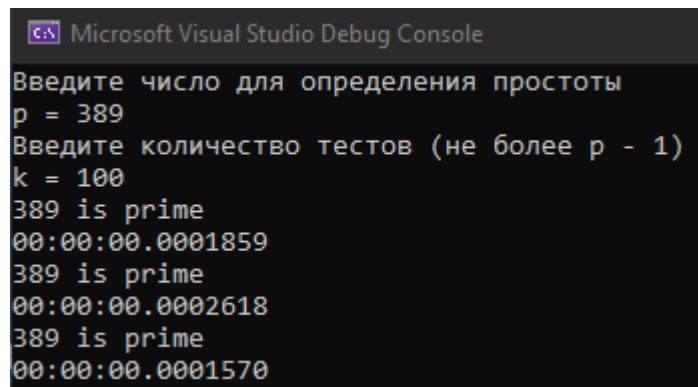
    if  $x = 1$  then return 0
  end if
  if  $x = p - 1$  then Перейти к следующей итерации A
  end if
  return 0
end for
return 1
end for

```

3.2 Результаты тестирования

По вышеописанному псевдокоду был написан код на языке C#, приведённый в приложении А. Также были зафиксированы результаты работы алгоритмов для чисел разного порядка и разного количества тестов. Например, для чисел порядка 10^2 создавались 100 уникальных случайных чисел в диапазоне $0 < a < p$ (т.е. такого же порядка, какого и число). Для числа 10^8 создавалось 10^8 чисел, которые также были уникальны и попадали в вышеуказанный диапазон. Время измерялось только на действии алгоритма, т.е. лишние действия не попадали в время измерения алгоритма и не «портили» его результат.

На рисунке 1 приведён результат тестирования числа порядка 10^2 , остальные скриншоты результатов приведены в приложении Б.



```

Microsoft Visual Studio Debug Console
Введите число для определения простоты
p = 389
Введите количество тестов (не более p - 1)
k = 100
389 is prime
00:00:00.0001859
389 is prime
00:00:00.0002618
389 is prime
00:00:00.0001570

```

Рисунок 1 – Результат вычислений для числа порядка 10^2

ЗАКЛЮЧЕНИЕ

В заключение реферата хочется отметить, что вышеописанные алгоритмы проверки чисел на простоту по-прежнему имеют актуальность для криптографов (особенно тех, кто только начинает изучать основы данной области компьютерных наук). Алгоритмы определения простоты числа являются одними из фундаментальных в криптографии, и прорывы в области их изучения могут изменить наше представление о безопасности наших методов шифрации и дешифрации передаваемой информации.

Что же касается эффективности алгоритмов, можно утверждать, что алгоритм Миллера-Рабина является наиболее эффективным с точки зрения как вероятности распознавания составных чисел, так и скорости работы. Тест Соловея-Штрассена, несмотря на эффективное вычисление символа Якоби, является не столь быстрым. Тест Ферма не распознаёт ни одного числа Кармайкла в отличие от алгоритма Миллера-Рабина, что позволяет говорить об отказоустойчивости последнего.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Статья о криптографии.
URL: <https://www.symmetron.ru/articles/zachem-nuzhna-kriptografiya/>
- 2 Реферат по алгоритмам Миллера-Рабина и тесту Ферма.
URL: https://studentlib.com/chitat/referat-256890-test_chisla_na_prostotu.html
- 3 Бухштаб А. А. Теория чисел. — М.: «Просвещение», 1966. — 384 с.
- 4 Сизый С. В. §4. Теория сравнений // Лекции по теории чисел. — М.: Физматлит, 2008. — с. 87-88, 91
- 5 Эйлера функция // Математическая энциклопедия (в 5 томах). — М.: Советская Энциклопедия, 1985. — Т. 5. — С. 934.
- 6 Виноградов И. М. Основы теории чисел. — М.—Л.: ГИТТЛ, 1952. — 180 с.
- 7 Статья Википедии об тестах простоты.
URL: https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%BF%D1%80%D0%BE%D1%81%D1%82%D0%BE%D1%82%D1%8B
- 8 Видео о тесте Ферма.
URL: https://www.youtube.com/watch?v=jbias_aHNUQ
- 9 Маховенко Е. Б. «Теоретико-числовые методы в криптографии: Учебное пособие» — М.: Гелиос АРВ, 2006, с. 169-191
- 10 Михелович М.Х. Теория чисел. 1967. §3.9. с. 107—109
- 11 Балабанов А. А., Агафонов А. Ф., Рыку В. А. «Алгоритм быстрой генерации ключей в криптографической системе RSA» — Вестник научно-технического развития, 2009 № 7(23). — С. 11.
- 12 Карл Фридрих Гаусс. Труды по теории чисел / Общая редакция академика И. М. Виноградова, комментарии члена-корр. АН СССР Б. Н. Делоне. — М.: Изд-во АН СССР, 1959. — С. 126. — 297 с. — (Классики науки).
- 13 Ознакомительная информация с тестом Миллера-Рабина
URL: <https://foxford.ru/wiki/informatika/test-prostoty-millera-rabina>

- 14 Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ. — 3. — Москва: Вильямс, 2013. — С. 1012—1015. — 1328 с.
- 15 Брюс Шнайер. Прикладная Криптография. — Москва: Триумф, 2013. — С. 298. — 816 с.
- 16 Презентация на тему «Генерация простых чисел»
URL: <http://www.myshared.ru/slide/1320763/>
- 17 Статья о реализации теста на простоту Соловея-Штрассена на языке C#
URL: <https://cyberleninka.ru/article/n/realizatsiya-veroyatnostnogo-testa-na-prostotu-soloveya-shtrassena-na-yazyke-c>
- 18 Коблиц Н. Курс теории чисел и криптографии. — 2-ое. — М.: Научное издательство ТВП, 2001. — С. 149—160. — 254 с.
- 19 Псевдокод и реализация алгоритма Миллера-Рабина.
URL: <https://vscode.ru/prog-lessons/test-millera-rabina-na-prostotu-chisla.html>

ПРИЛОЖЕНИЕ А

Приложение А

```
using System;
using System.Diagnostics;
using System.Numerics;
using System.Security.Cryptography;

namespace Main
{
    class Program
    {
        //Вычисление наибольшего общего делителя
        static long GCD(long a, long b)
        {
            if (b == 0)
            {
                return a;
            }
            return GCD(b, a % b);
        }

        //Быстрое возведение в степень по модулю
        public static long BinPow(long a, long n, long mod)
        {
            if (n == 0)
            {
                return 1;
            }
            if (n % 2 == 1)
            {
                return (BinPow(a, n - 1, mod) * a) % mod;
            }
            else
```

```

    {
        long b = BinPow(a, n / 2, mod);
        return (b * b) % mod;
    }
}

//Вычисление символа Якоби
public static long Yacobi(long a, long p)
{
    int r = 1;
    while (a != 0)
    {
        int t = 0;
        while (a % 2 == 0)
        {
            t++;
            a /= 2;
        }
        if (t % 2 != 0)
        {
            long temp = p % 8;
            if (temp == 3 || temp == 5)
            {
                r = -r;
            }
        }
        long a4 = a % 4, b4 = p % 4;
        if (a4 == 3 && b4 == 3)
        {
            r = -r;
        }
        long c = a;
        a = p % c;
        p = c;
    }
}

```

```

    }
    return r;
}
//Тест простоты "в лоб"
static bool IsPrime(long n)
{
    for (long i = 2; i <= Math.Sqrt(n); i++)
        if (n % i == 0)
            return false;
    return true;
}
//Тест простоты Ферма
static bool FermatTest(long k, long p, long [] a)
{
    if (p == 1)
        return false;
    if (p == 2)
        return true;
    for (int i = 0; i < k; i++)
    {
        if (GCD(a[i], p) != 1)
            return false;
        if (BinPow(a[i], p - 1, p) != 1)
            return false;
    }
    return true;
}
//Тест простоты Соловея-Штрассена
static bool SolovayStrassenTest(long k, long p, long[] a)
{
    if (p == 1)
        return false;
    if (p == 2)
        return true;

```

```

    long x;
    long y;
    for (long i = 0; i < k; i++)
    {
        if (GCD(a[i], p) != 1)
            return false;
        else
        {
            x = BinPow(a[i], (p - 1) / 2, p);
            y = Yacobi(a[i], p);
            if (y == -1)
                y += p;
            if (x != y)
                return false;
        }
    }
    return true;
}

//Тест простоты Миллера-Рабина
static bool MillerRabbinTest(long k, long p, long[] a)
{
    if (p == 1)
        return false;
    if (p == 2)
        return true;
    long t = p - 1;
    long s = 0;
    long x;
    while (t % 2 == 0)
    {
        t /= 2;
        s += 1;
    }
    for (int i = 0; i < k; i++)

```

```

{
    if (GCD(a[i], p) != 1)
        return false;
    // представим  $p - 1$  в виде  $(2^s) \cdot t$ , где  $t$  нечётно,
    ↪ это можно сделать последовательным делением  $p$ 
    ↪  $- 1$  на 2
    x = BinPow(a[i], t, p);
    if (x == 1 || x == p - 1)
        continue;
    for (int j = 0; j < s - 1; j++)
    {
        x = BinPow(x, 2, p);
        if (x == 1)
            return false;
        if (x == p - 1)
            break;
    }
    if (x != p - 1)
        return false;
}
return true;
}

static void Main()
{
    var sw = new Stopwatch();
    var rand = new Random();
    HashSet<long> testNumbers = new HashSet<long>();
    Console.Write("Введите число для определения
    ↪ простоты\n" + "p = ");
    long p = long.Parse(Console.ReadLine());
    Console.Write("Введите количество тестов (не более p -
    ↪ 1)\n" + "k = ");
    long k = long.Parse(Console.ReadLine());

```

```

if (k >= p)
{
    Console.WriteLine("Слишком большое кол-во тестов
        ↪ (не более p - 1)");
    return;
}
if (k == 0)
{
    Console.WriteLine("Нет тестов");
    return;
}
long[] a = new long[k];
while (testNumbers.Count != k)
{
    testNumbers.Add(rand.NextInt64(p - 1) + 1);
}
int j = 0;
foreach (long i in testNumbers)
{
    a[j] = i;
    j++;
}
bool ans;
sw.Start();
ans = FermatTest(k, p, a);
sw.Stop();
Console.WriteLine(ans == true ? $"{p} is prime" :
    ↪ $"{p} is not prime");
Console.WriteLine(sw.Elapsed);
sw.Reset();
sw.Start();
ans = SolovayStrassenTest(k, p, a);
sw.Stop();

```



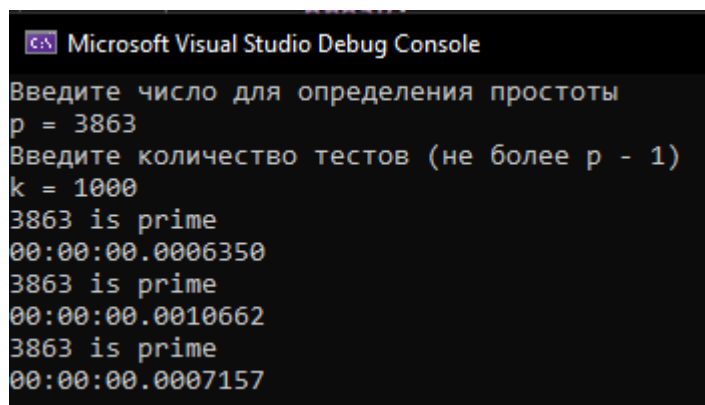
```

        Console.WriteLine(ans == true ? $"{p} is prime" :
            ↳ $"{p} is not prime");
        Console.WriteLine(sw.Elapsed);
        sw.Reset();
        sw.Start();
        ans = MillerRabbinTest(k, p, a);
        sw.Stop();
        Console.WriteLine(ans == true ? $"{p} is prime" :
            ↳ $"{p} is not prime");
        Console.WriteLine(sw.Elapsed);
    }
}
}

```

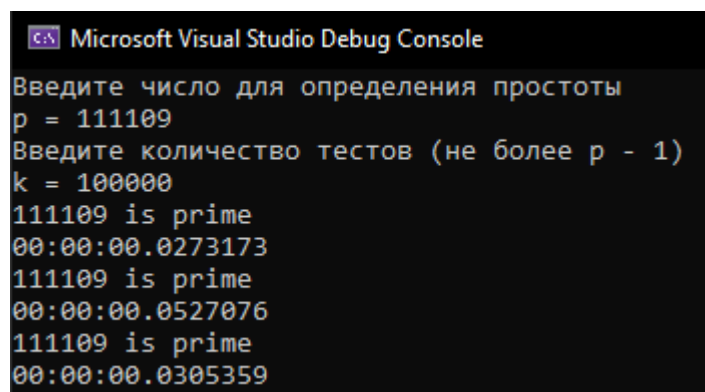
ПРИЛОЖЕНИЕ Б

Приложение Б



```
Microsoft Visual Studio Debug Console
Введите число для определения простоты
p = 3863
Введите количество тестов (не более p - 1)
k = 1000
3863 is prime
00:00:00.0006350
3863 is prime
00:00:00.0010662
3863 is prime
00:00:00.0007157
```

Рисунок 2 – Результат вычислений для числа порядка 10^3



```
Microsoft Visual Studio Debug Console
Введите число для определения простоты
p = 111109
Введите количество тестов (не более p - 1)
k = 100000
111109 is prime
00:00:00.0273173
111109 is prime
00:00:00.0527076
111109 is prime
00:00:00.0305359
```

Рисунок 3 – Результат вычислений для числа порядка 10^5

```
Microsoft Visual Studio Debug Console
Введите число для определения простоты
p = 1154789
Введите количество тестов (не более p - 1)
k = 1000000
1154789 is prime
00:00:00.3754396
1154789 is prime
00:00:00.6689193
1154789 is prime
00:00:00.4082267
```

Рисунок 4 – Результат вычислений для числа порядка 10^6

```
Microsoft Visual Studio Debug Console
Введите число для определения простоты
p = 86028157
Введите количество тестов (не более p - 1)
k = 10000000
86028157 is prime
00:00:05.0197128
86028157 is prime
00:00:08.8718545
86028157 is prime
00:00:05.3081927
```

Рисунок 5 – Результат вычислений для числа порядка 10^7

```
Microsoft Visual Studio Debug Console
Введите число для определения простоты
p = 160481219
Введите количество тестов (не более p - 1)
k = 100000000
160481219 is prime
00:00:51.5135436
160481219 is prime
00:01:30.6448393
160481219 is prime
00:00:52.4652410
```

Рисунок 6 – Результат вычислений для числа порядка 10^8