

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ФАКУЛЬТЕТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК**

«Рекомендовать к защите»

Заведующий кафедрой прикладной  
математики и компьютерных наук

\_\_\_\_\_ Соколов Е. В.  
протокол заседания кафедры № \_\_\_\_\_  
от «\_\_\_\_\_» \_\_\_\_\_ 20\_\_ г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА  
(МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)**

**ПСЕВДОПРОСТЫЕ ЧИСЛА ФРОБЕНИУСА  
БЕЗ БОЛЬШИХ МНОЖИТЕЛЕЙ**

Направление подготовки:	02.04.01 Математика и компьютерные науки
Направленность образовательной программы:	Математические методы в компьютерных науках
Выпускную квалификационную работу выполнил:	студент 2 курса очной формы обучения _____ Голубев Владислав Дмитриевич
Руководитель выпускной квалификационной работы:	доцент кафедры прикладной математики и компьютерных наук, кандидат физико-математических наук _____ Хашин Сергей Иванович

# СОДЕРЖАНИЕ

Введение .....	3
Глава 1. Теоретическая часть .....	5
1.1 Простые и составные числа .....	5
1.2 Символы Якоби и Лежандра .....	8
1.3 Поле Галуа .....	10
1.4 Квадратичные иррациональности .....	11
1.5 Тест Фробениуса .....	12
1.6 Согласованные простые делители .....	15
1.7 Обход дерева .....	16
1.8 Описание библиотеки MPIR .....	18
Глава 2. Практическая часть .....	20
2.1 Алгоритм поиска псевдопростых по Фробениусу чисел .....	20
2.2 Результаты работы .....	21
Заключение .....	24
Список использованных источников .....	25
Приложение .....	26

## ВВЕДЕНИЕ

Развитие теории чисел тесно и непосредственно связано с развитием целого ряда разделов математики. Теория чисел не только широко использует методы, разработанные в смежных математических дисциплинах, но и сама влияет на формирование этих дисциплин.

В трудах Евклида теоретико-числовые исследования занимают сравнительно небольшое место, однако уже у него мы встречаем ряд основных положений теории делимости и хотя простой, но чрезвычайно важный результат: бесконечность множества простых чисел.

Греческим математикам был известен способ выделения простых чисел из натурального ряда, получивший название решето Эратосфена.

В 18-ом веке Л. Эйлер значительно продвинул вперед развитие теории чисел. Он обобщил основной результат Ферма для случая делимости на составные числа, создал общую теорию так называемых степенных вычетов.

Большое значение имеют работы великого русского математика П.Л. Чебышева. Его работы по теории простых чисел являются основой для целого ряда последующих исследований в этой области.

Проблема определения простоты числа интересна, как с научной точки зрения, так как до сих пор не найдено единой аналитической записи для всех простых чисел, так и с практической точки зрения для применения в криптосистемах с открытым ключом.

Простые числа исследуются уже довольно долго, но наибольшее развитие тема вероятностных проверок приобрела во второй половине двадцатого века, из-за необходимости генерировать большие простые числа для криптосистем.

На современном этапе развития информационного общества применение простых чисел получило широкое распространение в области защиты информации, прежде всего это вызвано изобретением криптографии с асимметричным ключом, применяющейся в алгоритмах электронной цифровой подписи.

На сегодняшний день известно достаточно много алгоритмов проверки чисел на простоту. Их можно разделить на две категории: вероятностные тесты простоты и истинные тесты простоты. Истинные тесты простоты всегда дают факт простоты либо составности числа. Вероятностные тесты определяют, является ли число простым или нет с некоторой долей вероятности. Иными словами, вероятностный тест говорит, что число, скорее всего, является простым, однако оно может оказаться как простым, так и составным.

В теории чисел псевдопростым числом Фробениуса называется псевдопростое число, прошедшее тест принадлежности к вероятно простым числам, разработанный Джоном Грантамом (Jon Grantham) в 1996 году.

Актуальность выбранной темы обусловлена, во-первых, развитием области защиты информации, во-вторых, недостаточной разработанностью данной темы. Рассмотрение проблем и вопросов, связанных с данной темой, носит как теоретическую, так и практическую значимость.

Целью работы является исследовать тест Фробениуса на наличие ошибок в определенных границах.

Структура дипломной работы состоит из введения, двух глав, раскрывающих теоретические основы и практическую реализацию алгоритма, заключения и списка используемой литературы.

Введение раскрывает актуальность данной темы, определяет научную новизну, раскрывает теоретическую и практическую значимость работы.

В основной части работы рассматриваются теоретические основы, разработка и реализация алгоритма для исследования теста Фробениуса.

В заключении подводятся итоги исследования, формируются окончательные выводы по рассматриваемой теме, полученные в ходе проведенной работы.

# ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 1.1 Простые и составные числа

Натуральный ряд чисел будет, как обычно, обозначаться в виде  $1, 2, 3, 4, \dots$ , а отдельные его элементы будут называться натуральными числами.

Понятие натурального числа, появляющееся как результат постепенного абстрагирования, является основой всего дальнейшего развития математики. Изучение свойств натуральных чисел, начатое в примитивной форме математиками давно ушедших поколений, занимает большое место в современной математике, составляя основное содержание одного из ее ведущих разделов, который называется теорией чисел.

**Определение 1.1.1.** Натуральное число  $p$  называется простым, если  $p > 1$  и  $p$  не имеет положительных делителей, отличных от 1 и  $p$ . [1]

Первые простые числа в натуральном ряду:  
 $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, \dots$

**Определение 1.1.2.** Натуральное число  $n > 1$  называется составным, если  $n$  имеет по крайней мере один положительный делитель, отличный от 1 и  $n$ . [1]

Таким образом, множество натуральных чисел разбивается на три подмножества: 1) простые числа, 2) составные числа и 3) число 1, которое не причисляется ни к простым, ни к составным числам.

**Теорема 1.1.1.** Для любого натурального числа  $n > 1$  наименьший, отличный от единицы положительный делитель всегда представляет собой простое число. [1]

**Теорема 1.1.2.** Каждое натуральное число, отличное от 1, можно представить в виде произведения простых чисел. [1]

**Теорема 1.1.3.** Для каждого натурального числа  $n > 1$  существует единственное разложение на простые множители. [1]

**Определение 1.1.3.** Каноническим разложением целого числа  $a > 1$  называется представление  $a$  в виде  $a = p_1^{\alpha_1} \dots p_k^{\alpha_k}$ , где  $p_1, \dots, p_k$  – попарно различные простые числа,  $\alpha_1, \dots, \alpha_k$  – натуральные числа. Каноническим разложением целого числа  $a < -1$  называется аналогичное представление в виде  $a = -p_1^{\alpha_1} \dots p_k^{\alpha_k}$ . [1]

При  $\alpha_1 = \dots = \alpha_k = 1$ , т.е.  $a = p_1 \dots p_k$ , число  $a$  называют свободным от квадратов.

**Теорема 1.1.4.** Если  $a = \pm p_1^{\alpha_1} \dots p_k^{\alpha_k}$  – каноническое представление числа  $a$ , то положительное число  $d$  является делителем  $a$  тогда и только тогда, когда  $d = p_1^{\beta_1} \dots p_k^{\beta_k}$ ,  $0 \leq \beta_1 \leq \alpha_1, \dots, 0 \leq \beta_k \leq \alpha_k$ . [1]

Простые способы нахождения первоначального списка простых чисел до некоторого значения дают решето Эратосфена, решето Сундарама и решето Аткина.

Однако, на практике, часто вместо получения списка простых чисел необходимо проверить, является ли данное число простым. Алгоритмы, применяемые для решения этой задачи, называются тестами простоты. Имеется огромное количество полиномиальных тестов простоты, но многие из них являются вероятностными и применяются для работы в криптографии.

**Теорема 1.1.5.** Множество простых чисел бесконечно.

Доказательство. «Предположим, что множество простых чисел конечно и состоит из чисел  $2, 3, 5, \dots, p$ , где  $p$  – последнее, самое большое простое число. Рассмотрим натуральное число  $N = 2 \cdot 3 \cdot 5 \dots p + 1$ .

$2 \nmid N, 3 \nmid N, \dots, p \nmid N$ , так как непосредственно видно, что при делении  $N$  на все числа  $2, 3, 5, \dots, p$  получается остаток, равный 1. Таким образом,  $N$  не делится ни на одно простое число, т.е.  $N = 1$ ; а вместе с тем непосредственно видно, что  $N > 1$ . Предположение, что множество простых чисел конечно, привело нас к противоречию, т.е. простые числа образуют бесконечное множество.» [1]

Процесс представления чисел в каноническом виде мы будем называть факторизацией. Общий метод факторизации заданного числа заключается в том, что  $n$  пробуют делить последовательно на простые числа  $2, 3, 5, \dots, p_r \leq \sqrt{n}$  до тех пор, пока не найдется простое число  $p$ , такое, что  $p|n$ . Если такое  $p$  находится, факторизация  $n$  сводится к факторизации меньшего числа; если же среди этих всех простых чисел нет ни одного делителя  $n$ , то  $n$  простое. Для больших  $n$  этот алгоритм требует долгих вычислений. Имеются таблицы, с помощью которых можно производить факторизацию чисел, лежащих в достаточно широких пределах.

**Определение 1.1.4.** Общим делителем целых чисел  $a_1, \dots, a_n$  называется любое целое  $d$ , такое, что  $d|a_1, \dots, d|a_n$ . [1]

**Определение 1.1.5.** Наибольшим общим делителем целых чисел  $a_1, \dots, a_n$  называется такой положительный общий делитель  $a_1, \dots, a_n$ , который делится на любой другой общий делитель этих чисел. [1]

Наибольший общий делитель числа  $a_1, \dots, a_n$  обозначается  $(a_1, \dots, a_n)$ .

**Определение 1.1.6.** Пусть  $a_1, \dots, a_n$  – отличные от нуля целые числа. Наименьшим общим кратным этих чисел называют наименьшее положительное число, кратное всем этим числам. [1]

Наименьшее общее кратное чисел  $a_1, \dots, a_n$  обозначается  $[a_1, \dots, a_n]$ .

**Определение 1.1.7.** Числа  $a_1, \dots, a_n$  называются взаимно простыми, если  $(a_1, \dots, a_n) = 1$ , т.е. если наибольший общий делитель этих чисел равен 1. [1]

**Определение 1.1.8.** Числа  $a_1, \dots, a_n$  называются попарно взаимно простыми, если  $(a_i, a_j) = 1$  при всех  $i \neq j$  ( $1 \leq i \leq n, 1 \leq j \leq n$ ). [1]

**Теорема 1.1.6.** Два числа  $a$  и  $b$ , отличные от 0 и  $\pm 1$ , взаимно просты тогда и только тогда, когда их канонические разложения не содержат одинаковых простых множителей. [1]

Псевдопростые числа – это числа, на которых метод проверки ошибается. Псевдопростых чисел достаточно много.

## 1.2 Символы Якоби и Лежандра

**Определение 1.2.1.** Класс чисел по модулю  $p$  называется классом квадратичных вычетов по этому модулю, если для чисел  $a$ , принадлежащих этому классу, сравнение  $x^2 \equiv a \pmod{p}$  имеет два решения. [1]

**Определение 1.2.2.** Класс чисел по модулю  $p$  называется классом квадратичных невычетов, если для чисел  $a$ , принадлежащих этому классу, сравнение  $x^2 \equiv a \pmod{p}$  не имеет решений. [1]

Соответственно этому все числа, принадлежащие классам квадратичных вычетов, т.е. все числа  $a$ , для которых сравнение  $x^2 \equiv a \pmod{p}$  имеет два решения, будем называть квадратичными вычетами по модулю  $p$  и аналогично все числа  $a$ , для которых это сравнение не имеет решений, квадратичными невычетами по этому модулю. Числа  $a$ , принадлежащие классу  $\bar{0}$ , для которых сравнение  $x^2 \equiv a \pmod{p}$  имеет одно решение, не причисляются ни к квадратичным вычетам, ни к квадратичным невычетам.

**Определение 1.2.3.** Пусть  $p$  – простое число,  $a$  – целое число. Определим символ Лежандра следующим образом:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & a \equiv 0 \pmod{p} \\ 1 & a - \text{квадратичный вычет по модулю } p \\ -1 & a - \text{квадратичный невычет по модулю } p \end{cases}$$

*Свойства символа Лежандра.*

1. Если  $b \equiv a \pmod{p}$ , то  $\left(\frac{b}{p}\right) = \left(\frac{a}{p}\right)$ .
2. Если  $a$  не делится на  $p$ , то  $\left(\frac{a^2}{p}\right) = 1$ .
3.  $\left(\frac{1}{p}\right) = 1$ .
4. Критерий Эйлера:

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



5. Закон взаимности. Для двух любых нечетных простых чисел  $p$  и  $q$  имеем:

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}.$$

Обобщением символа Лежандра является символ, введенный Якоби.

**Определение 1.2.4.** Пусть нечетное число  $m = p_1 p_2 \dots p_s$ , где  $p_i$  – простые числа, среди которых могут быть одинаковые,  $(a, m) = 1$ .

Символ Якоби  $\left(\frac{a}{m}\right)$  определяется равенством

$$\left(\frac{a}{m}\right) = \left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right) \dots \left(\frac{a}{p_s}\right),$$

где  $\left(\frac{a}{p_i}\right)$  при  $i = 1, 2, \dots, s$  – символы Лежандра.

Символ Лежандра  $\left(\frac{a}{p}\right)$  является частным случаем символа Якоби.

При  $m = p$ , где  $p$  – простое число, символ Якоби  $\left(\frac{a}{p}\right)$  является по определению вместе с тем и символом Лежандра.

Символ Якоби почти никогда не вычисляют по определению. Часто для вычисления применяют свойства символа Якоби, чаще всего квадратичный закон взаимности.

*Свойства символа Якоби.*

1. Если  $a \equiv b \pmod{m}$ , то символы Якоби  $\left(\frac{a}{m}\right)$  и  $\left(\frac{b}{m}\right)$  равны.

2.  $\left(\frac{a^2}{m}\right) = 1$ .

3.  $\left(\frac{ab}{m}\right) = \left(\frac{a}{m}\right)\left(\frac{b}{m}\right)$ .

4. Закон взаимности. Пусть  $m$  и  $n$  – нечетные числа, большие 1. Тогда

$$\left(\frac{n}{m}\right)\left(\frac{m}{n}\right) = (-1)^{\frac{m-1}{2}\frac{n-1}{2}}.$$

### 1.3 Поле Галуа

**Определение 1.3.1.** Полем называется множество  $P$ , содержащее как минимум два элемента, на котором заданы две алгебраические операции (сложение и умножение), удовлетворяющие аксиомам:

1. Коммутативность сложения:  $(\forall a, b \in P) [a + b = b + a]$ ;
2. Ассоциативность сложения:  $(\forall a, b, c \in P) [(a + b) + c = a + (b + c)]$ ;
3. Существование нулевого элемента:  $(\exists 0 \in P) (\forall a \in P) [a + 0 = a]$ ;
4. Существование противоположного элемента:  
$$(\forall a \in P) (\exists (b) \in P) [a + b = 0];$$
5. Коммутативность умножения:  $(\forall a, b \in P) [a \cdot b = b \cdot a]$ ;
6. Ассоциативность умножения:  $(\forall a, b, c \in P) [(a \cdot b) \cdot c = a \cdot (b \cdot c)]$ ;
7. Существование единичного элемента:  $(\exists 1 \in P) (\forall a \in P) [a \cdot 1 = a]$ ;
8. Существование обратного элемента:  $(\forall a \in P \setminus \{0\}) (\exists b \in P) [a \cdot b = 1]$ ;
9. Дистрибутивность умножения относительно сложения:

$$(\forall a, b, c \in P) [(a + b) \cdot c = a \cdot c + b \cdot c].$$

Конечное поле (поле Галуа) в общей алгебре – это поле, состоящее из конечного числа элементов. Число элементов в поле называется его порядком.

Примерами полей являются числовые поля  $\mathbb{Q}$ ,  $\mathbb{R}$ ,  $\mathbb{C}$ , поле  $F_p$  классов вычетов целых чисел по простому модулю  $p$ .

Понятие конечного поля применяется в теории чисел, теории групп, алгебраической геометрии, криптографии, в разработке криптографических шифров, таких как AES. Для работы с информацией при кодировании и декодировании данных все арифметические операции выполняются в полях Галуа. Имеется великое множество криптографических протоколов и криптосистем, основанных на применении конечных полей.

## 1.4 Квадратичные иррациональности

**Определение 1.4.1.** Квадратичной иррациональностью называется число вида  $z = a + b \cdot \sqrt{c}$ , где  $a, b, c$  – целые числа, причем  $c$  свободно от квадратов. Число  $a$  называется рациональной частью  $z$ ,  $a = \text{Rat}(z)$ , а число  $b$  – иррациональной частью,  $b = \text{Irr}(z)$ . [2]

Через  $z \bmod n$  будем обозначать число  $(a \bmod n) + (b \bmod n) \cdot \sqrt{c}$ .

Сопряженным числом будем называть число  $\bar{z} = a - b \cdot \sqrt{c}$ .

Сопряжение мультипликативно:  $\overline{z_1 \cdot z_2} = \bar{z}_1 \cdot \bar{z}_2$ .

Нормой  $N(z)$  называется целое число  $N(z) = \bar{z} \cdot z = a^2 - b^2 \cdot c$ .

Норма мультипликативна:  $N(z_1 \cdot z_2) = N(z_1) \cdot N(z_2)$ ;

$$N(z \bmod n) = N(z) \bmod n.$$

Например, пусть  $z = 1 + \sqrt{3}$ , тогда  $\bar{z} = 1 - \sqrt{3}$ ,  $N(z) = N(\bar{z}) = -2$ .

Вычисление  $z^a \bmod n$  вдвое сложнее, чем та же операция для целых чисел.

**Теорема 1.4.1.** Если символ Якоби  $Jacobi(c/p) = -1$ , то квадратичные иррациональности по простому модулю  $p$  образуют поле (поле Галуа) из  $p^2$  элементов, обозначаемое  $\mathbb{Z}_p[\sqrt{c}]$ . [2]

## 1.5 Тест Фробениуса

**Теорема (Фробениуса) 1.5.1.** Пусть  $p$  – простое число, символ Якоби  $Jacobi(c/p) = -1$  и  $z$  принадлежит  $\mathbb{Z}_p[\sqrt{c}]$ . Тогда  $z^p \bmod p = \bar{z}$ . [2]

**Определение 1.5.1.** Пусть  $n$  нечетное натуральное число, не являющееся полным квадратом. Его индексом Фробениуса  $Ind_F(n)$  будем называть наименьшее среди чисел  $[-1, 2, 3, 5, 7, 11, \dots]$ , такое, что символ Якоби  $Jacobi(c/n) = -1$ . [3]

Нетрудно выяснить, когда индекс Фробениуса принимает маленькие значения:

- если  $n \equiv 3 \bmod 4$ , то  $Ind_F(n) = -1$ ;
- если  $n \equiv 5 \bmod 8$ , то  $Ind_F(n) = 2$ ;
- если  $n \equiv 17 \bmod 24$ , то  $Ind_F(n) = 3$ ;
- если  $n \equiv 1 \bmod 24$ , то  $Ind_F(n) \geq 5$ .

Теорему Фробениуса можно использовать для построения вероятностного теста простоты числа.

Пусть  $n > 1$  – нечетное число, числа  $a, b, c$  таковы, что  $a, b, c$  взаимно просты с  $n$ ,  $a^2 - b^2 \cdot c \not\equiv 0, 1 \pmod{n}$  и  $Jacobi(c/n) = -1$ . Тогда, если  $(a + b \cdot \sqrt{c})^n \equiv (a - b \cdot \sqrt{c}) \bmod p$ , то число  $n$ , скорее всего, простое.

Как и в других аналогичных тестах, для повышения надежности, можно было бы выполнить этот тест несколько раз с различными  $a, b$  (менять  $c$  не имеет смысла), но это излишне.

На сегодняшний день не известно ни одного контрпримера к этому тесту.

Учитывая отсутствие контрпримеров, вместо того, чтобы говорить о произвольном (случайном) выборе параметров  $a, b, c$ , мы их просто зафиксируем.

**Определение 1.5.2.** Пусть  $n$  – нечетное натуральное число, не являющееся полным квадратом, и пусть  $c = \text{Ind}_F(c)$  – его индекс Фробениуса. Тогда

$$z = \begin{cases} 2 + \sqrt{c}, & c = -1, 2; \\ 1 + \sqrt{c}, & c \geq 3. \end{cases}$$

**Определение 1.5.3.** Число  $n$  будет простым по Фробениусу, если

$$z^n \equiv \bar{z} \pmod{n}.$$

Если составное число просто по Фробениусу, то будем называть его *псевдопростым по Фробениусу (Frobenius pseudoprime, FPP)*.

Таким образом, если  $n$  псевдопросто по Фробениусу, то  $n$  псевдопросто по основанию  $N(z)$ , то есть тест Фробениуса включает в себя тест Ферма. Сравнение иррациональных компонент фактически является тестом Люка (Lucas). Следовательно, тест Фробениуса является объединением тестов Ферма и Люка.

**Пример 1.5.1.** Пусть  $n = 19$ , тогда  $c = -1$ ,  $z = 2 + i$ ,

$$z^n = -3565918 + 2521451i \equiv 2 - i \pmod{n}.$$

**Пример 1.5.2.** Пусть  $n = 33$ , тогда  $c = -1$ ,  $z = 2 + i$ ,

$$z^n \equiv 2 + 22i \pmod{n}.$$

**Пример 1.5.3.** Пусть  $n = 17$ , тогда  $c = 3$ ,  $z = 1 + \sqrt{3}$ ,

$$z^n = 13160704 + 7598336\sqrt{3} \equiv 1 - \sqrt{3} \pmod{n}.$$

**Гипотеза.** Псевдопростых по Фробениусу чисел не существует. [2]

Другими словами, тест Фробениуса никогда не ошибается.

Вероятность найти контрпример прямым перебором очень маленькая. Гораздо больше шансов найти его в виде произведения простых.

**Определение 1.5.4.** Пусть натуральное число  $n$  псевдопросто по Фробениусу,  $c = \text{Ind}_F(n)$  – его индекс Фробениуса. Простой делитель  $p$  числа  $n$  называется  $\Phi$  - отрицательным, если символ Якоби  $\text{Jacobi}(c/p) = -1$  и  $\Phi$  - положительным, если  $\text{Jacobi}(c/p) = 1$ . [2]

Согласно определению, если  $Jacobi(c/p) = 1$ , то  $c$  является квадратом по простому модулю  $p$ , то есть существует  $d$ , такое, что  $c \equiv d^2 \pmod{p}$ .  $\Phi$  - положительные делители у псевдопростых по Фробениусу чисел встречаются очень редко.

**Теорема 1.5.2.** Пусть  $p$  – простое,  $n = p^2 q$  для некоторых  $p, q$  (не исключаем, что  $q$  кратно  $p$ ) и пусть  $n$  псевдопростое по Фробениусу с параметрами  $(a, b, c)$ . Тогда

$$z^p \equiv \bar{z} \pmod{p^2}.$$

*Следствие.* Если  $n = p^2 q$  – псевдопросто по Фробениусу, то  $N(z)^{p-1} \equiv 1 \pmod{p^2}$ , где  $N(z)$  – норма  $z$ .

Пусть натуральное  $n$  псевдопросто по Фробениусу с индексом Фробениуса  $c$  и  $p$  – простой делитель  $n$ , такой, что  $Jacobi(c/n) = +1$ . В этом случае кольцо  $R_p = \mathbb{Z}_p[\sqrt{c}]$  изоморфно декартову произведению  $\mathbb{Z}_p \times \mathbb{Z}_p$ , изоморфизм задается формулой

$$a + b\sqrt{c} \rightarrow (a + b \cdot g, a - b \cdot g),$$

где  $g^2 \equiv c \pmod{p}$ .

**Теорема 1.5.3.** Пусть  $n$  псевдопросто по Фробениусу,  $z = a + b\sqrt{c}$  и  $p$  –  $\Phi$  - положительный простой делитель  $n$ ,  $n = p \cdot q$ ,  $c \equiv d^2 \pmod{p}$ . Введем обозначения (два элемента из  $\mathbb{Z}_p$ ):

$$z_1 = a + b \cdot d \pmod{p},$$

$$z_2 = a - b \cdot d \pmod{p}.$$

Тогда

$$z_1^q \equiv z_2 \pmod{p},$$

$$z_2^q \equiv z_1 \pmod{p}.$$

## 1.6 Согласованные простые делители

**Теорема 1.6.1.** Пусть  $n$  – псевдопростое по Фробениусу число,  $p$  – его простой делитель. Тогда

$$n \equiv p \mod Q_p,$$

где  $Q_p = \text{ord}(z, p)$  – порядок числа  $z \mod p$ , т.е. наименьшая степень, в которую нужно возвести  $z$ , чтобы получить  $1 \mod p$ . [2]

Если мы рассмотрим два простых делителя  $p_1$  и  $p_2$ , то получим:

$$n \equiv p_1 \mod Q_{p_1},$$

$$n \equiv p_2 \mod Q_{p_2}.$$

Объединяя эти два сравнения, получим:

$$p_1 \equiv p_2 \mod \text{GCD}(Q_{p_1}, Q_{p_2}).$$

Такую пару простых чисел будем называть согласованной.

Псевдопростое  $n$  раскладывается на два простых множителя  $n = p_1 p_2$  тогда и только тогда, когда

$$p_1 \equiv 1 \mod (\text{ord}[a, p_2]),$$

$$p_2 \equiv 1 \mod (\text{ord}[a, p_1]).$$

Псевдопростое  $n$  раскладывается на три простых множителя  $n = p_1 p_2 p_3$  тогда и только тогда, когда

$$p_1 p_2 \equiv 1 \mod (\text{ord}[a, p_3]),$$

$$p_1 p_3 \equiv 1 \mod (\text{ord}[a, p_2]),$$

$$p_2 p_3 \equiv 1 \mod (\text{ord}[a, p_1]).$$

## 1.7 Обход дерева

Для начала рассмотрим некоторые понятия, необходимые для реализации алгоритма. Дерево – связный граф без циклов. Корень – самая верхняя вершина дерева. Лист – вершина, не имеющая потомков. Обход дерева – систематический просмотр всех вершин, при котором каждая вершина встречается один раз.

Некоторые определения из теории графов. Граф  $G = [R, A]$  – это совокупность двух множеств: множества точек, которые называются вершинами, и множества рёбер  $A$ . Каждый элемент  $a \in A$  есть упорядоченная пара  $(P_i, P_j)$  элементов множества  $R$ , вершины  $P_i$  и  $P_j$  называются концевыми точками или концами ребра  $a$ .

Граф называется конечным, если множества  $R$  и  $A$  конечны. В определение ребра можно принимать или не принимать во внимание порядок расположения двух его концов. Если этот порядок несущественен, т.е. если  $(P_i, P_j) = (P_j, P_i)$ , то говорят, что  $a$  есть неориентированное ребро. Если же этот порядок существенен, то  $a$  называется ориентированным ребром.

Граф называется неориентированным, если каждое его ребро не ориентировано, и ориентированным, если ориентированы все его рёбра.

Рассмотрим два популярных случая обхода дерева – поиск в ширину и поиск в глубину.

Поиск в ширину (обход по уровням) – один из алгоритмов обхода графа. Метод лежит в основе некоторых других алгоритмов близкой тематики. Поиск в ширину подразумевает поуровневое исследование графа: вначале посещается корень – произвольно выбранный узел, затем все потомки данного узла, после этого посещаются потомки потомков и т.д. Вершины просматриваются в порядке возрастания их расстояния от корня, т.е. сначала обрабатываются все вершины, смежные с текущей, а лишь потом их потомки.



В моей работе потребовался обход дерева в глубину, рассмотрим этот случай более подробно.

Рассматривая поиск в глубину, удобно представлять себе ориентированный граф как образ дерева. Более точно, пусть есть ориентированный граф, одна из вершин которого выделена. Будем предполагать, что все вершины доступны из выделенной по ориентированным путям. Построим дерево, которое можно было бы назвать «универсальным накрытием» нашего графа. Его корнем будет выделенная вершина графа. Из корня выходят те же стрелки, что и в графе – их концы будут сыновьями корня. Из них в дереве выходят те же стрелки, что и в графе и так далее. Разница между графом и деревом в том, что пути в графе, ведущие в одну и ту же вершину, в дереве «расклеены». В других терминах: вершина дерева – это путь в графе, выходящий из корня. Ее сыновья – это пути, продолженные на одно ребро. Заметим, что дерево бесконечно, если в графе есть ориентированные циклы.

Имеется естественное отображение дерева в граф (вершин в вершины). Обход дерева (посещение его вершин в том или ином порядке) одновременно является и обходом графа, только каждая вершина посещается многократно.

Будем предполагать, что для каждой вершины дерева выходящие из нее рёбра упорядочены (например, пронумерованы). Будем обходить дерево следующим образом. Сначала корень, а потом поддеревья (в порядке ведущих в них рёбер).

Другими словами, на путях, выходящих из выделенной вершины, введем порядок: путь предшествует своему продолжению; если два пути расходятся в некоторой вершине, то меньшим считается тот, который выходит из нее по меньшему ребру. Вершины теперь упорядочиваются в соответствии с минимальными путями, в них ведущими. Обход вершин графа в указанном порядке называется поиском в глубину.

## **1.8 Описание библиотеки MPIR**

### **(Multiple Precision Integers and Rationals)**

Разрядная сетка компьютеров ограничена. Когда-то использовались 8-разрядные процессоры, затем 16-, 32-, 64-разрядные. Для бытовых целей, чисел такой разрядности достаточно как для представления целых чисел, так и для вещественных вычислений. Тем не менее, существует много приложений, где требуются вычисления с большими числами, например в криптографии. Также зачастую нужно выполнять точные вычисления (с целыми числами) (например, в финансовой области) либо вычисления с произвольно заданной точностью. Ясно, что напрямую такие вычисления выполнить невозможно, и в мире создано большое количество библиотек, облегчающих труд программистов в этой области. Как правило, большие числа представляются в этих библиотеках в виде структур. Рассмотрим основные библиотеки, позволяющие выполнять вычисления с большими целыми числами, а также с вещественными числами с произвольной точностью.

Наиболее известная библиотека с открытым исходным кодом для выполнения вычислений с произвольной точностью – GMP ([11]). Библиотека написана на Си и ассемблере, предназначена для Unix-систем, однако, имеется много портов под Windows. Как указано на сайте библиотеки, вычисление миллиарда знаков числа  $\pi$  с помощью ее функций занимает полчаса времени на обычном компьютере. [11]

Другая заслуживающая упоминания библиотека – MPIR ([12]). Библиотека написана на C++, предназначена для Ubuntu и Debian. Есть порты на Windows. Библиотека используется в библиотеке Boost для C++, также имеется версия для Matlab, которая позволяет выполнять вычисления в этом пакете с произвольной точностью.

MPIR представляет собой портативную библиотеку, написанную на C для произвольной точности арифметики с целыми числами, рациональными

числами и числами с плавающей точкой. Она обеспечивает максимально возможную арифметику для всех приложений, которые требуют более высокую точность, чем поддерживаемые основные типы C.

Многие приложения используют только несколько сотен бит точности, но для некоторых приложений, возможно, потребуется тысячи или даже миллионы битов.

Скорость MPiR достигается за счет использования fullwords в качестве основного арифметического типа, использующего сложные алгоритмы, и тщательно оптимизированный код.

Эта библиотека предназначена для работы как в C, так и в C ++ компиляторах.

В данной библиотеке, целое число обычно означает многократной точности число как, определенное с помощью MPiR библиотеки.

Есть пять классов функций в библиотеке MPiR:

1. Функции для целочисленной арифметики. Есть около 150 функции в этом классе.
2. Функции для рациональных чисел. Есть около 40 функции в этом классе.
3. Функции с плавающей точкой. Есть около 60 функций этого класса.
4. Быстрые низкоуровневые функции, которые работают на натуральных числах.
5. Разные функции. Функции для установки пользовательских распределения и функции для генерации псевдослучайных чисел.

## ГЛАВА 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

### 2.1 Алгоритм поиска псевдопростых по Фробениусу чисел

Представим алгоритм для поиска псевдопростых по Фробениусу чисел. Возьмем  $N$  простых чисел, у которых индекс Фробениуса равен  $-1$ , т.е.  $n \equiv 3 \pmod{4}$ , будем называть их допустимыми.

Алгоритм состоит в переборе всех подмножеств множества допустимых простых чисел, каждая пара в котором является согласованной.

Будем предполагать, что  $n$  раскладывается на нечётное количество различных допустимых множителей  $n = p_0 \cdot p_1 \cdot \dots \cdot p_k$ . Как было сказано выше, каждая пара этих простых чисел должна быть согласованной.

Возьмем допустимое простое число  $p_0$ . Построим для него список согласованных с ним чисел. Выбираем из него по очереди все возможные числа  $p_1$ , согласованные с  $p_0$ , и строим список чисел, согласованных с  $p_0, p_1$ . Выбираем из него по очереди все возможные  $p_2$ , строим список чисел, согласованных с  $p_0, p_1, p_2$ . Проверяем, не является ли произведение чисел  $p_0 \cdot p_1 \cdot p_2$  – псевдопростым по Фробениусу числом.

Далее, выбираем из построенного списка по очереди  $p_3$ . Строим список согласованных чисел с  $p_0, p_1, p_2, p_3$ . Проверяем произведение чисел  $p_0 \cdot p_1 \cdot p_2 \cdot p_3$  по определению FPP и т.д., пока не переберем все.

Алгоритм был реализован с помощью обхода дерева в глубину, т.е. была реализована функция обхода дерева, которая делает следующее:

- 1) пытается добавить еще один множитель в произведение, т.е. увеличить текущее количество согласованных множителей, при этом заполняя массив, состоящий из простых чисел;

- 2) если добавить множитель не получается, пытается «пойти вправо» на дереве;

- 3) если «пойти вправо» не получается, то уменьшает высоту и снова «идет вправо»;

## 2.2 Результаты работы

Для реализации описанного алгоритма была написана программа на языке C++ (Visual C++ 2010). Сложность задачи заключалась в том, что количество множителей неограниченно. Поэтому, хотя каждый из них небольшой (меньше заданной константы), но произведение может превышать  $2^{64}$ . Это делает неизбежным обращение к библиотекам поддержки длинной арифметики.

Для работы с числами произвольной длины, как было сказано выше, на сегодняшний день наиболее популярны и эффективны библиотеки GMP([11]) и MPIR ([12]). В моем случае более подходящей оказалась библиотека MPIR, с помощью ее был реализован алгоритм проверки чисел на простоту тестом Фробениуса.

При всех вычислениях требуемое количество памяти было сравнительно незначительным, нигде не требовался даже 1 Гбайт памяти, поэтому объем памяти ограничением не являлся, проблему составляло только время работы.

Рассмотрим время работы программы при различных верхних границах. Индекс Фробениуса в данном случае равен  $-1$ . Результаты оформим в виде таблицы.

Количество простых чисел	Верхняя граница множителей	Время работы программы
10	67	5 сек.
50	499	15 сек.
100	1187	1 мин.
150	1907	8 часов
200	2707	26 часов

Таблица 2.2.1 Зависимость времени работы программы от количества простых чисел.

Если взять одну сотню простых чисел (максимальное  $p = 1187$ ), то полный перебор все возможных произведений занял одну минуту. Но для обработки 200 простых чисел (до 2707) потребовалось более суток.

Учитывая столь быстрый рост объема вычислений, следует признать, что дальнейшее увеличение границы подобным способом практически невозможно. Для этого требуется разрабатывать существенно более эффективные методы.

Результат работы программы доказывает, что не существует чисел, псевдопростых по Фробениусу с индексом Фробениуса равным  $-1$ , раскладывающихся в произведение множителей, каждый из которых не превосходит 2707.

Также были проверены и другие небольшие индексы Фробениуса (от 2 до 127). Количество простых чисел во всех случаях равнялось 70. Результаты работы программы оформим в виде таблицы.

Индекс Фробениуса	Нижняя граница множителя	Верхняя граница множителя
2	3	739
3	5	773
5	7	787
7	11	823
11	13	769
13	19	787
17	23	823
19	23	797
23	31	769
29	31	827
31	37	821
37	43	853
41	47	883
43	47	907
47	59	857

53	61	809
59	61	827
61	67	887
67	71	887
71	79	811
73	83	919
79	83	919
83	89	991
89	101	919
97	107	941
101	103	883
103	107	907
107	109	953
109	127	971
113	137	1051
127	131	997

Таблица 2.2.2 Нижняя и верхняя граница множителя при заданном индексе Фробениуса.

В результате работы программы, получились следующие результаты. Не существует чисел, псевдопростых по Фробениусу с индексами Фробениуса от 2 до 127, раскладывающихся в произведение множителей, каждый из которых не превосходит определенной верхней границы. Время работы программы в этом случае составило около 3 часов.

С помощью данного алгоритма проверены все составные числа с индексом Фробениуса от  $-1$  до 127, раскладывающиеся в произведение множителей, каждый из которых не превосходит определенной верхней границы. Общее количество таких произведений слишком велико, чтобы их можно было все перебрать. Поэтому пришлось разработать алгоритм, чтобы работа программы уложилась в разумное время.

## ЗАКЛЮЧЕНИЕ

Одной из важных задач криптографии является разработка эффективных методов проверки чисел на простоту. Даже самый простейший из них – метод последовательного деления на все простые числа не так плох. Но в современной криптографии имеют практическое значение большие простые числа. Таким образом, с практической точки зрения, нам надо уметь проверять простоту чисел длиной до 1000 десятичных знаков. Для этого требуются уже другие метод. Одним из них является метод Фробениуса проверки числа на простоту.

В ходе данной работы, мною были выполнены все поставленные задачи. Во-первых, раскрыты теоретические аспекты. Во-вторых, разработан и реализован алгоритм для исследования теста Фробениуса проверки чисел на простоту. Код реализации алгоритма на языке программирования C++ в приложении.

На основании проделанного мною исследования, можно сделать вывод, что предложенный метод нахождения псевдопростых по Фробениусу чисел существенно более эффективный, чем прямой перебор. С помощью данного алгоритма проверены все составные числа с индексом Фробениуса равным  $-1$ , раскладывающиеся в произведение множителей, каждый из которых не превосходит 2707. Также проверены все составные числа с индексом Фробениуса от 2 до 127, раскладывающиеся в произведение множителей, каждый из которых не превосходит определенной верхней границы (см. таблицу 2.2.2). Количество таких чисел достаточно велико, но ни на одном из них тест Фробениуса не ошибается.

Таким образом, найден еще один факт, свидетельствующий в пользу гипотезы о том, что чисел, псевдопростых по Фробениусу, не существует.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Бухштаб А.А. *Теория чисел* // Просвещение, 1966.
2. Хашин С.И. *Кратные множители псевдопростых чисел* // Вестн. Иван. гос. ун-та. 2013. Вып. 2. С. 102-107.
3. Хашин С.И. *Натуральные числа с большим индексом Фробениуса* // Вестн. Иван. гос. ун-та. 2015. Вып. 2. С. 75-78.
4. Baillie R. *Lucas Pseudoprimes* // Samuel S., Wagstaff Jr., Mathematics of Computation, 35(152):1391-1417. doi:10.1090/S0025-5718-1980-0583518-6.
5. Crandall R.E. *Prime Numbers: A Computational Perspective* // Pomerance C., (Second Edition), Springer-Verlag, 2005. - 597 Pages.
6. Damgard I.B. *An Extended Quadratic Frobenius Primality Test with Average and Worst-Case Error Estimate* // Frandsen G.S., Journal of Cryptology, Volume 19, Number 4 (2006), 489-520, DOI: 10.1007/s00145-006-0332-x.
7. Grantham J. *Frobenius pseudoprimes* // Math. of computation, Volume 70, Number 234, p.873-891 S 0025-5718(00)01197-2.
8. Khashin S.I. *Counterexamples for Frobenius primality test* // 2013.
9. Seysen M. *A Simplified Quadratic Frobenius Primality Test* // Cryptology ePrint Archive, 2005/462.
10. Sorenson J. *Strong Pseudoprimes to Twelve Prime Bases* // Webster J., 2015, arXiv.org:1509.00864 [math.NT].
11. GMP // URL: <http://www.gmpilib.org> (дата обращения: 20.03.2018).
12. MPIR // URL: <http://www.mpir.org> (дата обращения: 20.03.2018).

## ПРИЛОЖЕНИЕ

### *Код программы на языке программирования C++ (Visual C++ 2010)*

```
#include <cstdlib>
#include <iostream>
#include <conio.h>
#include <fstream>
#include <vector>

#include "frob.h"
#include "z64.h"

using namespace std;

const int np = 200; // количество простых чисел

u64 mp[np]; // массив, содержащий простые числа
u64 ord[np]; // массив, содержащий порядки простых чисел
int t1; // текущее количество согласованных простых множителей
// t1 начинается с 0 и может расти вплоть до np
mpz_class product; // произведение простых чисел
u64 arrT[np]; // массив, содержащий индексы выбранных в mp простых чисел
u64 arrT0; // переменная, содержащая индекс 1-го выбранного в массиве mp простого числа
vector<u64> zero; // пустой вектор
vector<vector<u64>> msp;
// вектор, состоящий из простых чисел, согласованных с arrT[0], arrT[1], ..., arrT[nsp]
// Например, при выбранном первом множителе mp[arrT[0]] он будет
// содержать индексы в массиве mp чисел, согласованных с mp[arrT[0]].
// При выбранных двух множителей mp[arrT[0]], mp[arrT[1]] он будет
// содержать индексы в массиве mp чисел, согласованных
// с mp[arrT[0]] и с mp[arrT[1]].
int mspN[np]; // количество чисел в массиве msp[i]

// функция init заполняет массивы mp, ord
// ind_c - индекс Фробениуса
void init (int ind_c)
{
    printf("Index_Frobenius = ");
    printf("%d\n", ind_c);

    // заполняем массив простыми числами
    u64 mp1 = 3;
    for (int i=0; i<np; i++)
    {
        while (frob::minC(mp1)!=ind_c)
        {
            mp1 = z64::NextPrime(mp1);
        }
        mp[i] = mp1;
        mp1 = z64::NextPrime(mp1);
    }

    printf("np = ");
    printf("%d\n", np);
    printf("mp[1] = ");
    printf("%d\n", mp[0]);
    printf("mp[np] = ");
    printf("%d\n", mp[np-1]);
    printf("\n\n");

    // находим порядки для простых чисел
    for (int i=0; i<np; i++)
    {
        if (ind_c <= 2)
            ord[i] = z64::FrobOrder(2,1,ind_c, mp[i]);
        else
            ord[i] = z64::FrobOrder(1,1,ind_c, mp[i]);
    }
}

// функция заполняет массив согласованных, устанавливает первый множитель равным mp[i]
void sepP0 (int i)
{
    int nsp = 0;
    t1 = 0;
```

```

        msp.clear();
        msp.push_back(zero);

        arrT0 = i;
        arrT[0] = i;

        for (int i1=i+1; i1<np; i1++)
        {
            if (((mp[i1]-mp[i])%(z64::gcd(ord[i],ord[i1])))) == 0)
            {
                msp[0].push_back(i1); // msp[0][nsp] = i1
                nsp++;
            }
        }
        mspN[0] = nsp;
    }

    // функция заполняет массив согласованных, устанавливает j-й множитель равным mp[i]
    void sepPj (int i, int j)
    {
        t1 = j;
        arrT[j] = 0;
        int nsp = 0;

        while (msp.size() <= j)
            msp.push_back(zero);
        msp[j].clear();

        for (int i1=i+1; i1<mspN[j-1]; i1++)
        {
            if (((mp[msp[j-1][i1]]-mp[msp[j-1][i]])%(z64::gcd(ord[msp[j-1][i]],ord[msp[j-1][i1]]))) == 0)
            {
                msp[j].push_back(msp[j-1][i1]); // msp[j][nsp] = msp[j-1][i1]
                nsp++;
            }
        }
        mspN[j] = nsp;
    }

    // функция делает шаг вправо
    void shag_vpravo()
    {
        arrT[t1]++;
        sepPj(arrT[t1],t1+1);
    }

    // функция делает шаг вверх
    void shag_vverh()
    {
        t1--;
    }

    // функция делает шаг вниз, если возможно
    bool yest_snizu__shag_vniz()
    {
        if(mspN[t1]==0)
            return false;
        t1++;
        sepPj(0,t1);
        return (mspN[t1]!=0);
    }

    bool yest_sprava()
    {
        return (arrT[t1] < mspN[t1]);
    }

    bool yest_vyshe()
    {
        return (t1 > 0);
    }

```

```

// функция проверяет произведение простых чисел, попарно согласованных друг с другом, по определению FPP
void obrabot_list()
{
    int t2 = 0;    // счетчик попарно согласованных друг с другом чисел
    for (int k=0; k<t1; k++)
    {
        if (msp[k].size() > arrT[k])
        {
            t2++;
        }
    }

    // если количество чисел < 3 или четно, то выходим
    if (((t2+1)<3)||((t2+1)%2==0))
        return;

    // выводим простые числа, попарно согласованные друг с другом
    printf("%llu ", mp[arrT0]);
    product = mp[arrT0];
    for (int k=0; k<t1; k++)
    {
        if (msp[k].size() > arrT[k])
        {
            printf("%d\t", mp[msp[k][arrT[k]]]);
            product *= mp[msp[k][arrT[k]]];
        }
    }
    printf("%\n\n");

    // проверяем произведение по определению FPP
    if (frob::FrobTest(product) == 1)
    {
        printf("%s\n FPP - ", product.get_str().c_str());
        system("PAUSE");
    }
}

// обход дерева
void obhod ()
{
    for (int i=0; i<np; i++)
    {
        sepP0(i);

        // если чисел, попарно согласованных друг с другом, нет, то выполняем пропуск оставшейся
        // части кода тела цикла и переходим к следующей итерации цикла
        if ((mspN[0] == 0)|| (mspN[0] == 1))
            continue;

        int sost = 1;
        while ((sost!=2)|| (yest_vyshe()))
        {
            if ((sost==1)&&(yest_snizu__shag_vniz())) { }
            else if ((sost==1))
            {
                obrabot_list();
                sost = 2;
            }
            else if ((sost==2)&&(yest_sprava()))
            {
                shag_vpravo();
                sost = 1;
            }
            else
            {
                shag_vverh();
                if ((t1==0)&&(yest_sprava()))
                {
                    shag_vpravo();
                    sost = 1;
                }
            }
        }
    }
}

```