НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

В. М. Зюзьков

Компьютерная алгебра

Издательство Томского университета
2014

УДК 512.55, 519.7

Рецензенты:

кафедра алгебры Томского государственного университета, заведущий кафедрой доктор физико-математических наук, профессор П.А. Крылов;
заслуженный работник высшей школы Российской Федерации, профессор кафедры высшей математики ФГБОУ ВПО ТГАСУ, доктор физико-математических наук, профессор М.И. Слободской

В учебном пособии излагаются основные свойства евклидовых и факториальных колец, связанные с делимостью. Рассматриваемые вопросы посвящены алгоритмам нахождения наибольшего общего делителя, распознавания простых элементов и факторизации в кольцах $\mathbb{Z}$, $\mathbb{Z}[i]$, $\mathbb{Q}[x]$ и $\mathbb{Z}[x]$. Описанные алгоритмы доведены до программ для системы Mathematica. Рассматриваются процедурное, функциональное и основанное на правилах преобразований программирование на языке Wolfram. Дано введение в параллельное программирование с системой Mathematica. Приведены задачи и упражнения.

Для студентов, обучающихся по направлению «Математика и компьютерные науки»; будет полезно также всем, кто имеет дело с символьными вычислениями и криптографией.

Рисунок на обложке создан Enrique Zeleny с помощью программы Mathematica

Оглавление

Предисловие

В соответствии с традицией мы рассматриваем термин «компьютерная алгебра» как синоним терминов подобных словам «символьные преобразования» и «аналитические вычисления».

Для кого предназначена эта книга? В первую очередь, эта книга – учебное пособие для студентов направления «Математика и компьютерные науки» по дисциплине «Компьютерная алгебра». Но автор надеется, что она будет полезна и для профессионалов-математиков, которые пожелают использовать в своей работе такой мощный программный инструмент как система Mathematica. Применение Mathematica позволяет эффективно вычислять математические объекты, что проливает свет на используемые математические понятия. Причем использование Mathematica не требует глубоких знаний программирования.

Изложение замкнуто – необходимые знания по алгебре и теории чисел даются в книге, при этом многие утверждения приводятся вместе с доказательствами. Математические вопросы концентрируются вокруг понятия делимости в евклидовых и факториальных кольцах. Рассматриваются алгоритмы нахождения наибольшего общего делителя, распознавания простых элементов и факторизации в кольцах целых чисел, гауссовых целых чисел и в кольце многочленов над целыми числами. Приводятся обоснования алгоритмов и их программы в системе Mathematica. Дается краткое описание системы и программирования на языке Wolfram – языке системы Mathematica – в объеме необходимом для понимания приведенных программ. Достаточно полное описание Mathematica требует отдельной книги. Последняя глава служит введением в возможности параллельных вычислений с системой Mathematica

Хотя учебное пособие не затрагивает вопросы криптографии, тем не менее, приводимые алгоритмы с простыми числами будут полезны и для студентов изучающих криптографию.

Книга не предназначена для разработчиков систем компьютерной алгебры, поэтому проблемы реализации системы компьютерной алгебры, такие, как представление данных и упрощение выражений рассмотрены только в общем виде и отдельно для системы Mathematica.

Текст учебного пособия разбит на главы, главы на параграфы, внутри параграфа идет нумерация пунктов (определений, теорем, примеров и др.). Ссылки внутри главы даются с помощью двойной нумерации, ссылка на пункт из другой главы дополняется явным указанием номера главы.

Большинство глав заканчиваются упражнениями и задачами, которые можно использовать для более глубокого изучения математического материала и Mathematica. Целью автора при выборе задач для этой книги было желание показать эффективность системы компьютерной алгебры как инструмента для решения математических задач.

И последнее: чтение этой книги без компьютера принесет мало пользы. Читайте и используйте Mathematica, выполняйте программы и экспериментируйте, получайте удовольствие от компьютерной алгебры!

Невозможно избавиться от чувства, что математические формулы живут собственной жизнью и обладают собственным разумом, что они умнее нас, умнее даже тех, кто их открыл, что мы получаем из этих формул больше, чем в них было изначально заложено.

Генрих Герц

Глава 1. Системы компьютерной алгебры

§ 1. Краткая история символьных вычислений

История математики до компьютерной эры содержит много примеров трудоемких вычислений. Некоторые вычисления сводились к сложным и громоздким преобразованиям формул, другие вычисления использовали небольшие формулы, но требовали выполнения операций с большим количеством цифр в числах.

Великий Леонард Эйлер (1707 – 1783) был непревзойдённым мастером формальных выкладок и преобразований, в его трудах многие математические формулы и символика получили современный вид (например, ему принадлежат обозначения для e и π). Наглядными примерами мастерства Эйлера служат его вычисление суммы обратных квадратов и получение необычайной формулы, связывающей суммы делителей натуральных чисел [31, стр. 39–43, 11–122].

В 19 веке очень много вычислений было проделано в астрономии. Например, французский математик Урбен Леверье проводил расчет орбиты Нептуна. Расчет был основан на аналитических вычислениях возмущенной орбиты Урана. И этот расчет собственно и привел к открытию Нептуна.

Впечатляющие вычисления с карандашом и бумагой проделал французский астроном Чарльз-Евгений Делоне для вычисления орбиты Луны. Он вывел около 40000 формул. На их вывод потребовалось 10 лет и еще 10 лет ушло на проверку формул. Окончательная формула занимала 128 страниц его книги с результатами работы. Проверка его аналитических преобразований было проведена двумя американскими математиками с помощью компьютера в 70-е годы 20 века. Компьютеру потребовалось двое суток работы.

Большие усилия тратили математики на определение числа π, вручную вычисляя большое количество цифр. Так, например, наилучший результат к концу XIX века был получен англичанином Вильямом Шенксом. Он потратил 15 лет для того, чтобы вычислить 707 цифр, хотя из-за ошибки только первые 527 были верными. Он использовал формулу Мэчина (Johm Machin, 1680 – 1751)

$$\frac{\pi}{4} = 4\,\mathrm{arctg}\frac{1}{5} - \mathrm{arctg}\frac{1}{239}.$$

Ошибку Шенкса обнаружил в 1944 году Фергюсон (D. E. Ferguson); он считал по формуле, подобной формуле Мэчина,

$$\frac{\pi}{4} = 3\,\mathrm{arctg}\frac{1}{4} + \mathrm{arctg}\frac{1}{20} + \mathrm{arctg}\frac{1}{1985}$$

на настольном механическом калькуляторе.

В начале 50-х годов стали появляться первые программы, производящие частично аналитические вычисления. В 1951 году с помощью компьютера EDSAC 1 было открыто наибольшее известное простое число $180(2^{127} - 1)^2 + 1$ с 79 десятичными цифрами [33, стр. 46]. Оно будет использоваться несколько раз в этой книге в примерах работы алгоритмов с простыми числами.

В 1952 году математики Эмиль Артин и Джон фон Нейман проделали большие вычисления, связанные с эллиптическими кривыми, на компьютере MANIAC [11].

В 1953 было показано, как алгоритмы в теории групп могут быть реализованы на компьютере [2].

В 60-х годах 20 века стали создаваться первые системы компьютерной алгебры Система компьютерной алгебры (computer algebra system) – программа для выполнения символьных (математических) вычислений. Основная определяющая функциональность таких систем – это операции с выражениями в символьной форме.

Первые системы были ограничены по своим возможностям и предназначались для какой-то отдельной области математики. Системы компьютерной алгебры общего назначения (универсальные) – это те, в которых реализованы основные математические алгоритмы и есть возможность пользователю самому создать новые алгоритмы на языке программирования системы.

Назовем некоторые исторически значимые и используемые в настоящее время системы компьютерной алгебры.

Macsyma является системой компьютерной алгебры, которая первоначально разрабатывалась с 1968 по 1982 (первая распространяемая версия появилась в 1978 году) в Массачусетском технологическом институте как научный проект MAC (Mathematics and Computation), а затем стала продаваться как коммерческий продукт Maxima (компания Symbolics, последняя версия 2.4, 1999 г). Это была первая общего назначения символическая система математики и одна из самых ранних систем, основанная на математических знаниях; многие его идеи были позже восприняты Mathematica, Maple и другими системами. Macsyma была реализована на MacLisp.

Reduse – система компьютерной алгебры общего назначения, созданная физиком Херном (Anthony C. Hearn) в 60-х годах главным образом для применения в физике; первая распространяемая версия появилась в 1968 г., последняя версия создана в 2009 г. С 2008 г. система распространяется бесплатно.

Maple начала разрабатываться в 1980 сотрудниками Symbolic Computation Group в Университете Waterloo (первая распространяемая версия в 1984 году). С 1988 года система продается компанией Maplesoft (версия 17, 2013).

Фундамент Maple составляет небольшое ядро, написанное на *C*. Это ядро в свою очередь обеспечивает функционирование языка Maple. Функциональность системы основана на различных библиотеках. Различные алгоритмы Maple используют численные данные в разных форматах. Символические выражения хранятся в памяти как направленные ациклические графы. Стандартный интерфейс и вычислительный интерфейс реализованы на Java.

Mathematica – система компьютерной алгебры общего назначения, используется во многих научных, инженерных, математических и вычислительных областях. Система была задумана Стивеном Вольфрамом (физик, математик и программист) и в дальнейшем разработана в компании Wolfram Research (Шампейн, штат Иллинойс, США). Начало разработки – 1986 г.; первая версия – 1988 г.; последняя 9-я версия – 2012 г. [13].

Только человек, по роду своей деятельности имеющий дело с математическими вычислениями, глубоко понимающий их специфику и потребности, мог создать подобный программный продукт.

В этой книге будет использоваться система компьютерной алгебра Mathematica. Глава 2 содержит описание Mathematica и введение в программирование на языке Wolfram.

§ 2. Представление данных

Системы компьютерной алгебры для получения точных результатов работают с целыми числами, рациональными и алгебраическими числами. Причем количество цифр в числе не ограничивается.

Символьные вычисления обычно оперируют с математическими выражениями произвольного вида. В отличие от вычислительной математики, где структуры данных (числа, массивы) не меняют своего размера, в алгоритмах компьютерной математики выражения меняют свою структуру и размер.

Даже в том случае, когда мы можем ограничить размер входных выражений, и знаем какой размер выходных выражений, мы часто сталкиваемся с ситуацией, когда во время вычисления размер промежуточных выражений становится непредсказуемо большой (см. глава 4, пример 2.10). Поэтому необходимо использовать при реализации алгоритмов динамические структуры данных. В качестве такие структур обычно используют списки. Причем списки, как правило, имеют иерархическую структуру, т.е. элементом списка является снова список.

В книге [25] хорошо описано, как представлять числа неограниченной длины и как эффективно выполнять арифметические операции с ними. Алгоритмы работы со списковыми структурами также хорошо изучены, см. [24].

Основный класс выражений, используемых в символьных вычислениях – это многочлены, понимаемые в обобщенном смысле. Для того чтобы прояснить ситуацию, приведем пример. Преобразование

$$(x-y)(x+y) \;\rightarrow\; x^2-y^2 \tag{1}$$

является многочленным (полиномиальном) вычислением, но таким же является и преобразование

$$(\cos a - \sin b)\,(\cos a + \sin b) \rightarrow \cos^2 a - \sin^2 b,$$

в котором фактически произведено то же вычисление, что и в (1) с заменой x на $\cos a$ и y на $\sin b$.

Обычно системы компьютерной алгебры (SCA) могут работать с многочленами произвольного числа переменных. Их можно складывать, вычитать, умножать и делить.

2.1. Обычно различают для многочленов два типа представлений: **плотные** и **разреженные**. Традиционно этим понятиям не придают четкого математического смысла, поскольку они касаются возможных реализаций и носят обычно прикладной характер. Следуя этой традиции, можно сказать, что разреженное представление какого-либо класса объектов называется представление, ориентированное на экономное (в том или ином смысле) представление выделенного класса объектов. Плотным представлением называется представление, не обладающее упомянутым свойством.

Например, сравним следующие два представления многочлена:

1) представление многочлена $a_0 + a_1x + \ldots + a_nx^n$ списком его коэффициентов

$$\{a_0, a_1, \ldots, a_n\}, \tag{2}$$

2) представлением многочлена списком пар упорядоченных чисел (коэффициент, степень), где пара с нулевым коэффициентом пропускается, так что многочлен $1 - x^8$ представляется в виде списка

$$\{\{1, 0\}, \{-1, 8\}\}. \tag{3}$$

Заметим, что здесь следует ввести еще представление нулевого многочлена (оно должно быть особым).

Если считать, что под каждое число отводится одинаковая память (не зависящая от числа), то представление (2) экономнее, если число ненулевых слагаемых больше $n/2$, иначе экономнее представление (3). Мы, конечно, не учитываем здесь дополнительные технические детали.

2.2. Представление (2) принято называть **плотным**, а представление (3) – **разреженным представлением многочленов**.

Полезность понятия «представление» иллюстрируется на примере проверки соотношения

$$(x^{1000} + 1)(x^{1000} - 1) = x^{2000} - 1. \quad (4)$$

Здесь применение представления (2) привело бы к миллиону умножений слева, а в представлении (3) можно обойтись лишь четырьмя умножениями.

Конечно, равенство (4) можно проверить и без умножения (что и делается в современных SCA).

2.3. Представление (2) и (3) являются **внутренними** представлениями многочленов, связанные с реализацией. Но очевидно имеются и **внешние** представления – те, с которых пользователь SCA работает с многочленами при вводе-выводе и которые используются в математике. Примерами таких представлений многочлена являются его изображение в виде

$$\sum_{i=0}^{n} a_i x^i \quad (5)$$

и четыре изображения

$1 - x^8$,

$-(-1 + x)\,(1 + x)\,(1 + x^2)\,(1 + x^4)$,

$(-1 + x)\,(1 + x)\left(-1 + \sqrt{2}\ x - x^2\right)\left(1 + x^2\right)\left(1 + \sqrt{2}\ x + x^2\right)$

и

$-(-1 + x)\,(-i + x)\,(i + x)\,(1 + x)\left(-i + x^2\right)\left(i + x^2\right)$

одного и того же многочлена.

Представление (5) можно практически использовать в SCA, в отличие от записи в виде $a_0 + a_1 x + \ldots + a_n x^n$. Представление многочлена $1 - x^8$ в виде произведения множителей уже учитывает математические свойства выражений.

В этой книги все алгоритмы работают с математическими выражениями во внешнем представлении. Единственным исключением служит алгоритм 1.11 из главы 4.

В главе 2 мы рассмотрим подробно представление данных в системе Mathematica.

§ 3. Задача упрощения выражений

Какие преобразования математических выражений обычно совершаются? Наиболее общий вид преобразований – это преобразование из одного математического представления в другое представление с сохранением семантики выражения. Такие преобразования обычно называют **эквивалентными**, так как исходное выражение и результат удовлетворяют какому-то фиксированному отношению эквивалентности.

Использование эквивалентных преобразований связано с определенными требованиями к внешнему представлению математических выражений.

Наиболее существенным является требование о том, чтобы выбор представления был **каноническим**, т. е. во множестве всех эквивалентных выражений нужно выбрать единственное выражение, которое представляло бы этот класс эквивалентности.

Часто целью эквивалентного преобразования является упрощение выражений.

Задача упрощения выражений имеет две разновидности:

1) построение эквивалентного данному объекту более простого объекта;
2) нахождение единственного представления для эквивалентных объектов.

Более точно, пусть T – класс объектов (выражений), например, термов в языке первого порядка, ограниченных классов логических формул или ограниченных классов программ; и пусть ~ обозначает отношение эквивалентности на классе T, например, функцио-

нальное равенство, равносильность логических формул, равенство по модулю идеала и т.п.

3.1. Задача построения эквивалентного более простого объекта состоит в нахождении алгоритмического отображения S: $T \to T$, такого, что для любого объекта t из T выполнены следующие свойства:

$S(t) \sim t$,
$S(t) \leq t$.

Здесь $\leq$ обозначает некоторое рассматриваемое отношение упрощения. Например, $s \leq t$ может означать:

- s короче t, или
- для представления s в компьютере требуется меньше памяти, чем для t, или
- s более устойчиво численно, чем t, или
- структура s более понятна, чем структура t и т.д.

Конечно, желательно, чтобы $S(t) < t$ для «большинства» t.

3.2. Задача вычисления единственного представления для эквивалентных объектов (иными словами, задача *канонического упрощения*) состоит в нахождении алгоритма S: $T \to T$ («канонизации» для $\sim$ на T), такого, что для любых объектов s, $t \in T$ справедливы следующие свойства

$S(t) \sim t$,
из $s \sim t$ следует $S(s) = S(t)$,

т.е. S указывает единственного представителя в каждом классе эквивалентности, $S(t)$ называется **каноническим видом** объекта t.

Пример канонизации: преобразование элементарных арифметических выражений к полиномиальному виду,

$(1/2)(2x + 2)(x - 1) \to x^2 - 1$.

Указанные два типа задач не являются полностью независимыми: последовательное итеративное уменьшение размера выражения относительно некоторой меры «простоты» может иногда привести к канонизации или дать идею конструкции канонизации. Обратно, канонизация определяет соответствующее понятие упрощения: можно сказать, что результат канонизации объекта «проще», чем сам объект.

С другой стороны, практические процедуры, которые в интуитивном смысле "упрощают" выражения, могут приводить к двум различным упрощенным видам для двух эквивалентных выражений, и, обратно. В некоторых случаях канонические виды простых выражений могут быть достаточно сложными, например, если считать каноническим видом многочлена – сумму одночленов, то канонический вид многочлена $(x - 1)^{10}$ сложнее его исходного вида.

3.3. Примеры отношения эквивалентности в задачах упрощения:
1. Элементарные арифметические выражения $(x - 1)^{10}$ и $1 - 10x + 45x^2 - 120x^3 + 210x^4 - 252x^5 + 210x^6 - 120x^7 + 45x^8 - 10x^9 + x^{10}$ синтаксически не тождественны, но они эквивалентны, как представляющие один и тот же полином из $\mathbb{Z}[x]$.
2. Выражения $x^6 - 1$ и $x - 1$ представляют из себя разные полиномы, как в кольце $\mathbb{Q}[x]$, так и в кольце $\mathbb{Z}_2[x]$[1]. Но они функционально эквивалентны (т.е. представляют одну и ту же функцию) над полем $\mathbb{Z}_2$.

Чтобы сформулировать теорему, говорящую о связи канонического упрощения и распознавания эквивалентности необходимо напомнить некоторые известные понятия [30]: функция называется вычислимой, если существует алгоритм, вычисляющий ее значения; множество называется разрешимым, если его характеристическая функция вычис-

[1] $\mathbb{Z}_2[x]$ – кольцо многочленов от переменной x над полем $\mathbb{Z}_2$.

лима; бинарные разрешимые отношения $M(x, y)$ – это в точности те, для которых множество $\{<x,y> \mid M(x,y)\}$ разрешимо.

Теорема 3.4. Пусть T – разрешимое множество объектов и ~ обозначает отношение эквивалентности на T. Отношение ~ разрешимо тогда и только тогда, когда существует канонизация S для ~.

Так как отношение эквивалентности не всегда разрешимо, то и не всегда существует канонизация для такого отношения.

Это является одной из причин, почему иногда на представления налагаются более слабые требования, чем те, что они канонические. Одним из таких условий является условие нормальности.

Как правило, рассматриваемая структура данных снабжена некоторым набором арифметических операций, часть из которых определена не для всех значений аргументов. В частности, недопустимо деление на нуль. Тем самым нуль приобретает некоторое особое положение, и возрастает значение задачи определения равенства элемента нулю.

3.5. Представление, в котором все эквивалентные нулю выражения представляются одним и тем же образом (0), называется **нормальным**.

Любое каноническое представление является нормальным, но обратное верно не всегда. Однако во многих случаях наличие нормального представления позволяет построить каноническое. Если же рассматриваемая структура данных такова, что в ней имеются, кроме нуля, и другие "особые" элементы, то определение нормального представления должно быть усложнено.

Но даже нормальность не всегда существует. Как известно, не существует алгоритма, который для произвольной математической функции мог бы установить, является ли эта функция тождественным нулем или нет [23, стр. 110]. Но совсем необязательно искать алгоритм для произвольных функций. Функции могут быть достаточно простые.

Теорема 3.6. (Ричардсон, [9]) Пусть R класс выражений созданных из переменной x, рациональных чисел и двух вещественных чисел π и $\ln 2$ с помощью операций сложения, умножения, композиции и функций sin, exp и абсолютная величина. Тогда, если выражение $E \in R$, то предикат «$E = 0$» неразрешим.

3.7. Во время вычислений SCA применяет некоторые «очевидные» правила упрощения из алгебры и тригонометрии для удаления лишних символов из выражений и преобразования их к некоторому стандартному виду. Такой процесс называется **автоматическим упрощением**. Пример автоматического упрощения в системе Mathematica:

$$x + 2x + y\, y^2 + z^0 + \text{Sin}[\pi/4]$$

$$1 + \frac{1}{\sqrt{2}} + 3\,x + y^3$$

В большинстве систем компьютерной алгебры (в том числе и в Mathematica) все выражения в диалоге с пользователем и при символьных вычислениях преобразуются в **контексте автоматического упрощения**. Это означает, (1) при вводе все операнды математических операторов автоматически упрощаются перед тем как операторы применяются; (2) результат вычисления выражения представлен в автоматически упрощенном виде.

Автоматическое упрощение описывается множеством упрощающихся правил преобразования, применяемых без участия пользователя. Существует множество упрощающих правил. Но только часть из них следует применять во всех ситуациях. Применять или нет других правила – решает пользователь. Во-первых, во многих случаях «упрощенный вид выражения» зависит от контекста вычисления; во-вторых, применение некоторых упрощающих правил математически допустимо только при определенных условиях. Таким образом, кроме автоматического упрощения, должны существовать явно используемые правила упрощения.

Каким образом задача упрощения решается в Mathematica, описывается в следующей главе.

Устройство нашего мира непостижимо
без знания Mathematica.
Приписывается Роджеру Бэкону

Глава 2. Mathematica

§ 1. Обзор системы

Mathematica представляет собой модульную систему программного обеспечения, в которой ядро, фактически выполняющее вычисления, отделено от интерфейса, отвечающего за взаимодействие с пользователем.

Такая конструкция имеет много преимуществ по сравнению с монолитной системой. К примеру, интерфейс Mathematica может быть запущен на локальном компьютере с расширенными графическими возможностями, в то время как ядро Mathematica может выполняться на удаленном компьютере с более мощным процессором, или несколько ядер могут быть запущены из одного интерфейса.

Интерфейсный процессор – это и есть пользовательский интерфейс, включающий в себя окно редактирования, в которое мы вводим данные, строку меню, палитру инструментов, упрощающих ввод данных. Таким образом, типичный процесс взаимодействия пользователя с Mathematica состоит из следующих шагов:

- ввод данных в окно редактирования (пользователь и интерфейсный процессор);
- отправка введённых данных в ядро для выполнения вычислений (интерфейсный процессор);
- выполнение вычислений и отправка обратно в интерфейсный процессор (ядро);
- вывод результатов на экран в окно редактирования (интерфейсный процессор).

Интерфейсный процессор предоставляет графический интерфейс, который позволяет создавать и редактировать документы Mathematica, называемые блокнотами (notebooks). Блокноты позволяют «смешивать» в любой пропорции исходные данные и результаты вычислений Mathematica с текстом, графикой, рисунками и другими материалами. Вы можете использовать блокнот, как для выполнения текущих расчетов, так и в качестве средства для презентации или публикации результатов Вашей работы. Все содержание и форматирование блокнота могут быть сгенерированы алгоритмически или интерактивно редактироваться.

Несмотря на 25-летнюю историю системы Mathematica, ее язык программирования не имел общепринятого названия. Часто его называли языком Mathematica. Но в настоящее время этот язык стал использоваться и в других программных продуктах компании Wolfram Research, и поэтому с июня 2013 года он получил официальное название язык Wolfram. Этот мультипарадигмальный язык программирования спроектирован как максимально универсальный язык, с акцентом на символьные вычисления, функциональное программирование и программирование, основанное на правилах. Его создатели заложили в язык возможность реализовывать произвольные структуры и данные.

Язык Wolfram является интерпретирующим языком – это означает, что команды выполняются в реальном времени без необходимости компилировать их в программы.

Вы печатаете выражение, нажимаете комбинацию Shift+Enter, Mathematica отметит ввод выражения надписью `In[n]:=` и после этого ядро вычисляет значение выражения. Соответствующий вывод результата располагается под вводом и отмечается надписью `Out[n]=` [2].

[2] Во всех примерах в этой книге нумерация ввода/вывода отсутствует; это можно сделать при настройке параметров системы. Строки ввода и вывода различаются по формату: ввод выделяется полужирным шрифтом *Verdana* и вывод всегда следует после ввода.

Наряду со стандартной текстовой формой ввода, Mathematica поддерживает общую, нетекстовую форму ввода, такую как графика и управление интерактивными моделями, произвольно смешиваемую с вводом текста.

1. 1. Примеры 1.1. Вычисляем факториал:

100!

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

Находим предел отношения двух соседних чисел Фибоначчи в бесконечности:

Limit[Fibonacci[n] / Fibonacci[n - 1], n → Infinity]

$$\frac{1}{2}\left(1+\sqrt{5}\right)$$

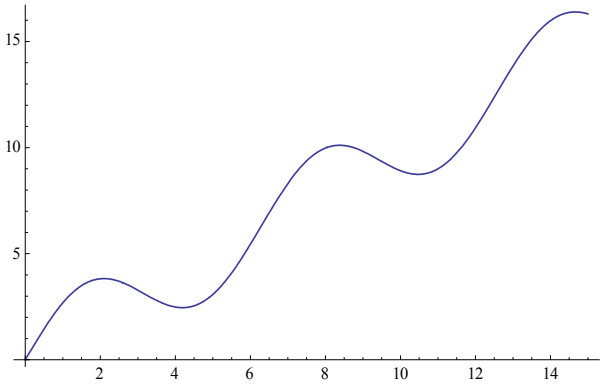
Находим разложение выражения на множители:

Factor[x^10 - 1]

$$(-1+x)(1+x)\left(1-x+x^2-x^3+x^4\right)\left(1+x+x^2+x^3+x^4\right)$$

График функции 2 sin(*x*) + *x* на отрезке от 0 до 15:

Plot [2 Sin[x] + x, {x, 0, 15}]



Имеются важные соглашения о нотации.

- Прописные и строчные буквы считаются разными знаками.
- Все встроенные символы, в частности, имена Mathematica-функций начинаются с большой буквы (функции и переменные, которые вы определяете, не имеют такого ограничения).
- Пробел в большинстве случаев интерпретируется как умножение, поэтому *a b*
- есть то же самое, что и *a*b* (но *ab* обозначает переменную с именем «*ab*»).
- Круглые скобки, ( ), используются только для группировки, как в выражении 2(3+5).
- Квадратные скобки, [ ], используются для аргументов функций.
- Фигурные скобки, { }, используются для обозначения списков, таких как {1, 2 , 3}.
- Запятые используются для разделения аргументов (список это тоже функция с именем List и с любым количеством аргументов).

1.2. Основные возможности и состав Mathematica

- Библиотека традиционных и специальных математических функций.
- Инструменты для работы с матрицами и данными, включая поддержку разреженных массивов.

- Поддержка комплексных чисел, рациональных и вещественных чисел произвольной точности, интервальной арифметики и символьных вычислений.
- Поддержка 2D и 3D-данных и функций визуализации и анимации.
- Решение систем уравнений, диофантовых уравнений, обыкновенных дифференциальных уравнений, уравнений в частных производных, дифференциальных алгебраических уравнений, дифференциальных уравнений с задержкой аргумента, стохастических дифференциальных уравнений и рекуррентных соотношений.
- Численные и символические инструменты для классического и дискретного анализа.
- Библиотеки многомерной статистики, включая интерполяцию и аппроксимацию данных, проверку гипотез, и расчеты вероятности и математического ожидания для более 100 различных распределений.
- Расчеты и моделирование случайных процессов и очередей.
- Локальная и глобальная оптимизации с ограничениями и без них.
- Язык Wolfram поддерживает различные парадигмы: процедурное, функционально и объектно-ориентированное программирование. Ещё один стиль программирования, основанный на правилах преобразований, непосредственно присущ системе, поскольку именно он лежит в основе возможности выполнения алгебраических преобразований.
- Инструментарий для создания пользовательского интерфейса для расчетов и приложений.
- Инструменты для 2D и 3D-обработки изображений и морфологической обработки изображений, включая распознавание изображений.
- Инструменты для визуализации и анализа графов.
- Инструменты для задач комбинаторики.
- Инструменты для интеллектуального анализа данных, такие как кластерный анализ, выравнивания последовательностей и сопоставление с образцом.
- Библиотека функций теории чисел.
- Инструменты для финансовых расчетов.
- Инструменты теория групп и тензоров.
- Библиотеки для обработки сигналов, в том числе вейвлет-анализ звуков, изображений и данных.
- Непрерывные и дискретные интегральные преобразования.
- Коллекция баз данных математической, научной и социально-экономической информации и доступ к Wolfram|Alpha данным и вычислений.

Wolfram|Alpha – база знаний и набор вычислительных алгоритмов. Wolfram|Alpha в ответ на запрос вычисляет ответ, основываясь на собственной базе знаний, которая содержит данные о математике, физике, астрономии, химии, биологии, медицине, истории, географии, политике, музыке, кинематографе, а также информацию об известных людях и интернет-сайтах. Он способен переводить данные между различными единицами измерения, системами счисления, подбирать общую формулу последовательности, находить возможные замкнутые формы для приближенных дробных чисел, вычислять суммы, пределы, интегралы, решать уравнения и системы уравнений, производить операции с матрицами, определять свойства чисел и геометрических фигур.

- Техническая обработка текстов, включая редактирование формулы и автоматизированную генерация отчетов.
- Инструменты для подключения к DLL. SQL, Java,. NET, C + +, Fortran, CUDA, OpenCL и http-систем.
- Инструменты для параллельного программирования.
- Использование как «свободной формы языкового ввода» (естественного языка пользовательского интерфейса) так и языка Wolfram при взаимодействии с Интернетом.

1.3. Способы выполнения приложений, написанных в системе Mathematica

1. Ноутбук выполняется в среде Mathematica. Любой ноутбук Mathematica может быть структурирован с использованием иерархии ячеек, что позволяет разбивать документ на секции, показывать и прятать различные виды информации. Документы могут быть представлены в виде слайдов для презентаций. Ноутбуки и их содержание представлены в виде выражений Mathematica, которые могут быть созданы, изменены или проанализированы программы Mathematica. Это позволяет совершать преобразования в другие форматы, например, TeX или XML.

Кроме выполнения ноутбуков в среде Mathematica есть несколько способов выполнение приложения, написанного на Mathematica, вне среды.

2. **Mathematica Player Pro** является средой выполнения, в которой работают любые приложения Mathematica, но эта среда не позволяет редактировать или создавать код.

3. Бесплатная программа **Wolfram CDF Player** предусмотрена для запуска ноутбуков Mathematica, которые были сохранены в формате вычислимого документа (CDF). Он также может просматривать стандартные Mathematica файлы, но не запускать их.

Формат вычисляемых документов (Computable Document Format или CDF) – это электронный формат документов, созданный с целью облегчения создания динамически сгенерированного интерактивного контента. Файлы CDF создаются в системе Mathematica..

CDF-документы предусматривает такие графические элементы пользователя, как ползунки, меню и кнопки. Содержимое документа обновляется с использованием встроенной вычислительной подсистемы при взаимодействии с графическими элементами пользователя. В документе могут использоваться текст, таблицы, изображения, звуки и анимации. Формат CDF предусматривает использование печатной вёрстки и символики математики и других наук и областей знаний. Также поддерживаются структурная иерархия документа. Этот формат включает в себя плагины для основных браузеров для Windows и Macintosh, которые позволяют вкладывать CDF-материалы в HTML-страницы.

Формат вычисляемого документа отличается от статических форматов, таких как PDF, и от таких форматов, как Adobe Flash, где динамическая информация предварительно сгенерирована. Благодаря тому, что программа CDF Player целиком содержит библиотеку времени исполнения системы Mathematica, содержимое документа может генерироваться в ответ на действие пользователя с помощью любых алгоритмов или визуализационных функций, которые можно описать в системе Mathematica. Это делает CDF особенно уместным для научного, инженерного и другого материала, а также электронных учебников.

4. Mathematica-код может быть преобразован в код C или автоматически созданный модуль DLL.

§ 2. Представление данных в Mathematica

Mathematica имеет дело со многими различного различными видами вещей: математическими формулами, списками, графиками и т. д. Хотя они выглядят различными Mathematica представляет все эти вещи одним способом – все они есть выражения.

Типичным примером выражения в Mathematica является `f[x, y]`. Вы можете использовать `f[x, y]` для представления математической функции $f(x, y)$. Эта функция имеет имя f и два аргумента x и y. Но вы не всегда пишите `f[x, y, z, ...]`. Например, `x + y` есть также выражение. Когда вы пишите выражение `x + y`, Mathematica преобразует его в стандартную полную форму `Plus[x, y]`. Затем когда это печатается снова, выражение изображается в первоначальном виде как `x + y`.

Фактически, все, что вы пишите в Mathematica, трактуется как выражения в стандартной форме. Рассмотрим примеры.

Таб. 2.1. Исходное представление и полное стандартное представление

«Естественное» (общепринятое) представление	Стандартное полное представление

x + y + z (сумма)	Plus[x, y, z]
x y z (умножение указывается пробелом)	Times[x, y, z]
x^n (возведение в степень)	Power[x, n]
{a, b, c} (список)	List[a, b, c]
a → b (правило для подстановки)	Rule[a, b]
a = b (присваивание)	Set[a, b]
a == b (равенство логическое)	Equal[a, b]

Вы можете получить стандартное полное представление любого объекта с помощью функции FullForm, например:

FullForm[1+x^2+(y+z)^2]

Plus[1, Power[x, 2], Power[Plus[y, z], 2]]

Объект f в выражении f[x, y, z, ...] называется **заголовком** выражения. Заголовок выражения можно получить, используя функцию Head.

Head[g[a, b]]

g

Head[a b c]

Times

Head[876]

Integer

Head[3.14]

Real

Head[Pi]

Symbol

Стандартная полная форма выражения позволяет легко сослаться на части любого выражения, так же как на части списка. Например, определим вложенный список с помощью присваивания

list = {a, {3, b},{6, x + y}};

Тогда его стандартная полная форма есть List[a, List[3, b], List[Plus[x, y]]]. Поэтому выражение x + y можно получить с помощью индексирования:

list[[3, 2]]

x + y

Точно также доступ к подвыражению x + y в выражение a + 3^b + f[4, x + y] получаем с помощью такого же индексирования

(a + 3^b + f[4, x + y])[[3, 2]]

x + y

§ 3. Автоматическое упрощение

Правила преобразования при автоматическом упрощении связаны с аксиомами поля. Рассмотрим применение в Mathematica некоторых аксиом.

Следствие закона дистрибутивности $(a + b)\, c = a\, c + b\, c$:

2x + y + (3/2)x

$$\frac{7\,x}{2} + y$$

С другой стороны этот закон не применяется при автоматическом упрощении выражения $a\,x + b\,x + b\,y$ (из-за неопределенности, какой символ рассматривать как коэффициент), а применяется частично:

2 x + √2 x + 3 x

$5\,x + \sqrt{2}\,x$

Общей практикой в системах компьютерной алгебры является группировка вместе только рациональных коэффициентов с последующим автоматическим упрощением.

Законы ассоциативности

$a + (b + c) = (a + b) + c$,
$(a \times b) \times c = a \times (b \times c)$

приводят к удалению лишних скобок.

Примеры (стрелка показывает результат автоматического упрощения):

$(w + x) + (y + z) \to w + x + y + z$,
$((w\,x)\,y)\,z \to w\,x\,y\,z$,
$(((u + v)\,w)\,x + y) + z \to (u + v)\,w\,x + y + z$.

Mathematica автоматически упорядочивает слагаемые и сомножители в суммах и произведениях:

{x + y, y + x}

{x + y, x + y}

{1 + x + y, 1 + y + x, y + 1 + x, y + x + 1, x + 1 + y, x + y + 1}

{1 + x + y, 1 + x + y, 1 + x + y, 1 + x + y, 1 + x + y, 1 + x + y}

2(x y z) + 3 x (y z) + 4 (x y) z

9 x y z

Одноместный минус заменяется произведением: $-u \to (-1) \times u$. Двуместный минус заменяется суммой: $u - v \to u + (-1) \times v$.

FullForm[-u]

Times[−1, u]

FullForm[u − v]

Plus[u, Times[−1, v]]

Частное представляется в виде выражения без знака дроби:

FullForm[u/v]

Times[u, Power[v, −1]]

Следующие основные тождественные преобразования применяются при автоматическом упрощении:

$u + 0 \to u$;

$u \times 0 \to 0$;

$u \times 1 \to u$;

$0^w \to 0$, если w – положительное целое или дробь,

$\to$ *неопределенно*, иначе;

$1^w \to 1$;

$v^0 \to 1$, если $v \neq 0$,

$\to$ *неопределенно*, если $v = 0$;

$u^1 \to u$.

В результате сумма не содержит нулевых слагаемых, произведение не содержит множителей 0 или 1, степень не содержит 0 или 1 в качестве показателя степени или основания.

3.1. Следующие правила преобразования при целом n выполняются автоматически

$$u^v \cdot u^w \to u^{v+w},$$

$$(u^v)^n \to u^{v \cdot n},$$

$$(u \cdot v)^n \to u^n \cdot v^n.$$

Почему последние два правила нельзя применять и в случае, когда n – не целое число? Чтобы исключить следующие ошибочные преобразования (показаны стрелками):

$$1 = 1^{1/2} = (i^4)^{1/2} \to i^2 = -1,$$

$$1 = 1^{1/2} = ((-1) \cdot (-1))^{1/2} \to (-1)^{1/2}(-1)^{1/2} = i^2 = -1.$$

Как работает Mathematica в этих случаях?

```
Sqrt[i^4]
```

1

```
Sqrt[(-1) (-1)]
```

1

§ 4. Упрощение алгебраических выражений

Существует много ситуаций, когда вы желаете написать алгебраическое выражение в наиболее простой форме. Трудно точно сформулировать, что мы подразумеваем под «простейшей» формой, но можно предложить следующую процедуру. Рассматриваем множество различных форм выражения и выбираем ту форму, которая состоит из наименьшего числа частей.

4.1. Функция Simplify[expr] пытается найти простейшую форму выражения, применяя различные стандартные алгебраические преобразования.

Функция FullSimplify[expr] пытается найти простейшую форму выражения, используя не только алгебраические преобразования, но и преобразования многих других видов.

```
Simplify[x^2 + 2x + 1]
```

$(1 + x)^2$

```
Simplify[x^10 - 1]
```

$-1 + x^{10}$

Мы часто используем Simplify, чтобы упростить какое-то выражение, полученное в результате вычисления. Интегрируем:

```
Integrate[1/(x^4 - 1), x]
```

$$-\frac{\text{ArcTan}[x]}{2}+\frac{1}{4}\text{Log}[1-x]-\frac{1}{4}\text{Log}[1+x]$$

Дифференцируем (символ % обозначает последний результат вычисления):

```
D[%,x]
```

$$-\frac{1}{4\ (1-x)}-\frac{1}{4\ (1+x)}-\frac{1}{2\ \left(1+x^2\right)}$$

Упрощаем результат:

```
Simplify[%]
```

$$\frac{1}{-1+x^4}$$

Функция Simplify не может упростить гиперболическое выражение, а FullSimplify преобразует в экспоненциальную форму:

```
Simplify[Cosh[x] - Sinh[x]]
```

```
Cosh[x] - Sinh[x]
```

```
FullSimplify[Cosh[x] - Sinh[x]]
```

$$e^{-x}$$

С помощью Simplify мы легко доказываем следующую теорему.

Теорема 4.2. Любое рациональное число a представимо в виде суммы трех кубов рациональных чисел.

Доказательство.

```
Simplify[((a^3 - 729)/(9 a^2 + 81 a + 729))^3 + ((-a^3 + 243 a + 729)/(9 a^2 + 81 a + 729))^3 + ((27 a^2 + 243 a)/(9 a^2 + 81 a + 729))^3]
```

```
a
```

♦

4.3. Преобразование выражений в различные формы. Функция Expand[expr] выполняет умножение и возведение в степень, а функция ExpandAll[expr] раскрывает все скобки. Присвоим e дробно-рациональное выражение

```
e = (x - 1)^2 (2 + x) / ((1 + x)(x + 3)^2)
```

$$\frac{(-1+x)^2\ (2+x)}{(1+x)\ (3+x)^2}$$

```
Expand[e]
```

$$\frac{2}{(1+x)\ (3+x)^2}-\frac{3\ x}{(1+x)\ (3+x)^2}+\frac{x^3}{(1+x)\ (3+x)^2}$$

```
ExpandAll[e]
```

$$\frac{2}{9+15x+7x^2+x^3}-\frac{3x}{9+15x+7x^2+x^3}+\frac{x^3}{9+15x+7x^2+x^3}$$

Together[expr] приводит все дроби к общему знаменателю у полученного выражения.

Together[%]

$$\frac{2-3x+x^3}{(1+x)(3+x)^2}$$

Функция Apart[expr] представляет выражение в виде суммы простых дробей.

Apart[$\frac{2-x+x^3}{(-3+x)^2(1-x)}$]

$$-1-\frac{13}{(-3+x)^2}-\frac{13}{2(-3+x)}-\frac{1}{2(-1+x)}$$

Factor[expr] представляет полученное выражение в виде произведения:

Factor[%]

$$-\frac{2-x+x^3}{(-3+x)^2(-1+x)}$$

Когда у нас выражение с одной переменной, то мы можем записать это выражение в виде суммы, произведения и т. д. Если выражение содержит несколько переменных, то здесь даже шире выбор возможных форм. Функция Collect[expr, x] группирует вместе термы по степеням x.

v = Expand[(3 + 2x)^2 (x + 2y)^2]

$$9x^2+12x^3+4x^4+36xy+48x^2y+16x^3y+36y^2+48xy^2+16x^2y^2$$

Collect[v, x]

$$4x^4+36y^2+x^3(12+16y)+x^2\left(9+48y+16y^2\right)+x\left(36y+48y^2\right)$$

Collect[v, y]

$$9x^2+12x^3+4x^4+\left(36x+48x^2+16x^3\right)y+\left(36+48x+16x^2\right)y^2$$

Функция TrigExpand[expr] преобразует тригонометрическое выражение в сумму термов.

TrigExpand[Tan[x] Cos[2x]]

$$\frac{3}{2}\text{Cos}[x]\,\text{Sin}[x]-\frac{\text{Tan}[x]}{2}-\frac{1}{2}\text{Sin}[x]^2\,\text{Tan}[x]$$

TrigFactor[expr] преобразует тригонометрическое выражение в произведение термов.

TrigFactor[%]

$$2\,\text{Sin}\left[\frac{\pi}{4}-x\right]\text{Sin}\left[\frac{\pi}{4}+x\right]\text{Tan}[x]$$

Simplify[%]

Cos[2 x] Tan[x]

4.4. Упрощение с допущениями. Преобразования выражений функциями, подобными Expand и Factor, всегда корректны, независимо от значений входящих переменных. Но в общем случае это не так.

Simplify$\left[\sqrt{x^2}\right]$

$\sqrt{x^2}$

Mathematica автоматически не упрощает Sqrt[x^2], так Sqrt[x^2] == x истинно только для некоторых значений x.

{Sqrt[4^2], Sqrt[(−4)^2]}

{4, 4}

Используем упрощение с допущениями:

{Simplify[Sqrt[x^2], x > 0], Simplify[Sqrt[x^2], x <= 0]}

{x, −x}

И автоматическое упрощение не всегда может быть выполнено:

2a + 2 Sqrt[a - Sqrt[- b]] Sqrt[a + Sqrt[- b]]

$2\,a + 2\sqrt{a - \sqrt{-b}}\ \sqrt{a + \sqrt{-b}}$

Simplify[2a + 2 Sqrt[a - Sqrt[- b]] Sqrt[a + Sqrt[- b]], a>0&&b>0]

$2\left(a + \sqrt{a^2 + b}\right)$

Пример с тригонометрическими функциями:

Simplify[ArcSin[Sin[x]], −Pi/2 < x < Pi/2]

x

В допущениях можно указывать некоторые области значений. Element[x, dom] утверждает, что x принадлежит области dom. Element[{x_1, x_2, …, x_n}, dom] утверждает, что все x_1, x_2, …, x_n принадлежат области dom.

Возможные области:
Integers – целые числа,
Reals – вещественные числа,
Primes – простые числа.

Simplify[Sqrt[x^2], Element[x, Reals]]

Abs[x]

Simplify[Sin[x + 2n Pi], Element[n, Integers]]

Sin[x]

Используется малая теорема Ферма (см. глава 6, теорема 5.7):

Simplify[Mod[a^p, p], Element[p, Primes] && Element[a, Integers]]

Mod[a,p]

Сейчас используется факт, что sin(x) (но не arcsin(x)) является вещественным числом, когда x – вещественное число.

Simplify[{Re[Sin[x]], Re[ArcSin[x]]}, Element[x, Reals]]

```
{Sin[x], Re[ArcSin[x]]}
```

§ 5. Процедурное программирование

Wolfram – необычный язык программирования. Хотя он имеет общие черты с типичными процедурными языками, такими, как Pascal и C, этот язык радикально отличается от всех предшествующих языков. Мы можем программировать в системе Mathematica, используя такие парадигмы программирования, как процедурное программирование, функциональное программирование и программирование, основанное на правилах преобразований. Кроме того, по-видимому, все алгоритмизуемые на данный момент математические операции присутствуют в Mathematica в виде встроенных функций.

По своей широте и глубине программные аспекты Mathematica настолько феноменальны, что мы можем рассмотреть только некоторые моменты в этом кратком обзоре. Ограничимся рассмотрением только тех особенностей языка программирования Wolfram, которые требуются для понимания приведенных программ в дальнейших главах.

Подробнее о языке программирования Wolfram см. в [32, 19, 14] или в help'е Mathematica – интерактивном электронном учебнике, содержащем более чем 100 000 примеров. Все примеры могут быть запущены или изменены прямо в документации (причем изменения не сохраняются, и help не изменяется), позволяя легко изучать новый функционал.

Опишем в первую очередь, как в Mathematica реализуется процедурное программирование. Как и во всех процедурных языках, в Mathematica используется понятие переменной. Переменные обычно задаются строчными буквами. В качестве переменной может использоваться целое слово. Переменная имеет значение, но поскольку возможно использование символьных вычислений, то значением переменной может быть любое математическое выражение, в частности, снова переменная. Если переменной не присваивалось никакое значение, то значением переменной является тогда она сама.

Например, вычислим значение переменной a:

```
a
```

a

Вычислим число π с 50 значащими цифрами и присвоим полученное значение переменной a:

```
a = N[Pi, 50];
```

Точка с запятой подавляет вывод результата, однако команда выполняется.

```
a^2
```

9.8696044010893586188344909998761511353136994072 41

Функция N аппроксимирует точное число приближенным. Mathematica все вычисления с числами проводит точно, если числа точные. Действия с приближенными числами выполняются с любой желаемой точностью.

```
b = 25/15 + 45/20 + Sin[c]
```

$$\frac{47}{12} + \text{Sin[c]}$$

Атомарные выражения суть числа и символы. Числа делятся на четыре класса: целые, рациональные, вещественные и комплексные. Количество цифр в числе не ограничивается. Символы – это последовательность букв, цифр и знака $, не начинающаяся с цифры. К символам относятся и именованные константы, имеющие предопределенное значения. Имена таких констант начинаются с большой буквы. Примерами таких констант являются Pi, E, Infinity; соответствующие математические объекты суть π, *e* и ∞.

Одним из видов сложных выражений являются списки. Список представляет собой удобную форму структурирования данных, по строению представляет собой множество элементов, заключённых в фигурные скобки { } и разделённых запятыми. Mathematica имеет большое количество встроенных функций, как математических, так и системных, но для программирования пользователь может определить свои функции.

Встроенную функцию Table можно использовать для создания списка.

Table[i^2, {i, 1, 10}]

{1, 4, 9, 16, 25, 36, 49, 64, 81, 100}

Выражение {i, 1, 10} называется **итератор** в этом контексте. Начальное значение 1 может быть опущено (тогда оно подразумевается по умолчанию). Или шаг может быть добавлен как в {i, 1, 10, 2}.

Некоторые списки, подобные списку целых чисел от *a* до *b*, могут быть получены более просто, используя Range.

Range[10, 20]

{10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}

Наряду со встроенными функциями, можно определить свою функцию, например, возводящую аргумент в квадрат.

f[x_] := x^2;

Знак подчерка представляет формальный параметр функции. Лексема := указывает на отложенное присваивание – это означает, что правая часть определения функции f не вычисляется до фактического вызова функции f.
Список из трех вызовов функции f:

{f[12], f[a], f[{a,b,c}]}

{144, a^2, {a^2, b^2, c^2}}

Почти все встроенные одноместные числовые функции (например, x^2) являются дистрибутивными: если такую функцию вызвать для списка элементов, то она выдаст список значений функции для каждого элемента. Тем самым, в данном случае можем обойтись без программирования.

Для выбора элементов списка, обладающих определенным свойством, удобно использовать функцию Select. Например, выбрать нечетные числа из первых 20 натуральных чисел:

Select[Range[20],EvenQ]

{2, 4, 6, 8, 10, 12, 14, 16, 18, 20}

Определим логическую функцию good, истинную для простого числа вида $100k + 1$. Мы можем легко найти все такие числа меньшие 1000.

good[x_] := PrimeQ[x] && Mod[x, 100] == 1;

Select[Range[1000], good]

{101, 401, 601, 701}

Типичные конструкции в процедурном программировании – ветвление и цикл. В Mathematica это можно реализовать различными способами. Остановимся сейчас на простейших.

Условная функция If[condition, t, f] выдает *t*, если condition есть True, в противном случае выдает f. Только одна ветвь в действительности вычисляется:

x = 1;
If[x ≠ 0, 1/x, Indeterminate]

1

Indeterminate – встроенная константа Mathematica, имеющая неопределенную величину.

Функция While[test, body] есть цикл с предусловием: body повторяется до тех пор, пока вычисление test дает True.

Пример: выполняем программу, вычисляющую наибольший общий делитель чисел $a = 27$ и $b = 6$:

```
{a, b} = {27, 6};
While[b ≠ 0, {a, b} = {b, Mod[a, b]}];
a
```

3

Для того чтобы можно было различать глобальные и локальные объекты, употребляется функция Module[{*x*, *y*,...}, *expr*], которая указывает, что переменные *x*, *y*,... – локальны. Локальным переменным при описании можно задавать начальные значения. В следующем примере *x* не имеет числового значения после вычисления функции f[2.0].

```
f[x0_] := Module[{x = x0},While[x > 0 ,x = Log[x]]; x]
f[2.0]
```

-0.366513

```
x
```

x

§ 6. Функциональное программирование

Программирование заключается в написании множества инструкций, которые будут применяться к входным данным. В то время как процедурная программа пошагово выполняет инструкции, функциональное программирование сводится к применению функций к своим аргументам.

Например, требуется из списка пар составить новый список, переставляя элементы в парах.

Традиционный процедурный подход:

```
lis = {{a, 1}, {b, 2}, {c, 3}};
temp = lis;
Do[{temp[[i, 1]], temp[[i, 2]]} = {temp[[i, 2]], temp[[i, 1]]},
                                  {i, 1,   Length[lis]}];
temp
```

{{1, a}, {2, b}, {3, c}}

Сначала мы создаем список temp, такой же как lis. Затем заменяем в цикле шаг за шагом пары в temp. И, окончательно, возвращаем новое значение temp.

Следующая простая процедура решает эту же задачу, используя структурную итерацию.

```
Table[{lis[[i, 2]], lis[[i, 1]]}, {i, 1, Length[lis]}]
```

{{1, a}, {2, b}, {3, c}}

Вот как выглядит функциональный подход для решения той же задачи:

```
Map[Reverse, lis]
```

{{1, a}, {2, b}, {3, c}}

Этот простой пример иллюстрирует несколько ключевых особенностей функционального программирования. В то время, как процедурный подход требует итерацию по циклу, используя на каждом шаге меняющееся значение i, функциональная программа применяет функцию Reverse сразу ко всему списку. Функциональный подход часто дает более прямую реализацию решения многих задач, особенно когда используются рекурсивные структуры данных. Заметим, что основной тип данных Mathematica – выражения – имеют как раз рекурсивную структуру.

До этого мы рассматривали встроенные функции Mathematica и сразу применяли их для решения задачи. Сейчас перейдем к изучению мощных и полезных конструкций функционального программирования, и будем программировать свои собственные функции, в первую очередь для манипулирования выражениями.

Обыкновенные функции, работающие с определенными структурами данных, мы абстрактно представляем как отображение данных из области определения в область значения функции. Абстракция существенно основывается на возможности представить отображения (функции) с помощью ламбда-выражений[3]. Но в ламбда-выражениях нет никаких ограничений на типы параметров, они могут быть не только простыми данными, но и функциями. Таким образом, возможен более высокий уровень абстракции – **абстракция вычислений**, которая качественно отличается от связанной с данными **абстракции отображения.**

Аппликативная форма записи в виде <оператор> <операнд> имеет более общий смысл в функциональном программировании, чем в обычном императивном программировании. Если традиционно считается, что для того чтобы вызвать функцию, как правило, может потребоваться вычисление аргумента (операнда), то в функциональном программировании может быть необходимо вычислить и оператор.

Функции в функциональных языках являются **величинами первого класса**, наравне с традиционными видами данных, такими, как числа или списки. Они могут быть результатами вычислений других функций, или параметрами других функций, или комбинациями более простых функций и т.д. В функциональном программировании имеется возможность оперировать функциями всевозможными способами, без ограничений. В частности, функция сама по себе может рассматриваться как структура данных (или как её часть).

Функциями **первого порядка** называются функции, аргументами для которых служат величины, сами не являющиеся функциями. Функции второго порядка оперируют с функциями первого порядка и т. д. **Функцией высокого порядка** называется функция, не являющаяся функцией первого порядка, другими словами, такая функция имеет в качестве аргумента какую-нибудь функцию.

Рассмотрим три наиболее мощные, общие и полезные функции – это Map, Apply и Thread. Эти функции обеспечивают весьма сложные и эффективные способы преобразования выражений. По сути, использование их являются сущностью функционального программирования в Mathematica.

6.1. Функция **Map** применяет свой первый аргумент-функцию к каждому элементу списка, являющегося вторым аргументом.

```
Map[Head, {3, 22/7, Pi}]
```

```
{Integer, Rational, Symbol}
```

[3] Функциональное программирование полностью описывается в рамках ламбда-исчисления, которое было изобретено Алонзом Чёрчем (Alonzo Church, американский логик и математик) около 1930 г. Современные функциональные языки (например, Haskell [12]) используют ламбда-исчисление как теоретическую модель и как промежуточный код при трансляции. В нашем изложении функционального программирования в Mathematica мы не будем явно использовать ламбда-исчисление; желающие могут познакомиться с ним по книгам [18, 22].

Можем оперировать с неопределенной одноместной функцией f и одноуровневым списком:

Map[f, {a, b, c}]

{f[a], f[b], f[c]}

Но, в более общей ситуации, когда второй аргумент является не только списком, а, например, выражением g[a, b, c], функция f «обёртывает» (to wrap) каждый элемент g.

Map[f, g[a, b, c]]

g[f[a], f[b], f[c]]

Действие Map на список есть частный случай. Чтобы в этом убедиться достаточно вспомнить полную форму списка (g есть List).

FullForm[{a, b, c}]

List[a, b, c]

6.2. Thread и MapThread. Функция Thread может выполнять достаточные сложные структурные преобразования выражений. Рассмотрим только несколько примеров использования.

Thread[f[{a, b, c}, {x, y, z}]]

{f[a, x], f[b, y], f[c, z]}

Подобное можно сделать с помощью функции MapThread:

MapThread[f, {{a, b, c}, {x, y, z}}]

{f[a, x], f[b, y], f[c, z]}

В отличие от Thread функция MapThread имеет два аргумента: первый аргумент – функция, а второй список списков.

Примеры конкретных функций f:

MapThread[Power, {{a, b, c}, {7, 5, 2}}]

$\{a^7, b^5, c^2\}$

MapThread[Equal, {{a, b, c}, {x, y, z}}]

{a == x, b == y, c == z}

Второй аргумент у MapThread может быть списком любого количества подсписков, например:

MapThread[List, {{a, b, c}, {x, y, z}, {1, 2, 3}}]

{{a, x, 1},{b, y, 2},{c, z, 3}}

Следующие примеры более интересны. Подробности смотрите в Help'е Mathematica.

Конвертирование уравнений для списков в список уравнений.

Thread[{a, b, c} == {x, y, z}]

{a == x, b == y, c == z}

Применение функции к обеим сторонам уравнения.

Thread[Log[x == y], Equal]

Log[x] == Log[y]

Формирование пар с постоянным вторым элементом.

Thread[{{a, b, c}, 0}]

```
{{a, 0}, {b, 0}, {c, 0}}
```

6.3. Apply. В то время как Map применяется, чтобы выполнить одну и ту же операцию над каждым элементом выражения, функция Apply меняет структуру выражения, точнее меняет заголовок выражения:

```
Apply[h, g[a, b, c]]
```

```
h[a, b, c]
```

Примеры:

```
Apply[Plus, f[1, 2, 3, 4]]
```

```
10
```

```
Apply[h, {1, 2, 3, 4}]
```

```
h[1, 2, 3, 4]
```

6.4. Анонимные функции. В процедурном программировании каждая функция имеет свое имя. Это удобно, если такая функция вызывается несколько раз в разных местах программы или во время интерактивного сеанса. Но в функциональном программировании функция может иметь функциональный параметр, который используется только раз и именовать функцию-параметр специально для этого вызова совершенно излишне.

Например, мы определяем функцию

```
f[x_] := x^2 + x^3;
```

чтобы преобразовать каждый элемент списка

```
Map[f, {1, 2, 3, 4}];
```

Для этого случая нет необходимости давать имя функции f, и те же действия можно сделать с помощью **анонимной функции** – функции без имени.

```
Map[#^2+#^3&, {1, 2, 3, 4}]
```

```
{2, 12, 36, 80}
```

Символ # обозначает формальный параметр анонимной функции (вместо него при вызове будет подставляться фактический параметр), а символ & показывает окончание определения анонимной функции. Если у анонимной функции несколько параметров, то соответствующие параметры обозначаются #1, #2 и т.д. Следующие примеры иллюстрируют применение анонимных функций.

Прибавляем один и тот же вектор к каждому вектору в списке:

```
Map[(# + {x, y})&, {{1,1}, {2,2}, {3,3}, {4,4}}]
```

```
{{1 + x, 1 + y}, {2 + x, 2 + y}, {3 + x, 3 + y}, {4 + x, 4 + y}}
```

Помещаем простые числа в рамку:

```
Map[If[PrimeQ[#], Framed[#], #]&, Range[20]]
```

$$\left\{1, \boxed{2}, \boxed{3}, 4, \boxed{5}, 6, \boxed{7}, 8, 9, 10, \boxed{11}, 12, \boxed{13}, 14, 15, 16, \boxed{17}, 18, \boxed{19}, 20\right\}$$

Применение функций из списка к соответствующим аргументам:

```
MapThread[#1[#2]&, {{f, g, h}, {x, y, z}}]
```

```
{f[x], g[y], h[z]}
```

6.5. Суперпозиция функций. Важнейшей особенностью функционального программирования является последовательное применение или суперпозиция функций. Встроенные функции Nest и Fold автоматизируют применение одной и той же функции.

Вызов Nest[f, expr, n] выдает выражение после *n*-кратного применения функции f к исходному выражению expr. Например,

Nest[(1 + #)^2&, x, 5]

$(1 + (1 + (1 + (1 + (1 + x)^2)^2)^2)^2)^2$

NestList выдает список последовательный применений:

NestList[#^2&, 2, 6]

{2, 4, 16, 256, 65536, 4294967296, 18446744073709551616}

Мы можем итерировать функцию-список:

```
Nest[{(#[[1]] + #[[2]])/2, Sqrt[#[[1]] #[[2]]]} &, {0.5, 1.0}, 10]
```

{0.728396, 0.728396}

Используем алгоритм Ньютона для получения приближенного значения $\sqrt{2}$:

Nest[(# + 2/#)/2&, 1.0, 5]

1.41421

6.6. Функции **Fold** и **FoldList** являются аналогами Nest и NestList применительно к функциям от двух аргументов. Выражение FoldList[f, x,{ a, b, …}] порождает список

{x, f[x, a], f[f[a,x], b], …},

а функция Fold от тех же аргументов имеет значением последний элемент этого списка. Например, определение факториала:

```
fact[n_] := Fold[Times, 1, Range[n]];
fact[20]
```

2432902008176640000

Еще несколько примеров из справочной службы Mathematica:

Fold[List, x, {a, b, c, d}]

{{{{x, a}, b}, c}, d}

Fold[#1^#2&, x, {a, b, c, d}];

$\left(\left(\left(x^a\right)^b\right)^c\right)^d$

Создаем полином по схеме Горнера:

Fold[x #1 + #2&, 0, {a, b, c, d, e}]

e + x (d + x (c + x (b + a x)))

Формируем число из цифр:

Fold[10 #1 + #2&, 0, {4, 5, 1, 6, 7, 8}]

451678

Находим знакочередующуюся сумму:

Fold[#2 – #1&, 0, Reverse[{a, b, c, d, e}]]

a – b + c – d + e

Накапливаем сумму элементов списка:

FoldList[Plus, 0, {a, b, c, d}]

{0, a, a + b, a + b + c, a + b + c + d}

§ 7. Программирование с правилами преобразований

Применение правил для преобразования выражений из одной формы в другую есть мощное и полезное средство в языке Wolfram. Множество из нескольких тысяч встроенных правил может неограниченно дополняться правилами, определенными пользователем. Правила могут быть созданы для изменения формы выражений, для фильтрации данных в соответствии с заданными критериями, и могут применяться для обширных классов выражений или для небольших множеств выражений с помощью использования определенных шаблонов (образцов). Правила могут решать те же задачи, которые обычно реализуются с помощью процедурного или функционального программирования.

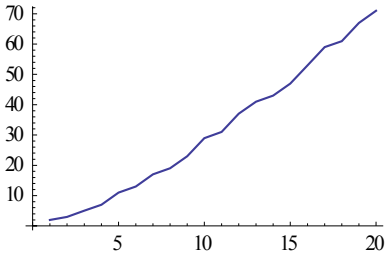
Пользователь Mathematica обычно встречает правила, когда использует некоторые встроенные функции. Например, правила появляются в выводе функции Solve, применяемой для решения уравнения.

```
solve = Solve[a x^2 + b x + c == 0, {x}]
```

$$\left\{\left\{x \to \frac{-b-\sqrt{b^2-4\,a\,c}}{2\,a}\right\}, \left\{x \to \frac{-b+\sqrt{b^2-4\,a\,c}}{2\,a}\right\}\right\}$$

Правила также используются для указания значений опций встроенных функций, например, на следующем графике правило Joined → True указывает, что точки графика должны быть соединены.

```
ListPlot[Table[Prime[n], {n, 20}], Joined → True]
```



7. 1. Создание и использование правил. Краткая форма для записи правила использует правую стрелку, которая создается вводом –> (без пробелов между – и >). Пользовательский интерфейс Mathematica автоматически преобразует –> в → как только продолжится ввод символов с клавиатуры. Оба символа являются краткой формой записи функции Rule.

Создадим следующее правило преобразования, которое может быть понято как «x преобразуется в 3»:

```
x → 3
```

```
x → 3
```

Глядя на вывод, замечаем, что правило ничего не делает: вывод является самим правилом. Это происходит потому, что правила ничего не делают, будучи сами по себе. Вам нужно использовать правило вместе с выражением, для которого правило было создано, чтобы извлечь из него пользу. Правила могут быть применены к выражению при помощи /. (это краткая форма записи функции ReplaceAll). Общий синтаксис этой функции следующий: *выражение* /. *правило*(*–а*)).

Используем /. для того, чтобы применить правило к выражению:

```
Sin[x] + x Cos[x] /. x → 3
```

```
3 Cos[3] + Sin[3]
```

Можно применить сразу несколько правил к выражению, поместив их в список:

Sin[x] + y Cos[z] /. {x → 3, y → 4, z → 5}

4 Cos[5] + Sin[3]

Вы можете менять подвыражение на любое выражение:

3x^4 + 5x^3 + a x^2 + 7 /. 5x^3 + 7→ y+x

x + a x^2 + 3 x^4 + y

Используем правило для f[x]. Обратите внимание, что это правило сопоставлено именно выражению f[x] и не влияет на f[y]:

1 + f[x] + f[y] /. f[x] → p

1 + p + f[y]

Для того, чтобы заменить функцию f независимо от ее аргумента, мы должны применить в правиле шаблон (образец). Правило f[x_] → x^2 может быть интерпретировано как "f[*любое*] заменяется на *любое*^2":

1 + f[x] + f[y] /. f[x_] → x^2

1 + x^2 + y^2

Шаблон x_ обозначает произвольное выражение, но мы можем ограничить возможные значения для x. Например, шаблон x_h обозначает любое выражение с заголовком h. Поэтому получаем

{a + b, 7, c, 5/7, 4 + 6x, 8} /. x_Integer → f[x]

{a + b, f[7], c, 5/7, f[4] + x f[6], f[8]}

{a + b, 7, c, 5/7, 4 + 6x, 8} /. x_Plus → g[x]

{g[a + b], 7, c, 5/7, g[4 + 6 x], 8}

Или мы можем потребовать для x выполнение некоторого предиката:

{6, -7, 3, 2, -1, -2} /. x_ /; x < 0 → w

{6, w, 3, 2, w, w}

Range[100, 110] /. x_ /; PrimeQ[x] → Framed[x]

$$\left\{100, \boxed{101}, 102, \boxed{103}, 104, 105, 106, \boxed{107}, 108, \boxed{109}, 110\right\}$$

В последнем примере встроенная функция Framed в качестве результата выдает аргумент, помещенный в рамку.

В Mathematica имеются различные функции для решения разнообразных уравнений. Большей частью они дают решения в виде правил. Необходимо уметь использовать эти правила для изучения и интерпретации полученных результатов. И, несмотря на то, что многие из методов использования таких решений являются специфичными для конкретной решаемой задачи, неизменно понадобится выполнение двух основных действий: извлекать правило, представляющее собой решение, из списка, а затем применять его к выражению.

Функция Solve предназначена для решения различных систем уравнений и неравенств от нескольких неизвестных. Рассмотрим простой пример решения квадратного уравнения:

s = Solve[x^2+b x + c == 0, x]

$$\left\{\left\{x \to \frac{1}{2}\left(-b - \sqrt{b^2 - 4c}\right)\right\}, \left\{x \to \frac{1}{2}\left(-b + \sqrt{b^2 - 4c}\right)\right\}\right\}$$

Чтобы использовать это решение, например, для вычисления x^2 + x, прежде всего, нужно извлечь его из вложенного списка s. Получаем два правила:

s[[1]]

$$\left\{x \to \frac{1}{2}\left(-b - \sqrt{b^2 - 4c}\right)\right\}$$

s[[2]]

$$\left\{x \to \frac{1}{2}\left(-b + \sqrt{b^2 - 4c}\right)\right\}$$

Теперь можно использовать решения при помощи /., подставив эти решения в выражение x^2 + x и упростив:

Simplify[x^2 + x /. s[[1]]]

$$\frac{1}{4}\left(-2 + b + \sqrt{b^2 - 4c}\right)\left(b + \sqrt{b^2 - 4c}\right)$$

Simplify[x^2 + x /. s[[2]]]

$$\frac{1}{4}\left(-b + \sqrt{b^2 - 4c}\right)\left(2 - b + \sqrt{b^2 - 4c}\right)$$

В то время как /. применяет правило к выражению только один раз, возможно, будет полезным многократно применить правило к выражению, пока последнее не перестанет изменяться. Это можно сделать при помощи //.

Используем /. для того, чтобы применить правило, или список правил, к каждой части выражения лишь один раз:

a + x^2 + y^6 /. {x → 2 + a, a → 1}

1 + (2 + a)^2 + y^6

Используем //. для того, чтобы применить правила многократно, до тех пор, пока выражение более не изменяется:

a + x^2 + y^6 //. {x → 2 + a, a → 1}

10 + y^6

Аналогичным образом:

log[a b c d] /. log[x_ y_] → log[x] + log[y]

log[a] + log[b c d]

log[a b c d] //. log[x_ y_] → log[x] + log[y]

log[a] + log[b] + log[c] + log[d]

Задача 7.2. Чтобы получить последовательность Туэ-Морса, начинаем с 0 и затем повторяем на каждом шаге замену: 0 заменяем 01 и 1 заменяем 10. Первые четыре члена последовательности:
0
01
0110
01101001
Определить функцию, вычисляющую *n*-й член последовательности.

Мы можем использовать подстановки с правилами:

f[n_] := FromDigits[Nest[Flatten[# /. {0 -> {0, 1}, 1 -> {1, 0}}]&, {0}, n]]

```
f[3]
1101001
```

Встроенная функция FromDigits преобразует список цифр в соответствующее число, а функция Flatten преобразует вложенный список в одноуровневый список. Детали смотрите в справочной службе Mathematica.

Предложенного описания системы Mathematica, на наш взгляд, достаточно, что понять приведенные в книге программы. Некоторые программы будут дополнительно комментироваться.

§ 8. Псевдокод для алгоритмов

Все алгоритмы в этой книге написаны на особом псевдокоде, который можно охарактеризовать как смешение естественного языка, языка Wolfram и Паскаля. Надеемся, что такое смешение поможет математикам понимать алгоритмы, а программистам переводить алгоритмы в программы. Wolfram и Паскаль используются в псевдокоде в тех случаях, когда эти языки программирования помогают проще и понятнее естественного языка изобразить необходимые структуры данных и элементы управления. Естественный язык применяется в тех случаях, когда используется математическая терминология и когда используемые предложения не приводят к двусмысленности.

По-видимому, проще всего на примерах пояснить, как следует понимать фрагменты псевдоязыка.

Комментарии в тексте алгоритм выглядят как
(* комментарий *),
причем могут занимать несколько строчек.

Если необходимо изобразить последовательное выполнение шагов алгоритма, то они разделяются между собой символом «;» и иногда шаги нумеруются.

Например (см. алгоритм 4.4, глава 6),

Находим c – решение сравнения $c\,m_1 \equiv 1 \pmod{m_2}$;
$b = r_1 \bmod m_1$;
$s = (r_2 - r_1)c \bmod m_2$;
$x = b + s\,m_1$;
return(x)

Здесь символ «=» обозначает присваивание переменным соответствующих значений, и оператор return возвращает результат работы алгоритма.

Если нужно сделать ветвление в алгоритме, то оно описывается с помощью условного оператора Паскаля (см. алгоритм 6.9, глава 6):

If $p \equiv 3, 7 \pmod 8$ then begin $x = a^{(p+1)/4} \bmod p$; return(x) end;

При необходимости используются begin и end, и случай «иначе» задается при помощи else.

Повторение в алгоритме описывается с помощью циклов Паскаля while и for, например (см. алгоритм 6.8, глава 6),

```
While a – четно do
        begin
        a = a/2;
        If m mod 8 ∈ {3, 5} then t = – t;
        end;
```

и (см. алгоритм 4.7, глава 6)

```
for k = 2 to n do
    begin
    вычисляем x – выход алгоритма 4.4 с входом x, r_k, P, m_k;
    P = P·m_k
```

```
end;
```

Из языка Wolfram взято одновременное присваивание, например, выполнение

```
{x, y} = {exprx, expry}
```

дает x значение exprx, а y – значение expry.

И, наконец, символы == обозначают логическое равенство, т.е. операцию, которая выдает истину или ложь.

Правильному применению методов можно научиться, только применяя их на разнообразных примерах.

Иероним Георг Цейтен – датский математик и историк математики

Трудность решения в какой-то мере входит в само понятие задачи: там, где нет трудности, нет и задачи.

Д. Пойа

Задачи и упражнения

1. Целое x удовлетворяющее уравнению

$$2682440^4+15365639^4+18796760^4 = x^4$$

опровергает гипотезу Эйлера, что сумма трех четвертых степеней никогда не является сама четвертой степенью. Это предположение было открытым почти 200 лет, пока Ноам Элкис (Noam Elkies) из Гарварда не обнаружил данный контрпример в 1988 г. Найдите x.

2. Проверить, что формула Эйлера x^2+x+41 дает простое число для всех целых x от 0 до 39.

3. Докажите, что

$$\text{Tan}[3\,\text{Pi}\,/\,11] + 4\,\text{Sin}[2\,\text{Pi}\,/\,11] == \sqrt{11}$$

4. Какова вероятность, что случайно выбранное 12-значное число будет простым?

5. Пусть $a = 98$ и $b = 75$. Покажите, что $a^a\ b^b$ оканчивается точно на 98 нулей.

6. Докажите, что произведение четырех последовательных натуральных чисел в сумме с 1 есть полный квадрат.

7. Представьте

$$\frac{1}{1+x} + \frac{1}{1+\frac{1}{1+x}} + \frac{1}{1+\frac{1}{1+\frac{1}{1+x}}}$$

в виде одной дроби, и преобразуйте ее в сумму дробей с минимальными знаменателями.

8. Покажите, что среди первых 450 чисел Фибоначчи количество нечетных чисел в два раза больше чем четных.

9. Пусть m – натуральное число и

$$A = \frac{(m+3)^3+1}{3m}$$

Найдите все m меньшие 1000, такие что A – целое.

10. Найдите единственное положительное $n < 1000$ такое, что $n! + (n + 1)!$ есть полный квадрат.

11. Какая цифра не является последней у первых 20 чисел Фибоначчи?

12. Найдите первые пять положительных целых чисел n таких, что $n^6 + 1091$ есть простое число. Убедитесь, что все они находятся между 3500 и 8500.

13. Покажите, что среди первых 200 простых чисел p числа {3, 7, 13, 43, 137} – это как раз те, для которых остаток от деления 19^{p-1} на p^2 равен 1.

14. Объясните, что делают следующие функции:

```
g[n_] := Times@@Apply[Plus, Inner[List, x^Range[n], 1/x^Range[n], List], 1]
t[n_]:=Times@@Apply[Plus, Thread[List[x^Range[n],1 /x^Range[n]]], 1]
```

15. Объясните, что вычисляют следующие функции

```
Power@@(x+y);
Plus@@@(x^y+y^z);
```

16. Докажите, что

$(1 + 2 + 3 + ... + n)^2 = 1^3 + 2^3 + 3^3 + ... + n^3$.

17. Вычислите

$$\sum_{k=1}^{n} \frac{k}{k^4 + k^2 + 1}$$

18. Напишите функцию для вычисления ряда

$$s(n) = \frac{1}{1} + \frac{1}{1+2} + ... + \frac{1}{1+2+...+n}.$$

19. Определите

$$S(k, n) = \sum_{i=1}^{n} i^k$$

Покажите, что

$$\sum_{a=0}^{n} \frac{S(2, 3a + 1)}{S(1, 3a + 1)}$$

является всегда точным квадратом.

20. Определите, для каких целых n из диапазона $1 \le n \le 10$ число $\sin(\pi/n)$ выражается в радикалах? Тот же вопрос для $1 \le n \le 1000$.

21. Сколькими нулями оканчивается 10000!? Напишите программу для получения ответа.

22. Определите количество простых чисел среди первой тысячи чисел Фибоначчи.

23. Найдите все целые $0 < n < 20000$, такие, что 1997 делит $n^2 + (n + 1)^2$. Сделайте тоже самое для 2009.

24. Заметим что $12^2 = 144$ и $21^2 = 441$, т.е. числа 12 и 21 и их квадраты являются взаимно обратимыми (цифры написаны в противоположном порядке). Найдите все числа до 10000, обладающие этим свойством.

25. Числовой палиндром – это натуральное число, которое читается слева направо и справа налево одинаково. Найдите все палиндромы, меньшие 5000, которые являются простыми числами. Таких чисел – 20.

26. Обращение натурального числа – это число, получаемое из исходного, когда цифры записываются в обратном порядке. Найдите все натуральные числа, меньшие 5000, которые вместе со своими обращениями являются простыми числами.

27. Найдите все такие целые числа $0 < m < 10^5$, что четвертая степень количества положительных делителей числа m равна m.

28. Для каких чисел $n \le 5000$ выполнено `Mod[Fibonacci[n], n] == 0`? Предложите гипотезу.

29. Определите функцию `squareFreeQ[n]`, которая возвращает `True`, если `n` не делится на какой-нибудь квадрат, и `False` в противном случае. Протестируйте ее с помощью встроенной функции `SquareFreeQ`.

30. Число с n цифрами называется циклическим, если, умножая его на 1, 2, 3, ..., n, мы получаем каждый раз те же цифры, но в другом порядке. Найдите единственное шестизначное циклическое число.

31. Функцию $cs(n) = 1^3 + 2^3 + ... + n^3$
можно определить как

```
cs[n_] := Apply[Plus, Range[n]^3]
```

или, используя сокращение для `Apply`, можно записать как

```
cs[n_] := Plus @@ (Range[n]^3)
```

Определите подобным образом функции $ep(n) = 1 + 1/1! + 1/2! + ... + 1/n!$ и
$p(n) = (1 + x)(1 + x^2) \cdots (1 + x^n)$.

32. Покажите, что 2 и 3 – единственные числа n, меньшие 1000, для которых
$3^n + 4^n + ... + (n + 2)^n = (n + 3)^n$.

33. Число называется Harshad-числом, если оно делится на сумму его цифр (например, 12 есть Harshad-число, так как оно делится на 1+2=3). Найдите все двухзначные Harshad-числа. Как много сушествует пятизначных Harshad-чисел?

Harshad переводится с санскрита, как «дающее радость». Эти числа определил и назвал индийский математик D. Kaprekar.

34. Найдите все числа в первом миллионе, которые имеют следующее свойство: если $n = d_1d_2 ...d_k$, то $n = d_1! + d_2! + ...+ d_k!$ (например, $145 = 1! + 4! + 5!$).

35. Число называется совершенным, если оно равно сумме его собственных делителей, например, $6 = 1 + 2 + 3$, но $18 \neq 1 + 2 + 3 + 6 + 9$. Напишите программу, находящую все совершенные числа до 10000.

36. Среди первых 100000 найдите наибольшее число n, которое делится на все положительные числа $\le \sqrt{n}$.

37. Покажите, что сумма всех делителей числа
608655567023837898967037173424316962265783077335188597052832486051279169126 4
есть совершенное число (см. задачу 35).

38. Определите функцию Коллотца:

```
collatz[x_Integer?EvenQ]:=x/2
collatz[x_Integer]:=3x+1
```

Коллотц (L. Collatz) в 1937 высказал гипотезу, что если эту функцию многократно применять к любому натуральному числу, то, в конце концов, получим 1. Найдите, сколько раз функцию Коллатца необходимо применять к числам меньшим 1000, чтобы получить 1.

39. Напишите функцию для конвертирования $\{x_1, x_2, \ldots, x_k, x_{k+1}\}$ в $\{x_1 \to x_2, x_2 \to x_3, \ldots, x_k \to x_{k+1}\}$.

40. Докажите тождество Эйлера $(a^2 + b^2 + c^2 + d^2)\,(t^2 + x^2 + y^2 + z^2) =$
$= (a\,t + b\,z - c\,y + d\,x)^2 + (a\,y + b\,x + c\,t - d\,z)^2 + (a\,z - b\,t + c\,x + d\,y)^2 + (a\,x - b\,y - c\,z - d\,t)^2$.

41. Доказать, что среднеарифметическое двух неотрицательных чисел не меньше среднегеометрического этих чисел. Используйте Mathematica.

42. Для натурального $n>1$ пусть $f(n)$ равно 1 плюс сумма простых делителей числа n, причем каждое простое число в сумме берется такое число раз, какое стоит в показателе степени в разложении на простые множители числа n. Например, так как $680400 = 2^4 \cdot 3^5 \cdot 5^2 \cdot 7$, то $f(680400) = 1+ 4\cdot2+5\cdot3+2\cdot5+7 = 41$.

Имеем $f(7) = 8$ и $f(8)=7$. Проверьте, что для любого натурального $6 < n \le 1000$ последовательность $f(n), f(f(n)), f(f(f(n))), f(f(f(f(n))))$,… всегда приводит к циклу (7, 8).

43. Введем обозначение S_n для суммы первых n простых чисел $S_n = 2 + 3 + 5 + 7 + 11 + \ldots + p_n$. Проверьте, что для любого натурального $1 < n \le 1000$ между S_n и S_{n+1} всегда есть точный квадрат.

44. Господин *S* и господин *P*.
Выберем два натуральных числа, больших единицы, но меньших ста. Значение их суммы сообщено господину *S*, значение их произведения – господину *P*. Господин *P* звонит по телефону господину *S*.

P: Я не могу найти эти два числа.

S: Я знаю, что вам это и не удалось бы.

P: Ах, так… . Но тогда я их знаю!

S: Ну, тогда и я тоже их знаю!

Найдите эти числа с помощью Mathematica.

Глава 3. Пропедевтика

В главе описываются алгебраические понятия, используемые в книге, и вводятся понятия сложности алгоритмов и вычислительных задач.

Математика — это искусство давать
одно и то же имя разным предметам.
Анри Пуанкаре

§ 1. Алгебра

1.1. Всюду в этой книге мы будем обозначать:
множество натуральных чисел – $\mathbb{N}$,
натуральные числа вместе с нулем – $\mathbb{N}_0$,
множество целых чисел – $\mathbb{Z}$,
множество рациональных чисел – $\mathbb{Q}$,
множество вещественных чисел – $\mathbb{R}$,
множество комплексных чисел – $\mathbb{C}$.

1.2. Полугруппа $(S, *)$ есть множество S вместе с ассоциативной бинарной операцией $*$ на S. Примером полугруппы является множество натуральных чисел с операцией сложения. Но с операцией вычитания даже целые числа не образуют полугруппу, так как вычитание не ассоциативно.

1.3. Моноид $(S, *, e)$ есть полугруппа с **нейтральным элементом** e; т.е. $e * x = x * e = x$ для всех $x \in S$. Если операция $*$ коммутативна, то полугруппа или моноид называются **коммутативными**. Примерами коммутативных моноидов являются множества $\mathbb{N}_0$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ и $\mathbb{C}$ с операцией сложения и нулем в качестве нейтрального элемента, а $(\mathbb{N}, +)$ – только полугруппа. Но $(\mathbb{N}, \cdot, 1)$ – коммутативный моноид.

1.4. Группа $(G, *, \lozenge)$ есть моноид $(G, *, e)$ вместе с унарной операцией **обращение** (или инверсия) $\lozenge$, для которой $x * (\lozenge x) = e = (\lozenge x) * x$ для всех $x \in G$. Обычно элемент $\lozenge x$ называется **симметричным** для x. Группа G называется **коммутативной** или **абелевой**, если операция $*$ коммутативна. Моноиды $(\mathbb{Z}, +, –)$, $(\mathbb{Q}, +, –)$, $(\mathbb{R}, +, –)$ и $(\mathbb{C} +, –)$ являются группами, где $– x$ обозначает изменение знака числа x. Моноид $(\mathbb{N}_0, +, 0)$ не являются группой (обратный элемент существует только для 0; таким образом, например, 1 уже не имеет обратного элемента).

1.5. Кольцом $(R, +, –, \cdot, 0)$ называется абелева группа $(R, +, –, 0)$ с нейтральным элементом 0 и одновременно полугруппа $(R, \cdot)$, удовлетворяющая законам дистрибутивности $x \cdot (y + z) = x \cdot y + x \cdot z$ и $(y + z) \cdot x = y \cdot x + z \cdot x$. **Коммутативное кольцо** есть кольцо, для которого операция $\cdot$ коммутативна. **Кольцо с единицей** есть кольцо R вместе с элементом $1 (\neq 0)$, таким что $(R, \cdot, 1)$ есть моноид. Обычно мы будем использовать символы $+$, $–$, $\cdot$, 0, 1 для операций кольца. Мы будем называть эти операции **сложением**, **минусом**, **умножением**, **нулем** (нульарная операция, которая всегда выдает 0), **единицей** (нульарная операция, которая всегда выдает 1). Умножение записывается обычно просто конкатенацией. Запись a^n обозначает $aaa\ldots a$ (элемент a повторяется n раз), $a^0 = 1$. Операция **вычитания** (которая записывается также –) определяется как $x – y := x + (– y)$ для $x, y \in R$.

Ряд свойств типа

$$a \cdot (b – c) = a \cdot b – a \cdot c,$$
$$a \cdot 0 = 0 \cdot a = 0$$

являются следствиями определения кольца. Элемент $a – b$ является решением уравнения $b + x = a$.

Примеры 1.6.

- Четные числа с обычными операциями сложения и умножения составляют кольцо, нечетные – нет.
- $\mathbb{Z}$ – кольцо целых чисел с обычными операциями сложения и умножения. Множество $m\mathbb{Z}$ целых чисел, делящихся на m, будет в $\mathbb{Z}$ подкольцом (без единицы при $m > 1$).
- Кольцами с единицей являются множество рациональных чисел $\mathbb{Q}$ и множество вещественных чисел $\mathbb{R}$, множество комплексных чисел $\mathbb{C}$, причём естественные включения $\mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R} \subseteq \mathbb{C}$ определяют цепочки подколец кольца $\mathbb{C}$.
- Кольцо квадратных матриц порядка n также можно рассматривать над некоторым кольцом R**,** подразумевая, что элементы матриц берутся из R. Сложение и умножение матриц происходит по обычным матричным формулам с интерпретацией операций в R. Если $n > 1$, то умножение квадратных матриц в общем случае не перестановочно, поэтому кольца квадратных матриц в общем случае не коммутативны.

1.7. Гауссовы целые числа – это комплексные числа, у которых вещественная и мнимая части есть целые числа: $\{a + bi \mid a, b \in \mathbb{Z}\}$. Множество гауссовых целых чисел образует подкольцо в $\mathbb{C}$ и это кольцо обозначается через $\mathbb{Z}[i]$.

1.8. Пусть a и b – ненулевые элементы коммутативного кольца R с единичным элементом 1. Говорят, что «***a* делит *b***» или «***a* – делитель *b***» (и пишут $a \mid b$), если и только если существует элемент $x \in R$ такой, что $ax = b$. Делимость транзитивна: если a делит b и b делит c, то a делит c. Если a делит b и c, то a делит также их сумму $b + c$ и разность $b – c$. Для кольца R с единицей элементы $a \in R$, которые делят 1, называются **единицами** или **делителями единицы** или **обратимыми**. Если $ab = 1$, то говорят, что b есть результат деления 1 на a в кольце R, и это записывается как $b = a^{-1}$. Тем самым в кольце вводится унарная операция «обращение» a^{-1}, применимая только для обратимых элементов. Все обратимые элементы кольца R с единицей составляют группу $U(R)$ по умножению. Если $a = bu$ для единицы u, то a и b называются **ассоциированным***и.* Если c делит $a – b$, то мы говорим, что ***a* сравнимо с *b* по модулю *c***; пишем $a \equiv b \pmod{c}$. Для каждого c отношение сравнения по модулю c есть отношение эквивалентности.

1.9. Если $a \cdot b = 0$ при $a \neq 0$ и $b \neq 0$ в кольце R, то a называется **левым**, а b – **правым делителем нуля** (в коммутативном кольце R говорят просто о **делителях нуля**). Обратимый элемент в кольце не может быть делителем нуля.

Сам нуль в кольце $R \neq 0$ – **тривиальный делитель нуля**. Если других делителей нуля нет (кроме 0), то R называется **кольцом без делителей нуля**. Коммутативное кольцо с единицей $1 \neq 0$ и без делителей нуля называют **целостным кольцом** (**кольцом целостности** или **областью целостности**). В области целостности R выполняется **закон сокращения**: если $ac = bc$ и $c \neq 0$, то $a = b$.

Кольца $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ и $\mathbb{C}$ являются целостными.

Отметим следующие основные свойства делимости в целостном кольце R.

1) Если $a|b$, $b|c$, то $a|c$. Действительно, мы имеем $b = ab'$, $c = bc'$, где $b', c' \in R$. Поэтому $c = (ab')c' = a(b'c')$.

2) Если $c|a$ и $c|b$, то $c|(a \pm b)$. В самом деле, по условию $a = ca'$, $b = cb'$ для некоторых $a', b' \in R$, и ввиду дистрибутивности $a \pm b = c(a' \pm b')$.

3) Если $a|b$, то $a|bc$. Ясно, что $b = ab'$ влечет $bc = (ab')c = a(b'c)$.

Комбинируя 2) и 3) получаем

4) Если каждый из элементов $b_1, b_2, \ldots, b_m \in R$ делится на $a \in R$, то на a будет делиться также элемент $b_1c_1 + b_2c_2 + \ldots + b_mc_m$, где $c_1, c_2, \ldots, c_m$ – произвольные элементы.

1.10. Поле $(K, +, –, \cdot^{-1}, 0, 1)$ есть коммутативное кольцо с единичным элементом $(K, +, –, 0, 1)$ и одновременно группа $(K\backslash\{0\}, \cdot^{-1}, 1)$. Если все операции в K вычислимы, то мы называем K **вычислимым полем**. Кольца $\mathbb{Q}$, $\mathbb{R}$ и $\mathbb{C}$ являются полями. Любое поле не имеет делителей нуля, но обратное утверждение неверно. В кольце $\mathbb{Z}$ целых чисел нет делителей нуля, но оно не является полем.

Если D есть область целостности, то $Q(D)$ есть **поле частных** (или **поле отношений**) области D, которое определяется как фактор-множество

$$Q(d) = \{\frac{a}{b} \mid a, b \in D, b \neq 0\} / \sim,$$

где $\frac{a}{b} \sim \frac{a'}{b'} \Leftrightarrow ab' = a'b.$

Операции +, –, ·, $^{-1}$ могут быть определены на представителях классов эквивалентности $Q(D)$ следующим образом

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}, \quad \frac{a}{b} \cdot \frac{c}{d} = \frac{ac}{bd},$$
$$-\frac{a}{b} = \frac{-a}{b}, \quad (\frac{a}{b})^{-1} = \frac{b}{a}.$$

Классы 0/1 и 1/1 есть нулевой и единичный элементы соответственно. $Q(D)$ есть наименьшее поле, содержащее D.

Поле $\mathbb{Q}$ есть поле отношений для $\mathbb{Z}$.

1.11. Пусть $(R, +, –, 0, 1)$ – коммутативное кольцо с единицей 1. **Многочленом (полиномом) над *R* от одной переменной** называется формальное выражение

$$f = f(x) = f_n x^n + f_{n-1} x^{n-1} + \ldots + f_1 x + f_0, \qquad (1)$$

где $f_0, \ldots, f_n \in R$ называются коэффициентами (и обозначаются coeff(f, i)), а переменная x рассматривается как формальный символ без определенного значения. Мы говорим, что (1) является многочленом **степени *n* со старшим коэффициентом f_n**, если $f_n \neq 0$; в этом случае запишем $\deg(f) = n$ и $\mathrm{lc}(f) = f_n$. Кроме того, по определению $\deg(0) = -\infty$ и $\mathrm{lc}(0) = 0$, где 0 обозначает **нулевой** многочлен, т.е. многочлен, у которого все коэффициенты равны нулю. Причем мы считаем, что для $-\infty$ выполняются свойства: $-\infty + (-\infty) = -\infty$, $-\infty + n = -\infty$, $-\infty < n$ для каждого $n \in \mathbb{N}_0$. Коэффициент coeff(f, 0) называется постоянным членом. **Старшим мономом** многочлена $f(x)$ будем называть $\mathrm{lt}(f) = x^{\deg(f)}$. Мы говорим также, что, что f – **нормированный** многочлен, если его $\mathrm{lc}(f) = 1$.

Сложение, вычитание и умножение многочленов определяется естественным образом, как если бы переменная x была бы элементом R: мы складываем или вычитаем многочлены посредством вычисления коэффициентов при одинаковых степенях x. Умножение выполняется согласно правилу

$$(u_r x^r + \ldots + u_0)\,(v_s x^s + \ldots + v_0) = (w_{r+s} x^{r+s} + \ldots + w_0),$$

где

$$w_k = u_0 v_k + u_1 v_{k-1} + \ldots + u_{k-1} v_1 + u_k v_0.$$

Множеством всех многочленов над R от переменной x вместе с операциями сложения и умножения образует кольцо; нейтральными элементами служит нулевой 0 и единичный элемент 1 ($p_0 = 1$, $p_k = 0$ для $k > 0$). Это кольцо обозначается $R[x]$.

Непосредственно из определения операций сложения и умножения в $R[x]$ следует, что для любых двух многочленов f и g степеней n и m соответственно имеют место неравенства

$$\deg(f + g) \leq \max(\deg(f), \deg(g)), \quad \deg(fg) \leq \deg(f) + \deg(g).$$

Второе из этих неравенств на самом деле заменяется равенством

$$\deg(fg) = \deg(f) + \deg(g)$$

всякий раз, когда произведение $\mathrm{lc}(f)\,\mathrm{lc}(g)$ старших коэффициентов многочленов f и g отлично от нуля.

1.12. Подполем F поля P называется подкольцо в P, само являющееся полем. Например, поле рациональных чисел $\mathbb{Q}$ — подполе поля вещественных чисел $\mathbb{R}$.

В случае $F \subset P$ говорят также, что поле P является **расширением** своего подполя F. Из определения подполя следует, что нуль и единица поля P будут содержаться также в F и служить для F нулём и единицей. Если взять в P пересечение G всех подполей, содержащих F и некоторый элемент $a \in P$, не принадлежащий F, то G будет минимальным полем, содержащим множество $\{F, a\}$.

Говорят, что расширение G поля F получено **присоединением** к F элемента a, и отражают этот факт записью $G = F(a)$. Аналогично можно говорить о подполе $G = F(a_1, a_2, \ldots, a_n)$ поля P, полученном присоединением к F n элементов $a_1, a_2, \ldots, a_n$ поля P.

Пусть K, L – поля такие, что $K \subset L$. Пусть элемент $\alpha \in L \setminus K$ такой, что $f(\alpha) = 0$ для некоторого неприводимого многочлена $f \in K[x]$. Тогда элемент α называется **алгебраическим над K степени deg(f)**. Многочлен f определяется с точностью до постоянного множителя и называется **минимальным многочленом для α над K**. Через $K(\alpha)$ обозначается наименьшее поле, содержащее K и α. Поле $K(\alpha)$ называется **алгебраическим расширением** поля K.

Пример 1.13. Рассмотрим множество вещественных чисел вида $a + b\sqrt{2}$, где a и b – рациональные числа. Покажем, что это множество с обычными операциями сложения и умножения вещественных чисел есть поле. Проверим, что множество замкнуто относительно сложения и умножения. Итак,

$$(a + b\sqrt{2}) + (c + d\sqrt{2}) = (a + c) + (b + d)\sqrt{2},$$
$$(a + b\sqrt{2})(c + d\sqrt{2}) = (ac + 2bd) + (bc + ad)\sqrt{2},$$

Сумма и произведение имеют нужную форму. Обратный элемент для $a + b\sqrt{2}$ относительно сложения есть $(-a) + (-b)\sqrt{2}$. Чтобы найти обратный относительно умножения для $a + b\sqrt{2} \neq 0$, мы должны найти рациональные числа x и y, такие, что $(a + b\sqrt{2})(x + y\sqrt{2}) = 1$. Раскроем скобки и получаем $(ax + 2by) + (bx + ay)\sqrt{2} = 1$. Должно быть $bx + ay = 0$, потому что иначе $\sqrt{2}$ был бы рациональным числом. Поэтому имеем два уравнения $bx + ay = 0$, $ax + 2by = 1$. Решая их, находим

$$(a + b\sqrt{2})^{-1} = x + y\sqrt{2} = \frac{a}{a^2 - 2b^2} + \frac{-b}{a^2 - 2b^2}\sqrt{2}.$$

Знаменатель $a^2 - 2b^2$ всегда не равен 0. Почему? Сначала рассмотрим случай $b = 0$. Тогда из $a + b\sqrt{2} \neq 0$ следовало бы, что $a \neq 0$ и поэтому $a^2 - 2b^2 \neq 0$. Пусть теперь $b \neq 0$ и если бы было $a^2 - 2b^2 = 0$, то следовало бы $\sqrt{2} = \sqrt{\frac{a^2}{b^2}} = |a / b|$, т.е. $\sqrt{2}$ был бы рациональном числом.

Данное поле обозначается $\mathbb{Q}(\sqrt{2})$, что указывает, что оно получено из поля рациональных чисел $\mathbb{Q}$ присоединением элемента $\sqrt{2}$.

Пример 1.14. Рассмотрим множество выражений вида $a + b\,i$, где $i = \sqrt{-1}$ и числа a и b – рациональные. Определим операции сложения и умножения:

$$(a + b\,i) + (c + d\,i) = (a + c) + (b + d)\,i,$$
$$(a + b\,i) \cdot (c + d\,i) = (b\,c - b\,d) + (c\,b + a\,d)\,i.$$

Получаем поле. Обратный элемент относительно сложения для $a + b\,i$ есть $(-a) + (-b)\,i$. Если $a + b\,i \neq 0$, то обратный элемент есть

$$(a+bi)^{-1} = \frac{a^2}{a^2+b^2} + \frac{(-b)}{a^2+b^2} i.$$

Это поле обозначается $\mathbb{Q}(i)$ и называется гауссовым полем рациональных чисел. Подобным образом определяется поле $\mathbb{C}$ комплексных чисел, где a и b – вещественные числа.

Пример 1.15. Рассмотрим множество выражений вида

$$u = a + b\sqrt{2} + c\sqrt{3} + d\sqrt{2}\sqrt{3},$$

где a, b, c и d – рациональные числа с обычными операциями сложения и умножения. Это поле обозначается как $\mathbb{Q}(\sqrt{2}, \sqrt{3})$, что показывает на то, что поле конструируется из поля $\mathbb{Q}$, присоединением двух чисел $\sqrt{2}$ *и* $\sqrt{3}$. Группируя слагаемые, имеем

$$a + b\sqrt{2} + c\sqrt{3} + d\sqrt{2}\sqrt{3} = (a + b\sqrt{2}) + (c + d\sqrt{2})\sqrt{3},$$

т.е. это поле есть $(\mathbb{Q}(\sqrt{2}))(\sqrt{3})$. Таким образом, сначала к $\mathbb{Q}$ присоединяем $\sqrt{2}$, чтобы получить $\mathbb{Q}(\sqrt{2})$, а затем присоединяем $\sqrt{3}$ к полю $\mathbb{Q}(\sqrt{2})$.

§ 2. Сложность алгоритмов и вычислительных задач

Пусть дана задача и некоторый алгоритм для ее решения. Как сравнить этот алгоритм с другими алгоритмами, решающими ту же задачу? Как оценить его качество? Вопросы такого рода интересуют и математиков, и программистов.

Введем в первую очередь некоторые понятия, связанные с асимптотической оценкой функций. Мы будем рассматривать функции, заданные на натуральных числах и с положительными действительными значениями.

2.1. Если $f(n)$ и $g(n)$ – некоторые функции, то запись $\boldsymbol{f(n) = O(g(n))}$ означает, что найдётся такая константа $c > 0$ и такое n_0, что $f(n) \le cg(n)$ для всех $n \ge n_0$. Говорят, что функция g **мажорирует** функцию f, если $f(n) = O(g(n))$. Символ $O(g(n))$ читается как «***O*** **большое от** $\boldsymbol{g(n)}$»; при этом говорят, что $f(n)$ имеет порядок O большое от (n).

Например, пусть $f(n) = n^2 + 3n$ и $g(n) = n^2$. Тогда $f(n) = O(g(n))$, так как достаточно взять $n_0 = 3$ и $c = 2$.

Отыскивая асимптотически оценку для суммы, мы можем отбрасывать члены меньшего порядка, которые при больших n становятся малыми по сравнению с основным слагаемым. Заметим также, что коэффициент при старшем члене роли не играет (он может повлиять только на выбор констант n_0 и c). Поэтому для любого положительного a мы имеем $an^2 + 3n = O(n^2)$. Конечно, в этом случае также выполнено $n^2 = O(an^2 + 3n)$.

Отношение «O большое» наиболее полезно для сравнения асимптотического роста функции. Следующая теорема устанавливает свойства этого отношения для основных функций. Доказательство см. в [17].

Теорема 2.2.

1. Если r и s – действительные числа, $r \le s$ и $n > 1$, тогда $n^r \le n^s$. Следовательно, $n^r = O(n^s)$.
2. Если $f(n) = O(g(n))$, то $cf(n) = O(g(n))$.
3. Если $f(n) = O(g(n))$ и $h(n) = O(g(n))$, то $(f+h)(n) = O(g(n))$.
4. Если $f(n) = O(g(n))$ и $h(n) = O(e(n))$, то $(f \times h)(n) = O(g \times e)(n))$.
5. Если $p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$, то $p(n) = O(n^k)$.
6. Для целых чисел a и b, больших единицы, $\log_a(n) = O(\log_b(n))$.
7. Пусть n – неотрицательное целое число. Тогда $n < 2^n$ и, следовательно, $n = O(2^n)$.
8. Если $f(n) = O(g(n))$ и $g(n) = O(h(n))$, то $f(n) = O(h(n))$.
9. Для целых чисел a, больших единицы, $\log_a(n) = O(n)$.

10. Пусть n – неотрицательное целое число, тогда $n! < n^n$ и, следовательно, $n! = O(n^n)$.

11. Пусть $a > 1$ и n – неотрицательное целое число, тогда $\log_a(n!) \le n \log_a(n)$ и, следовательно, $\log_a(n!) = O(n \log_a(n))$.

Пример 2.3. Определим число арифметических операций, необходимых для сложения двух матриц.

Пусть матрицы имеют размеры $m \times k$. Тогда алгоритм сложения матриц $A + B = C$ можно описать на Паскале следующим образом:

```
for i := 1 to m do
    for j := 1 to k do
        C[i, j] := A[i, j] + B[i, j];
```

Как видим, сложение выполняется для каждого i и каждого j. Поскольку i принимает m значений, а j принимает k значений, то выполняется mk операций сложения. Пусть $n = \max(m, k)$. Тогда число выполняемых арифметических операций имеет порядок $O(n^2)$.

Пример 2.4. Определим число арифметических операций, необходимых для умножения двух матриц.

Пусть матрицы A и B имеют размеры $m \times p$ и $p \times k$, соответственно. Тогда алгоритм умножения матриц $A \times B = C$ можно описать следующим образом:

```
for i:=1 to m do
    for j := 1 to k do
      begin
      C[i, j] := 0;
      for s := 1 to p do C[i, j] := C[i, j] + A[i, s]*B[s, j];
      end;
```

Поскольку s принимает значения от 1 до p, то выполняется p операций сложения и p операций умножения. Величина s изменяется от 1 до p для каждого i и каждого j, поэтому s пробегает значения от 1 до p mk раз. Таким образом, выполняется mkp операций сложения и столько же операций умножения. Следовательно, всего выполняется $2mkp$ операций. Пусть $n = \max(m,k,p)$. Тогда число выполняемых арифметических операций имеет порядок $O(n^3)$.

Пример 2.5. Сравним количество операций, которое требуется для непосредственного вычисления значения многочлена традиционным способом и по схеме Горнера.

Пусть требуется вычислить $p(c)$, где $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$. Если $p(c)$ вычисляется непосредственно, то для подсчета c^k требуется выполнить k–1 операций умножения. Еще одна операция нужна для умножения на a_k, так что вычисление $a_k x^k$ требует k операций умножения. Таким образом, нужно выполнить $1 + 2 + \ldots + n = n(n\text{-}1)/2$ умножений. Для того чтобы найти сумму n+1 слагаемых, требуется выполнить n сложений, так что общее число арифметических операций равно $n(n\text{-}1)/2 + n$ и имеет порядок $O(n^2)$.

При вычислении многочлена $a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ по схеме Горнера, мы переписываем полином в виде $x(x(x(a_4x + a_3) + a_2) + a_1) + a_0$ и замечаем, что выражение включает четыре операции умножения и четыре операции сложения. Очевидно, в общем случае $p(x) = x(x(\ldots (x(a_nx+a_{n-1}) + a_{n-2}) + \ldots a_2) + a_1) + a_0$ включает n операций сложения и n операций умножения. Таким образом, общее число арифметических операций равно $2n$ и имеет порядок $O(n)$.

В рассмотренных примерах с помощью отношения «O большое» мы смогли оценить вычислительную сложность используемых алгоритмов. Дадим точные определения в общем случае.

Класс однородных вычислительных задач мы будем называть **проблемой** (также используется понятие **массовая задача** или **абстрактная задача**). Индивидуальные случаи проблемы Q мы будем называть **частными случаями** проблемы Q.

Мы можем, например, говорить о проблеме умножения матриц. Частные случаи этой проблемы суть пары матриц, которые нужно перемножить.

2.6. Более формально, мы принимаем следующую абстрактную модель вычислительной задачи. Абстрактная задача есть произвольное бинарное отношение Q между элементами двух множеств – множества **условий** (или входных данных) I и множества **решений** S.

Например, в задаче умножения матриц входными данными являются две конкретные матрицы – сомножители, а матрица – произведение является решением задачи. В задачах с натуральными числами, например, разложения числа на простые множители простоты числа, входными данными и решением являются натуральные числа.

Нам бы хотелось связать с каждым частным случаем проблемы некоторое число, называемое его **размером**, которое выражало бы меру количества входных данных. Например, размером задачи умножения матриц может быть наибольший размер матриц-сомножителей. Размером задачи о разложении числа на простые множители обычно считают количество десятичных цифр в исходном числе.

Решение задачи на компьютере можно осуществлять с помощью различных алгоритмов. Прежде чем подавать на вход алгоритма исходные данные (то есть элемент множества I), надо договориться о том, как они представляются «в понятном для компьютера виде»; мы будем считать, что исходные данные закодированы последовательностью битов. Формально говоря, **представлением** элементов некоторого множества M называется отображение e из M во множество битовых строк. Например, натуральные числа 0, 1, 2, 3, … обычно представляют битовыми строками 0, 1, 10, 11, 100, … (при этом, например, $e(17) = 10001$).

Фиксировав представление данных, мы превращаем абстрактную задачу в **строковую**, для которой входным данным является битовая строка, представляющая исходное данное абстрактной задачи. Естественно считать размером строковой задачи длину строки.

2.7. (Асимптотическая временная сложность алгоритмов) Будем говорить, что алгоритм **решает** строковую задачу за **время** $O(T(n))$, если на входном данном битовой строки длины n алгоритм работает время $O(T(n))$.

В качестве временной оценки работы алгоритма вместо общего числа шагов мы можем подсчитывать число шагов некоторого вида, таких как арифметические операции при алгебраических вычислениях, число сравнений при сортировке или число обращений к памяти.

Быстрыми являются линейные алгоритмы, которые обладают сложностью порядка $O(n)$, где n – размерность входных данных. К линейным алгоритмам относится школьный алгоритм нахождения суммы десятичных чисел, состоящих из n_1 и n_2 цифр. Сложность этого алгоритма – $O(n_1 + n_2)$. Есть алгоритмы, которые быстрее линейных, например, алгоритм двоичного поиска в линейном упорядоченном массиве имеет сложность $O(\log n)$, n – длина массива.

Другие хорошо известные алгоритмы – деление, извлечение квадратного корня, решение систем линейных уравнений и др. – попадают в более общий класс полиномиальных алгоритмов.

Полиномиальным алгоритмом (или алгоритмом полиномиальной временной сложности) называется алгоритм, у которого временная сложность равна $O(n^k)$, где k – положительное целое число. Алгоритмы, для временной сложности которых не существует такой оценки, называются **экспоненциальными**.

2.8. Сложность задачи – это асимптотическая временная сложность наилучшего алгоритма, известного для ее решения.

Основной вопрос теории сложности: насколько успешно или с какой стоимостью может быть решена заданная проблема Q? Мы не имеем в виду никакого конкретного алгоритма решения Q. Наша цель – рассмотреть все возможные алгоритмы решения Q и по-

пытаться сформулировать утверждение о вычислительной сложности, внутренне присущей Q. В то время как всякий алгоритм для Q дает верхнюю оценку величины сложности Q, нас интересует нижняя оценка. Знание нижней оценки представляет интерес математически и, кроме того, руководит нами в поиске хороших алгоритмов, указывая, какие попытки заведомо будут безуспешны.

Задачи, для которых существуют полиномиальный алгоритм считаются **легко решаемые**. Если для решения задачи полиномиальные алгоритмы неизвестны или доказано, что их не существует, то задача относится к **трудно решаемым задачам**. Понятие полиномиально разрешимой задачи принято считать уточнением идеи «практически разрешимой» задачи.

Мы будем приводить оценки сложности только для некоторых алгоритмов.

Задачи и упражнения

1. Докажите, что все обратимые элементы кольца R с единицей составляют группу по умножению.

2. Докажите, что обратимый элемент в кольце не может быть делителем нуля.

Господь сотворил целые числа;
остальное – дело рук человека.
Леопольд Кронекер

Глава 4. Евклидовы и факториальные кольца

В этой главе мы будем рассматривать, как компьютерная алгебра помогает изучать свойства колец $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}[i]$, $\mathbb{Q}[x]$ и $\mathbb{Z}[x]$, связанные с делимостью.

§ 1. Евклидовы кольца

Начнем с деления в $\mathbb{Z}$, так как понятия, связанные с делимостью в различных кольцах, являются обобщением и развитием понятий для целых чисел.

В системе Mathematica функция Divisible[m, n] выдает True, если n делитель m, иначе результат False. Функция работает с целыми числами любого размера, рациональными числами, точными вещественными и численными величинами в символьной форме.

```
Divisible[10000!, 772223]
```

True

```
Divisible[10000!, 10007]
```

False

```
Divisible[3/4, 1/4]
```

True

```
Divisible[Sqrt[8], Sqrt[2]]
```

True

```
FullSimplify[Divisible[2x, x], x ∈ Integers]
```

True

Теорема 1.1. Для целых чисел a и $b\neq 0$ существуют единственные целые q и r такие, что $a = q\,b + r$ и $0 \le r \le |b| - 1$.

Доказательство. Ясно, что одно представление числа a равенством $a = q\,b + r$ мы получим, если возьмем $q\,b$ равным наибольшему кратному числа b, не превосходящему a.

Тогда, очевидно, $0 \le r \le |b| - 1$. Докажем единственность такого представления. Пусть $a = q\,b + r$ и $a = q_1\,b + r_1$ – два таких представления. Значит,

$$0 = a - a = q\,b + r - q_1\,b + r_1 = b\,(q - q_1) + r - r_1.$$

Здесь 0 делится на b; $b\,(q - q_1)$ делится на b, следовательно, $r - r_1$ обязано делится на b. Так как $0 \le r \le |b| - 1$ и $0 \le r_1 \le |b| - 1$, то $r - r_1 < b$ и $r - r_1$ делится на b, значит $r - r_1$ равно нулю, а значит, и $q - q_1$ равно нулю, т.е. два таких представления совпадают. ♦

1.2. Число q называется **неполным частным**, а число r – **остатком** от деления a на b.

Очевидно, остаток от деления a на b равен нулю тогда и только тогда, когда b – делитель a.

Иногда мы будем использовать обозначения $a \bmod b$ для r. Так как для вещественных чисел широко используется функция «целая часть» (другое название «пол») $\lfloor x \rfloor$ – наибольшее целое число, не превосходящее x, то неполное частное q можно представить как $\lfloor a/b \rfloor$.

Пример 1.3. Имеем

$8 = q\,3 + r = 2\,3 + 2,$

$8 = q\,(-3) + r = (-2)\,(-3) + 2,$

$-8 = q\ 3 + r = (-3)\ 3 + 1,$
$-8 = q\ (-3) + r = 3\ (-3) + 1.$

Пример 1.4. В системе Mathematica частное и остаток вычисляются с помощью функций `Quotient[a, b]` и `Mod[a, b]`, и остаток выбирается так, чтобы выполнялись свойства $|r| \le |b| - 1$ и $r\ b \ge 0$. Поэтому имеем

$8 = q\ 3 + r = 2\ 3 + 2,$
$8 = q\ (-3) + r = (-3)\ (-3) - 1,$
$-8 = q\ 3 + r = (-3)\ 3 + 1,$
$-8 = q\ (-3) + r = 2\ (-3) - 2.$

Таким образом, `Quotient[a, b]` и `Mod[a, b]` совпадают с математическим определением q и r только тогда, когда $b > 0$.

Функции `Quotient[a, b]` и `Mod[a, b]`, также как и `Divisible`, работают не только с целыми аргументами.

Понятия, связанные с делимостью, являются обобщением и развитием понятий для целых чисел. В кольце $\mathbb{Z}$ делителей единицы два: 1 и –1. Ассоциированными элементами могут быть только пары ненулевых чисел a и $-a$.

Рассмотрим кольцо $\mathbb{Z}[i]$.

1.5. Определим **норму** для гауссова числа $a + bi$ как квадрат его модуля:

$$N(a + bi) = a^2 + b^2 = (a + bi)\ (a - bi).$$

Очевидно, норма равна нулю только для нуля. В остальных случаях норма – положительное число.

Норма обладает важным свойством **мультипликативности**:

$$N(u\ v) = N(u)N(v).$$

Отсюда также сразу следует, что обратимыми элементами кольца (делителями единицами) являются только те элементы, у которых норма равна 1, то есть $\{1, -1, i, -i\}$.

Все гауссовы числа делятся на делители единицы, поэтому любое гауссово число, отличное от делителей единицы, имеет как минимум 8 делителей: 4 делителя единицы и 4 их произведения на само это число. Эти делители называются **тривиальными**.

Для числа $z = a + bi$ ассоциированными числами в $\mathbb{Z}[i]$ являются $w = \varepsilon z$, где ε – делитель единицы. Поэтому ассоциированными для $z = a + bi$ являются четыре комплексных числа

$$z = a + bi,\ iz = -b + ai,\ -z = -a - bi,\ -iz = b - ai.$$

1.6. Несколько простых свойств деления в $\mathbb{Z}[i]$:

- Все делители гауссова числа z являются также делителями его нормы $N(z) = z \cdot \bar{z}$.
- Если гауссово число u нацело делится на гауссово число v, то и норма $N(u)$ (в силу мультипликативности) делится нацело на $N(v)$.

Из мультипликативности нормы в $\mathbb{Z}[i]$ следует, что $\mathbb{Z}[i]$ является целостным кольцом.

Для кольца $\mathbb{Z}[i]$ Mathematica использует такую же функцию `Divisible`, как и для целых чисел:

```
Divisible[3 + I, 2 – I]
```

```
True
```

```
(3 + I) / (2 – I)
```

```
1 + I
```

Пусть R – кольцо. Рассмотрим кольцо многочленов $R[x]$.

Совершенно очевидно, что многочлен $f \in R[x]$ обратим в точности тогда, когда $\deg(f) = 0$ и $f = f_0$ – обратимый элемент кольца R, поскольку $fg = 1$ влечет $\deg(f) + \deg(g) = \deg(1) = 0$. Поэтому ассоциированность многочленов $f, g \in R[x]$ означает, что они отличаются лишь обратимым множителем из R. Если R – целостное кольцо, то $R[x]$ также является целостным кольцом.

В качестве R мы можем взять кольца $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ и $\mathbb{C}$.

Деление с остатком в $\mathbb{Z}$ можно обобщить для целостных колец определенного вида.

1.7. Рассмотрим целостное кольцо R, в котором каждому элементу $a \neq 0$ поставлено в соответствие неотрицательное целое число $\delta(a)$, т.е. определено отображение

$$\delta: R\backslash\{0\} = R^* \to \mathbb{N}_0$$

так, что при этом выполняются условия:

E_1) $\delta(ab) \geq \delta(a)$ для всех $a, b \neq 0$ из R;

E_2) каковы бы ни были $a, b \in R$, $b \neq 0$, найдутся $q, r \in R$ (q – « частное», r – «остаток»), для которых

$$a = qb + r; \ \ \delta(r) < \delta(b) \text{ или } r = 0. \tag{1}$$

Целостное кольцо R с этими свойствами называется **евклидовым кольцом**. А операция, которая для данных $a, b \in R$, $b \neq 0$, находит частное и остаток, называется **евклидовым делением**. Отображение δ называют **евклидовой нормой**.

Как мы уже знаем (теорема **1.1),** $\mathbb{Z}$ – евклидово кольцо с $\delta(a) = |a|$ для $a \in \mathbb{Z}$.

Рассмотрим евклидово деление в $R[x]$.

Теорема 1.8. Пусть R – целостное кольцо и v – многочлен в $R[x]$ со старшим коэффициентом, обратимым в R. Тогда каждому многочлену $u \in R[x]$ сопоставляется одна и только одна пара многочленов $q, r \in R[x]$, для которых

$$u = qv + r, \ \ \deg(r) < \deg(v).$$

Доказательство [26, стр. 187–188]. Пусть

$$u = a_n x^n + a_{n-1}x^n + \ldots + a_0,$$

$$v = b_m x^m + b_{m-1}x^m + \ldots + b_0,$$

где $a_0 b_0 \neq 0$ и $b_0|1$. Применим индукцию по n. Если $n = 0$ и $m = \text{dev}(v) > \text{dev}(u) = 0$, то положим $q = 0$, $r = u$, а если $n = m = 0$, то $r = 0$ и $q = a_0 b_0^{-1}$. Допустим, что теорема доказана для всех многочленов степени $< n$ ($n > 0$). Без ограничения общности считаем $m \leq n$, поскольку в противном случае возьмём $q = 0$ и $r = u$. Раз это так, то

$$u = a_0 b_0^{-1} x^{n-m} \cdot v + h$$

где $\text{dev}(h) < n$. По индукции мы можем найти v и r, для которых $h = vv + r$, причём $\text{dev}(r) < m$. Положив

$$q = a_0 b_0^{-1} x^{n-m} + v,$$

мы придём к паре многочленов с нужными свойствами.

Обращаясь к свойству единственности частного q и остатка r, предположим, что

$$qv + r = u = q'v + r'.$$

Тогда $(q' - q)v = r - r'$. Имеем $\text{dev}(r - r') = \text{dev}(q' - q) + \text{dev}(v)$, что в наших условиях возможно только при $r' = r$ и $q' = q$.

Наконец, приведённые рассуждения показывают, что коэффициенты частного q и остатка r принадлежат тому же целостному кольцу R, т.е. $u, v \in R[x] \Rightarrow q, r \in R[x]$. ♦

Очевидно, что u делится на v, если остаток r равен нулю.

Следствие 1.9. Если R является полем, то любой ненулевой элемент R обратим и, следовательно, евклидово деление многочленов над полем всегда возможно. В этом случае, для любого многочлена $u \in R[x]$ значение $\delta(u)$ определяется как $\deg(u)$.

Алгоритм 1.10. Евклидово деление многочленов известно из курса элементарной алгебры под названием **деление столбиком**.

Рассмотрим пример деления в $Z[x]$: $u(x) = x^3 - 12x^2 - 42$, $v(x) = x - 3$. Частное и остаток от деления u на v могут быть найдены в ходе выполнения следующих шагов:

1. Делим первый элемент делимого на старший элемент делителя, помещаем результат под чертой ($x^3/x = x^2$).

$$\begin{array}{l} x^3 - 12\,x^2 - 42 \mid \underline{x - 3} \\ \qquad\qquad\quad \mid x^2 \end{array}$$

2. Умножаем делитель на полученный выше результат деления (на первый элемент частного). Записываем результат под первыми двумя элементами делимого ($x^2 \cdot (x - 3) = 3x^3 - x^2$).

$$\begin{array}{l} x^3 - 12\,x^2 + 0\,x - 42 \mid \underline{x - 3} \\ x^3 - 3\,x^2 \qquad\qquad\quad \mid x^2 \end{array}$$

3. Вычитаем полученный после умножения многочлен из делимого, записываем результат под чертой ($x^3 - 12x^2 + 0x - 42 - (x^3 - 3x^2) = -9x^2 + 0x - 42$).

$$\begin{array}{l} x^3 - 12\,x^2 + 0\,x - 42 \mid \underline{x - 3} \\ \underline{x^3 - 3\,x^2 \qquad\qquad\quad} \mid x^2 \\ \quad - 9\,x^2 + 0\,x - 42 \end{array}$$

4. Повторяем предыдущие три шага, используя в качестве делимого многочлен, записанный под чертой.

$$\begin{array}{l} x^3 - 12\,x^2 + 0\,x - 42 \mid \underline{x - 3} \\ \underline{x^3 - 3\,x^2 \qquad\qquad\quad} \mid x^2 - 9x \\ \quad - 9\,x^2 + 0\,x - 42 \\ \quad \underline{- 9\,x^2 + 27x \qquad} \\ \qquad\quad - 27x - 42 \end{array}$$

5. Повторяем шаг 4.

$$\begin{array}{l} x^3 - 12\,x^2 + 0\,x - 42 \mid \underline{x - 3 \qquad\quad} \\ \underline{x^3 - 3\,x^2 \qquad\qquad\quad} \mid x^2 - 9x - 27 \\ \quad - 9\,x^2 + 0\,x - 42 \\ \quad \underline{- 9\,x^2 + 27x \qquad} \\ \qquad\quad - 27x - 42 \\ \qquad\quad \underline{- 27x + 81} \\ \qquad\qquad\quad -123 \end{array}$$

6. Конец алгоритма.

Таким образом, $q(x) = x^2 - 9x - 27$ – частное деления, а $r(x) = -123$ – остаток.

Опишем алгоритм деления многочленов над полем при плотном представлении многочленов. Будем представлять любой многочлен в виде списка коэффициентов – элементов поля, не исключая и нулевые.

Алгоритм 1.11. Деление многочленов над полем – плотное представление. Вход: два списка коэффициентов $\{u_m, \ldots, u_1, u_0\}$ и $\{v_n, \ldots, v_1, v_0\}$ многочленов

$$u(x) = u_m x^m + \cdots + u_1 x + u_0, \qquad v(x) = v_n x^n + \cdots + v_1 x + v_0,$$

где $v_n \neq 0$ и $m \geq n \geq 0$. Выход: два списка коэффициентов $\{q_{m-n}, \ldots, q_1, q_0\}$ и $\{r_{n-1}, \ldots, r_1, r_0\}$ многочленов

$$q(x) = q_{m-n} x^{m-n} + \cdots + q_1 x + q_0, \qquad r(x) = r_{n-1} x^{n-1} + \cdots + r_1 x + r_0,$$

удовлетворяющих условию (1).

1. (Итерация по k.) Выполнять шаг 2 для $k = m - n, m - n - 1, \ldots, 0$; затем прекратить выполнение алгоритма с $\{r_{n-1}, \ldots, r_1, r_0\} = \{u_{n-1}, \ldots, u_1, u_0\}$.
2. (Цикл деления.) Присвоить сначала $q_k = u_{n+k}/v_n$, а затем $u_j = u_j - q_k v_{j-k}$ для $j = n + k - 1, n + k - 2, \ldots, k$. (Действие последней операции состоит в замещении $u(x)$ на $u(x) - q_k x^k v(x)$, многочлен $< n + k$.)

Примеры 1.12. Функция PolynomialQuotient[p, q, x] в Mathematica выдает частное от деления p на q, которые рассматриваются как полиномы из кольца $\mathbb{Q}[x]$:

```
PolynimialQuotient[x^2, x + a, x]
```

```
- a + x
```

```
Expand[% (x + a)]
```

$$-a^2 + x^2$$

Функция PolynomialRemainder выдает остаток от деления:

```
PolynomialRemainder[x^2, x + a, x]
```

$$a^2$$

В общем случае неполное частное и остаток являются рациональными выражениями от входных коэффициентов:

```
PolynomialQuotient[x^2 + x + 1, 2x + 1, x]
```

$$\frac{1}{4} + \frac{x}{2}$$

```
PolynomialRemainder[x^2 + x + 1, 2x + 1, x]
```

$$\frac{3}{4}$$

```
PolynomialQuotient[x^2 + b x + 1, a x + 1, x]
```

$$-\frac{1}{a^2} + \frac{b}{a} + \frac{x}{a}$$

```
PolynomialRemainder[x^2 + b x + 1, a x + 1, x]
```

$$1 + \frac{1}{a^2} - \frac{b}{a}$$

Для получения одновременно частного и остатка полезно использовать функцию PolynomialQuotientRemainder.

```
{f, g} = {x^2 + 4 x +1, x + 2};
PolynomialQuotientRemainder[f, g, x]
```

```
{2 + x, -3}
```

Рассмотрим евклидово деление в $\mathbb{Z}[i]$.

Теорема 1.13. Если положить $\delta(z) = N(z) = |z|^2$, то $\mathbb{Z}[i]$ станет евклидовым кольцом.

Доказательство. Свойство E_1 для евклидовых колец (определение **1.7)** следует из мультипликативности нормы $N(z)$.

Докажем свойство E_2. Пусть u и $v \neq 0$ – гауссовы целые числа. Среди точек на плоскости с целочисленными координатами выберем целое гауссово q как возможно ближе к комплексному числу u/v, так чтобы

$$|q - u/v| \le \frac{\sqrt{2}}{2}.$$

Имеем $N(u - qv) \le N(v)/2$. Из этого неравенства получаем, что $r = u - qv$ удовлетворяет условию $N(r) < N(v)$. Таким образом, существует, по крайней мере, одна пара $q, r \in \mathbb{Z}[i]$, удовлетворяющая условиям:

$$u = vq + r \text{ и } N(r) < N(v). \quad \blacklozenge$$

Замечание. Пара (q, r) не единственна в общем случае. Каждое гауссово целое q, достаточно близкое к u/v дает решение. Если вещественная и мнимая части числа u/v имеют вид $n + 1/2$, то существует четыре возможных значения для q. Если разделить $u = 7 + 2i$ на v = $3 - i$, то имеем три различных подходящих пары (q, r):

$$7 + 2i = (3 - i)(2 + i) + i = (3 - i)(1 + i) + 3 = (3 - i)(2 + 2i) - 1 - 2i.$$

Примеры 1.14. Для кольца $\mathbb{Z}[i]$ Mathematica использует такие же функции Quotient и Mod, как и для целых чисел:

```
q = Quotient[7 + 2 I ,3 – I]
```

```
2 + I
```

```
r = Mod[7 + 2 I, 3 – I]
```

```
I
```

```
q (3 – I) + r
```

```
7 + 2 I
```

Хотя возможно несколько пар (частное, остаток), Mathematica находит только одну пару.

§ 2. Наибольший общий делитель

2.1. Пусть R – целостное кольцо. **Наибольшим общим делителем** двух элементов $a, b \in R$ мы будем называть элемент $d \in R$, обозначаемый $\gcd(a, b)$ и обладающий двумя свойствами:

i) $d|a, d|b$;

ii) $c|a, c|b \Rightarrow c|d$.

Ясно, что вместе с d свойствами i), ii) обладает любой ассоциированный с ним элемент. Обратно: если c и d – два наибольших делителя элементов a и b, то будем иметь $c|d$, $d|c$, так что c и d ассоциированы. Обозначение $\gcd(a, b)$ относится к любому из них, т.е. в этой записи ассоциированные элементы не различаются. С учетом такого соглашения к определяющим свойствам i), ii) наибольшего общего делителя добавятся следующие:

iii) $\gcd(a, b) = a \Leftrightarrow a|b$;

iv) $\gcd(a, 0) = a$;

v) $\gcd(ta, tb) = t \gcd(a, b)$;

vi) $\gcd(\gcd(a, b), c) = \gcd(a, \gcd(b, c))$.

Проверка их не вызывает никаких трудностей и оставляется читателю. Свойство vi) позволяет также распространить понятие gcd на произвольное конечное число элементов.

2.2. По аналогии с $\gcd(a, b)$ вводится двойственное понятие **наименьшего общего кратного** $m = \operatorname{lcm}(a, b)$ элементов $a, b \in R$, также определённого с точностью до ассоциированности двумя свойствами:

i') $a|m, b|m$.

ii') $a|c, b|c \Rightarrow m|c$.

В частности, полагая $c = ab$, получаем $m|ab$.

Теорема 2.3. Пусть для элементов a, b целостного кольца R существуют $\gcd(a, b)$ и $\operatorname{lcm}(a, b)$.

Тогда:

а) $\gcd(a, b) = 0 \Leftrightarrow a = 0$ или $b = 0$.

б) $a, b \neq 0$, $m = \operatorname{lcm}(a, b)$, $ab = dm \Rightarrow d = \gcd(a, b)$.

Доказательство предоставляется читателю.

2.4. Элементы a, b целостного кольца, в котором существует gcd, называются **взаимно простыми**, если $\gcd(a, b) = 1$.

Из свойств i), ii), i'), ii') или из теоремы 2.3 нельзя извлечь ни способа вычисления, ни доказательства существования $\gcd(a, b)$ и $\operatorname{lcm}(a, b)$.

Пример 2.5. Наибольший общий делитель существует не всегда. Рассмотрим кольцо $\mathbb{Z}[\sqrt{-5}]$ всех целых чисел с добавленным элементом $\sqrt{-5}$. Число 2 делит 6 и $2 + 2\sqrt{-5}$. Точно также $1 + \sqrt{-5}$ является делителем и $2 + 2\sqrt{-5}$ и 6, так как $6 = (1 + \sqrt{-5})(1 - \sqrt{-5})$. Но здесь нет кратного двух элементов 2 и $1 + \sqrt{-5}$, которое бы было делителем и 6 и $2 + 2\sqrt{-5}$.

Но для евклидовых колец наибольший общий делитель существует и вычисляется просто.

Теорема 2.6. В евклидовом кольце R любые два элемента a, b имеют наибольший общий делитель и наименьшее общее кратное. Более того, существует алгоритм, с помощью которого можно найти $\gcd(a, b)$ и такие $u, v \in R$, что выполнено **соотношение Безу**:

$$\gcd(a, b) = au + bv.$$

Доказательство. Мы явно укажем алгоритм, который выдает нужный результат. Пусть даны ненулевые элементы a, b евклидова кольца R. Применяя достаточное количество раз деление с остатком, мы получим систему равенств типа (определение 1.7**,** (1)) с последним нулевым остатком:

$$\begin{aligned} a &= q_1 b + r_1, & \delta(r_1) &< \delta(b), \\ b &= q_2 r_1 + r_2, & \delta(r_2) &< \delta(r_1), \\ r_1 &= q_3 r_2 + r_3, & \delta(r_3) &< \delta(r_2), \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & & \\ r_{k-2} &= q_k r_{k-1} + r_k, & \delta(r_k) &< \delta(r_{k-1}), \\ r_{k-1} &= q_{k+1} r_k, & r_{k+1} &= 0. \end{aligned} \qquad (2)$$

Очевидно, строго убывающая цепочка неотрицательных целых чисел должна закончиться, а это может произойти только из-за обращения в ноль одного из остатков.

Последний отличный от нуля остаток r_k является как раз наибольшим общим делителем элементов a и b. В самом деле, по условию $r_k | r_{k-1}$. Двигаясь в системе (2) снизу вверх и используя свойство 4) отношения делимости, сформулированное в главе 3, пункт 1.9, получим цепочку $r_k | r_{k-1}$, $r_{k-1} | r_{k-2}$, …, $r_k | r_2$, $r_k | r_1$ и, наконец, $r_k | b$, $r_k | a$. Стало быть, r_k – общий делитель элементов a и b. Обратно: пусть d – любой другой делитель тех же элементов; тогда $d | r_1$, и, двигаясь теперь в системе (2) сверху вниз, мы получим цепочку отношений делимости $d | r_2$, $d | r_3$, …, $d | r_k$. Последнее из них окончательно убеждает нас в том, что $\gcd(a, b)$ существует, причем имеет место равенство

$$r_k = \gcd(a, b). \qquad (3)$$

Обратим, далее, внимание на то обстоятельство, что каждый остаток r_i в системе (2) выражается в виде линейной комбинации с коэффициентами в R двух предыдущих остатков r_{i-1} и r_{i-2}. При этом r_1 выражается через a и b: $r_1 = a - q_1 b$, а r_2, выражаясь через b и r_1, тем самым является опять линейной комбинацией a и b. Последовательная подстановка в r_i выражений r_{i-1} и r_{i-2} через a и b даст нам при $i = k$ выражение

$$r_k = au + bv \qquad (4)$$

с какими-то элементами $u, v \in R$.

Сопоставляя (3) и (4) и принимая во внимание предложение 2.3 б) мы заканчиваем доказательство. ♦

Описанный алгоритм для нахождения наибольшего общего делителя $r_k = \gcd(a, b)$ носит имя Евклида, а если на этом не останавливаться и найти еще u и v из (4), то мы получаем расширенный алгоритм Евклида.

Замечание. Пара элементов x, y в соотношении Безу

$$ax + by = \gcd(a, b).$$

не единственная. На самом деле таких пар бесконечно много. Например, если мы возьмем u, v, для которых выполнено (4), и произвольное целое k, то, как несложно проверить,

$$a(u + kb) + b(v - ka) = \gcd(a, b).$$

Утверждение теоремы 2.6 легко распространяются на случай произвольного конечного числа элементов евклидова кольца.

Следствие 2.7. Пусть a, b, c – элементы евклидова кольца R.
i) Если $\gcd(a, b) = 1$ и $\gcd(a, c) = 1$, то $\gcd(a, bc) = 1$.
ii) Если $a|bc$ и $\gcd(a, b) = 1$, то $a|c$.
iii) Если $b|a$, $c|a$ и $\gcd(b, c) = 1$, то $bc|a$.

Следствие 2.8. Если R – одно из колец $\mathbb{Z}$, $\mathbb{Z}[i]$ или $P[x]$, где P – поле, то для любых двух элементов $a, b \in R$ (одновременно ненулевых) существует $\gcd(a, b)$, для которого выполняется соотношение Безу, и наибольший общий делитель можно найти с помощью алгоритма Евклида.

Рассмотрим свойства gcd в различных кольцах. Начнем с $\mathbb{Z}$.

Для вычисления $\gcd(m, n)$ заданных целых чисел $0 \le m < n$ можно в алгоритме Евклида использовать рекурсию

$\gcd(0, n) = n$,
$\gcd(m, n) = \gcd(n \bmod m, m)$ при $m>0$.

Указанная рекурсия законна потому, что любой общий делитель чисел m и n должен быть также общим делителем чисел m и $n \bmod m$.

Количество операций mod в алгоритме оценивается сверху величиной $5\log_{10} m$ [4, стр. 171-173].

Текст программы **алгоритма Евклида** на языке Wolfram $\gcd(m, n)$:

```
gcd[m_, n_] := If[m == 0, n, gcd[Mod[n, m], m]]
```

Алгоритм 2.9. Расширенный рекурсивный алгоритм Евклида (A Extended Euclidean Algorithm) вычисляет для целых m, n тройку чисел x, y, d, удовлетворяющих соотношению Безу

$$x\,m + y\,n = d = \gcd(m, n).$$

Вход алгоритма: m, n. Выход: $\{x, y, d\} = \mathrm{EEA}(m, n)$:
If $m = 0$ then return($\{0, 1, n\}$)
else begin
  $r = n \bmod m$;
  $\{r_1, m_1, d\} = \mathrm{EEA}(r, m)$; (* рекурсивный вызов *)
  return($\{m_1 - \lfloor n/m \rfloor r_1, r_1, d\}$)
end

Текст программы в системе Mathematica:

```
eea[m_, n_] := Module[{d, r1, m1},
   If[m == 0, {0, 1, n}, r = Mod[n, m];
     {r1, m1, d} = eea[r, m];
     {m1 – Quotient[n, m] r1, r1, d}]]
```

Функция GCD[n_1, n_2, …, n_k] в Mathematica вычисляет наибольший общий делитель n_1, n_2, …, n_k. Аргументы могут быть и рациональными числами.

```
GCD[100!, 500!, 1000!, 10000!] == 100!
```

True

Функция ExtendedGCD[n_1, n_2, …, n_k] вычисляет расширенный наибольший общий делитель n_1, n_2, …, n_k.

```
{g, {a, b, c}} = ExtendedGCD[6, 15, 30]
```

{3, {–2, 1, 0}}

```
6a + 15b + 30c == g
```

True

Нахождение gcd в кольце $\mathbb{Q}[x]$

Если f и g – многочлены относительно одной общей переменной и с коэффициентами из $\mathbb{Q}$, то PolynomialGCD[f, g] вычисляет gcd(f, g).

```
PolynomialGCD[3(1+x)^2(2+x)(4+x), 2(1+x)(2+x)(3+x)]
```

$(1 + x)(2 + x)$

Если коэффициенты многочленов целые, то результатом является нормированный многочлен, то есть старший коэффициент gcd равен 1.

Все символьные параметры в многочленах трактуются функцией PolynomialGCD как переменные, поэтому при наличии нескольких переменных эта функция вычисляет наибольший общий делитель для многочленов от нескольких переменных (см. § 4).

Функция находит gcd и в случае более двух многочленов:

```
PolynomialGCD[x^2-1, x^3-1, x^4-1, x^5-1, x^6-1, x^7-1]
```

– 1 + x

Вычислительные трудности алгоритма Евклида

При нахождении gcd многочленов над полем $\mathbb{Q}$ с помощью алгоритма Евклида часто происходит значительное увеличение размеров рациональных чисел в промежуточных вычислениях, если даже результат содержит небольшие числа.

Пример 2.10.

```
u = x^8 + 5 x^7 + 7 x^6 – 3 x^5 + 4 x^4 + 17 x^3 – 2 x^2 – 6 x + 3;
v = x^8 + 6 x^7 + 3 x^6 + x^5 + 10 x^4 + 8 x^3 + 2 x^2 + 9 x + 8;
```

Получаем последовательность остатков:

```
r1 = PolynomialRemainder[u, v, x]
```

$$-5 - 15\,x - 4\,x^2 + 9\,x^3 - 6\,x^4 - 4\,x^5 + 4\,x^6 - x^7$$

```
r2 = PolynomialRemainder[v, r1, x]
```

$$-42 - 146\,x - 53\,x^2 + 94\,x^3 - 41\,x^4 - 45\,x^5 + 39\,x^6$$

```
r3 = PolynomialRemainder[r1, r2, x]
```

$$-\frac{327}{169} - \frac{2749\,x}{507} - \frac{655\,x^2}{169} + \frac{132\,x^3}{169} - \frac{101\,x^4}{169} - \frac{896\,x^5}{507}$$

```
r4 = PolynomialRemainder[r2, r3, x]
```

$$\frac{17\,428\,125}{802\,816} - \frac{8\,166\,587\,x}{802\,816} - \frac{36\,161\,099\,x^2}{802\,816} - \frac{3\,461\,627\,x^3}{200\,704} - \frac{3\,280\,121\,x^4}{802\,816}$$

r_5 = PolynomialRemainder[r_3, r_4, x]

$$\frac{2\,198\,164\,799\,488}{63\,663\,868\,489} - \frac{2\,030\,741\,536\,768\,x}{63\,663\,868\,489} - \frac{4\,783\,122\,333\,696\,x^2}{63\,663\,868\,489} - \frac{554\,215\,997\,440\,x^3}{63\,663\,868\,489}$$

r_6 = PolynomialRemainder[r_4, r_5, x]

$$\frac{1\,272\,943\,070\,806\,564\,261}{13\,664\,195\,416\,883\,200} - \frac{103\,051\,366\,981\,906\,031\,x}{1\,115\,444\,523\,827\,200} - \frac{10\,141\,289\,265\,339\,652\,563\,x^2}{54\,656\,781\,667\,532\,800}$$

r_7 = PolynomialRemainder[r_5, r_6, x]

$$-\frac{126501666385563244213043200}{12426532615055757392751971 7} - \frac{126501666385563244213043200\,x}{12426532615055757392751971 7}$$

r_8 = PolynomialRemainder[r_6, r_7, x]

0

Simplify[r_7]

$$-\frac{126\,501\,666\,385\,563\,244\,213\,043\,200\,(1+x)}{124\,265\,326\,150\,557\,573\,927\,519\,717}$$

PolynomialGCD[u, v]

1 + x

Очевидно, r_7 является также по определению наибольшим общим делителем многочленов u и v, но Mathematica выдает нормированный многочлен.

Последовательность многочленов r_1, r_2, r_3, …, 0 в алгоритме Евклида называется **последовательностью полиномиальных остатков.**

Рост коэффициентов последовательности полиномиальных остатков может быть минимизирован, если каждый член, как только он получен, нормируется. Определим функцию для определения старшего коэффициента

```
lc[poly_, x_] := Coefficient[poly, x, Exponent[poly, x]]
```

И очередной полиномиальный остаток будет вычисляться по предыдущим остаткам следующим образом

```
rk+2 = (p = PolynomialRemainder[rk, rk+1, x];
  Expand[If[Not[SameQ[p, 0]], p / lc[p, x], p]])
```

Встроенная функция SameQ[p, 0] проверяет, является ли многочлен p нулевым.

Таким образом, для данных многочленов u и v мы получаем последовательность полиномиальных остатков

$$5 + 15\,x + 4\,x^2 - 9\,x^3 + 6\,x^4 + 4\,x^5 - 4\,x^6 + x^7$$

$$-\frac{14}{13} - \frac{146\,x}{39} - \frac{53\,x^2}{39} + \frac{94\,x^3}{39} - \frac{41\,x^4}{39} - \frac{15\,x^5}{13} + x^6$$

$$\frac{981}{896} + \frac{2749\,x}{896} + \frac{1965\,x^2}{896} - \frac{99\,x^3}{224} + \frac{303\,x^4}{896} + x^5$$

$$-\frac{103\,125}{19\,409} + \frac{48\,323\,x}{19\,409} + \frac{213\,971\,x^2}{19\,409} + \frac{81\,932\,x^3}{19\,409} + x^4$$

$$-\frac{684\,517}{172\,585} + \frac{2\,529\,523\,x}{690\,340} + \frac{851\,133\,x^2}{98\,620} + x^3$$

$$-\frac{79\,978\,996}{159\,294\,267}+\frac{79\,315\,271\ x}{159\,294\,267}+x^2$$

```
1 + x
```

```
0
```

Действительно, мы видим, что коэффициенты растут медленнее, но расплачиваемся за это вычислением gcd целых чисел на каждом шаге, чтобы максимально редуцировать дроби.

Заметим, что многочлены u и v на самом деле принадлежат кольцу $\mathbb{Z}[x]$ и $\gcd(u, v)$ также принадлежит кольцу $\mathbb{Z}[x]$. Пока мы еще не рассмотрели вопрос: всегда ли существует наибольший общий делитель для элементов кольца $\mathbb{Z}[x]$? Кольцо $\mathbb{Z}[x]$ не является евклидовым и нельзя применить теорему 2.6.

Забегая вперед, предположим, что в кольце $\mathbb{Z}[x]$ всегда существует наибольший общий делитель. Но из предыдущего примера видно, что использовать арифметику рациональных чисел для вычисления gcd в $\mathbb{Z}[x]$ нецелесообразно; с одной стороны, число требуемых для максимального редуцирования коэффициентов вычислений gcd целых чисел слишком велико, и с другой стороны, отказ от редукции ведет к стремительному росту выражения. Мы вернемся к этому вопросу в главе 7, § 2.

Используя расширенный алгоритм Евклида, мы можем найти не только наибольший общий делитель двух многочленов в $\mathbb{Q}[x]$, но и дополнительные многочлены, входящие в соотношение Безу. В системе Mathematica имеется соответствующая функция PolynomialExtendedGCD.

Расширенный gcd многочленов $f(x)$ и $g(x)$ относительно переменной x есть такой список $\{d, \{r, s\}\}$, что $d = \gcd(f, g)$ и $f r + g s = d$:

```
{f, g} = {2 x^5 - 2 x, (x^2 - 1)^2};
{d, {a, b}} = PolynomialExtendedGCD[f, g, x]
```

$$\left\{-1+x^2, \left\{\frac{x}{4}, \frac{1}{4}\left(-4-2x^2\right)\right\}\right\}$$

```
Expand[a f + b g == d]
```

```
True
```

Полиномы могут иметь символьные коэффициенты:

```
f = 2 a (x^2 - 1)^2 (x^3 - 2);
g = 4 a (x - 1)^3 (x^2 - 3);
{d, {r, s}} = PolynomialExtendedGCD[f, g, x]
```

$$\left\{1-2x+x^2, \left\{-\frac{166-32x-42x^2}{736\,a}, -\frac{-12+188x+224x^2+158x^3+42x^4}{1472\,a}\right\}\right\}$$

Наибольший общий делитель вычисляется с точностью до множителя, не содержащего x.

```
PolynomialGCD[f, g]
```

$$2a(-1+x)^2$$

```
PolynomialExtendedGCD[(x - a)(b x - c)^2, (x - a)(x^2 - b c), x]
```

$$\left\{-a+x, \left\{\frac{b^3c+c^2+2bcx}{b^6c^2-2b^3c^3+c^4}, \frac{-b^5c+3b^2c^2-2b^3cx}{b^6c^2-2b^3c^3+c^4}\right\}\right\}$$

Задача 2.11. Для трех многочленов

```
a = (x – 2) (x –1);
b = (x – 1)^2 (x^3 + 3);
c = (x –1) (x^2 – 7);
```

найти такие многочлены f и g, чтобы выполнялось равенство $f\,a + g\,b = c$.

Решение. Найдем расширенный gcd

```
{d, {r, s}} = PolynomialExtendedGCD[a, b, x]
```

$$\left\{-1+x,\ \left\{\frac{1}{11}\left(-7-2\,x-x^2-x^3\right),\ \frac{1}{11}\right\}\right\}$$

Решение существует, если c делится на d.

```
h = Expand[c / d]
```

$$-7+x^2$$

```
{f, g} = h {r, s}
```

$$\left\{\frac{1}{11}\left(-7+x^2\right)\left(-7-2\,x-x^2-x^3\right),\ \frac{1}{11}\left(-7+x^2\right)\right\}$$

Проверка:

```
Expand[f a + g b == c]
```

True

Задача 2.12. Имеется дробь

```
e1 = (1 + Sqrt[2])^(2/3) / (1 + (1 + Sqrt[2])^(1/3) + (1 + Sqrt[2])^(2/3))
```

содержащая алгебраическое иррациональное число

```
α = (1 + Sqrt[2])^(1/3)
```

Преобразовать дробь так, чтобы иррациональность исчезла из знаменателя.

Решение. Определим два многочлена $h, g \in \mathbb{Q}[x]$:

```
h[x_] := x^2;
g[x_] := x + x^2 +1;
```

Теперь дробь $e1$ есть значение дробно-рациональной функции $h(x)/g(x)$ при $x = \alpha$. Далее мы снова собираемся воспользоваться соотношением Безу в расширенном алгоритме Евклида.

Найдем многочлен $f \in \mathbb{Q}[x]$, для которого число α является корнем. После этого применим к многочленам f и g расширенный алгоритм Евклида. Если f и g будут взаимно просты, то задача легко решается. Действительно, пусть $f\,u + g\,v = 1$ – полученное соотношение Безу. Имеем тогда

$$\frac{h(\alpha)}{g(\alpha)} = \frac{h(\alpha)v(\alpha)}{g(\alpha)v(\alpha)} = \frac{h(\alpha)v(\alpha)}{1-f(\alpha)u(\alpha)} = h(\alpha)v(\alpha),$$

так как $f(\alpha) = 0$.

Проделаем это в Mathematica. Многочлен f можно найти при помощи встроенной функции для нахождения минимального многочлена (см. глава 3, пункт 1.12):

```
f = MinimalPolynomial[α, x]
```

$-1-2x^3+x^6$

Проверка подтверждает правильность нахождения многочлена.

```
Expand[f /. x → α]
```

0

Теперь надо убедиться, что многочлены *f* и *g* взаимно просты.

```
PolynomialGCD[f, g[x]]
```

1

Применяем расширенный алгоритм Евклида для *f* и *g*.

```
{d, {u, v}} = PolynomialExtendedGCD[f, g[x], x]
```

$$\left\{1, \left\{-\frac{1}{2}, \frac{1}{2}\left(1-x-x^3+x^4\right)\right\}\right\}$$

Проверим:

```
Expand[f u  + g[x] v == d]
```

True

Теперь остается только вычислить искомое значение

```
e2 = h[α] (v /. x →  α)
```

$$\frac{1}{2}\left(1+\sqrt{2}\right)^{2/3}\left(-\sqrt{2}\ -\left(1+\sqrt{2}\right)^{1/3}+\left(1+\sqrt{2}\right)^{4/3}\right)$$

И проверка говорит о правильном преобразовании

```
Simplify[e1 == e2]
```

True

Нахождение gcd в кольце $\mathbb{Z}[i]$

Функция GCD в Mathematica также находит наибольший общий делитель для целых гауссовых чисел:

```
GCD[– 10 + 8 I, 5 – 4 I]
```

4 + 5 I

Пусть $w = x + y\,i$ есть наибольший общий делитель двух целых гауссовых чисел z_1 и z_2, тогда, по определению, ассоциированные числа $-w$, iw, $-iw$ также являются $\gcd(z_1, z_2)$. Mathematica из четырех значений выдает то, у которого вещественная и мнимая часть неотрицательны:

```
GCD[{1, – 1, I, – I} (– 10  + 8 I), 5 – 4 I]
```

{4 + 5 I, 4 + 5 I, 4 + 5 I, 4 + 5 I}

Функция ExtendedGCD находит расширенный наибольший общий делитель для целых гауссовых чисел:

```
ExtendedGCD[– 10 + 8 I, 5 – 4 I]
```

{4 + 5 I, {0, 1}}

§ 3. Простые элементы и факториальные кольца

3.1. Элемент *a* из кольца *R* называется **неприводимым**, если он не является ни нулевым, ни обратимым и если каждое *b*, делящее *a*, есть либо единица, либо является элементом ассоциированным с *a*.

Если кольцо R без делителей нуля, то a – неприводимый элемент тогда и только тогда, когда из $a = bc$ следует, что b или c – обратимый элемент в R. В поле каждый ненулевой элемент обратим, и, следовательно, там отсутствуют неприводимые элементы.

3.2. Элемент p называется **простым**, если он не единица и из того, что $p|ab$, следует $p|a$ или $p|b$.

Теорема 3.3. В целостном кольце R простой элемент является неприводимым.

Доказательство. От противного. Пусть p – простой, но разложимый элемент, т.е. $p = ab$ и оба элемента a и b не обратимы. Так как $p|ab$, то из простоты p следует, что $p|a$ или $p|b$. Для определенности будем считать, что $p|a$. Тогда $a = pa'$ для некоторого $a' \in R$. Тогда $p = (pa')b = p(a'b)$. Сокращая обе части равенства на p, получаем $1 = a'b$, что противоречит необратимости b. ♦

Утверждение, обратное теореме 3.3, в общем случае не имеет место. Пусть R – кольцо $\mathbb{Z}[i\sqrt{5}]$, т. е. кольцо комплексных чисел вида $a + ib\sqrt{5}$, где $a, b \in \mathbb{Z}$. В кольце $\mathbb{Z}[i\sqrt{5}]$ число 3 неприводимо, но не простое, потому что 9, квадрат 3, может быть факторизовано двумя способами в кольце, именно, $(2+\sqrt{-5})(2-\sqrt{-5})$ и 3×3. Поэтому $3|(2+\sqrt{-5})(2-\sqrt{-5})$, но 3 не делит ни $2+\sqrt{-5}$, ни $2-\sqrt{-5}$ Числа 3 и $2\pm\sqrt{-5}$ – неприводимые, так как здесь не существует числа $x = a + b\sqrt{-5}$ такого, что x | 3 или $x \mid 2\pm\sqrt{-5}$ из-за того, что $a^2 + 5b^2 = 3$ не имеет целых решений.

Но для колец $\mathbb{Z}$, $\mathbb{Z}[i]$ и $R[x]$ неприводимый элемент является простым.

Теорема 3.4. Пусть R – евклидово кольцо. Элемент p неприводим в R тогда и только тогда, когда он простой в R.

Доказательство. Пусть p есть неприводимый элемент. Требуется доказать, что если $p|ab$, то $p|a$ или $p|b$. От противного. Пусть p не делит a. Тогда $\gcd(p, a) = 1$. По соотношению Безу, существуют такие целые x, y, что $xa + yp = 1$. Умножим обе части равенства на b. Имеем $xab + ypb = b$. Имеем $p|ypb$ и $p|xab$, следовательно, $p|b$. В силу теоремы 3.3 данная теорема доказана. ♦

3.5. Говорят, что целостное кольцо R – кольцо **с разложением на неприводимые множители**, если любой элемент $a \neq 0$ из R можно представить в виде

$$a = up_1p_2\ldots p_r, \tag{5}$$

где u – обратимый элемент, а $p_1, p_2, \ldots, p_r$ – неприводимые элементы (не обязательно попарно различные). Кольцо R называется **кольцом с однозначным разложением на неприводимые множители** (или **факториальным кольцом**), если для каждого ненулевого элемента R разложение (5) единственно с точностью до ассоциированных элементов, точнее: из существования другого такого разложения $a = vq_1q_2\ldots q_s$ следует, что $r = s$ и при надлежащей нумерации элементов p_i и q_j будет

$$q_1 = u_1p_1,\ q_2 = u_2p_2,\ \ldots,\ q_r = u_rp_r,$$

где $u_1, u_2, \ldots, u_r$ – обратимые элементы.

Заметим, что если p – неприводимый, а u – обратимый элемент, то ассоциированный с p элемент up тоже неприводимый.

Часто говорят, что о разложении на *простые* множители, а не на *неприводимые*. Эта терминология вполне допустима, так как в факториальные кольцах из неприводимости элементов следует их простота.

Лемма 3.6. Всякое евклидово кольцо R является кольцом с разложением (т.е. любой ненулевой элемент из R записывается в виде (5).

Доказательство [26, стр. 196]. Пусть элемент $a \in R$ обладает собственным делителем b: $a = bc$, где b и c – необратимые элементы (другими словами, a и b не ассоциированы). Докажем, что $\delta(b) < \delta(a)$.

В самом деле, согласно E_1) непосредственно имеем $\delta(b) \le \delta(bc) = \delta(a)$. Предположив выполнение равенства $\delta(b) = \delta(a)$, мы воспользуемся условием E_2) и найдем q, r с $b = qa + r$, где $\delta(r) < \delta(a)$ или же $r = 0$. Случай $r = 0$ отпадает ввиду неассоциированности a и b. По той же причине $1 - qc \neq 0$. Стало быть, снова по E_2) (поменять местами a и b) имеем

$$\delta(b) = \delta(b) \le \delta(b(1 - cq)) = \delta(b - qa) = \delta(r) < \delta(a)$$

– противоречие. Итак, $\delta(b) < \delta(a)$.

Если теперь $a = a_1a_2\ldots a_n$, где все a_i необратимы, то $a_{m+1}a_{m+2}\ldots a_n$ – собственный делитель $a_ma_{m+1}\ldots a_n$, и по доказанному

$$\delta(a) = \delta(\, a_1a_2\ldots a_n) > \delta(\, a_2\ldots a_n) > \ldots > \delta(\, a_n) > \delta(1).$$

Эта строго убывающая цепочка неотрицательных целых чисел имеет длину $n \le \delta(a)$. Значит, для элемента $a \in R$ имеется разложение максимальной длины, которое и будет разложением на простые множители. ♦

Лемма 3.7. Пусть R – произвольное целостное кольцо с разложением на простые множители. Однозначность разложения в R (факториальность R) имеет место тогда и только тогда, когда свойства неприводимости и простоты элементов в R равносильны.

Доказательство см. в [26, стр. 191-192].

Теорема 3.8. Всякое евклидово кольцо R факториально (т.е. обладает свойством разложения на простые множители).

Доказательство следует из леммы **3.6**, теоремы **3.4** и леммы **3.7**.

Следствие 3.9. Кольца $\mathbb{Z}$, $\mathbb{Z}[i]$ и $R[x]$ (R – поле) факториальны.

Теорема 3.10 [27, стр. 154]. Если R есть факториальное кольцо, то и кольцо многочленов $R[x]$ факториально.

Следствие 3.11. Кольцо многочленов $\mathbb{Z}[x]$ факториально.

§ 4. Кольца многочленов от нескольких переменных

Как ввести кольца многочленов от нескольких переменных? Делается это очень просто. Вспомним, что конструкция кольца $B = A[x]$ включала произвольное коммутативное кольцо A с единицей. Мы можем заменить в нашей конструкции кольцо A на B и построить кольцо $C = B[y]$, где y – новая независимая переменная, играющая по отношению к B ту же роль, что и x по отношению к A. Элементы из C однозначно записываются в виде

$$\sum b_j y^j,\ b_j \in B\,,$$

причем B отождествляется с подкольцом в C, а именно с множеством элементов $b\cdot y^0 = b\cdot 1$. Так как в свою очередь

$$b_j = \sum a_{ij}x^i$$

– однозначная запись элементов $b_j \in B$, то любой элемент из C имеет вид

$$\sum_{i=0}^{k}\sum_{j=0}^{l} a_{ij}x^i y^j,\ a_{ij} \in B,$$

причем подразумевается по (смыслу конструкции), что a_{ij} перестановочны с x и y, а переменная x перестановочна с y. Кольцо C называется кольцом многочленов над A от двух переменных x и y.

Повторив достаточное число раз эту конструкцию, мы получим кольцо $A[x_1, x_2, \ldots, x_n]$ многочленов (полиномов) над A от n переменных. Любой многочлен $f \in A[x_1, x_2, \ldots, x_n]$ однозначно записывается в виде суммы мономов (одночленов)

$$\sum a_{i_1 i_2 \ldots i_n} x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}, \; a_{i_1 i_2 \ldots i_n} \in A.$$

Единственность записи непосредственно вытекает из следующего утверждения.

Теорема 4.1. [26, стр. 186]. Многочлен f равен нулю тогда и только тогда, когда равны нулю все его коэффициенты $a_{i_1 \ldots i_n}$.

Под степенью многочлена f относительно переменной x_k называется наибольшее целое число, которое встречается в качестве показателя при x_k в

$$\sum a_{i_1 i_2 \ldots i_n} x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}.$$

Например, многочлен 1 + x + $xy^3 + x^2y^2$ имеет степень 2 относительно x и степень 3 относительно y.

Целое число $i_1 + i_2 + \ldots + i_n$ называется **(полной) степенью одночлена**

$$x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}.$$

Степенью deg(*f*) (или полной степенью) многочлена f будет максимальная из полных степеней его одночленов. Полагаем $\deg(0) = -\infty$.

На кольцо $A[x_1, x_2, \ldots, x_n]$ переносятся многое результаты для $A[x]$. Например, справедливы следующие утверждения.

Теорема 4.2. [26, стр. 186]. Если A – целостное кольцо, то и $A[x_1, x_2, \ldots, x_n]$ является целостным.

Из теоремы 3.10 следует

Теорема 4.3. Если A – факториальное кольцо, то и $A[x_1, x_2, \ldots, x_n]$ является факториальным кольцом.

Задачи и упражнения

1. Покажите, что для целых a, b и $n > 0$ выполняется равенство

$b^n - a^n = (b - a)(b^{n-1} + b^{n-2}a + b^{n-3}a^2 + \ldots + ba^{n-2} + a^{n-1})$.

2. Пусть $n > m$ – натуральные числа, r – остаток от деления n на m.
(1) Покажите, что остаток от деления $2^n - 1$ на $2^m - 1$ равен $2^r - 1$.
(2) Покажите, что если r – четное число, то остаток от деления $2^n + 1$ на $2^m + 1$ равен $2^r + 1$.

Подсказка: для вычисления частного воспользуйтесь упражнением 1; утверждение вытекает из единственности остатка.

3. Докажите, что если R – целостное кольцо, то $R[x]$ также является целостным кольцом.

4. Докажите теорему 2.3.

5. Докажите следствие 2.7. Подсказка: для доказательства i) и ii) воспользуйтесь соотношением Безу; для доказательства iii) используйте утверждение б) теоремы 2.3.

6. Докажите свойства наибольшего общего делителя 2.1 iii) – vi).

7. Пусть z и $\bar{z}$ – два сопряженных целых гауссовых числа. С помощью Mathematica, перебирая случайным образом z, вычисляйте $\gcd(z, \bar{z})$. Выскажите гипотезу о значении наибольшего общего делителя сопряженных чисел. Докажите её.

8. Пусть n – натуральное число большее единицы. Покажите, что

(1) gcd(n, 2n + 1) = 1;

(2) gcd(2n + 1, 3n + 1) = 1;

(3) gcd(n! + 1, (n+1)! + 1) = 1.

9. Пусть $n > m$ – натуральные числа. С помощью упражнения 2 вычислите

$$\gcd(2^{2^n}+1, 2^{2^m}+1).$$

10. Каждое число, начиная с третьего, в последовательности Фибоначчи 1, 1, 2, 3, 5, 8, 13,… является суммой двух предыдущих. Обозначая n-е число последовательности через f_n, мы получаем $f_0 = f_1 = 1$ и $f_n = f_{n-1} + f_{n-2}$.
(1) Покажите, что gcd(f_n, f_{n+1}) = 1.
(2) Сколько делений с остатком необходимо выполнить для подсчета gcd(f_n, f_{n+1})?

11. Цель настоящего упражнения – выяснить экспериментально, какая часть случайно сгенерированных пар целых чисел состоит из взаимно простых чисел. В Mathematica есть функция RandomInteger, с помощью которой можно генерировать пары псевдослучайных чисел. Ваша программа получает на входе натуральное число m, общее количество генерируемых пар, К каждой их этих пар применяется алгоритм Евклида, который находит наибольший общий делитель, а затем подсчитывается число пар, для которых он равен 1. Выходом служит величина (число пар взаимно простых чисел)/m. Эта дробь задает меру вероятности того, что псевдослучайно выбранная пара состоит из взаимно простых чисел. Чтобы получить хорошее приближение, программу нужно выполнять при больших значениях m. Какие значения вы получили? Теоретический анализ дает правильную вероятность $6/\pi^2$ [25]. Насколько полученные вами значения согласуются с этой величиной?

12. Пусть R – кольцо $\mathbb{Z}[i\sqrt{5}]$, т. е. кольцо комплексных чисел вида $a + ib\sqrt{5}$, где $a, b \in \mathbb{Z}$. Покажите
(1) R есть целостное кольцо.
(2) R не обладает однозначной разложимостью (например, 6 и 9 имеют два различных разложения).

13. Докажите или опровергните следующие правила:
a) gcd(km, kn) = k gcd(m, n);
b) lcm(km, kn) = k lcm(m, n).

Ни одному другому разделу теории чисел не свойственно столько загадочности и изящества, как разделу, занимающемуся изучением простых чисел – непокорных упрямцев, упорно не желающих делиться ни на какие числа, кроме единицы и самих себя.

Мартин Гарднер. Математические досуги

Глава 5. Простые целые числа

§ 1. Основные свойства простых чисел

В силу теоремы 3.4 из главы 4 для кольца $\mathbb{Z}$ мы можем использовать традиционное определение простого числа.

1.1. Целое число $p \in \mathbb{Z}$, называется **простым**, если p удовлетворяет следующим двум условиям:

- $p \neq \pm 1$;
- делителями числа p являются только числа $1, -1, p, -p$.

Остальные целые числа (кроме ± 1) принято называть **составными**.

Лемма 1.2. Наименьший положительный делитель любого числа $n \in \mathbb{N}$, отличный от 1, есть число простое.

Доказательство. Пусть $c \mid n$, $c \neq 1$ и c – наименьшее с этим свойством. Если c – не простое число, то существует делитель c_1 числа c, $1 < c_1 < c$; очевидно, c_1 есть также делитель n. Это противоречит тому, что c есть наименьшее число с указанными выше свойствами. ♦

Лемма 1.3. Наименьший положительный отличный от 1 делитель составного числа $n \in \mathbb{N}$, не превосходит $\sqrt{n}$.

Доказательство. Пусть $c \mid n$, $c \neq 1$ и c – наименьшее число с этим свойством. Следовательно, $n = cn_1$, $n_1 \mid n$, $n_1 \geq c$, значит, $nn_1 \geq c^2 n_1$, $n \geq c^2$ и $c \leq \sqrt{n}$. ♦

Теорема 1.4. (Евклид). Простых натуральных чисел бесконечно много.

Доказательство. От противного. Пусть $p_1, p_2, \ldots, p_n$ – все простые, какие только есть. Рассмотрим число $a = p_1 p_2 \times \ldots \times p_n + 1$. Его наименьший положительный отличный от 1 делитель c, будучи простым (лемма 1.2), не может совпадать ни с одним из $p_1, p_2, \ldots, p_n$, так как иначе $c \mid 1$. ♦

Исторически первым эффективным алгоритмом нахождения простых чисел был метод, названный **решетом Эратосфена** в честь древнегреческого ученого Эратосфена Киренского. Шаги алгоритма следующие:

1. Выпишем числа 2, 3, 4, 5, 6, 7, … . Мы собираемся в этой последовательности отметить только простые числа. Пока ни одно число не отмечено. Все составные числа из последовательности будут удалены.
2. Отметим первый не отмеченный элемент в последовательности как простое число p.
3. Удалим из списка все числа, кратные p (само p не удаляется).
4. Вернемся к шагу 2.

Так как исходная бесконечная последовательность чисел начинается с 2, то 2 – первое простое число. Убирая числа, кратные 2, получаем **2**, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, … – так, что 3 – простое число; убираем теперь числа, кратные 3, получаем: **2**, **3**, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37, 41, 43, … – так, что 5 – простое число;

теперь удаляем все числа, кратные 5, получаем: **2**, **3**, **5**, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 49, 53, … и т. д.

По лемме 1.3 следует, что составление таблицы всех простых чисел, меньших n, закончено сразу, как только вычеркнуты все кратные простым, меньшим $\sqrt{n}$.

Определим функцию eratosthenes[n] в Mathematica, которая выдает список всех простых чисел, меньших данного натурального n. Функция использует список s из n булевских значений. Первоначально список содержит только значения True, в конечном виде s[[k]] == False тогда и только тогда, когда k – не простое число. Окончательно, все простые числа, меньшие n, размещаются в списке primeList, который первоначально пуст.

```
eratosthenes[n_] :=
  Module[{s = Table[True, {n}], primesList = {}, m, k},
    s[[1]] = False;
For[m = 2,  m ≤ IntegerPart[Sqrt[n]],  m++,
 If[s[[m]], For[k = m, k m ≤ n, k++, s[[k m]] = False]]];
For[m = 2, m ≤  n, m++, If[s[[m]], AppendTo[primesList, m]]];
primesList]
```

Функция AppendTo[s, a] выдает в качестве результата список s, после добавления в него элемента a.

```
eratosthenes[100]
```

```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89,
97}
```

Мы уже знаем, что кольцо факториально (следствие А.3.9), но приведем независимое доказательство.

Теорема 1.5. (Основная теорема арифметики). Всякое натуральное число большее 1 единственным образом (с точностью до порядка сомножителей) разложимо в произведение простых чисел.

Доказательство. Пусть $a > 1$, p_1 – его наименьший простой делитель. Значит, $a = p_1a_1$. Если, далее, $a_1 > 1$, то пусть p_2 – его наименьший простой делитель и $a_1 = p_2a_2$, т.е. $a = p_1p_2a_2$, и так далее, пока a_n не станет равным единице. Имеем, таким образом, $a = p_1p_2\ldots p_n$, и возможность разложения доказана.

Покажем единственность. Допустим, существует другое разложение на простые множители, т.е. $a = p_1p_2\ldots p_n = q_1q_2\ldots q_k$. Обе части делятся на p_1, значит, какое-то число q_i делится на p_1, но q_i – простое, поэтому $p_1 = q_i$. Аналогично p_2 равно какому-то числу q_j и так далее до исчерпания либо всех p_t, либо всех q_s. Если p_t, и q_s не кончаются одновременно – возникает противоречие, что доказывает единственность разложения. ♦

Следствие 1.6. Всякое рациональное число однозначно представимо в виде $p_1^{\alpha_1}p_2^{\alpha_2}\ldots p_k^{\alpha_k}$, где $\alpha_1, \alpha_2, \ldots, \alpha_k \in \mathbb{Z}$.

Следствие. **1.7.** Если $a = p_1^{\alpha_1}p_2^{\alpha_2}\ldots p_k^{\alpha_k}$ и $b = p_1^{\beta_1}p_2^{\beta_2}\ldots p_k^{\beta_k}$ – целые числа, то наибольший общий делитель a и b равен $p_1^{\gamma_1}p_2^{\gamma_2}\ldots p_k^{\gamma_k}$, а наименьшее общее кратное a и b равно $p_1^{\delta_1}p_2^{\delta_2}\ldots p_k^{\delta_k}$, где $\gamma_i = \min\{\alpha_i, \beta_i\}$, а $\delta_i = \max\{\alpha_i, \beta_i\}$.

§ 2. Формулы для простых чисел

Много усилий в истории математики было потрачено на получении формул (алгоритмов), дающих простые числа.

Многочлен найденный Эйлером $x^2 + x + 41$ дает 40 простых чисел при $0 \le x \le 39$. Он же нашел и еще несколько многочленов, например, $x^2 - 79x + 1601$ при $0 \le x \le 79$. Но справедлива

Теорема 2.1 (Эйлер). Любой многочлен с целыми коэффициентами (отличный от константы) при некотором натуральном значении аргумента принимает значение, представляющее собой составное число.

Доказательство. Пусть $f(x) = a_0x^n + a_1x^{n-1} + \ldots + a_n$, где все a_i – целые числа. Предположим, что при некотором k значение многочлена $f(x)$ – простое число, т.е. $f(k) = p$, где p – простое. Многочлен степени n принимает одно и то же значение не более чем в n точках. (Действительно, если $f(x) = y_0$ более чем в n точках $x_1, x_2, \ldots, x_{n+1}$, то многочлен $g(x) = f(x) - y_0$ имеет корни $x_1, x_2, \ldots, x_n + 1$, а, как известно, любой многочлен не может иметь более n действительных корней.) Итак, при некотором t имеем: $f(k + pt) \neq 0$ и $f(k + pt) \neq p$. Разлагая $f(k + pt)$ по степеням pt (используя бином Ньютона), получим

$$f(k + pt) = f(k) + c_1pt + c_2(pt)^2 + \ldots + c_n(pt)^n,$$

где все c_i – некоторые целые числа. Поскольку $f(k) = p$, из предыдущего равенства получаем, что $f(k + pt)$ делится на p, причём $f(k + pt) \neq 0$ и $f(k + pt) \neq p$, так что $f(k + pt)$ – составное число. ♦

Вооружившись современной вычислительной техникой, сегодня можно для различных многочленов эмпирически анализировать вероятность принять простое значение на разнообразных промежутках (см. упр. 3).

Приведем несколько примеров формул для простых чисел.

Теорема 2.2. [10]. Пусть p_m – m-е простое число и

$$\alpha = \sum_{m=1}^{\infty} (p_m / 10^{2^m})$$

(этот ряд сходится), тогда n-е простое число

$$p_n = \left\lfloor 10^{2^n} \alpha \right\rfloor - 10^{2^{n-1}} \left\lfloor 10^{2^{n-1}} \alpha \right\rfloor.$$

Теорема 2.3. [5]. Существует вещественное число θ, такое, что при каждом $n \geq 1$ формула

$$f(n) = \left\lfloor \theta^{(3n)} \right\rfloor$$

дает простое число. Число $\theta \approx 1,3064 \ldots$.

Теорема 2.4. [16]. Существует вещественное число ω, такое, что при каждом $n \geq 1$ формула

$$g(n) = \left\lfloor 2^{2^{\cdot^{\cdot^{2^{\omega}}}}} \right\rfloor$$

(башня состоит из n двоек) дает простое число. Число $\omega \approx 1,9287800 \ldots$.

Так как числа θ и ω известны только приближенно и числа, даваемые этими формулами, растут очень быстро, то функции $f(n)$ и $g(n)$ можно рассматривать не более как математические курьезы. Например, $g(1) = 3$, $g(2) = 13$, $g(3) = 16381$, и $g(4)$ имеет более 5000 цифр.

Еще одна формула:

$$p_n = \sum_{m=0}^{n^2+1} \operatorname{sgn}\left(n - \sum_{k=2}^{m} \left(((k-1)!)^2 - k \left\lfloor \frac{((k-1)!)^2}{k} \right\rfloor\right)\right).$$

С практической точки зрения важно эффективно генерировать достаточно большие простые числа, Наиболее популярны до сих пор числа Мерсенна:

$$p = 2^k - 1. \qquad (1)$$

Если k в (1) составное, $k = m\,n$, то p делится на $2^m - 1$. Поэтому (1) может быть источником простых чисел, лишь в случаях простых k (см. упр. 1). Бесконечна ли совокупность простых чисел Мерсенна – неизвестно.

Аналогична ситуация с простыми числами Ферма:

$$p = 2^{2^k} + 1. \tag{2}$$

Среди чисел $2^n + 1$ простыми могут быть лишь числа вида (2). В любом другом случае число $2^n + 1$ составное. Ферма предполагал, что все числа простые (см. упр. 2). Бесконечно ли множество простых чисел Ферма – неизвестно.

Таким образом, простые числа элементарно характеризуются и легко перечисляются (скажем, с помощью решета Эратосфена). Но не ясны закономерности ряда

$$2, 3, 5, 7, 11, 13, 17, 19, 23, \ldots, p_k, \ldots .$$

Кое-что просматривается, но какая-то таинственная часть остается за пределами понимания. Например, отсутствует «хорошая» формула для получения p_k. Или что то же самое, нет «хорошего» алгоритма для p_k, все существующие алгоритмы по сути – переборные.

Отметим один неожиданный результат. Сначала напомним определение. Множество натуральных чисел называется **перечислимым**, если оно есть множество значений некоторой вычислимой функции (иными словами, алгоритма), заданной на $\mathbb{N}$.

Советский математик Юрий Матиясевич доказал в 1970 году, что любое перечислимое множество может быть получено в результате нахождения положительных значений некоторого многочлена [30].

Очевидно, множество простых чисел перечислимо. Найдено несколько соответствующих многочленов для этого множества. Например, множество простых чисел порождают положительные значения следующего полинома от 26 переменных от a до z степени 25,

$$\begin{aligned}
(k+2)\{1 &- [n + l + v - y]^2 - \\
&- [wz + h + j - q]^2 - \\
&- [ai + k + 1 - l - i]^2 - \\
&- [2n + p + q + z - e]^2 - \\
&- [(a^2 - 1)l^2 + 1 - m^2]^2 - \\
&- [(a^2 - 1)y^2 + 1 - x^2]^2 - \\
&- [(gk + 2g + k + 1)(h + j) + h - z]^2 - \\
&- [e^3(e + 2)(a + 1)^2 + 1 - \mathrm{o}^2]^2 - \\
&- [16\mathrm{r}^2\mathrm{y}^4(\mathrm{a}^2 - 1) + 1 - u^2]^2 - \\
&- [z + pl(a - p) + t(2ap - p^2 - 1) - pm]^2 - \\
&- [16(k + 1)^3(k + 2)(n + 1)^2 + 1 - f^2]^2 - \\
&- [q + y(a - p - 1) + s(2ap + 2a - p^2 - 2p - 2) - x]^2 - \\
&- [((a + u^2(u^2 - a))^2 - 1)(n + 4dy)^2 + 1 - (x + cu)^2]^2 - \\
&- [p + l(a - n - 1) + b(2an + 2a - n^2 - 2n - 2) - m]^2\}.
\end{aligned}$$

Если каждую квадратную скобку приравнять нулю, – получится система из 14 полиномиальных уравнений. Извлекая из ее положительных решений $\{a, b, \ldots, z\}$ значения k, получаем список простых чисел $v = k + 2$, – но в крайне запутанном порядке.

§ 3. Метод пробных делений

Современные приложения теории чисел тесно связаны с двумя вопросами:

- как проверить, что данное целое число простое?
- как эффективно раскладывать данное целое число на множители?

Рассмотрим простые алгоритмы, дающие ответы на эти вопросы.

Метод пробных делений заключается в последовательном делении числа n на пробные числа для получения частичного или полного разложения числа n на множители. Сначала берется первое простое число, 2, и мы начинаем делить данное число n на 2 до тех пор, пока это возможно. Затем берется следующее простое число, 3, и мы начинаем процесс деления на 3 оставшейся части числа n и т.д. Если мы достигнем пробного делителя, превосходящего квадратный корень из неразложенной части, то на этом можно остановиться, поскольку сама неразложенная часть есть простое число. Если же каждое простое число, не превосходящее квадратного корня из n, не являются делителем n, то n само является простым числом. Таким образом, метод пробных делений может выяснить, является ли данное число n простым, и если – нет, то разложить его на простые множители.

Пример 3.1. Пусть дано число n = 283503. Это число на 2 не делится. Пробуем разделить на 3. Это делитель и частное равно 94501. Снова пробуем 3, но оно не делит 94501. Теперь пробуем разделить 94501 на 5 и 7 и находим, что эти числа делителями не являются. Далее пробуем число 11. Это делитель, причем частное равно 8591. Снова пробуем число 11 и видим, что деление опять возможно, причем частное равно 781. И в третий раз 11 делит неразложенную часть – частное теперь равно 71. Еще раз пробуем число 11, но оно не делит 71. Теперь 11 превосходит квадратный корень из 71, и, следовательно, 71 – простое число. Таким образом, мы получили разложение на простые множители $282503 = 3 \cdot 11^3 \cdot 71$.

Пробные делители не обязаны непременно быть простыми числами. Если мы предпримем попытку деления на составное число d, предварительно исключив из числа n простые делители числа d, то просто окажется, что это число d не делит оставшуюся часть числа n. Таким образом, мы лишь потеряем некоторое время, но это не помешает найти разложение на простые множители.

Рассмотренный процесс можно ускорить, заметив, что после 2 все простые числа нечетны. Таким образом, в качестве пробных делителей достаточно использовать 2 и все нечетные числа. Приведем описание алгоритма метода пробных делений на 2 и все нечетные числа, большие двух.

Алгоритм 3.2. (метод пробных делений). Дано целое число $n > 1$. Алгоритм создает мультимножество S простых делителей числа n. (Термин «мультимножество» означает, что элементы этого множества могут повторяться.)

```
(* Деление на 2 *)
S = {} ;                          (* Пустое мультимножество *)
m = n;
While (2 | m)
        begin
        m = m/2;
        S = S ∪ {2}
        end;
(* Основной цикл деления *)
d = 3;
While(d^2 ≤ m)
        begin
        While (d | m)
                begin
                m = m/d;
                S = S ∪ {d}
                end;
        d = d + 2
        end;
If (m == 1) then return(S) else return(S ∪ {m})
```

В общем случае, метод пробных делений, не эффективен. Но если число n имеет небольшой делитель, то алгоритм 3.1 достаточно быстро установит, что n – составное число.

Функция trialDivision – реализация этого алгоритма в Mathematica.

```
trialDivision[n_] :=
 Module[{s = {}, m = n, d},
        While[Divisible[m, 2], m = m/2; AppendTo[s, 2]];
        d = 3;
        While[d^2 ≤ m, While[Divisible[m, d], m = m/d;
                                              AppendTo[s, d]];
                       d = d + 2];
        If[m == 1,  s,  AppendTo[s, m]]]
```

§ 4. Метод Ферма разложения на множители

Рассмотрим ситуацию, когда нечетное составное число n равно произведению приблизительно равных сомножителей a и b. Метод Ферма достаточно быстро находит a и b. Точнее, такие a и b, что $n = ab$ и разность

$$\frac{a+b}{2} - \left\lfloor \sqrt{n} \right\rfloor$$

– наименьшая из возможных. Если n – простое, то единственно возможные значения для a и b – это $a = n, b = 1$.

Очевидно, a и b должны быть нечетными числами и поэтому можно определить целые $x = (a + b)/2$ и $y = (a – b)/2$. В этом случае $n = x^2 – y^2 = (x + y)(x – y)$, и, следовательно, $x^2 – n$ является точным квадратом. Поиск квадрата такого вида начинается с $x = \left\lfloor \sqrt{n} \right\rfloor$ – наименьшего числа, при котором разность $x^2 – n$ неотрицательна.

Алгоритм 4.1. (алгоритм Ферма разложения на множители). Дано нечетное натуральное число $n > 1$. Алгоритм выдает нетривиальное разложение числа n или сообщает о том, что n – простое число.

$x = \left\lfloor \sqrt{n} \right\rfloor$;

While ($\sqrt{x^2 - n}$ – не целое число) do x = x+1;

$y = \sqrt{x^2 - n}$; (* y – целое *)

If x+y == n then return("простое число") else return({x + y,x – y})

Почему этот алгоритм заканчивается? Потому что в случае простого n число x достигает значения $(n + 1)/2$, для которого число $\sqrt{x^2 - n}$ целое.

Функция partFactorFermat реализует алгоритм 4.1 в Mathematica.

```
partFactorFermat[n_] :=
 Module[{x = ⌊√n⌋, y},
  While[Not[IntegerQ[√x^2 - n]], x = x + 1]; y = √x^2 - n ;
  If[x + y == n, Return["простое число"], Return[{x + y, x - y}]];
  x = x + 1]
```

Метод пробных делений и алгоритм Ферма, используемые по отдельности, не эффективны. Но, комбинируя их, мы можем сделать большее.

Например, рассмотрим число 13506273942517991. Пробные деления не могут разложить это число на простые множители за разумное время. Но алгоритм Ферма выдает два сомножителя.

```
partFactorFermat[13506273942517991]
```

{116216501, 116216491}

Но с этими сомножителями алгоритм Ферма не может справиться, а метод пробных делений сразу выясняет, что они простые числа.

trialDivision[116216491]

{116216491}

trialDivision[116216501]

{116216501}

§ 5. Mathematica: функции с простыми числами

Рассмотрим основные функции Mathematica, связанные с простыми числами. Функция PrimeQ[n] является тестом на простоту целого числа *n*. Функцию можно применять сразу к списку целых чисел:

PrimeQ[Range[-7, 7]]

{True, False, True, False, True, True, False, False, False, True, True, False, True, False, True}

Функция Prime выдает простое число по заданному порядковому номеру:

{Prime[1], Prime[1000 000 000 000]}

{2, 29 996 224 275 833}

Для функции $\pi(x)$, вычисляющей количество простых чисел, не превосходящих x, используется функция PrimePi. Функция PrimePi является обратной к функции Prime.

PrimePi[29 996 224 275 833]

1000 000 000 000

Функция $\pi(x)$ аппроксимируется тремя функциями: отношением $x/\ln(x)$, интегральным логарифмом

$$\mathrm{li}(x) = \int_0^x \frac{1}{\ln t}\,dt$$

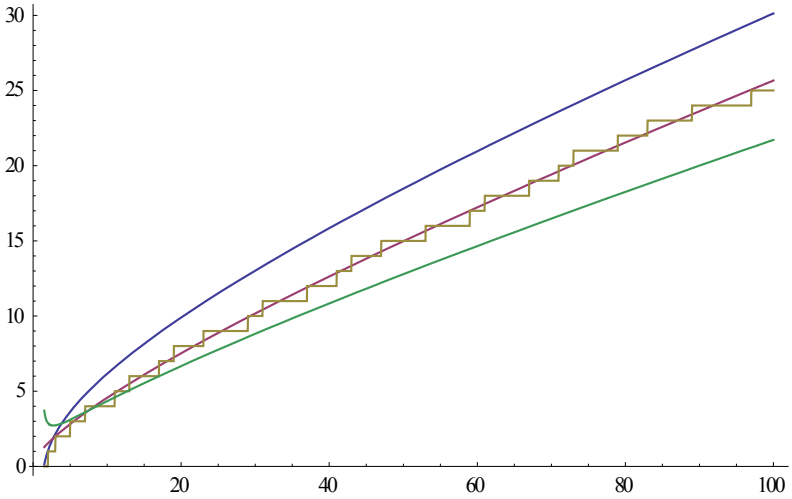
и функцией Римана

$$R(x) = \sum_{n=1}^{\infty} \mu(x)\mathrm{li}(x^{1/n})/n,$$

где μ обозначает функцию Мебиуса[4]. Предел в бесконечности отношения каждой из этих функций к $\pi(x)$ равен 1, но аппроксимация становится все более точной, если перечислять эти функции в следующем порядке: $x/\ln x$, $\mathrm{li}(x)$, $R(x)$. Графики функций на рисунке располагаются в порядке $\mathrm{li}(x)$, $R(x)$, $x/\ln x$; ступенчатая функция – это $\pi(x)$.

Plot[{LogIntegral[n], RiemannR[n], PrimePi[n], n / Log[n], {n, 1.5, 100}]

[4] Если m – произведение четного количества различных простых чисел, то $\mu(m) = 1$; если m – произведение нечетного количества различных простых чисел, то $\mu(m) = -1$ и, наконец, если m имеет кратный простой делитель, то $\mu(m) = 0$.



Функция NextPrime выдает следующее простое число, или предыдущее:

{NextPrime[1000, –1], NextPrime[1000]}

{997, 1009}

Мы можем использовать функцию NextPrime, чтобы найти пары простых чисел – близнецов:

{#, # + 2}& /@ Select[Range[1000], PrimeQ[#] && NextPrime[#] == 2 + #&]

{{3, 5}, {5, 7}, {11, 13}, {17, 19}, {29, 31}, {41, 43}, {59, 61}, {71, 73}, {101, 103}, {107, 109}, {137, 139}, {149, 151}, {179, 181}, {191, 193}, {197, 199}, {227, 229}, {239, 241}, {269, 271}, {281, 283}, {311, 313}, {347, 349}, {419, 421}, {431, 433}, {461, 463}, {521, 523}, {569, 571}, {599, 601}, {617, 619}, {641, 643}, {659, 661}, {809, 811}, {821, 823}, {827, 829}, {857, 859}, {881, 883}}

Функция FactorInteger[n] дает список всех простых делителей числа *n* вместе с их степенями:

FactorInteger[10^101 + 1]

{{11, 1}, {607, 1}, {809, 1}, {1213, 1}, {1327067281, 1},
{11500490394117824585468796003575163076836624586334794818271756072956027758946488969, 1}}

Функция работает и с рациональными числами:

FactorInteger[38 / 81]

{{2, 1}, {3, –4}, {19,1}}

Найдем все натуральные числа до 100, которые являются простыми или степенями простых:

Select[Range[100], Length[FactorInteger[#]] == 1&]

{1, 2, 3, 4, 5, 7, 8, 9, 11, 13, 16, 17, 19, 23, 25, 27, 29, 31, 32, 37, 41, 43, 47, 49, 53, 59, 61, 64, 67, 71, 73, 79, 81, 83, 89, 97}

Задачи и упражнения

1. Найдите несколько простых чисел k, для которых число Мерсенна $2^k - 1$ не является простым.

2. Проверьте гипотезу Ферма о том, что все числа Ферма простые.

3. [28, стр. 69]. Может случиться так, что данный многочлен принимает простые значения на определенных промежутках довольно часто. Покажите при помощи вычислений, что многочлен Дресса и Оливье

$$f(x) = x^2 + x - 1354363$$

обладает тем потрясающим свойством, что для случайного целого числа $x \in [1, 10^4]$ вероятность простоты числа величины $|f(x)|$ превосходит ½. Занимательное следствие таково: если Вам удастся запомнить семизначный телефонный номер 135–43–63, то тем самым Вы запомните и несколько тысяч простых чисел.

4. Почему формула 2.2 абсолютно бесполезна?

5. Докажите, что для любых целых чисел $n_1, n_2, \ldots, n_k$, $k \geq 2$, выполняется соотношение Безу, т.е. существуют такие целые $x_1, x_2, \ldots, x_k$, что выполнено

$$\gcd(n_1, n_2, \ldots, n_k) = x_1 n_1 + x_2 n_2 + \ldots + x_k n_k.$$

6. Докажите, что $\gcd(m, n) \cdot \mathrm{lcm}(m, n) = m \cdot n$.

7. Покажите, что в общем случае для трех целых чисел $abc \neq \gcd(a, b, c)\ \mathrm{lcm}(a, b, c)$.

8. Пусть p_n обозначает n-е простое число. Верно ли, что все числа вида $p_1 p_2 \cdots p_n + 1$ являются простыми?

9. Докажите, что если p, $p + 2$ и $p + 4$ – положительные простые числа, то $p = 3$.

10. Пусть k – натуральное число, k > 1. Покажите, что все числа в последовательности

k! + 2, k! + 3, …, k! + k

составные. Используйте этот факт для доказательства существования отрезков последовательных составных чисел произвольной длины.

Это последнее (исчисление сравнений – В. З.) появилось на свет во второй половине 18 века у Эйлера, Лагранжа, Лежандра и Гаусса. При этом была замечена аналогия его с теорией алгебраических уравнений.

Н. Бурбаки. Очерки по истории математики

Глава 6. Сравнения. Кольца классов вычетов

§ 1. Определения и основные свойства

Пусть m – фиксированное натуральное число, $m > 1$. Множество $m\mathbb{Z}$ всех целых чисел, кратных m, очевидно, замкнуто не только относительно операции сложения, но и относительно операции умножения, и удовлетворяет всем трём аксиомам кольца.

Теперь, используя подкольцо $m\mathbb{Z} \subset \mathbb{Z}$, построим ненулевое кольцо, состоящее из конечного числа элементов. С этой целью введём определение

1.1. Два целых числа a, b называются **сравнимыми по модулю *m***, если при делении на m они дают одинаковые остатки. При этом пишут $a \equiv b \pmod m$, а число m называют **модулем сравнения**. Получается разбиение $\mathbb{Z}$ на классы чисел, сравнимых между собой по модулю m и называемых **классами вычетов по модулю *m***.

Каждый класс вычетов имеет вид

$$\{r\}_m = r + m\mathbb{Z} = \{r + mk \mid k \in \mathbb{Z}\},$$

так что

$$\mathbb{Z} = \{0\}_m \cup \{1\}_m \cup \ldots \cup \{m-1\}_m.$$

Легко проверить, что сравнение является отношением эквивалентности. По определению $a \equiv b \pmod m$ тогда и только тогда, когда $a - b$ делится на m. Удобство записи $a \equiv b \pmod m$ для отношения делимости $m \mid (a - b)$ состоит в том, что с такими сравнениями можно оперировать совершенно так же, как с обычными равенствами (см. теорему С.1.5). Точнее, если $a \equiv b \pmod m$ и $c \equiv d \pmod m$, то выполнено $a \pm c \equiv b \pm d \pmod m$ и $a \cdot c \equiv b \cdot d \pmod m$.

Таким образом, каждым двум классам $\{a\}_m$ и $\{b\}_m$ независимо от выбора в них представителей a, b можно сопоставить классы, являющиеся их суммой или произведением, т.е. на множестве $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z}$ классов вычетов по модулю m однозначным образом индуцируются операции + и ×:

$$\{a\}_m + \{b\}_m = \{a + b\}_m,$$
$$\{a\}_m \times \{b\}_m = \{a \times b\}_m.$$

Так как определения этих операций сводятся к соответствующим операциям над числами из классов вычетов, т.е. над элементами из $\mathbb{Z}$, то $\mathbb{Z}_m$ будет также коммутативным кольцом с единицей $\{1\}_m = 1 + m\mathbb{Z}$. Оно называется **кольцом классов вычетов по модулю *m***. Совокупность вычетов, взятых по одному из каждого класса вычетов, называется **полной системой вычетов по модулю *m*** (в полной системе вычетов, таким образом, всего m штук чисел). Непосредственно сами остатки при делении на m называются **наименьшими неотрицательными вычетами** и, конечно, образуют полную систему вычетов по модулю m.

При фиксированном модуле и некотором навыке при операциях с классами вычетов по модулю m используют записи с фиксированным множеством представителей этих классов, чаще всего – с множеством остатков $\{0, 1, \ldots, m-1\}$. Например, в соответствии с этим соглашением, вместо $m - k \equiv -k \pmod m$, $2(m-1) \equiv -2 \equiv m - 2 \pmod m$ пишут соответственно $m - k = -k$ и $2(m-1) = -2 = m - 2$.

Теорема 1.2. Делителями единицы в $\mathbb{Z}_m$ являются классы взаимно простых чисел с числом m.

Доказательство. Пусть $\varepsilon \in \mathbb{Z}_m$ такое, что существует $\mu \in \mathbb{Z}_m$, для которого $\varepsilon\mu = 1$ (в данном случае классы вычетов мы обозначаем также как некоторые представители этих классов). Поэтому в $\mathbb{Z}$ мы имеем равенство $\varepsilon\mu + km = 1$, что означает ε и m взаимно просты. Обратное также очевидно. ♦

Следствие 1.3. Кольцо $\mathbb{Z}_p$ является полем (конечным) тогда и только тогда, когда p есть простое число.

Доказательство. Если $\mathbb{Z}_p$ – поле, то все элементы, не равные нулю являются обратимыми. Переходя к $\mathbb{Z}$, мы видим, что все целые между 1 и $p - 1$ взаимно просты с p. Обратное также очевидно. ♦

Замечание 1.4. Если $\gcd(a, m) = d > 1$, то класс вычетов a в $\mathbb{Z}_m$ являются делителем нуля.

Следующие свойства сравнений легко доказываются.

Теорема 1.5.

1. Если $a \equiv b \pmod m$ и $c \equiv d \pmod m$, то $a \pm c \equiv b \pm d \pmod m$.
2. Слагаемое, стоящее в какой-либо части сравнения, можно переносить в другую часть, изменив его знак на противоположный.
3. К любой части сравнения можно прибавить любое число, кратное модулю.
4. Если $a \equiv b \pmod m$ и $c \equiv d \pmod m$, то $a \cdot c \equiv b \cdot d \pmod m$.
5. Если $a \equiv b \pmod m$ и $n \in \mathbb{N}$, то $a^n \equiv b^n \pmod m$.
6. Пусть $p(x) = p_n x^n + p_{n-1}x^{n-1} + \ldots + p_1 x + p_0$ и $q(y) = q_n y^n + q_{n-1}y^{n-1} + \ldots + q_1 y + q_0$, где все $p_i, q_j, x, y \in \mathbb{Z}$ и $n \in \mathbb{N}$. Если $x \equiv y \pmod m$ и $p_k \equiv q_k \pmod m$, для всех $k = 0, 1, \ldots, n$, то $p(x) \equiv q(x) \pmod m$.
7. Если $ac \equiv bc \pmod m$ и $\gcd(c, m) = 1$, то $a \equiv b \pmod m$.
8. Сравнения $a \equiv b \pmod m$ и $ac \equiv bc \pmod{mc}$, выполняются одновременно.
9. Если $a \equiv b \pmod{m_1}$ и $a \equiv b \pmod{m_2}$, то $a \equiv b \pmod{\operatorname{lcm}(m_1 m_2)}$; $\operatorname{lcm}(x, y)$ – наименьшее общее кратное x и y.
10. Если $a \equiv b \pmod m$ и $d \mid m$, то $a \equiv b \pmod d$.
11. Если одна часть и модуль делятся на некоторое число, то и другая часть сравнения должна делиться на то же число.

Следующая теорема показывает применение отношения сравнения.

Теорема 1.6. (Вильсон-Лагранж)**.** Число $p > 1$ является простым тогда и только тогда, когда

$$(p - 1)! \equiv -1 \pmod p.$$

Доказательство. Пусть p – простое число. Сопоставим каждому из чисел $1, 2, \ldots, p - 1$ обратное ему по модулю p. Обратное к a – это такое число a^{-1}, для которого $a \cdot a^{-1} \equiv 1 \pmod p$. Каждое число из ряда $1, 2, \ldots, p - 1$ имеет в этом ряду точно одно обратное. Число, обратное к a, может быть равно a; тогда $a^2 \equiv 1 \pmod p$, т.е. $a \equiv \pm 1 \pmod p$, откуда следует, что $a = 1$ или $a = p - 1$. Все остальные числа $2, 3, \ldots, p - 2$ (кроме 1 и $p - 1$) могут быть разбиты на пары, так что произведение чисел в каждой паре сравнимо с 1 по модулю p. Следовательно,

$$2 \cdot 3 \cdot 4 \cdot \ldots \cdot (p - 2) \equiv 1 \pmod p.$$

Если умножить это сравнение на $p - 1$, то получим сравнение $(p - 1)! \equiv -1 \pmod p$, так как $p - 1 \equiv -1 \pmod p$. Если p равно 2 или 3, то приведенное доказательство не проходит; в этих случаях, однако, результат устанавливается непосредственно.

Пусть теперь p – составное, тогда оно имеет простой делитель p_1, причём $p_1 < p$ и, следовательно, $p_1|(p-1)!$. Тогда получим, что $(p-1)! + 1$ не делится на p_1 и, тем более, на p. ♦

Теорема Вильсона-Лагранжа дает критерий простоты числа, но его разумно использовать для небольших чисел (имеющих не более 10 цифр).

§ 2. Возведение в степень

Возведение в степень является фундаментальной операцией теории чисел (см., например, теорему 5.6 Эйлера), и особый интерес представляет возведение в степень с приведением по модулю. Для возведения некоторого x в n-ю степень обычно не требуется перемножать все n символов $x \cdot x \cdot \ldots \cdot x$, как записано. Приведем более эффективный (для больших степеней) рекурсивный алгоритм. В качестве объектов, возводимых в степень, могут выступать целые числа, элементы конечного поля, многочлены или что-нибудь еще. Для алгоритма существенно лишь то, чтобы x принадлежал некоторой полугруппе, то есть множеству, на котором определено произведение $x \cdot x \cdot \ldots \cdot x$.

Алгоритм 2.1. (рекурсивный алгоритм для возведения в степень). Алгоритм вычисляет x^n, где x принадлежит некоторой полугруппе, а n – положительное целое число.

```
(*Рекурсивная функция pow(x, n) *)
If n = 1 then return x ;
If n – четно then return pow(x, n/2)^2;
            else  return x * pow(x, (n – 1)/2)^2 ;
```

Рекурсивную структуру алгоритма проиллюстрируем на примере числа 5^{15}. Легко видеть, что порядок операций будет следующий

$$pow(5,15) = 5\,pow(5,7)^2 = 5(5\,pow(5,3)^2)^2$$
$$= 5(5(5\,pow(5,1)^2)^2)^2 = 5(5(5\times 5^2)^2)^2.$$

Когда вычисляется $x^n \bmod m$, на каждом шаге алгоритма (четном или нечетном) требуется выполнять приведение по модулю. Например, вычисляя полученную выше цепочку степеней по модулю $m = 8$, легко получить ответ

$$5^{15} (\bmod 8) = 5(5(5\times 5^2)^2)^2 (\bmod 8)$$
$$= 5(5(5\times 1)^2)^2 (\bmod 8) = 5(5\times 25)^2 (\bmod 8)$$
$$= 5(5\times 1)^2 (\bmod 8) = 5\times 25 (\bmod 8)$$
$$= 5\times 1 (\bmod 8) = 5.$$

Встроенная функция в *Mathematica* для возведения в степень Power(x, n), или в другой записи x^n, вычисляется эффективно, но при вычислении по модулю надо использовать функцию PowerMod(x, n, m), так как тогда вычисления идут с небольшими числами и поэтому быстрее.

Пример 2.2. Оценим время при вычислении $7^{150000008} \bmod 12$ двумя разными способами. В первом случае остаток вычисляется всего один раз, а во втором случае на каждом шаге. Функция Timing выдает продолжительность вычисления выражения и его значение.

```
Mod[7^150000008, 12]//Timing
```

```
{20.062, 1}
```

```
PowerMod[7, 150000008, 12]//Timing
```

```
{0., 1}
```

§ 3. Нахождение обратного элемента по модулю

Рассмотрим задачу инвертирования, т. е. вычисления обратного элемента n^{-1} по модулю m. Эта операция, когда она возможна, возвращает единственное целое число $x \in [1, m-1]$, такое, что $nx \equiv 1 \pmod{m}$. Эта операция тесно связана с нахождение наибольшего общего делителя.

Напомним отношение Безу для gcd. Если n и m – не равные одновременно нулю целые числа, то найдутся такие a, b, что $an + bm = \gcd(n, m)$. Из этого отношения немедленно получаем следствие для инвертирования: если n и m – положительные целые числа, и $\gcd(n, m) = 1$, то уравнение $an + bm = 1$ разрешимо, и a является обратным к n по модулю m, т.е $n^{-1} \equiv a \pmod{m}$.

Мы рассматривали рекурсивную версию расширенного алгоритма Евклида (алгоритм 2.9, глава 4), сейчас предлагается версия с циклом.

Алгоритм 3.1. (расширенный алгоритм Евклида для вычисление gcd и инвертирования). По заданным целым числам m и n, $m \geq n \geq 0$ и $m > 0$ алгоритм возвращает тройку целых чисел (a, b, d), таких, что an + bm = d = gcd(m, n).

```
(*Инициализация*)
      (a, b, d, u, v, w) = (1, 0, x, 0, 1, y);
(*Цикл расширенного алгоритма Евклида*)
      While w > 0 do
            begin
            q = ⌊d/w⌋;
            (a, b, d, u, v, w) = (u, v, w, a – qu, b – qv, d – qw)
            end;
       return(a, b, d)
```

Таким образом, для решения сравнения $nx \equiv 1 \pmod{m}$, необходимо 1) вызывать этот алгоритм с входными данными m и $n \bmod m$ и 2) если $d = 1$, то $x = n^{-1} = b \bmod m$.

В системе *Mathematica* алгоритм 3.1. может быть представлен программой:

```
extendedgcd[m_, n_] :=
   Module[{a, b, d, u, v, w, q},
                  {a, b, d, u, v, w} = {1, 0, m, 0, 1, n};
                  While[w > 0,
                      q = Floor[d / w];
                     {a, b, d, u, v, w} = {u, v, w, a - q u, b-q v, d-q w}];
                  {a, b, d}
          ]
```

Функция inversion находит обратный элемент для n по модулю m.

```
inversion[n_, m_] :=
 Module[{x, y, a, b, d}, If[n > m, {x, y}={m, Mod[n, m]}, {x, y} = {m, n}];
                         {a, b, d} = extendedgcd[x, y];
                         If[d == 1, Mod[b, m], {}]
        ]
```

Пример 3.2. Рассмотрим два достаточно больших числа

$$m = 18496 \times 2^{18496} + 1;$$

```
n = Quotient[2 m, 3];
```

Числа n и m имеют соответственно 5572 и 5573 цифр:

```
IntegerLength[{n, m}]
```

```
{5572, 5573}
```

Легко проверить, что числа m и n взаимно просты:

extendedgcd[n, m]

{–3, 2, 1}

Вычислим обратный элемент для n по модулю m и сразу проверим вычисления:

Mod[n × inversion[n, m], m] == 1

True

Для нахождения обратного элемента для n по модулю m в Mathematica используется функция Solve.

Пример 3.3. Вычислить обратный элемент для 9876543210 по модулю 541:

Solve[9876543210 x == 1, {x}, Modulus → 541]

{{x → 227}}

Проверим

Mod[9876543210 × 227, 541] == 1

True

Функция Solve используется для решения различных уравнений и систем уравнений; опция Modulus → 541 говорит о том, что вычисления производятся по модулю 541. Для больших чисел, таких как n и m из примера 3.2 нахождение обратного элемента с помощью функции inversion происходит быстрее, чем с помощью Solve.

§ 4. Решение сравнений

Решение линейных сравнений

Теорема 4.1. Рассмотрим сравнение $a\,x \equiv b \pmod m$. Пусть $\gcd(a, m) = d$. Если b не делится на d, то сравнение $a\,x \equiv b \pmod m$ не имеет решений. Если $d = 1$, то сравнение имеет одно решение $x \equiv a^{-1}b \pmod m$. Если b кратно $d > 1$, то сравнение $a\,x \equiv b \pmod m$ имеет d штук решений.

Доказательство. Рассмотрим два случая.

Случай 1. Пусть $a\,x \equiv b \pmod m$, $\gcd(a, m) = 1$. Так как, a и m взаимно просты, то, умножая обе части сравнения на a^{-1}, получаем $a^{-1}ax \equiv a^{-1}b \pmod m$ и, следовательно, $x \equiv a^{-1}b \pmod m$.

Случай 2. $a\,x \equiv b \pmod m$, $\gcd(a, m) = d > 1$. В этом случае для разрешимости сравнения необходимо, чтобы d делило b, иначе сравнение вообще выполняться не может. Действительно, $a\,x \equiv b \pmod m$ бывает тогда, и только тогда, когда $a\,x - b$ делится на m, т.е. $a\,x - b = tm$, $t \in \mathbb{Z}$, откуда $b = a\,x - tm$, а правая часть последнего равенства кратна d.

Пусть $a = a_1 d$, $b = b_1 d$ и $m = m_1 d$. Тогда, в силу теоремы 1.5(8), обе части сравнения $a_1 d\,x \equiv b_1 d \pmod{m_1 d}$ и модуль поделим на d: $a_1\,x \equiv b_1 \pmod{m_1}$, где уже a_1 и b_1 взаимно просты. Согласно случаю 1 такое сравнение имеет единственное решение x_0:

$$x \equiv x_0 \pmod{m_1}. \qquad (1)$$

По исходному модулю m, числа (1) образуют столько решений исходного сравнения, сколько чисел вида (1) содержится в полной системе вычетов: 0. 1, 2, …, $m - 2$, $m - 1$. Очевидно, из чисел $x = x_0 + tm$ в полную систему наименьших неотрицательных вычетов попадают только x_0, $x_0 + m_1$, …, $x_0 + 2m_1$, … $x_0 + (d - 1)m_1$, т.е. всего d чисел. Значит, у исходного сравнения имеется d решений. ♦

Пример 4.2. Решение сравнения $20x \equiv 16 \pmod{12}$ получаем в Mathematica:

Solve[20 x == 16, {x}, Modulus → 12}

{{x → 2 + 3 C[1]}}

C[1] обозначает целочисленный параметр: различные решения по модулю 12 получаем при C[1] = 0, 1, 2, 3, т.е. решениями являются вычеты 2, 5, 8 и 11.

Легко написать специальную функцию для решения сравнения $a\,x \equiv b \pmod m$.

```
solveMod[a_, b_, m_] :=
Module[{d = GCD[a, b], x, xs},
       xs = Solve[a x==b, {x}, Modulus -> m];
       If[xs[[1]] === {}, {}, xs[[1, 1, 2]] /. C[1] -> #& /@ Range[0, d – 1]]
       ]
```

```
solveMod[20, 16, 12]
```

```
{2, 5, 8, 11}
```

```
solveMod[88, 48, 24]
```

```
{0, 3, 6, 9, 12, 15, 18, 21}
```

Китайская теорема об остатках

Системы линейных сравнений с одним неизвестным были известны в Древнем Китае, в Индии и Греции. Следующая задача, например, решалась китайскими математиками.

Найти число, которое дает остаток 1 при делении на 3, остаток 2 при делении на 5 и остаток 3 при делении на 7.

В общем случае можно сформулировать следующую задачу.

Пусть $r_1, \ldots, r_n$ и $m_1, \ldots, m_n > 1$ – целые числа, $m_1, \ldots, m_n$ попарно взаимно просты. Найти x – решение системы сравнений

$$x \equiv r_1 \pmod{m_1}, \ldots, x \equiv r_n \pmod{m_n}.$$

Мы начнем с $n = 2$.

Теорема 4.3. (Китайская теорема об остатках – частный случай). Если $m_1 > 1$ и $m_2 > 1$ – взаимно простые числа, то система сравнений

1. $x \equiv r_1 \pmod{m_1}$,
2. $x \equiv r_2 \pmod{m_2}$.

имеет единственное решение $x \in [0, m_1 m_2 - 1]$.

Доказательство. Решение сравнения (1) имеет форму $x + k\,m_1$ для произвольного $k \in \mathbb{Z}$. Более того, мы также имеем решение сравнения (2), если $x + k\,m_1 \equiv r_2 \pmod{m_2}$, т.е. если $k\,m_1 \equiv r_2 - r_1 \pmod{m_2}$. По соотношению Безу для $\gcd(m_1, m_2)$ существует некоторое целое число c такое, что $cm_1 \equiv 1 \pmod{m_2}$. Поэтому $s = (r_2 - r_1)c$ есть решение (2). ♦

Алгоритм 4.4. (решение двух линейных сравнений с одним неизвестным)

Вход: два остатка (правые части сравнений) r_1 и r_2 и два соответствующих модуля m_1 и m_2. Выход: решение x.

Находим c – решение сравнения $c\,m_1 \equiv 1 \pmod{m_2}$;
$b = r_1 \bmod m_1$;
$s = (r_2 - r_1)c \bmod m_2$;
$x = b + s\,m_1$;
return(x)

Общий случай $n > 2$ сводится к частному случаю. Эта редукция базируется на следующих фактах.

Лемма 4.5. 1. Пусть $m_1, \ldots, m_n > 1$ – целые попарно взаимно простые числа и положим $P = \prod_{i=1}^{n-1} m_i$. Тогда m_n и P взаимно просты.

2. Пусть a, b и m_1, $m_2 > 1$ – целые числа и m_1, m_2 взаимно просты. Тогда $a \equiv b \pmod{m_1}$ и $a \equiv b \pmod{m_2}$ только тогда, когда $a \equiv b \pmod{m_1 m_2}$.

Доказательство. См. упр. 2.

Пусть мы имеем n сравнений и r будет решением первых двух сравнений. Тогда решение исходных n сравнений будет такое же, как решение следующих $n - 1$ сравнений

$$x \equiv r \pmod{m_1 m_2},$$
$$x \equiv r_i \pmod{m_i},\ i = 3, 4, \ldots, n.$$

Повторяя этот процесс, мы приходим к случаю 2 сравнений, и решение получаем с помощью алгоритма 4.4.

Теорема 4.6. Система n линейных сравнений с одним неизвестным всегда имеет единственное решение по модулю $m_1 \cdot m_2 \cdot \ldots \cdot m_n$, и оно вычисляется с помощью алгоритма 4.7.

Алгоритм 4.7 (решение системы линейных сравнений с одним неизвестным – общий случай). Вход: список остатков (правые части сравнений) $r_1, \ldots, r_n$ и список модулей $m_1, \ldots, m_n$. Выход: решение x.

$P = m_1$;
$x = r_1 \bmod m_1$;
for k = 2 to n do
    begin
    вычисляем x – выход алгоритма 4.4 с входом x, r_k, P, m_k;
    $P = P \cdot m_k$
    end;
return(x)

Программы для алгоритмов 4.4 и 4.7 в Mathematica могут быть следующими:

```
cr2[r1_, r2_, m1_, m2_] :=
      Module[{c, b, s}, c = Solve[c m1 == 1, {c}, Modulus → m2][[1, 1, 2]];
                        b = Mod[r1, m1];
                        s = Mod[(r2 – r1)c, m2];
                        b + s m1]

cr[n_, rs_, ms_] :=
 Module[{p, x, k},
            p = ms[[1]]; x = Mod[rs[[1]], p]; k := 2;
            While[k ≤ n, x = cr2[x, rs[[k]], p, ms[[k]]]; p=p ms[[k]]; k++];
            x]
```

Найдем решение для китайской задачи:

```
cr[3, {1, 2, 3}, {3, 5, 7}]
```

52

Сравнение любой степени

Пусть левая часть сравнения есть многочлен относительно неизвестного x. Выясним, сколько может быть решений у такого сравнения.

Лемма 4.8. Пусть $h(x)$ – многочлен степени n с целыми коэффициентами. Для любого целого числа a найдется многочлен $q(x)$ степени $n - 1$, удовлетворяющий условию:

$$h(x) = (x - a)q(x) + h(a).$$

Доказательство. Разделив многочлен $h(x)$ на $x - a$, мы найдем такие многочлены $q(x)$ и $r(x)$, что

$$h(x) = q(x)(x - a) + r(x), \qquad (2)$$

причем либо $r(x) = 0$, либо его степень меньше степени $x - a$ (теорема 1.8 из главы 4). Таким образом, $r(x) = c$ – целое число. Заменим в уравнении (2) x на a:

$$h(a) = q(a)(a - a) + c = c.$$

Таким образом, мы можем переписать (2) в виде:

$$h(x) = (x - a)q(x) + h(a),$$

что завершает доказательство. ♦

Теорема 4.9. Пусть $f(x)$ – многочлен степени n с целыми коэффициентами и старшим коэффициентом 1. Если p – простое число, то сравнение $f(x) \equiv 0 \pmod p$ имеет не более n решений.

Доказательство. Докажем теорему индукцией по степени многочлена. Если $n = 1$, то имеем линейное сравнение по простому модулю, которое имеет только одно решение (теорема 4.1). Базис индукции доказан.

Предположим теперь, что любой многочлен степени $k - 1$ со старшим коэффициентом 1 имеет не более $k - 1$ корня в $\mathbb{Z}_p$. Докажем, что данное предположение влечет аналогичное утверждение для многочлена степени k со старшим коэффициентом 1.

Итак, пусть $f(x)$ – многочлен степени k с целыми коэффициентами и старшим коэффициентом 1. Если $f(x)$ не имеет корней в $\mathbb{Z}_p$, то доказывать нечего, ибо $0 \le k$. (Пример 4.10 такого многочлена приведен после доказательства.) Следовательно, мы можем предполагать, что $f(x)$ имеет корень a в $\mathbb{Z}_p$, т.е. $f(a) \equiv 0 \pmod p$. По лемме 4.8

$$f(x) = (x - a)q(x) + f(a), \tag{3}$$

где степень $q(x)$ равна $k - 1$. Так как старшие коэффициенты многочленов $f(x)$ и $x - a$ равны 1, то же самое справедливо и для многочлена $q(x)$. Значит, мы можем применить индуктивное предположение к $q(x)$.

Редуцируя равенство (3) по модулю p, мы получаем

$$f(x) \equiv (x - a)q(x) \pmod p. \tag{4}$$

Пусть $b \ne a$ – еще один корень $f(x)$ в $\mathbb{Z}_p$. Это означает, что $f(b) \equiv 0 \pmod p$, но $a - b \ne 0$ в $\mathbb{Z}_p$. Отсюда и из (4), заменяя x на b, получаем

$$0 \equiv f(b) \equiv (b - a)q(b) \pmod p.$$

Так как p – простое, то $\mathbb{Z}_p$ – поле и, следовательно, $\mathbb{Z}_p$ не имеет делителей нуля. Поэтому имеем $q(b) \equiv 0 \pmod p$. Мы заключаем, что если b – корень многочлена $f(x)$, отличный от a, то b является и корнем $q(x)$ в $\mathbb{Z}_p$. Иначе говоря, у $f(x)$ только на один корень больше (в $\mathbb{Z}_p$), чем у $q(x)$. А по предположению индукции у последнего многочлена не более чем $k - 1$ различных корней в Z_p. Следовательно, $f(x)$ имеет не больше k разных корней. Доказательство теоремы закончено. ♦

Пример 4.10. Найдем все возможные вычеты у многочлена $f(x) = x^2 + 3$ по модулю 5. Используем Mathematica:

```
Table[Mod[x^2 + 3, 5], {x, 0, 4}]
```

```
{3, 4, 2, 2, 4}
```

Таким образом, сравнение $x^2 + 3 \equiv 0 \pmod 5$ удовлетворяет условиям теоремы, и не имеет решений в $\mathbb{Z}_5$.

Пример 4.11. Рассмотрим, что происходит, когда мы ищем решение сравнения по составному модулю. Рассмотрим сравнение $x^2 - 17 \equiv 0 \pmod 8$. Мы и в этом случае в Matematica используем функцию Solve.

```
Solve[x^2 – 17 == 0, {x}, Modulus → 8]
```

{{x → 1}, {x → 3}, {x → 5}, {x → 7}}

В данном случае сравнение второй степени имеет 4 решения.

§ 5. Функция Эйлера φ

5.1. Функцией Эйлера $\varphi: \mathbb{N} \to \mathbb{N}$ называется функция, значение которой $\varphi(n)$ равно количеству целых чисел $1 \le k \le n$, таких что $\gcd(k, n) = 1$.

Первые несколько величин $\varphi(n)$ есть:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\varphi(n)$	1	1	2	2	4	2	6	4	6	4	10	4	12	6

Теорема 5.2. Функция Эйлера есть мультипликативная функция, что означает:

$$\forall m, n \in \mathbb{Z}, \gcd(m, n) = 1 \Rightarrow \varphi(mn) = \varphi(m)\varphi(n).$$

Для доказательства нам потребуется

Лемма 5.3. 1). Любые m штук попарно несравнимых по модулю m чисел образуют полную систему вычетов по модулю m.

2). Если a и m взаимно просты, а x пробегает полную систему вычетов по модулю m, то значения линейной формы $ax + b$, где b – любое целое число, тоже пробегает полную систему вычетов по модулю m.

Доказательство. Утверждение 1) очевидно. Докажем утверждение 2). Чисел $ax + b$ ровно m штук. Покажем, что они между собой не сравнимы по модулю m. Пусть для некоторых x_1 и x_2 из полной системы вычетов оказалось, что $ax_1 + b \equiv ax_2 + b \pmod m$. Тогда, по свойствам сравнений, получаем

$$ax_1 \equiv ax_2 \pmod m,$$
$$x_1 \equiv x_2 \pmod m$$

– противоречие с тем, что x_1 и x_2 различны и взяты из полной системы вычетов. ♦

Доказательство теоремы. Разместим числа от 1 до mn в таблице из m строк и n столбцов:

$$\begin{array}{lllll}
1 & m+1 & 2m+1 & \dots & (n-1)m+1 \\
2 & m+2 & 2m+2 & \dots & (n-1)m+2 \\
3 & m+3 & 2m+3 & \dots & (n-1)m+3 \\
\dots & & & & \\
r & m+r & 2m+r & \dots & (n-1)m+r \quad \leftarrow \text{строка с номером } r \\
\dots & & & & \\
m & 2m & 3m & \dots & nm
\end{array}$$

Пусть r будет положительное целое $\le m$, такое что $\gcd(r, m) = d > 1$. Покажем, что в строке с номером r нет элементов, взаимно простых с mn. Имеем $d|r$ и $d|m$, поэтому $d|km + r$ для любого целого k; т.е. d делит каждый элемент в строке с номером r.

Поэтому в строке с номером r нет чисел взаимно простых с m и, следовательно, с mn. Другими словами, числа в таблице могут быть взаимно простыми с mn только в строках с номерами r, для которых $\gcd(r, m) = 1$. По определению, $\varphi(m)$ равно количеству таких чисел r и, следовательно, имеется $\varphi(m)$ таких строк.

Рассмотрим сейчас какую-нибудь строку с номером r, для которой $\gcd(r, m) = 1$:

$$r, m + r, 2m + r, \dots, (n-1)m + r.$$

Так как gcd(m, n) = 1, то по лемме 5.3, числа этой строки есть полная система вычетов по модулю n, т.е. остатки этих чисел при делении на n образуют перестановку чисел 0, 1, 2,…, n – 1, среди которых чисел взаимно простых с n есть φ(n) штук. Поэтому ровно φ(n) элементов строки с номером r являются взаимно простыми с n и, следовательно, с mn.

Поэтому φ(m) строк содержат натуральные числа, взаимно простые с mn, и в каждой такой строке содержится φ(n) элементов, взаимно простых с mn. Таким образом, таблица содержит φ(m)φ(n) натуральных чисел ≤ mn и взаимно простых с mn; т.е. φ(mn) = = φ(m)φ(n). ♦

Свойство мультипликативности сводит вычисление φ(n) к вычислению φ(p^α), где p – простое и $\alpha \geq 1$. Чтобы вычислить значение φ(p^α) мы сначала посчитаем количество целых $h \in [1, p^\alpha]$, для которых gcd(h, p) > 1. Любое такое число должно делиться на p и поэтому имеет вид

$$h = pq, \qquad 1 \leq q \leq p^{\alpha-1}.$$

Поэтому количество таких чисел h равно $p^{\alpha-1}$. Это дает

$$\varphi(p^\alpha) = p^\alpha - p^{\alpha-1} = p^\alpha\,(1 - 1/p).$$

Если $n = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$ есть разложение числа n на простые множители, то мы получаем формулу

$$\varphi(n) = (p_1^{\alpha_1} - p_1^{\alpha_1-1}) \cdots (p_k^{\alpha_k} - p_k^{\alpha_k-1}) = n(1-1/p_1) \cdots (1-1/p_k).$$

В системе *Mathematica* функция Эйлера $\varphi(n)$ вычисляется с помощью EulerPhi[n]. В справочной службе системы Mathematica приведены несколько интересных численных значений функции Эйлера.

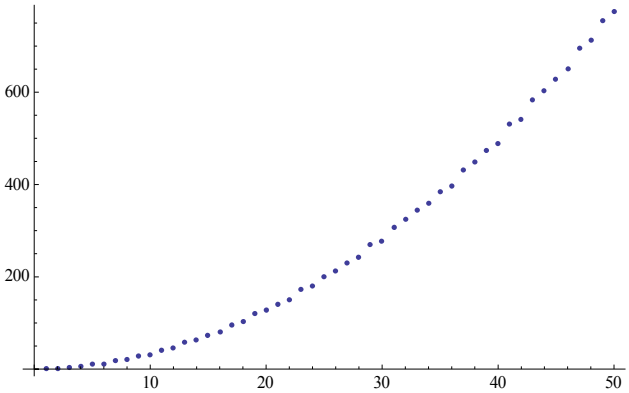
Функция Эйлера для степеней 10:

```
Table[EulerPhi[10^k], {k, 0, 10}]
```

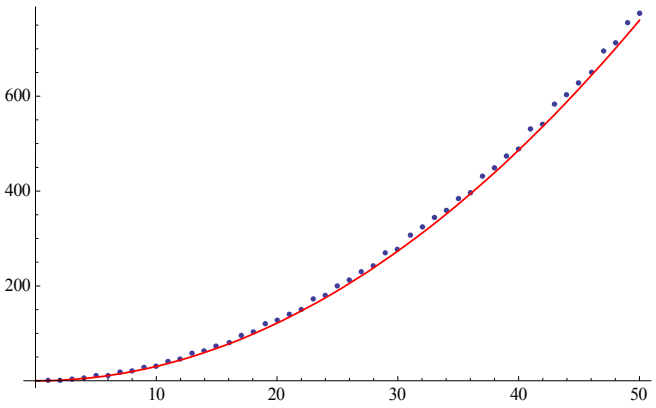
```
{1, 4, 40, 400, 4000, 40000, 400000, 4000000, 40000000, 40000000, 4000000000}
```

Накопленные суммы функции Эйлера аппроксимируются функцией $(3n^2)/\pi^2$:

```
ListPlot[Table[Sum[EulerPhi[k], {k, n}], {n, 50}]]
```




```
Show[%, Plot[3 n^2/π^2, {n, 0, 50}, PlotStyle → Red]]
```



О встроенной функции Show см. справочную службу Mathematica.

Прежде чем сформулировать теорему Эйлера, приведем необходимое определение.

5.4. Приведенной системой вычетов по модулю m называется совокупность всех вычетов из полной системы, взаимно простых с модулем m.

Приведенную систему обычно выбирают из наименьших неотрицательных вычетов. Ясно, что приведенная система вычетов по модулю m содержит $\varphi(m)$ штук вычетов.

Лемма 5.5. 1). Любые $\varphi(m)$ чисел, попарно не сравнимые по модулю m и взаимно простые с модулем, образуют приведенную систему вычетов по модулю m.

2). Если $\gcd(a, m) = 1$ и x пробегает приведенную систему вычетов по модулю m, то ax также пробегает приведенную систему вычетов по модулю m.

Доказательство. Утверждение 1) очевидно. Докажем утверждение 2). Числа ax попарно несравнимы (доказательство как в лемме 5.3), их ровно $\varphi(m)$ штук. Ясно также, что все они взаимно просты с модулем, так как $\gcd(a, m) = 1$ и $\gcd(x, m) = 1$ влечет $\gcd(ax, m) = 1$. Значит, числа ax образуют приведенную систему вычетов. ♦

Теорема 5.6 (Эйлер). Пусть $m > 1$, $\gcd(a, m) = 1$, тогда

$$a^{\varphi(m)} \equiv 1 (\mathrm{mod}\, m).$$

Доказательство. Пусть x пробегает приведенную систему вычетов по модулю m:

$$x = r_1, r_2, \ldots, r_n,$$

где $n = \varphi(m)$ – их число, $r_1, r_2, \ldots, r_n$ – наименьшие неотрицательные вычеты по модулю m. Следовательно, наименьшие неотрицательные вычеты, соответствующие числам ax, суть соответственно

$$p_1, p_2, \ldots, p_n$$

– тоже пробегают приведенную систему вычетов, но в другом порядке (лемма 5.5). Значит, для любого i, $1 \le i \le n$, имеем $ar_i \equiv p_j$ для некоторого j, $1 \le j \le n$. Перемножим эти n сравнений. Получится

$$a^n r_1 r_2 \cdot \ldots \cdot r_n \equiv p_1, p_2 \cdot \ldots \cdot p_n \pmod m.$$

Так как $r_1 r_2 \cdot \ldots \cdot r_n = p_1, p_2 \cdot \ldots \cdot p_n \neq 0$ и взаимно просто с m, то, поделив последнее сравнение на $r_1 r_2 \cdot \ldots \cdot r_n$, получим $a^{\varphi(m)} \equiv 1 (\mathrm{mod}\, m)$. ♦

Частный случай этой теоремы принадлежит П. Ферма.

Теорема 5.7 (**малая теорема Ферма**). Пусть p – простое число и p не делит a. Тогда

$$a^{p-1} \equiv 1 \pmod p.$$

Если p делит a, то $a^{p-1} \equiv 0 \pmod p$. Но можно подправить формулировку теоремы Ферма, чтобы снять ограничение взаимной простоты.

Следствие 5.8. Для любого целого a и простого p верно

$$a^p \equiv a \pmod p.$$

Доказательство. Умножим обе части сравнения $a^{p-1} \equiv 1 \pmod p$ на a. Ясно, что получится сравнение, справедливое и при a, кратном p. ♦

Из малой теоремы Ферма мы получаем еще один алгоритм вычисления обратного числа по простому модулю.

Следствие 5.9. Пусть p – простое число и p не делит a. Тогда

$$a^{-1} = a^{p-2} \bmod p. \tag{5}$$

Доказательство. По малой теореме Ферма $a^{p-1} \equiv 1 \pmod p$. Т.е. $a \cdot a^{p-2} \equiv 1 \pmod p$, поэтому a^{p-2} есть обратное число для a по модулю p. ♦

Таким образом, для нахождения обратного элемента по простому модулю, мы, кроме функций inversion и Solve (см. §3), также можем воспользоваться отношением (5).

a = 77 198 236 511; p = Prime[10^9]

22 801 763 489

x = PowerMod[a, p –2, p]

19 517 671 688

Mod[a x, p] == 1

True

§ 6. Квадратичные вычеты

Существование решения уравнения $x^2 = a$ определяется кольцом, в котором мы работаем. Для любого комплексного числа a решение существует. Вещественное число является квадратом, когда оно неотрицательное. Мы будем рассматривать эти уравнения в кольце классов вычетов по модулю m.

6.1. Пусть $p > 2$ – простое число. Назовем элемент $a \in \mathbb{Z}$ **квадратичным вычетом по модулю *p***, если a является квадратом в $\mathbb{Z}_p$, т.е. если существует такой $x \in \mathbb{Z}$, что $x^2 \equiv a \pmod p$. В противном случае a называется **квадратичным невычетом по модулю p**.

Приведем список квадратичных вычетов по модулю 7: 1, 2, 4. Вот все квадратичные вычеты по модулю 17: 1, 2, 4, 8, 9, 13, 15, 16.

Теорема6.2. Если ненулевое a является квадратичным вычетом по модулю p, то сравнение $x^2 \equiv a \pmod p$ имеет точно два решения.

Доказательство. Действительно, пусть x_1 – решение сравнения, тогда $x_2 = -x_1$ – тоже решение, ведь $(x_1)^2 = (-x_1)^2$. Эти два решения не сравнимы по модулю p, так как из $x_1 \equiv -x_1 \pmod p$ следует, что $2x_1 \equiv 0 \pmod p$, т.е. $x_1 \equiv 0 \pmod p$, что невозможно, поскольку $a \neq 0$.

Предположим сравнение имеет третье решение x_3. Тогда $(x_3)^2 \equiv (x_1)^2 \pmod p$, так что $p \mid (x_3)^2 - (x_1)^2$ и, следовательно, $x_3 \equiv x_1 \pmod p$ или $x_3 \equiv -x_1 \equiv x_2 \pmod p$. Т.е. на самом деле все равно имеется два решения. ♦

Теорема 6.3. Для нечетного простого p существует ровно $(p-1)/2$ квадратичных вычетов и $(p-1)/2$ квадратичных невычетов.

Доказательство. Из $(p-x)^2 \equiv p^2 - 2ap + a^2 \pmod p$ следует, что $x^2 \equiv (p-x)^2 \pmod p$, поэтому первая половина квадратов $1^2, 2^2, \ldots, (p-1)^2$ дает при делении на p те же остатки,

что и – последняя. Отсюда следует, что квадратичных вычетов не может быть более $(p-1)/2$. Далее, числа $1^2, 2^2,\ldots, ((p-1)/2)^2$ все различны по модулю. Действительно, при любых $a, b \in \{1^2, 2^2,\ldots, ((p-1)/2)^2\}$ разность $a^2 - b^2$ не может делиться на p, так как $a^2 - b^2 = (a+b)(a-b)$ и оба сомножителя меньше p. Поэтому остатки от деления чисел $1^2, 2^2,\ldots, ((p-1)/2)^2$ на p дают $(p-1)/2$ вычетов по модулю p. Оставшиеся $(p-1)/2$ числа из $1, 2,\ldots, p-1$ будут невычетами. ♦

6.4. Если p – нечетное простое число, то **символ Лежандра** $\left(\frac{a}{p}\right)$ определяется как

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{если } a \equiv 0 (\mathrm{mod}\ p), \\ 1, & \text{если } a - \text{квадратичный вычет } (\mathrm{mod}\ p), \\ -1, & \text{если } a - \text{квадратичный невычет } (\mathrm{mod}\ p). \end{cases}$$

Таким образом, символ Лежандра показывает, является ли ненулевой вычет a по модулю p квадратом $(\mathrm{mod}\ p)$.

Теорема 6.5 (**Критерий Эйлера**). Пусть a не кратно p. Тогда

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

Доказательство. По малой теореме Ферма, $a^{p-1} \equiv 1 (\mathrm{mod}\ p)$ т.е.

$$(a^{\frac{p-1}{2}} - 1)(a^{\frac{p-1}{2}} + 1) \equiv 0 (\mathrm{mod}\ p).$$

В левой части последнего сравнения в точности один сомножитель делится на p, ведь оба сомножителя на p делиться не могут, иначе их разность, равная двум, делилась бы на $p > 2$. Следовательно, имеет место одно и только одно из сравнений:

$$a^{\frac{p-1}{2}} \equiv 1 (\mathrm{mod}\ p),$$

$$a^{\frac{p-1}{2}} \equiv -1 (\mathrm{mod}\ p).$$

Но всякий квадратичный вычет a удовлетворяет при некотором x сравнению $a \equiv x^2 \pmod p$ и, следовательно, удовлетворяет также получаемому из него почленным возведением в степень $(p-1)/2$ сравнению $a^{(p-1)/2} \equiv x^{p-1} \equiv 1 \pmod p$ (вновь применяем теорему Ферма). При этом квадратичными вычетами и исчерпываются все решения сравнения $a^{(p-1)/2} \equiv 1 \pmod p$, т.к., будучи сравнением степени $(p-1)/2$, оно не может иметь более $(p-1)/2$ решений (теорема 4.9). Это означает, что квадратичные невычеты удовлетворяют сравнению $a^{(p-1)/2} \equiv -1 \pmod p$. ♦

С символом Лежандра тесно связан, хотя и имеет некоторые существенные отличия, символ Якоби.

6.6. Пусть m – нечетное натуральное число (не обязательно простое), тогда для любого целого числа *а символ Якоби* $\left(\frac{a}{m}\right)$ определяется в терминах (единственного) разложения числа m на простые множители

$$m = \prod p_i^{t_i}$$

как

$$\left(\frac{a}{m}\right) = \prod \left(\frac{a}{p_i}\right)^{t_i},$$

где $\left(\frac{a}{p_i}\right)$ – символы Лежандра. Подразумевается, что $\left(\frac{a}{1}\right)=1$.

Сразу необходимо заметить, что для составных нечетных чисел m символ Якоби $\left(\frac{a}{m}\right)$ может иногда принимать значение +1 и в том случае, когда сравнение $x^2 \equiv a \pmod m$ не имеет решений. Например,

$$\left(\frac{2}{15}\right)=\left(\frac{2}{3}\right)\left(\frac{2}{5}\right)=(-1)(-1)=1,$$

хотя число 2 в действительности не является квадратом по модулю 15. Однако, если

$$\left(\frac{a}{m}\right)=-1,$$

то a и m взаимно просты и сравнение $x^2 \equiv a \pmod m$ не имеет решений. Наконец, символ Якоби равен 0 тогда и только тогда, когда $\gcd(a, m) > 1$.

Понятно, что символ Якоби можно вычислить. Разлагаем число m на простые множители, затем вычисляем соответствующие символы Лежандра, определяя разрешимость сравнения $x^2 \equiv a \pmod p$ путем полного перебора. Однако символы Лежандра и Якоби не применялись бы столь успешно, если бы их нельзя было вычислить совсем просто, без факторизации и тестов на простоту, и, конечно, без полного перебора. В следующей теореме собраны некоторые замечательные свойства символов Лежандра и Якоби – свойства, превращающие их вычисление в элементарную задачу, сопоставимую по сложности с нахождением gcd.

Теорема 6.7 (соотношения для символов Лежандра и Якоби). Пусть p обозначает нечетное простое число, m и n – произвольные натуральные нечетные числа (допускаются и простые), и a и b – произвольные целые числа. Тогда справедлив критерий Эйлера для квадратичного вычета по простому модулю

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

а также мультипликативные законы

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

$$\left(\frac{a}{mn}\right)=\left(\frac{a}{m}\right)\left(\frac{a}{n}\right)$$

и специальные соотношения

$$\left(\frac{-1}{m}\right) \equiv (-1)^{(m-1)/2},$$

$$\left(\frac{2}{m}\right) \equiv (-1)^{(m^2-1)/8}.$$

Кроме того, для взаимно простых m, n выполняется закон квадратичной взаимности:

$$\left(\frac{m}{n}\right)\left(\frac{n}{m}\right)=(-1)^{(m-1)(n-1)/4}.$$

Из критерия Эйлера следует [28, стр. 118], что сложность вычисления символа Лежандра не превосходит $O(\ln^3 p)$. Но вычисления символа Якоби (значение которого в слу-

чае простого m совпадает со значение символа Лежандра) можно выполнить с помощью алгоритма со сложностью $O(\ln^2 m)$.

Алгоритм 6.8 (вычисление символа Лежандра / Якоби) [28, стр. 118] . Вход: нечетное целое m и целое a. Выход: символ Якоби $\left(\frac{a}{m}\right)$, который для простых нечетных m совпадает также с символом Лежандра.

```
a = a mod m;
t = 1;
while a ≠ 0 do
      begin
      while a – четно do
            begin
            a = a/2;
            If m mod 8 ∈ {3, 5} then t =  – t;
            end;
      (a, m) = (m, a);
      If a ≡ m ≡ 3  (mod 4) then t = – t;
      a = a mod m
      end;
if m == 1 return(t) else return(0)
```

Этот алгоритм в Mathematica можно реализовать в виде функции jacobiSymbol.

```
jacobiSymbol[a_,  m_] :=
     Module[{b, n, t},
       {b, n, t} = {Mod[a, m], m, 1};
       While[b ≠ 0,
          If[(b > n / 2) && OddQ[b], b = n – b; If[Mod[n, 4] == 3, t = –t]];
          While[EvenQ[b], b = b / 2;
                              If[MemberQ[{3, 5}, Mod[n, 8]], t = –t]
               ];
          {b, n} = {n, b};
          If[Mod[b, 4] == Mod[n, 4] == 3, t = –t];
          b = Mod[b, n]
             ];
        If[n == 1, t, 0]
             ]
```

В Mathematica имеется встроенная функция JacobiSymbol для вычисления символа Якоби. Возможно, ее алгоритм аналогичен алгоритму 6.8, но сравнение работы по времени показывает преимущество встроенной функции JacobiSymbol.

Мы видели, что для нечетного простого p и a – не кратного p, разрешимость сравнения $x^2 \equiv a \pmod p$ определяется значением символа Лежандра $\left(\frac{a}{p}\right)$. Если $\left(\frac{a}{p}\right) = 1$, важной задачей является поиск «квадратных корней» из a, которых будет два, каждый из них будет равен другому со знаком «минус» (по модулю p).

Опишем алгоритм извлечения квадратного по модулю простого числа p. Этот алгоритм достаточный сложный и его обоснование приведено в [28, стр. 120–122]. Отметим только некоторые его особенности. Во-первых, он существенно опирается на критерий Эйлера. Во-вторых, в алгоритме происходит разбор случаев различных остатков $p \bmod 8$. И, в-третьих, с помощью вычисления символа Якоби находится некоторый квадратичный невычет по модулю p. Неизвестно ни одного строгого **детерминированного** метода для быстрого поиска квадратичного невычета. Поэтому используется **случайный** алгоритм.

Алгоритм 6.9 (квадратный корень по модулю p). Пусть заданы нечетное простое число p и целое число a, являющееся квадратом по модулю p. Этот алгоритм возвращает решение x сравнения $x^2 \equiv a \pmod p$.

1. (* Простейшие случаи: $p \equiv 3, 5, 7 \pmod 8$ *)
   $a = a \bmod p$;
   If $p \equiv 3, 7 \pmod 8$ then begin $x = a^{(p+1)/4} \bmod p$; return(x) end;
   If $p \equiv 5 \pmod 8$ then
      begin
      $x = a^{(p+3)/8} \bmod p$;
      $c = x^2 \bmod p$; (* Тогда $c = \pm a \pmod p$ *)
      If $c \neq a \pmod p$ then $x = x\, 2^{(p-1)/4} \bmod p$;
      return(x)
      end;
2. (* Случай: $p \equiv 1 \pmod 8$ *)
   (* Найти случайный квадратичный невычет d (mod p) *)
   Найти случайное целое $d \in [2, p-1]$, для которого $\left(\frac{a}{p}\right) = -1$;
   (* Вычислить символ Якоби, алгоритм 6.8 *)
   Представить $p - 1 = 2^s t$, где t нечетно;
   $A = a^t \bmod p$; $D = d^t \bmod p$;
   $m = 0$; $i = 0$;
   While $i < s$ do begin $n = 2^{s-1-i}$; If $(A\, D^m)^n \equiv -1 \pmod p$ then $m = m + 2^i$ end;
   (* Таким образом, $A\, D^m \equiv 1 \pmod p$ *)
   $x = a^{(t+1)/2} D^{m/2} \bmod p$;
   return(x)

Сложность алгоритма определяется большим количеством необходимых возведений в степень, и поэтому составляет $O(s^2 + \ln t)$ модулярных операций. В случае наивной реализации арифметических подпрограмм сложность алгоритма в наихудшем случае (когда s велико) составляет $O(\ln^4 p)$ битовых операций. Но в усредненном случае получаем всего лишь $O(\ln^3 p)$ битовых операций [25, стр. 122].

Вычисление квадратного корня по простому модулю запрограммируем в Mathematica в виде функции

```
sqrtMo[p_, a_] :=
 Module[{b = Mod[p, 8], a1, x, c, d, d1, s, t, bo, at, dt, m, i},
   Which[b == 3 || b == 7,
          a1 = Mod[a, p]; PowerMod[a1, (p + 1) / 4, p],
         b == 5,
          a1 = Mod[a, p]; x = PowerMod[a1, (p + 3)/ 8, p];
          c = PowerMod[x, 2, p];
          If[c ≠ a1, Mod[x × PowerMod[2, (p – 1) / 4, p], p], x],
         b == 1,
          bo = True;
          While[bo, d = Random[Integer, {2, p – 1}];
                    (* Случайный квадратичный невычет d (mod p) *)
                    If[JacobiSymbol[d, p] == –1, bo = False]
               ];
          s = 0; t = p – 1;
          While[Mod[t, 2] == 0, s = s + 1; t = t / 2];
          (* Представить p – 1 = t × 2^s, где  t – нечетно *)
          at = PowerMod[a, t, p]; dt = PowerMod[d, t, p]; m = 0;
          For[i = 0, i < s, i ++,
            If[PowerMod[at × PowerMod[dt, m, p], 2^(s – 1 – i), p] == p – 1,
                                                          m = m + 2^i
              ]
              ];
          Mod[PowerMod[a, (t + 1) / 2, p] PowerMod[dt, m / 2, p], p]
         ]]
```

Пример 6.10. Вычисление квадратного корня. Возьмем наибольшее простое число, известное на начало 1951 года [33, стр. 46]:

p = 180(2^127 – 1)^2 + 1;

5 210 644 015 679 228 794 060 694 325 390 955 853 335 898 483 908 056 458 352 183 851 018 372 555 735 221

PrimeQ[p]

True

Найдем случайный квадратичный вычет a по модулю p в диапазоне $(p/2, p-1]$:

b = –1; While[b ≠ 1, a = RandomInteger[{⌈p/2⌉, p-1}]; b = JacobiSymbol[a, p]]; a

3 425 654 204 371 662 383 599 633 294 133 352 754 175 980 037 420 540 522 506 418 606 890 419 463070 787

JacobiSymbol[a, p]

1

sqrtMod[p, a]

3 230 641 831 664 651 200 054 008 126 717 036 128 945 641 609 021 119 819 993 323 728 861 475 600216 733

Пример 6.11. С помощью функции Solve в Mathematica можно извлечь и квадратный корень по модулю:

p = 180(2^127 – 1)^2 + 1;

Найдем случайный квадратичный вычет a по модулю p в диапазоне $(p/2, p-1]$:

b = –1; While[b ≠ 1, a = RandomInteger[{⌈p/2⌉, p-1}]; b = JacobiSymbol[a, p]]; a

3 310 827 153 276 843 693 465 864 058 346 379 698 721 003 376 032 123 416 099 990 356 421 740 322 988 130

Применяя функцию Solve, получаем два правила для нахождения корней:

rs = Solve[x^2 == a, Modulus → p];

И осуществляя подстановку по этим правилам, получаем два квадратных корня:

x /. rs

{1 745 881 723 062 704 116 132 284 001 035 314 231 594 859 894 983 76 879 866 951 304 156 361 214 860 675,

3 464 762 292 616 524 677 928 410 324 355 641 621 741 038 588 924 279 578 485 232 546 862 011 340 874 546}

Задачи и упражнения

1. Докажите теорему 1.5.

2. Докажите лемму 4.5.

3. Признак делимости на 9: число n делится на 9 тогда и только тогда, когда на 9 делится сумма цифр числа n. Докажите этот признак, используя сравнение $10 \equiv 1 \pmod 9$.

4. Признак делимости на 11: число n делится на 11 тогда и только тогда, когда на 11 делится его сумма цифр, стоящих на четных местах, минус сумма цифр на нечетных местах. Докажите этот признак, используя сравнение $10^k \equiv (-1)^k \pmod{11}$.

5. Пусть $\sigma(n)$ – сумма всех делителей целого числа n. С помощью Mathematica выскажите гипотезу, для каких натуральных чисел n имеем $\sigma(n) + \varphi(n) = 2n$.

6. Число x при делении на 2002 дает остаток 2001, а при делении на 2001 – остаток 1901. Какой остаток дает x при делении на $2002 \cdot 2001$?

Глава 7. Делимость многочленов над факториальными кольцами

Если мы ограничимся рассмотрением многочленов над полем, то не охватим многочлены над множеством целых чисел и многочлены от нескольких переменных. Поэтому мы рассмотрим более общую ситуацию, когда коэффициенты многочленов принадлежат факториальным кольцам.

§ 1. Псевдоделение многочленов

1.1. Многочлен над факториальным кольцом называется **примитивным**, если его коэффициенты взаимно просты.

Лемма 1.2 (Гаусс). Произведение примитивных многочленов над факториальным кольцом R является примитивным многочленом.

Доказательство. Пусть $u(x) = u_m x^m + \cdots + u_0$ и $v(x) = v_n x^n + \cdots + v_0$ – примитивные многочлены из $R[x]$. Пусть p – любой простой элемент из R; тогда следует показать, что p не делит все коэффициенты $u(x)v(x)$. По условию имеется такой индекс j, что u_j не делится на p, и такой индекс k, что v_k не делится на p. Пусть j и k – наименьшие такие индексы; тогда коэффициент при x^{j+k} в $u(x)v(x)$ равен

$$u_j v_k + u_{j+1}v_{k-1} + \cdots + u_{j+k}v_0 + u_{j-1}v_{k+1} + \cdots + u_0 v_{k+j}.$$

Так как первый член этой суммы не делится на p, а все остальные делятся, на p не делится вся сумма. ♦

Если ненулевой многочлен $u(x)$ над факториальным кольцом R не примитивен, можно записать $u(x) = p_1 \cdot u_1(x)$, где простой элемент из R, делящий все коэффициенты $u(x)$, а $u_1(x)$ – другой ненулевой элемент над R. Все коэффициенты $u_1(x)$ имеют на один простой множитель меньше соответствующих коэффициентов $u(x)$. Теперь, если $u_1(x)$ не примитивен, можно записать $u_1(x) = p_2 \cdot u_2(x)$ и т.д. Этот процесс прекратится при представлении $u(x) = c \cdot u_k(x)$, где $c \in R$ и $u_k(x)$ – примитивен.

Фактически имеется соответствующая лемме 1.2 лемма 1.3.

Лемма 1.3. Любой ненулевой многочлен $u(x)$ над факториальным кольцом R можно разложить на множители в виде $u(x) = c \cdot v(x)$, где c представляет собой элемент R, а $v(x)$ – примитивный многочлен. Кроме того, это представление единственно в том смысле, что если $u = c_1 \cdot v_1(x) = c_2 \cdot v_2(x)$, то $c_1 = a \cdot c_2$ и $v_2(x) = a \cdot v_1(x)$, где a – обратимый элемент R.

Доказательство. Мы уже показали, что такое разложение существует, так что доказательства требует только единственность разложения. Положим, что $c_1 \cdot v_1(x) = c_2 \cdot v_2(x)$, где $v_1(x)$, $v_2(x)$ – примитивные многочлены. Пусть p – некоторый простой элемент из R. Если p^k делит c_1, то оно делит и c_2; в противном случае p^k должно делить все коэффициенты $c_2 \cdot v_2(x)$ и p должно делить все коэффициенты $v_2(x)$. Таким образом, получается противоречие с посылкой примитивности $v_2(x)$. Точно так же p^k делит c_2 только в том случае, если p^k делит c_1. Следовательно, в силу единственности разложения $c_1 = a\, c_2$, где a обратимо; поскольку $0 = ac_2 \cdot v_1(x) - c_2 \cdot v_2(x) = c_2(a \cdot v_1(x) - v_2(x))$, получаем $a \cdot v_1(x) - v_2(x) = 0$. ♦

Таким образом, можно записать любой ненулевой многочлен $u(x)$ в виде

$$u(x) = \mathrm{cont}(u) \cdot \mathrm{pp}(u(x)),$$

где $\mathrm{cont}(u)$ – **содержание (content)** u, представляющий собой элемент R, а $\mathrm{pp}(u(x))$ – **примитивная часть (primitive part)**, являющаяся примитивным многочленом над R. В случае, когда $u(x) = 0$, удобно определить $\mathrm{cont}(u) = \mathrm{pp}(u(x)) = 0$. Комбинация лемм 1.2 и 1.3 приводит к соотношениям

$$\text{cont}(u \cdot v) = a\ \text{cont}(u)\ \text{cont}(v), \qquad (1)$$
$$\text{pp}(u(x) \cdot v(x)) = b\ \text{pp}(u(x))\ \text{pp}(v(x)),$$

где a и b – обратимые элементы, зависящие от пути вычисления содержимого, причем $ab = 1$. При работе с многочленами над целыми числами обратимыми элементами являются только +1 и –1, и поэтому удобно определить $\text{pp}(u(x))$ так, чтобы ее старший коэффициент был положителен; тогда соотношения (1) истинны при $a = b = 1$. При работе с многочленами над полем можно принять $\text{cont}(u(x)) = \text{lc}(u)$, так что $\text{pp}(u(x))$ будет нормированным многочленом; в этом случае (1) вновь выполняется при $a = b = 1$ для всех $u(x)$ и $v(x)$.

Пример 1.4. Рассмотрим многочлены из кольца $Z[x]$: $u(x) = -15x^4 + 60x^2 - 30$ и $v(x) = 4x - 4$. Тогда

$$\text{cont}(u) = -15,\ \text{pp}(u(x)) = x^4 - 4x^2 + 2,$$
$$\text{cont}(v) = 4,\ \text{pp}(v(x)) = x - 1,$$
$$u(x) \cdot v(x) = -60x^5 + 60x^4 + 240x^3 - 240x^2 - 120x + 120,$$
$$\text{cont}(u \cdot v) = -60,\ \text{pp}(u(x) \cdot v(x)) = x^5 - x^4 - 4x^3 + 4x^2 + 2x - 2.$$

Если коэффициенты многочленов не принадлежат полю, то старший коэффициент $\text{lc}(u)$ многочлена u может не делиться на старший коэффициент $\text{lc}(v)$ многочлена v. Важными примерами таких колец являются кольца $\mathbb{Z}[x]$ и кольцо многочленов от нескольких переменных. Фактически, здесь нет частного и остатка, удовлетворяющего условию

$$u(x) = q(x)\ v(x) + r(x) \quad \text{и} \quad \deg(r) \le \deg(v) \qquad (2)$$

Однако, соотношения (2) будут выполнены, если мы нормализуем многочлен u с помощью соответствующей степени старшего коэффициента $\text{lc}(v)$ многочлена v.

Теорема 1.5 [25, стр. 447]. Пусть R – целостное кольцо, $u(x), v(x) \in R[x]$, $v(x) \neq 0$, и $m = \deg(u) \ge n = \deg(v)$. Тогда существуют единственные многочлены $q(x), r(x) \in R[x]$ такие, что

$$\text{lc}(v)^{m-n+1} \cdot u(x) = q(x)\ v(x) + r(x) \quad \text{и} \quad \deg(r) \le \deg(v). \qquad (3)$$

Поэтому вводят **алгоритм псевдоделения** $u(x)$ на $v(x)$, который всё же позволяет получить **псевдочастное** $q(x)$ и **псевдоостаток** $r(x)$, которые будут сами по себе принадлежать множеству многочленов над кольцом R. Для данных

$$u(x) = u_m x^m + \cdots + u_0 \quad \text{и} \quad v(x) = v_n x^n + \cdots + v_0$$

многочлены $q(x)$ и $r(\text{x})$ – единственны, и поэтому их можно найти алгоритмом евклидова деления в $R[x]$ многочлена $v_n^{m-n+1}u(x)$ на многочлен $v(x)$.

Пример 1.6. Программа в Mathematica для псевдоделения в $\mathbb{Z}[x]$.

Функция lc находит старший коэффициент многочлена:

```
lc[p_, x_] := Coefficient[p, x, Exponent[p, x]]
```

Встроенная функция Exponent[p, x] определяет наибольшую степень переменной x в многочлене p. Встроенная функция Coefficient[p, x, n] определяет в многочлене p коээфициент при x^n.

Функции pquot и prem находят псевдочастное и псевдоостаток:

```
pquot[f_, g_, x_] := Module[{lc = lc[g,x]},
          PolynomialQuotient[lc^(Exponent[f, x] – Exponent[g, x] + 1)f, g, x]];

prem[f_, g_, x_] := Module[{lc = lc[g, x]},
         PolynomialRemainder[lc^(Exponent[f, x] – Exponent[g, x] + 1)f, g, x]];

u = x^8 + x^6 – 3x^4 – 3x^3 + 8x^2 + 2x – 5;

v = 3x^6 + 5x^4 – 2x^2 – 9x + 21;

pquot[u, v, x]
```

```
-6 + 9 x^2
```

prem[u, v, x]

```
-9 + 3 x^2 - 15 x^4
```

§ 2. Наибольший общий делитель в факториальных кольцах

Перейдем теперь к наибольшим общим делителям в факториальных кольцах. Предположим теперь, что R – факториальное кольцо. В этом случае неприводимые элементы и простые совпадают. Следствием разложимости любого элемента на простые множители являются существование наибольших общих делителей в факториальных кольцах. Обозначим через P множество простых элементов в R такое, что всякий простой элемент из R ассоциирован с одним и только одним элементом из P. Рассматривая разложения двух элементов $a, b \in R$ удобно считать, что в них входят одинаковые элементы из P, но некоторые, возможно, с нулевыми показателями, т.е.

$$\begin{aligned} &a = up_1^{k_1}\ldots p_r^{k_r}, \qquad b = up_1^{l_1}\ldots p_r^{l_r}, \\ &u \mid 1, v \mid 1;\ k_i \geq 0, l_i \geq 0;\ \ p_1 \in P;\ 1 \leq i \leq r. \end{aligned} \tag{4}$$

Любой простой элемент $p \in R$, делящийся произведение $ab \in R$ делит, по крайней мере, один из сомножителей a, b. Отсюда получается

Теорема 2.1. Пусть a, b – ненулевые элементы факториального кольца R, записанные в виде (4).

Справедливы утверждения:

$$\begin{aligned} &1)\ a|b \text{ тогда и только тогда, когда } k_i \leq l_i,\ i = 1, 2, \ldots, r; \\ &2)\ \gcd(a,b) = p_1^{s_1}\ldots p_r^{s_r}, \text{ где } s_i = \min\{k_i, l_i\}, i = 1,2,...,r; \\ &3)\ \operatorname{lcm}(a,b) = p_1^{t_1}\ldots p_r^{t_r}, \text{ где } t_i = \max\{k_i, l_i\}, i = 1,2,...,r. \end{aligned} \tag{5}$$

Таким образом, в качестве s_i нужно брать наименьший из двух показателей, а в качестве t_i – наибольший. В частности, элементы a, $b \in R$ взаимно просты в точности тогда, когда простые множители, входящие в разложение одного элемента, не входят в разложение другого.

Так как факториальное кольцо может иметь много обратимых элементов, то в определении наибольшего общего делителя присутствует некоторая неоднозначность. Если $d = \gcd(a, b)$, то εd, где ε – обратимый элемент, тоже является наибольшим общим делителем. Другими словами, не имеет смысла говорить о конкретном наибольшем делителе a и b. Существует целое множество наибольших общих делителей, каждый из которых отличается от других на обратимый множитель.

Поскольку кольцо $\mathbb{Z}[x]$ факториально (глава 4, теорема 3.10), то неприводимые элементы и простые элементы в $\mathbb{Z}[x]$ совпадают, и их принято называть **неприводимыми многочленами**.

Кроме того, наибольший общий делитель двух многочленов над $\mathbb{Z}$ есть общий делитель, который делится на наибольшее количество неприводимых многочленов (см. (5)). В кольце $\mathbb{Z}$ обратимыми элементами являются 1 и –1 и из двух асоцированных наибольших общих делителей обычно выбирают многочлен с положительным старшим коэффициентом.

Из соотношений (1) получаем важные соотношения

$$\begin{aligned} \operatorname{cont}(\gcd(u, v)) &= a\cdot\gcd(\operatorname{cont}(u), \operatorname{cont}(v)), \\ \operatorname{pp}(\gcd(u(x), v(x))) &= b\cdot\gcd(\operatorname{pp}(u(x)), \operatorname{pp}(v(x))), \end{aligned} \tag{6}$$

где a и b – обратимые элементы. Здесь $\gcd(u(x), v(x))$ обозначает любой из многочленов $u(x)$ и $v(x)$. Соотношения (6) позволяют свести задачу нахождения наибольших общих де-

лителей произвольных многочленов к задаче нахождения наибольших общих делителей примитивных многочленов. Ясно, что cont(u) – один из наибольших общих делителей коэффициентов многочлена u, а pp($u(x)$) = $u(x)$/cont(u). Используя алгоритм псевдоделения можно следующим образом построить «обобщенный алгоритм Евклида» для примитивных многочленов над факториальным кольцом. Пусть $u(x)$ и $v(x)$ – примитивные многочлены с $m = \deg(u) \geq \deg(v) = n$; при помощи алгоритма псевдоделения определим многочлен $r(x)$, удовлетворяющий (3). Имеем $\gcd(u(x), v(x)) = \gcd(v(x), r(x))$, ибо всякий общий делитель многочленов $u(x)$ и $v(x)$ делит $v(x)$ и $r(x)$; обратно, всякий общий делитель многочленов $v(x)$ и $r(x)$ делит $\mathrm{lc}(v)^{m-n+1}u(x)$, и этот общий делитель должен быть примитивны (поскольку $v(x)$ примитивен), значит, он делит $u(x)$. Если $r(x) = 0$, то $\gcd(u(x), v(x)) = v(x)$; если $r(x) \neq 0$, то в силу примитивности $v(x)$ мы имеем $\gcd(v(x), r(x)) = \gcd(v(x), \mathrm{pp}(r(x)))$ и, следовательно, этот процесс можно продолжить.

Алгоритм 2.2 Евклида для нахождения gcd многочленов с псевдоделением. По данным ненулевым многочленам $u(x)$ и $v(x)$ над факториальным кольцом R этот алгоритм вычисляет один из наибольших общих делителей многочленов $u(x)$ и $v(x)$. Мы считаем, что имеются вспомогательные алгоритмы для вычисления наибольших общих делителей элементов из R и для деления a на b в R, когда $b \neq 0$ и a кратно b.

1. (* Свести к примитивным.*) Применяя упомянутый вспомогательный алгоритм для вычисления gcd в R, вычислить d = gcd(cont(u), cont(v)). Заменить $u(x)$ многочленом pp($u(x)$); подобным же образом заменить $v(x)$ на pp($v(x)$).
2. (* Псевдоделение. *) Используя алгоритм псевдоделения, вычислить $r(x)$. Если $r(x)$ равен 0, то перейти на шаг 4. Если deg(r) равен 0, то заменить $v(x)$ на единичный многочлен 1 и перейти на шаг 4.
3. (* Сделать остаток примитивным.*) Заменить $u(x)$ на $v(x)$ и заменить $v(x)$ на pp($r(x)$). Перейти на шаг 2.
4. (* Присоединить содержание.*) return(d $v(x)$).

Программа в Mathematica

Функции prem и lc – ранее определенные функции (пример 1.6) для вычисления псевдоостатка и старшего коэффициента соответственно.

Функция cont вычисляет содержание многочлена:

```
cont[p_, x_] := GCD@@CoefficientList[p, x]
```

Функция pp вычисляет примитивную часть многочлена:

```
pp[p_, x_] := Expand[p / cont[p, x]]
```

Функция eap вычисляет наибольший общий делитель двух многочленов в $\mathbb{Z}[x]$, не используя операции с рациональными числами:

```
eap[f_, g_, x_] :=
  Module[{fc = cont[f, x], gc = cont[g, x], d, u, v, r, b = True},
      d = GCD[fc, gc]; u = Expand[f / fc]; v = Expand[g / gc];
      If[Exponent[u, x] < Exponent[v, x], {u, v} = {v, u}];
      While[b, r = prem[u, v, x];
            If[Not[SameQ[r, 0]], r = pp[r, x]; {u, v} = {v, r}, b = False]
        ];
      If[Exponent[v, x] == 0, d, Expand[d pp[v, x]]]
    ]
```

Пример вычисления gcd:

```
u = x^8 + 5x^7 + 7x^6 – 3x^5 + 4x^4 + 17x^3 – 2x^2 – 6x + 3;
v = x^8 + 6x^7 + 3x^6 + x^5 + 10x^4 + 8x^3 + 2x^2 + 9x + 8;

eap[u, v, x]
–1 – x
```

– Хотелось бы более четких инструкций.
– Полной ясности не будет никогда.
Не жди ее, привыкай действовать в
условиях частичной неопределенности.
В. Серкин. Хохот шамана

Глава 8. Распознавание простых и составных чисел

§ 1. Псевдопростые числа Ферма

Малая теорема Ферма (глава 6, теорема 5.7) дает необходимое условие для простоты натурального числа. Таким образом, мы имеем

Тест на разложимость 1.1. Пусть n – нечетное натуральное число. Если найдется такое целое число a, что
(1) $1 < a < n - 1$, и
(2) a^{n-1} не сравнимо с 1 по модулю n,
то n – составное число.

Пусть $k = (10^{3571}-1)/9$, т.е. десятичная запись k состоит из 3571 единицы (см. упр. 1). Проверим, является ли это число составным.

```
(PowerMod[2, # – 1, #] ≠ 1)& [(10^3571 – 1) / 9]
```

True

1.2. Составное число n называется **псевдопростым числом Ферма** по основанию a, $1 < a < n - 1$, если выполнено сравнение

$$a^n \equiv a \pmod{n}. \qquad (1)$$

В пределах первой тысячи псевдопростыми числами Ферма по основанию 2 являются 341 = 11 · 31, 561 = 3 · 11 · 17 и 645 = 3·5·43. По основанию 3 псевдопростыми числами, меньшими 1000, являются 91 = 7·13, 121 = 11·11, 561 = 3·11·17, 671 = 11·61, 703 = 19·37 и 949 = 13·73.

Теорема 1.3 [28, стр. 154]. Для каждого фиксированного основания $a \geq 2$ количество псевдопростых чисел Ферма по основанию a, которые не превосходят x, есть $o(\pi(x))$ при $x \to \infty$ ($\pi(x)$ – количество простых чисел, меньших x; $f(x) = o(g(x))$ при $x \to \infty$ означает, по определению, что предел $f(x) / g(x)$ равен нулю при $x \to \infty$). Иными словами, псевдопростые числа Ферма редки по сравнению с простыми числами.

1.4. Если для пары натуральных чисел n, a, где n – нечетное, $1 < a < n - 1$, выполняется сравнение

$$a^{n-1} \equiv 1 \pmod{n}, \qquad (2)$$

мы будем говорить, что n – **вероятное простое число по основанию *a***. Таким образом, если число n – простое, то оно является вероятным простым по всем основаниям a. Теорема 1.3 утверждает, что при фиксированном выборе основания a наиболее вероятные простые числа по основанию a скорее всего являются простыми.

Алгоритм 1.5 (тест для вероятного простого числа). Нам даны целые числа $n > 3$ и a, причем $2 \leq a \leq n - 2$. Алгоритм выдает или ответ «n – вероятное простое число по основанию a», или ответ «n – составное число».

1. (* Вычислить степенной вычет *)
$b = a^{n-1} \bmod n$.
2. (* Вынести решение *)
if b == 1 then return(«n – вероятное простое число по основанию a»)
else return(«n – составное число»)

Как сказано выше, вероятные простые числа, являющиеся составными достаточно редки. Однако это не мешает существованию бесконечного множества таких чисел.

Теорема 1.6 [28, стр. 155]. Для каждого целого числа $a \geq 2$ существует бесконечное множество псевдопростых чисел Ферма по основанию a.

§ 2. Числа Кармайкла

Рассмотрим натуральные числа, не превосходящие 10^9. Оказывается среди них лежит 50847544 простых и только 5597 псевдопростых по основанию 2. Кроме того, мы использовали только одно основание, а если использовать несколько разных оснований, то количество неопределяемых составных чисел значительно уменьшится. На самом деле, между 1 и 10^9 находится 1272 псевдопростых по основанию 2 и 3 и только 685 псевдопростых по основанию 2, 3 и 5. Возможно, существует конечное множество оснований такое, для которого не найдется псевдопростых чисел по всем основаниям этого множества. Если такое множество существует, то из вероятностного алгоритма 1.5 мы бы получили детерминированный алгоритм распознавания простых чисел.

К сожалению, наши надежды безосновательны. Число 561 = 3·11·17 является не только псевдопростым числом Ферма по основаниям 2 и 3, но оно псевдопростое по всем основаниям a. Кажется невероятным, что такие числа существуют, но, и в самом деле это так. Впервые они были найдены Р. Кармайклом в 1910 году и с тех пор такие числа носят имя этого математика.

2.1. Составное число n, для которого при всех натуральных числах a выполнено сравнение (1), называется **числом Кармайкла**.

Числа Кармайкла легко распознать по разложению на простые множители.

Теорема 2.2 (критерий Корсельта, [28, стр. 156]). Натуральное число n является числом Кармайкла тогда и только тогда, когда оно 1) составное, 2) не делится ни на какой квадрат и 3) для каждого простого числа p, делящего n, число $p - 1$ делит $n - 1$.

Следующая программа на языке Wolfram реализует критерий Корсельта.

```
carmichael[n_] :=
Module[{fs, ks, ps},
  Which[PrimeQ[n], False,
    fs = FactorInteger[n]; ks = Cases[fs, {p_, k_} → k]; (ks /. List → Times) > 1,
                              False,
    ps = Cases[fs, {p_, k_} → p];  Map[Divisible[n – 1, # – 1]&, ps] /. List → And,
                              True,
    True, False]]
```

Числа 561, 1105, 1729, 2465, 2821, 6601, 8911 являются числами Кармайкла. Это легко проверить:

```
Map[Carmichael, {561, 1105, 1729, 2465, 2821, 6601, 8911}]
```

{True, True, True, True, True, True, True}

Обозначим через $C(n)$ количество чисел Кармайкла в диапазоне от 1 до n. Доказано [15], что, начиная с достаточно большого n, выполнено двойное неравенство

$$n^{2/7} < C(n) < n \exp\left(-\frac{\ln n \ln\ln\ln n}{\ln\ln n}\right). \qquad (3)$$

Поскольку нижняя оценка в (3) – возрастающая функция, то отсюда следует, что чисел Кармайкла бесконечно много.

§ 3. Сильно вероятные простые числа

В этом параграфе опишем тест на простоту, которому удовлетворяют не все псевдопростые числа Ферма. Иными словами, новый алгоритм будет более эффективным, чем алгоритм 1.5.

Мы будем использовать иной вариант малой теоремы Ферма.

Теорема 3.1. Пусть n – нечетное простое число и $n - 1 = 2^s t$, где число t нечетно. Если число a не делится на n, то

$$\begin{cases} \text{или } a^t \equiv 1 (\mathrm{mod}\, n), \\ \text{или } a^{2^i t} \equiv -1 (\mathrm{mod}\, n) \quad \text{для некоторого } i, \text{где } 0 \le i \le s-1. \end{cases} \tag{4}$$

Доказательство. Если $s = 0$, то первый случай $a^t \equiv 1 \pmod n$ – это просто утверждение малой теоремы Ферма. Если $s > 0$, то рассмотрим вычеты по модулю n у следующей последовательности степеней:

$$a^t, a^{2t}, ..., a^{2^{s-1}t}, a^{2^s t}.$$

Разберемся, какими свойствами обладает эта последовательность. Теорема Ферма говорит нам, что

$$a^{2^s t} \equiv a^{n-1} \equiv 1 (\mathrm{mod}\, n).$$

Значит, последний вычет в последовательности всегда равен 1. Конечно, единица среди вычетов может встретиться и раньше. Пусть k – наименьший показатель, для которого

$$a^{2^k t} \equiv 1 (\mathrm{mod}\, n).$$

Тогда имеем

$$a^{2^k t} - 1 = (a^{2^{k-1} t} - 1)(a^{2^{k-1} t} + 1).$$

Число n – простое, и поэтому если n делит левую часть последнего равенства, то n делит один из сомножителей правой части (сумму или разность). Но в силу выбора показателя k, число n не может делить разность. Остается только одна возможность

$$n \mid a^{2^{k-1} t} + 1, \text{ т.е. } \quad a^{2^{k-1} t} \equiv -1 (\mathrm{mod}\, n).$$

Что и требовалось доказать. ♦

По аналогии с вероятными простыми числами дадим следующее определение.

3.2. Нечетное число $n > 3$, для которого (4) имеет место для всех a из промежутка $1 < a < n - 1$, называется **сильно вероятное простое число**.

Поскольку каждое сильно вероятное псевдопростое число по основанию a автоматически является и вероятным простым числом по основанию a, и поскольку каждое простое число, превосходящее $a + 1$, является сильно вероятным простым числом по основанию a, единственное отличие этих понятий заключено в том, что тест на сильно вероятное простое число пройдет меньше составных чисел.

Алгоритм 3.3 (тест на сильно вероятное простое число). Нам дано нечетное число $n > 3$ в виде $n = 1 + 2^s t$, где число t нечетно. Кроме того, нам задано целое число a из промежутка $1 < a < n - 1$. Алгоритм выдает или ответ «n – сильно вероятное простое число по основанию a», или ответ «n – составное число».

1. (* Нечетная часть числа $n - 1$ *)
$b = a^t \bmod n$; (* используем С.2.1 *)
If (b == 1) or (b == $n - 1$)

then return(«*n* – сильно вероятное простое число по основанию *a*»);
2. (* Степень 2 в числе *n* – 1 *)
k = 1; (* *k* – просто счетчик *)
While k ≤ n – 1 do
begin
b = b^2 mod n;
If *b* == *n* – 1 then return(«*n* — сильно вероятное простое число по основанию *a*»);
k = *k* + 1;
end;
return(«*n* – составное число»)

Соответствующая программа:

```
highlyProbablePrimeQ[n_, a_] :=
 Module[{t, s, b, k},
   s = 0; t = n – 1; While[Mod[t, 2] == 0, s = s + 1; t = t / 2];
   (* Представили n – 1 = t × 2^s, где t – нечетно *)
   b = PowerMod[a, t, n];
   If[b == 1 || b == n – 1,
        Return[{n ,"– сильно вероятное простое число по основанию ", a}]];
   For[k = 0, k < n, k ++, b = Mod[b^2, n];
    If[b == n – 1,
        Return[{n ,"– сильно вероятное простое число по основанию ",a}]]];
   Return[{n , "– составное число"}]]
```

Проверим этот тест на числах Кармайкла:

```
highlyProbablePrimeQ[#, 3]& /@ {561, 1105, 1729, 2465, 2821, 6601, 8911}
```

{{561, – составное число}, {1105, – составное число}, {1729, – составное число},

{2465, – составное число}, {2821, – составное число}, {6601, – составное число},

{8911, – сильно вероятное простое число по основанию , 3}}

Если мы изменим основание, то и число 8911 будет распознано как составное.

```
highlyProbablePrimeQ[8911, 2]
```

{8911, – составное число}

Тест на сильно вероятное простое число является более эффективным, чем тест на вероятное простое число. Так из 5597 псевдопростых чисел по основанию 2, лежащих между 1 и 10^9 алгоритм 3.3 оставляет всего 1282 числа, не распознавая их как составные.

Более того, для алгоритма 3.3 не существуют чисел аналогичных Кармайкловским числам. Это следует из результата, независимо полученного в работах [6, 7].

Теорема 3.4 (доступное доказательство см. в [28, стр. 159, 162-163]). Пусть $n > 9$ – нечетное натуральное число. Если тест на сильно вероятное простое число, примененный к n для более чем $n/4$ оснований, лежащих между 1 и $n – 1$, не распознает в n составного числа, то n – простое.

Объясним значение этого результата. Если нам дано нечетное число n и требуется исследовать его на простоту, мы можем сначала проверить условие (4) для некоторого числа a из промежутка $1 < a < n – 1$. Если условие (4) не выполнено, то составность числа n доказана.

3.5. Пусть для нечетного составного числа n и целого числа a из отрезка $[1, n – 1]$ не выполнено условие (4). Тогда будем говорить, что a является **свидетелем** для числа n. Иными словами, для нечетного составного числа n, свидетелем является то основание, по которому число n является составным.

Таким образом, для доказательства составности числа n достаточно найти хотя бы одного свидетеля.

Теорема 3.4 утверждает, что если n – нечетное составное число, то, как минимум, 3/4 всех целых чисел отрезка [1, n – 1] являются свидетелями для n. Поскольку алгоритм 3.3 допускает очень быстро работающую реализацию, проверить, является ли данное число a свидетелем числа n, достаточно легко.

Поэтому предъявить свидетеля для составного числа очень просто, если воспользоваться вероятностным методом. Следующий алгоритм – это просто сочетание алгоритма 3.3 со случайным выбором основания a.

Алгоритм 3.6 (тест Миллера-Рабина – вероятностный тест на составность). Нам задано нечетное число $n > 3$. Этот вероятностный алгоритм ищет свидетеля для n, чтобы доказать его составность. Если найден свидетель a, то выдается ответ (a, ДА). В противном случае выдается ответ (a, НЕТ).

1. (* Выбор возможного свидетеля *)
Выбор случайного числа $a \in [2, n - 2]$;
Посредством алгоритма 3.3 решается, является ли n сильно вероятным простым числом по основанию a;
2. (* Объявления результата *)
If n – сильно вероятное простое число по основанию a then return(a, НЕТ)
else return(a, ДА);

Согласно теореме 3.4, если $n > 9$ – нечетное составное число, то вероятность того, что алгоритму 3.6 не удастся найти свидетеля для n меньше 1/4. Кроме того, мы можем применить этот алгоритм и повторно. Вероятность неудачи при поиске свидетеля посредством k (независимых) итераций алгоритма 3 меньше $1/4^k$. Очевидно, выбирая число k большим, мы можем сделать эту вероятность пренебрежимо малой. Алгоритм 3.6 – очень эффективное средство распознавания составных чисел. Но каков будет результат его работы, если мы подадим на вход нечетное простое число? Разумеется, свидетеля найти не удастся, поскольку в силу теоремы 3.1 у простых чисел свидетелей нет.

Пусть нам неизвестно, является данное большое нечетное число n простым или составным. Предположим, что, скажем, 20 итераций алгоритма 3.6 не позволили найти для п свидетеля. Какой можно сделать вывод? Строго говоря, на этом основании ничего о простоте или составности числа n утверждать нельзя. Конечно, представляется логичным предположить, что число n, скорее всего, простое. Вероятность того, что 20 итераций алгоритма 3.6 не позволят найти свидетеля для данного составного нечетного числа, не превосходит 4^{-20}, что меньше, чем одна триллионная. Следовательно, число n и в самом деле наверняка простое. Но его простота остается недоказанной и, более того, это число может оказаться и составным.

Возникает вопрос: можно ли верхнюю границу для свидетелей в алгоритме 3.6 сделать меньше n – 2? Самое лучшее что известно, есть следующий результат

Теорема 3.7 [28, стр. 164]. Пусть n – нечетное составное число. При условиях справедливости расширенной гипотезы Римана наименьший свидетель для числа n меньше $2 \cdot \ln^2 n$.

Обычная гипотеза Римана состоит в предположении, что все нетривиальные нули дзета-функции, аналитически продолжающей ряд

$$\sum_{n=0}^{\infty} \frac{1}{n^s},$$

расположены на вертикальной прямой $\sigma = ½$. «Расширенная» – предполагает то же самое у любого ряда Дирихле

$$\sum_{n=1}^{\infty} \frac{\chi(s)}{n^s},$$

где функция $\chi(s)$ – характер Дирихле. Знаменитая гипотеза Римана и ее расширенный вариант, еще не доказаны, но имеется много свидетельств в их справедливости (см. 25, стр. 48-58, 29).

Приведем детерминистический тест Миллера на простоту. Он основан на теореме 3.7 и при условии расширенной гипотезы Римана устанавливает простоту числа n за полиномиальное время относительно количества цифр в n.

Алгоритм 3.8 (тест Миллера на простоту). Задано нечетное число $n > 1$. Алгоритм устанавливает, является ли число n простым (ДА) или нет (НЕТ). Если возвращен ответ «НЕТ», то число n непременно составное. Если же получен ответ «ДА», то или число n простое, или расширенная гипотеза Римана не верна.

$a = 2$;
While $a \le \lfloor 2 \ln^2 n \rfloor$ do
    begin
    При помощи алгоритма 3.3 проверяем, является ли число n сильно вероятным простым числом по основанию a;
    If n не является n сильно вероятным простым числом по основанию a then
                                                    return(НЕТ);
    a = a +1
    end;
return(ДА)

Для приложений простых чисел, например, в криптографии, важно уметь получать большое простое число. Причем на практике обычно достаточно часто используют числа, которые почти наверняка являются простыми, но их простота строго не доказана. Следующий алгоритм [28, стр. 161] можно применять для получения случайных чисел, которые, скорее всего, являются простыми.

Алгоритм 3.9. (генерация простого числа). Нам заданы целые числа $k \ge 3$ и $T \ge 1$. Этот вероятностный алгоритм генерирует k-битовое число (т.е. число из промежутка $[2^{k-1},2^k)$), которое не было признано составным после T итераций алгоритма 3.6.

1. (* Выбор кандидата *)
Выбор случайного нечетного числа n из промежутка $[2^{k-1},2^k)$;
2. (* Выполнить тесты на сильно вероятное простое число *)
$i = 1$;
While $i \le T$ do
    begin
    Применяем алгоритм 3.6 для поиска свидетеля для n;
    If свидетель для n найден then goto 1 else $i = i + 1$
    end;
return(n) (* n – сильно вероятное простое число *)

Mathematica имеет функцию для генерации простого числа:

RandomPrime[{10^{32}, 10^{33}}]

501 720 654 095 427 233 202 505 694 811 793

Другим источником больших простых чисел являются числа специального вида, например, простые числа Мерсенна $M_n = 2^n - 1$ (для простоты M_n необходимо, чтобы было простое n). За редкими исключениями рекорд максимального простого числа всегда принадлежал именно простым числам Мерсенна. В январе 2013 найдено простое число Мерсенна $M_{57885161}$. Оно сейчас наибольшее и среди всех известных простых чисел.

Для проверки на простоту чисел Мерсенна существует эффективный тест Люка-Лемера [28, стр. 210].

§ 4. Тестирование простоты – полиномиальная задача

Мы можем классифицировать тесты на простоту по наличию следующим свойств:

1. Универсальность. Многие тесты предназначены для проверки чисел из ограниченного множества. Например, тест Люка-Лемера.

2. Полиномиальность. Наибольшее время работы алгоритма ограничено полиномом от количества цифр в проверяемом числе. Алгоритмы 1.5, 3.3, 3.6 и 3.8 яв-

ляются полиномиальными. Алгоритмы из главы 5: алгоритм 3.2 (метод пробных делений) и алгоритм 4.1 (метод Ферма) не являются полиномиальными.

3. Детерминизм. Алгоритм гарантирует получение ответа. Вероятностные тесты, такие как алгоритмы 1.5, 3.3 и 3.6, могут проверить простоту числа за полиномиальное время, но при этом дают лишь вероятностный ответ. Детерминированным является алгоритм 3.8.

4. Безусловность. Корректность теста не зависит от каких-либо недоказанных гипотез. Напротив, тест Миллера (алгоритм 3.8) детерминирован и работает за полиномиальное время для любого входного числа, но его корректность зависит от недоказанной расширенной гипотезы Римана.

Все изученные нами алгоритмы обладают не более чем тремя из перечисленных свойств.

В августе 2002 года индийский математик М. Агравал и его два бывших студента Н. Кайл и Н. Саксена (они только за три месяца до этого защитили свою бакалаврскую работу) анонсировали детерминированный полиномиальный алгоритм для определения, является ли число простым. Улучшенный и упрощенный вариант алгоритма был опубликован через два года [1]. Сейчас он известен как *AKS-тест*.

Этот тест произвел сенсацию не только потому, что был получен давно ожидаемый результат, к которому различные исследователи уже подошли с разных сторон; но он замечателен еще своей простотой. *AKS*-тест является универсальным, полиномиальным, детерминированным и безусловным тестом. Сложность алгоритма установлена как $O(\ln^{7.5} n)$, но усовершенствования продолжаются, и, по-видимому, удастся получить версию алгоритма со сложностью $O(\ln^6 n)$ [28, стр. 228–247]. К сожалению, речь сейчас еще не идет о практическом применении алгоритма *AKS* из-за большого показателя степени.

Задачи и упражнения

1. Определим $r(n) = (10^n - 1)/9$. Десятичными цифрами чисел $r(n)$ являются только единицы; их n штук.

a) Покажите, что если число n составное, то число $r(n)$ – тоже составное. Подсказка: если n делится на m, то $r(n)$ делится на $r(m)$.

b) Рассмотрите все простые числа n в интервале от 2 до 500 включительно. Среди них имеется только пять чисел n, для которых соответствующие $r(n)$ не удовлетворяют тесту на разложимость. Найдите эти пять $r(n)$. Являются ли эти числа простыми?

2. Найдите все псевдопростые числа Ферма по основанию 2, 3 и 5 меньшие 10 000.

3. Найдите все числа Кармайкла, не превышающие 500 000. Проверьте, выполнено ли уже соотношение (3) для $n = 500\,000$.

4. Напишите программу, реализующую алгоритм 3.6.

5. Напишите программу, реализующую алгоритм 3.8.

6. Напишите программу, реализующую алгоритм 3.9.

Глава 9. Факторизация натуральных чисел

На настоящее время все используемые для факторизации чисел детерминированные алгоритмы, сложность которых строго проанализирована, являются экспоненциальными [28, стр. 254]. Это означает, что время их работы в худшем случае является фиксированной положительной степенью разлагаемого числа. В начале 70-х годов 20 века стали появляться более быстрые алгоритмы, но они являются эвристическими, то есть не гарантируют разложения любого составного числа. Большинство специалистов думает, что задача факторизации является экспоненциальной. Вся современная криптография основана на этом допущении.

Нам известны два алгоритма, с помощью которых мы можем разложить натуральное число на простые множители. Это метод пробных делений и метод Ферма разложения на множители из главы 5. Алгоритмы из главы 8 более быстры, чем вышеназванные, но они могут только сообщить о существовании делителей, не более.

Рассмотрим один из эвристических методов – ро-метод Полларда для разложения на множители.

Изложим этот алгоритм, следуя [28, стр. 259]. Рассмотрим случайную функцию f: $S \to S$, где $S = \{0, 1, \ldots, m - 1\}$. Выберем случайный элемент $s \in S$ и рассмотрим последовательность

$$s,\ f(s),\ f(f(s)),\ \ldots.$$

Так как функция f принимает значения из конечного набора, то очевидно, что с какого-то момента последовательность повторится и станет циклической. Мы можем изобразить поведение этой последовательности при помощи буквы ρ, хвост которой означает доциклическую часть последовательности, и овал – циклическую часть. Можно показать, что ожидаемая длина и хвоста и цикла имеет порядок $\sqrt{m}$.

Теперь предположим, что мы хотим разложить на множители число n, и пусть число p является наименьшим простым делителем числа n. Возьмем конкретную функцию f, отображающую $\{0, 1, \ldots, p - 1\}$ в $\{0, 1, \ldots, p - 1\}$, а именно, $f(x) = x^2 + 1 \bmod p$. Если эта функция «достаточно случайна», то мы можем ожидать, что последовательность итераций $(f^{(i)}(s))$, $k = 0, 1, \ldots$, начинающаяся со случайного числа $s \in \{0, 1, \ldots, p - 1\}$, начнет повторяться раньше, чем через $O(\sqrt{p})$ шагов. Т. е. мы ожидаем существование шагов $0 \le j < k$, $k = O(\sqrt{p})$), таких, что $s^{(j)}(s) = f^{(k)}(s)$.

Так как мы не знаем число p, то мы не можем вычислить последовательность, описанную выше. Однако мы можем вычислить значения функции F, определенной как $F(x) = x^2 + 1 \bmod n$. Но нам не требуется найти две равные итерации функции F, а требуется найти две равные итерации функции $f(x) = x^2 + 1 \bmod p$. Пусть для некоторых j и k, $j < k$, имеем $f^{(j)}(s) = f^{(k)}(s)$. Как можно найти j и k, зная только F? Очевидно, что $f(x) = F(x) \bmod p$. Следовательно, $F^{(j)}(s) \equiv F^{(k)}(s) \pmod p$. Т. е. $\gcd(F^{(j)}(s) - F^{(k)}(s), n)$ делится на число p. Если нам повезет и этот gcd не равен самому числу n, то мы получим нетривиальный делитель числа n.

В ро-методе Полларда есть еще одна трудность. Естественно, мы не можем перебирать все пары чисел j, k, удовлетворяющие условию $0 \le j < k$, и вычислять $\gcd(F^{(j)}(s) - F^{(k)}(s), n)$ для каждой пары. Это может занять больше времени, чем поиск простого делителя p при помощи пробного деления. Поэтому нам нужен другой способ поиска подходящей пары чисел j, k, отличный от полного перебора. Для этого используют замечательный прием, а именно, метод поиска циклов Флойда.

Пусть M – конечное множество и f: $M \to M$ – некоторое отображение, $a_0 \in M$ – произвольный элемент и последовательность элементов $a_0, a_1, a_2, \ldots$ определяется равен-

ством $a_{n+1} = f(a_n)$, $n = 0, 1, \ldots$. Метод Флойда основан на выполнении следующего условия: если выполнено равенство $a_n = a_{2n}$, то величина длины цикла делит n.

Итак, основная идея ро-метода Полларда состоит в вычислении последовательности $\gcd(F^{(i)}(s) - F^{(2i)}(s), n)$ для $i = 1,2,...$, и это вычисление должно закончиться нетривиальным разложением числа n на множители за $O(\sqrt{p})$ шагов, где число p является наименьшим простым делителем числа n.

Алгоритм 1 (ро-метод Полларда для разложения на множители). Дано составное число n. Этот алгоритм пытается найти нетривиальный делитель числа n.

```
d = n;
While d == n do
        begin
        (* Выбрать инициализаторы *)
        Выбираем случайное a ∈ [1, n - 3];
        Выбираем случайное s ∈ [1, n - 1];
        U = V = s;
        Определяем функцию F(x) = (x^2 + a) mod n;
        d = 1;
        While d == 1 do
                begin
                (* Поиск делителей *)
                U = F(U);
                V = F(F(V));
                d = gcd(U – V, n);
                end;
        end;
(* Успех *)
return(d)
```

Программа на языке Wolfram:

```
pollard[n_] :=
        Module[{s, a, u, v, d = n},
                While[d == n,
                    s = Random[Integer, {0, n – 1}];
                    a = Random[Integer, {0, n – 3}];
                    u = s; v = s; d = 1;
                     While[d == 1,
                       u = Mod[u^2 + a, n];
                       v = Mod[v^2 + a, n];
                       v = Mod[v^2 + a, n];
                       d = GCD[u – v, n];
                           ];
                      ];
                 d]
```

Протестируем программу:

```
Do[n = RandomInteger[{10^9, 10^10}];
   While[PrimeQ[n], n = RandomInteger[{10^9, 10^10}]];
   Print["n = ", n, ", d =  ", d = pollard[n], ", n/d = ", n/d], {5}]
```

```
n = 2835002809, d =  24469, n/d = 115861
n = 1073327771, d =  863, n/d = 1243717
n = 4020358968, d =  9, n/d = 446706552
n = 1992126523, d =  457, n/d = 4359139
n = 5157874644, d =  52, n/d = 99189897
```

Алгоритм работает очень быстро для чисел с маленькими делителями, но медленно в случае, когда все делители большие. Если n – простое число, то программа работает бесконечно долго, поскольку каждый раз во внешнем цикле получаем $d = n$.

Глава 10. Гауссовы целые числа

§ 1. Гауссовы простые числа

Вспомним определения.

1.1. Неприводимое гауссово целое число – это гауссово целое число, не имеющее других делителей, кроме тривиальных.

1.2. Ненулевое необратимое гауссово число z, по определению, является **простым гауссовым целым числом**, если из $z \mid u \cdot v$ следует $z \mid u$ или $z \mid v$. Для кольца $\mathbb{Z}[i]$ простые и неприводимые числа одни и те же. Это следует из следующей леммы.

Лемма 1.3. [3, стр. 241, теорема 218]. Если z – неприводимое гауссово число, то из $z \mid u \cdot v$ следует $z \mid u$ или $z \mid v$.

Число, не являющееся простым, называется *составным*. При этом делители единицы, подобно натуральной единице, не считаются ни простыми, ни составными числами.

Как мы ранее установили (глава 4, следствие 3.9) в $\mathbb{Z}[i]$ имеется аналог основной теоремы арифметики. Точнее, каждое гауссово целое число, не являющееся нулём или делителем единицы, разлагается на простые множители, причём это разложение однозначно с точностью до порядка и ассоциированности множителей.

Нам понадобится утверждение, известное как теорема Ферма-Эйлера. Но сначала несколько лемм.

Лемма 1.4. Для любого простого p, для которого $p \equiv 1 \pmod 4$, существует такое целое m, что $p \mid (m^2 + 1)$.

Доказательство. Пусть $p = 4n + 1$, тогда в качестве числа m годится $m = (2n)!$. Чтобы это увидеть, рассмотрим число

$$(p-1)! = 1\cdot 2\cdot\ldots\cdot(2n-1)\cdot(2n)\cdot(2n+1)\cdot\ldots\cdot(4n-1)\cdot(4n) =$$
$$1\cdot 2\cdot\ldots\cdot(2n-1)\cdot(2n)\cdot(p-2n)\cdot(p-(2n-1))\cdot\ldots\cdot(p-2)\cdot(p-1).$$

Оно даёт при делении на p такой же остаток, как и число

$$1\cdot 2\cdot\ldots\cdot(2n-1)\cdot(2n)\cdot(-1)^{2n}\cdot(2n)\cdot(2n-1)\cdot\ldots\cdot 2\cdot 1 = m^2.$$

Значит, $m^2 + 1$ при делении на p дает такой же остаток, как и число $(p-1)! + 1$. Последнее число кратно p по теореме 1.6 Вильсона-Лагранжа из главы 6. ♦

Лемма 1.5. Любой простой делитель p числа $m^2 + 1$, $m > 1$ – натуральное, представим в виде суммы двух квадратов натуральных чисел.

Доказательство. Поскольку пары чисел $(m, 1)$ и $(m, -1)$ есть пары взаимно простых чисел и $m^2 + 1 = (m + i)(m - i)$, то p не является делителем ни $m + i$, ни $m - i$ в $\mathbb{Z}[i]$. Поэтому, в силу леммы G.1.3, число p не является простым гауссовым числом. Значит $p = (a + bi)(c + di)$, где целые $a + bi$ и $c + di$ – не делители единицы. Имеем

$$N(p) = N(a + bi)\, N(c + di).$$

Поэтому выполнено $p^2 = (a^2 + b^2)(c^2 + d^2)$, откуда получаем $p = a^2 + b^2 = c^2 + d^2$. ♦

Теорема 1.6 (Ферма-Эйлер). Простое число $p > 2$ представимо в виде суммы двух квадратов тогда и только тогда, когда $p \equiv 1 \pmod 4$.

Доказательство. Достаточность сразу следует из лемм 1.4 и 1.5. Докажем необходимость. Число p – нечетное и равно сумме двух квадратов. Поэтому один квадрат четный, а другой нечетный. Квадрат четного числа нацело делится на 4, а квадрат нечетного числа при делении на 4 дает в остатке 1, а не 3. Следовательно, случай $p \equiv 3 \pmod 4$ невозможен. ♦

Известно, что разложение на сумму квадратов единственно, см. [21, стр. 119–120].

Теорема 1.7. Гауссово целое число z является простым тогда и только тогда, когда оно ассоциировано с числом из следующих списков:

(i) $1+i$;

(ii) $a + bi$, где $a^2 + b^2$ есть простое натуральное число и $a^2 + b^2 \equiv 1 \pmod 4$;

(iii) p, где p есть простое натуральное число и $p \equiv 3 \pmod 4$.

Доказательство. Покажем сначала, что принадлежность одному из списков (i), (ii) или (iii) достаточна для того, чтобы гауссово целое число было простое.

- *Гауссово целое типа (i) или (ii) есть простое.* Чтобы убедиться в этом, достаточно доказать, что $N(z)$ – простое в $\mathbb{Z}$ влечет z – простое в $\mathbb{Z}[i]$.

  От обратного, пусть $z = uv$ – не простое гауссово число, тогда $N(u) > 1$, $N(v) > 1$ и $N(z) = N(u)N(v)$. Следовательно, $N(z)$ – не простое. Противоречие.
- *Гауссово целое типа (iii) есть простое.* Если число $p = 4n + 3$ является произведением двух гауссовых целых чисел $p = (a + bi)(c + di)$, то, в силу мультипликативности нормы, выполнено равенство $p^2 = (a^2 + b^2)(c^2 + d^2)$. Значит, либо один из множителей $(a^2 + b^2)$ и $(c^2 + d^2)$ равен 1, а другой равен p^2, либо $p = a^2 + b^2 = c^2 + d^2$. В первом случае ясно, что число p было представлено в виде произведения делителя единицы и ассоциированного с p числа. Рассмотрим второй случай. Число $p \equiv 3 \pmod 4$ и равно сумме двух квадратов. Но это противоречит теореме 1.6.

Покажем теперь, что любое число w, ассоциированное с простым гауссовым числом $z = a + bi$, принадлежит одному из списков (i), (ii), (iii).

- Если $N(z)$ есть простое в $\mathbb{Z}$, то мы необходимо имеем $N(z) = 2$ с z типа (i) или $N(z) \equiv 1 \pmod 4$, так как $N(z)$ есть сумма двух квадратов, т.е. z принадлежит списку (ii).
- Если $N(z)$ – не простое в $\mathbb{Z}$, то $N(z) = p_1 \ldots p_k$, где все p_i есть простые в $\mathbb{Z}$ и $k \geq 2$. Так как z является простым в $\mathbb{Z}[i]$, и z делит норму $N(z) = z \cdot \bar{z}$, то из леммы 1.3 следует, что z делит одно из p_i. Таким образом, $p_i = uz$. Переходя к нормам имеем $p_i^2 = N(u)N(z)$. Равенство возможно в двух случаях. Первый случай – $p_i = N(u) = N(z)$. Но это противоречит тому, что $N(z)$ – не простое в $\mathbb{Z}$. Второй случай – $p_i^2 = N(z)$ и $N(u) = 1$. Это показывает, что z и p_i – ассоциированные числа. Число z не принадлежит типу (i), так как тогда бы $N(z)$ было бы простым. Число p_i при делении на 4 не дает в остатке 1, потому что тогда бы по теореме 1.6 имеем $p_i = a^2 + b^2$, и, следовательно, $p_i = (a + bi)(a - bi)$, и оба числа $a + bi$ и $a - bi$ не делители единицы, т.е. p_i не было бы простым в $\mathbb{Z}[i]$. Поэтому $p_i \equiv 3 \pmod 4$ и z принадлежит списку (iii). ♦

Для доказательства того, в каких случаях простое натуральное число p является простым гауссовым целым числом, нам потребуется следующая лемма.

Лемма 1.8. Простое натуральное число p нельзя представить в виде произведения более чем двух целых гауссовых чисел, не являющихся делителями единицы. (Другими словами, если p ассоциировано с произведением двух не являющихся делителями единицы целых гауссовых чисел, то эти числа простые.)

Доказательство. Если $p = (a + bi)(c + di)(e + fi)$, то в силу мутипликативности нормы выполнено $p^2 = (a^2 + b^2)(c^2 + d^2)(e^2 + f^2)$. Но квадрат простого числа никак не может быть произведением трех отличных от 1 натуральных чисел. ♦

Теорема 1.9. Простое натуральное число p является простым гауссовым целым числом тогда и только тогда, когда $p \equiv 3 \pmod 4$.

Доказательство. Достаточность доказана в теореме 1.7. Простое $p \equiv 1 \pmod 4$ не может быть простым гауссовым целым числом. Более точно, всякое простое натуральное число вида $p = 4n + 1$ разлагается на два сопряженных множителя: $p = (a + bi)(a - bi)$, при-

чем $a + bi$ и $a - bi$ – простые гауссовы числа (то, что они простые в $\mathbb{Z}[i]$ следует из леммы 1.8). Также $2 = -i(1+i)^2$ не является простым гауссовым целым числом. ♦

В Mathematica функция PrimeQ используется не только для целых чисел, но и для определения простоты гауссового целого числа. Надо просто добавить опцию GaussianIntegers → True, если речь идет о простых числах в кольце $\mathbb{Z}[i]$. Но если в изображении тестируемого числа уже присутствует мнимая единица, то опция необязательна.

В следующих примерах простое гауссово число помещается в рамочку (это делает функция Framed).

```
If[PrimeQ[#, GaussianIntegers → True], Framed[#], #]&
        /@ Table[Prime[k], {k, 1, 20}]
```

```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
 31, 37, 41, 43, 47, 53, 59, 61, 67, 71}
```

```
Table[If[PrimeQ[#], Framed[#], #]& [m + n I], {m, −3, 3}, {n, −3, 3}]
        // TableForm
```

−3 − 3 i	−3 − 2 i	−3 − i	−3	−3 + i	−3 + 2 i	−3 + 3 i
−2 − 3 i	−2 − 2 i	−2 − i	−2	−2 + i	−2 + 2 i	−2 + 3 i
−1 − 3 i	−1 − 2 i	−1 − i	−1	−1 + i	−1 + 2 i	−1 + 3 i
−3 i	−2 i	− i	0	i	2 i	3 i
1 − 3 i	1 − 2 i	1 − i	1	1 + i	1 + 2 i	1 + 3 i
2 − 3 i	2 − 2 i	2 − i	2	2 + i	2 + 2 i	2 + 3 i
3 − 3 i	3 − 2 i	3 − i	3	3 + i	3 + 2 i	3 + 3 i

Алгоритм 1.10 (Эйлер)

Опишем алгоритм Эйлера для разложения простого числа на сумму двух квадратов. В силу теоремы Ферма-Эйлера такое разложение возможно только для простых чисел $p \equiv 1 \pmod 4$.

Нам понадобится вариант евклидового деления целых чисел с **центрированным остатком**:

$$a = bq + r, \qquad -b/2 < r \le b/2.$$

Пара (q, r) единственна. Если q' и r' есть частное и остаток при обыкновенном евклидовом делении, то ясно, что

$$r = \begin{cases} r', & \text{если } 2r' \le b, \\ r' - b, & \text{если } 2r' > b; \end{cases} \qquad q = \begin{cases} q', & \text{если } 2r' \le b, \\ q' + 1, & \text{если } 2r' > b. \end{cases}$$

Первый шаг: нахождение частного решения $x^2 + 1 \equiv 0 \pmod p$

С помощью алгоритма 6.9 из главы 6 извлекаем квадратный корень из -1 по модулю p, т.е. находим решение $x^2 + 1 \equiv 0 \pmod p$. Существование решения сравнения $x^2 + 1 \equiv 0$

(mod p) следует из леммы 1.4, а, кроме того, символ Лежандра $\left(\frac{-1}{p}\right)$ равен 1 (см. теорему 6.7, глава 6).

Второй шаг: декомпозиция p на сумму двух квадратов

Пусть x и y – такие натуральные числа, что x и y не кратны p и $x^2 + y^2 \equiv 0 \pmod p$, в качестве таких чисел мы можем взять x, найденное на первом шаге, и $y = 1$. Тогда

$$x^2 + y^2 = pn_0. \tag{1}$$

Если $n_0 = 1$, то p уже равно сумме двух квадратов. Поэтому будем считать, что $n_0 > 1$. Мы делим x и y на p, используя деление с центрированным остатком:

$$\begin{aligned} x &= p\xi + r1, \quad |r1| < p/2, \\ y &= p\eta + r2, \quad |r2| < p/2. \end{aligned} \tag{2}$$

(Неравенства в (2) строгие, потому что p нечетно.) Из (1) и (2) следует, что $r1^2 + r2^2 \equiv \equiv x^2 + y^2 \equiv 0 \pmod p$, поэтому

$$r1^2 + r2^2 = mp, \quad m > 0. \tag{3}$$

Мы имеем $m > 0$, потому что из $m = 0$ следовало бы $r1 = 0$ и $r2 = 0$, но это противоречит исходному условию: x и y не кратны p.

Снова случай $m = 1$ сразу дает представление p в виде суммы двух квадратов. Поэтому будем считать, что $m > 1$. Поделим $r1$ и $r2$ на m:

$$\begin{aligned} r1 &= m\alpha + a, \quad |a| < m/2, \\ r2 &= m\beta + b, \quad |b| < m/2. \end{aligned} \tag{4}$$

Из (3) и (4) следует, что $a^2 + b^2 \equiv r1^2 + r2^2 \equiv 0 \pmod m$, т.е.

$$a^2 + b^2 = m\, n_1 \tag{5}$$

Выражая p из (3), (4) и (5), получаем

$$p = m(\alpha^2 + \beta^2) + 2(\alpha a + \beta b) + n_1. \tag{6}$$

Умножая обе части равенства (6) на n_1 и учитывая (5), получаем

$$\begin{aligned} pn_1 &= mn_1(\alpha^2 + \beta^2) + n_1(\alpha a + \beta b) + n_1^2 = \\ &(a^2 + b^2)(\alpha^2 + \beta^2) + n_1(\alpha a + \beta b) + n_1^2 = \\ &= (n_1 + \alpha a + \beta b)^2 + (a\beta - b\alpha)^2. \end{aligned} \tag{7}$$

Положим $x_1 = |n_1 + \alpha a + \beta b|$ и $y_1 = |a\beta - b\alpha|$, тогда (7) становится:

$$pn_1 = x_1^2 + y_1^2. \tag{8}$$

Оценим порядок величин m и n_1. Если $\xi \neq 0$ в (2), то мы можем написать $|x| \geq p\,|\xi| - |\,r1| \geq p - |\,r1| \geq p - |\,r1| > p/2 > |\,r1|$. Если же $\xi = 0$, то мы только имеем $|x| \geq |\,r1|$. Точно также получаем $|y| \geq |\,r2|$. Следовательно, из (1) и (3) имеем $n_0 \geq m \geq 1$. Мы также имеем $m\, n_1 = a^2 + b^2 \leq \mathrm{m}^2/2$, что дает $n_1 \leq \mathrm{m}/2$. Тем самым получаем $n_0/2 \geq m/2 \geq n_1$.

Покажем, что $n_1 \geq 1$. От противного, пусть $n_1 = 1$. Тогда из (5) следует $a = b = 0$ и поэтому $x = m\alpha$ и $y = m\beta$. Поэтому из (3) следует $m(\alpha^2 + \beta^2) = p$. Так как p – простое, то это означает, что $m = p$ или $m = 1$. Мы изначально предполагали, что $m > 1$. Но $m \neq p$, так как из (2) и (3) имеем $pm = r1^2 + r2^2 < p^2/2$, т.е. $m < p/2$. Полученное противоречие приводит к окончательному неравенству

$$n_0/2 \geq m/2 \geq n_1 \geq 1. \tag{9}$$

Если $n_1 = 1$, то из (8) мы получаем, что p есть сумма двух квадратов. Если $n_1 > 1$, то мы повторяем вычисления, полагая x равным x_1, а y равным y_1. (Если x_1 и y_1 кратны p, то из (8) следует, что $n_1 \equiv 0 \pmod p$). Но это невозможно, так как $p/4 \geq m/2 \geq n_1 > 1$.) В процессе вычисления получаем последовательность чисел $n_0 > n_1 > \ldots > n_k = 1$. Эта последовательность сходится достаточно быстро: $n_k/2 \geq n_{k+1}$.

Пусть процедура *centured_division*(a, b, q, r) выполняет деление a на b с неполным частным q и с центрированным остатком r. Тогда алгоритм Эйлера можно описать в следующем виде:

Ищем такое x, что $x^2 + 1 \equiv \pmod p$;
$y := 1$;
While $x^2 + y^2 > p$ do
  begin
  centured_division(x, p, ξ, $r1$);
  centured_division(y, p, η, $r2$);
  $m := (r1^2 + r2^2)$ div p;
  If $m = 1$ then begin $x := r1$; $y := r2$ end
          else begin
            centured_division($r1$, m, α, a);
            centured_division($r2$, m, β, b);
            $n := (a^2 + b^2)$ div m;
            $x := \text{abs}(n + \alpha a + \beta b)$;
            $y := \text{abs}(a\beta - b\alpha)$
            end
  end;
return({x,y})

Можно показать, что количество шагов в цикле while имеет порядок $O(\log_2 p)$.

Представим программу в системе Mathematica, реализующую алгоритм Эйлера.

Функция quotientRemaiderCent[m, n] осуществляет деление чисел m на n с центрированным остатком:

```
quotientRemainderCent[m_, n_] :=
      Module[{qr = QuotientRemainder[m, n], q, r, q1, r1},
            q = qr[[1]]; r = qr[[2]]; If[2r ≤ n, qr, {q + 1, r – n}]]
```

Функция summaSquares[p] находит такие числа x и y, что $p = x^2 + y^2$:

```
summaSquares[p_] :=
  Module[{x = sqrtMod[p, –1], y = 1, r1, r2, α, β, a, b, s, m, n},
    While[x^2 + y^2 > p,
          s = quotientRemainderCent[x, p]; r1 = s[[2]];
          s = quotientRemainderCent[y, p]; r2 = s[[2]];
          m = Quotient[r1^2 + r2^2, p];
          If[ m == 1, x = r1; y = r2,
                s=quotientRemainderCent[r1, m];
                α = s[[1]]; a = s[[2]]; s = quotientRemainderCent[r2, m] ;
                β = s[[1]]; b = s[[2]]; n = Quotient[a^2 + b^2, m];
                x = Abs[n + α a + β b]; y = Abs[a β – b α]
            ]
        ];
    {x, y}]
```

Пример 1.11. Возьмем простое число из примера 6.10 главы 6.

```
p = 180(2^127 – 1)^2 + 1;

summaSquares[p]
```

{435 323 051 965 400 292 202 079 655 882 171 312 489, 2 240 789 560 870 622 717 042 494 395 834 592 781 990}

Mathemaica может, прямо обращаясь к функции Solve, разложить простое число на сумму квадратов.

Пример 1.12. Число 18496 × 218496 + 1 является простым ([8, стр. 241], год открытия 1984, содержит 5573 цифры). Функция summaSquares находит решение

```
p = 18496 × 25573 + 1;
{x, y} = summaSquares[p];
```

Второе слагаемое равно 1, а первое есть квадрат числа, содержащего 2787 цифр:

```
{Short[x], y]
{1 145 351 552 576 294 <<2754>> 35 612 955 476 361 216, 1}
```

Очистим переменные x и y от значений и вызовем Solve в следующем виде

```
Clear[x, y]
{u, v} = {x, y} /. Solve[{x^2 + y^2 == p, x > 0 && y > 0}, {x, y}, Integers];
```

Получили два решения, совпадающие с {x, y} с точностью до порядка

```
{u[[1]], Short[u[[2]]]}
{1, 1 145 351 552 576 294 <<2754>> 35 612 955 476 361 216}
{Short[v[[1]]], v[[2]]}
{1 145 351 552 576 294 <<2754>> 35 612 955 476 361 216, 1}
```

§ 2. Факторизация гауссовых целых чисел

В разложении гауссова целого числа z на простые множители любой множитель z может быть заменен ассоциированным с ним числом: iz, $-z = i^2z$, $-iz = i^3z$. Таким образом, любое ассоциированное с z число i^kz может быть получено из z последовательным вращением на угол $\pi/2$ вокруг начала координат. В частности, ненулевое z или одно из ассоциированных с ним чисел находится в первой четверти или лежит на оси.

Мы выберем «нормальное» представление простых множителей так, чтобы они находились в первой четверти, т.е. вещественная и мнимая части не являлись отрицательными.

Построим алгоритм факторизации гауссова целого числа. Пусть

$$z = \varepsilon w_1^{k_1} w_2^{k_2} \cdots w_n^{k_n}$$

– разложение числа z на гауссовы простые множители w_1, w_2 , …, w_n с соответствующей кратностью k_1, k_2 , …, k_n и ε – делитель единицы. Тогда

$$N(z) = N(w_1)^{k_1} N(w_2)^{k_2} \cdots N(w_n)^{k_n} .$$

По теореме 1.7 простые делители w_i могут принадлежать только трем категориям.

1. w_i равно $1 + i$ с точностью до ассоциированности, тогда $N(z) = 2$.
2. $w_i = a \pm bi$, где $a^2 + b^2 = p$, p – простое число в $\mathbb{Z}$ и $p \equiv 1 \pmod 4$. Тогда $N(w_i) = p$.
3. $w_i = p$, где p – простое число в $\mathbb{Z}$ и $p \equiv 3 \pmod 4$. Тогда $N(w_i). = p^2$.

Рассмотрим $N(w_1)^{k_1} N(w_2)^{k_2} \cdots N(w_n)^{k_n}$. Каждое $N(w_i)$ – простое число, если w_i принадлежит категориям 1 и 2. И если w_i принадлежит категории 3, то $N(w_i)$ – квадрат простого числа. Являются ли все $N(w_i)$ различными? В общем случае – нет. Действительно, два разных сомножителя $w_i = a + bi$ и $w_j = a - bi$, принадлежащие категории 2, имеют одинако-

вую норму $N(w_i) = N(w_j) = a^2 + b^2 = p$. И если мы факторизуем в $\mathbb{Z}$ число $N(z)$, то двум таким сомножителям $w_i^{k_i}$ и $w_j^{k_j}$ будет соответствовать один множитель p^k, $k = k_i + k_j$.

Зная факторизацию $N(z)$, мы легко получаем факторизацию z. Для этого рассмотрим три случая для p^k – сомножителей в разложении $N(z)$ на простые множители:

1. $p = 2$. Тогда w^k ассоциировано с $(1 + i)^k$.
2. $p \equiv 1 \pmod 4$. Тогда находим единственные натуральные числа a и b, такие, что $a^2 + b^2 = p$. Определяем m – максимальную степень, с которой $(a + bi)^m$ делит z, и тогда $k - m$ будет максимальной степенью, с которой $(a - bi)^{k-m}$ делит z. И мы получаем два сомножителя $(a + bi)^m$ и $(a - bi)^{k-m}$ в разложении z или один сомножитель, если одно из чисел m или $k - m$ равно 0.
3. $p \equiv 3 \pmod 4$. Тогда k обязательно четно и w^k ассоциировано с $p^{k/2}$.

Пример 2.1. Пусть $z = 3 + 21i$. Факторизуем $N(3 + 21i) = 450$:

FactorInteger[450]

{{2, 1}, {3, 2}, {5,2}}

Число 2 дает множитель $1 + i$. Число 3^2 дает множитель 3, так как $3 \equiv 3 \pmod 4$. Число 5 дает остаток 1 при делении на 4. Нетрудно обнаружить, что $5 = 2^2 + 1^2$ и дважды входит в разложение 450. Так как z не делится на $2 + i$, то в разложение z входит $(2 - i)^2$. Выберем для $2 - i$ «нормальное» представление – это будет число $1 + 2i$. Таким образом, факторизация с точность до делителя единицы есть

$$(1 + i)\ 3\ (1 + 2i)^2.$$

Это произведение равно $-21 + 3i$. Остается выбрать делитель единицы. Для этого делим z на $-21 + 3i$. Получаем $\varepsilon = -i$. И, окончательно, получаем разложение

$$3 + 21i = (-i)\ (1 + i)\ 3\ (1 + 2i)^2.$$

Опишем алгоритм на псевдокоде:

Алгоритм 2.2. (факторизация гауссовых целых чисел). Вход: z – гауссово целое число с нормой > 1. Выход: s – список пар $\{g, m\}$, где g – гауссово простое число, m – его степень.

(* Находим разложение нормы z в $\mathbb{Z}$ *)
z* = комплексно сопряженное к z число;
ds = FactorInteger[z z*];
(* каждый элемент списка *ds* есть пара $\{p, k\}$, где p – простое число,
а k – степень, с которой число p входит в разложение нормы *)
Преобразуем список *ds* в новый список *s*, заменяя каждую пару $\{p, k\}$ одной или двумя парами вида $\{g, m\}$, где g – гауссово простое число, m – его степень, по следующим правилам:

```
  If  p == 2 then получаем {1 + i, k};
  If  p ≡ 3 (mod 4) then получаем {p, k/2};
  If  p ≡ 1 (mod 4) then
          begin
          Находим a и b по алгоритму 1.10 для данного p;
          Приводим a + bi и a – bi к «нормальному» представлению;
          Определяем максимальную степень m, с которой (a + bi)^m делит z;
          If 0 < m < k then получаем две пары {a + bi, m} и {a – bi, k – m};
          If m == 0 then получаем {a – bi, k};
          If m == k then получаем {a + bi, k}
          end
  end;
```

(* Теперь s – список пар $\{g, m\}$, где g – гауссово простое число, m – его степень *)
w = произведение всех g^m из списка s;
ε = z/w; (* ε – делитель единицы *);

If $\varepsilon \neq 1$ then добавляем в начало списка s элемент $\{\varepsilon, 1\}$;
return(s)

Программа состоит из нескольких функций.

Функция nrm[z] выдает ассоциированное с z число, находящееся в первой четверти.

```
nrm[w_] :=
 Module[{z},
   Which[Re[w] < 0 && Im[w] < 0, z = –w,
         Re[w] < 0, z = – I w,
         Im[w] < 0, z = w I,
         True, z = w];
    z]
```

Для функции pg[z, {p, k}] входными данными является: пара $\{p, k\}$ – простое число p в степени k, полученная при разложении числа $N(z)$ в $\mathbb{Z}$. Функция определяет соответствующие паре $\{p, k\}$ один или два множителя в разложении z и выдает вложенный список из множителя или из двух множителей. Эта функция использует функцию summaSquares (см 1.10).

```
pg[z_, {p_, k_}] :=
    Module[{a, b, w1, w2, m, u = z},
          Which[p == 2, {{1+I, k}},
                Mod[p, 4] == 3, {{p, k/2}},
                Mod[p, 4] == 1, {a,b} = summaSquares[p]; w1 = nrm[a+b I];
                                w2 = nrm[a - b I];  m = 0;
                                While[Divisible[u, w1], u = u/w1; m = m+1];
                                Which[m == 0, {{w2, k}},
                                      k == m, {{w1, k}},
                                      True, {{w1, m}, {w2, k - m}}]]]
```

Функция factory является главной, она факторизует свой аргумент.

```
factori[z_] :=
    Module[{s, w, e]},
        s = Flatten[Map[pg[z, #]&, FactorInteger[z Conjugate[z]]], 1];
        w = Map[#[[1]]^#[[2]]&, s]/. List → Times; e = z/w;
        If[e ≠ 1, PrependTo[s, {e, 1}]]; s]
```

В системе Mathematica встроенная функция FactorInteger с опцией GaussianIntegers → True факторизует гауссовы целые числа. Следующий пример показывает применение сразу двух функции FactorInteger и factori к большому числу.

```
z = 1785650004473308591568 – 898208239842554370543 5*I;

{factori[z], FactorInteger[z, GaussianIntegers → True]}

{{{–I, 1}, {31 + 20 I, 3}, {17 + 42 I, 2}, {2539 + 1726 I, 4}},

{{–I, 1}, {17 + 42 I, 2}, {31 + 20 I, 3}, {2539 + 1726 I, 4}}}
```

Задачи и упражнения

1. Если u и v – гауссовые целые числа u и v такие, что норма $N(u)$ делится нацело на $N(v)$, то u делится на v. Докажите или опровергните.
2. Докажите или опровергните, что если целое вещественное число n кратно ненулевому целому гауссову числу $a + bi$, то n кратно числу $(a^2 + b^2)/\gcd(a, b)$.
3. Доказать, что если числа a и b взаимно просты, то наименьшее натуральное число, кратное $a + bi$, есть $a^2 + b^2$.

Тех, кто полагает, будто вычислительная машина способна заменить математика, можно было бы сравнить с теми, кто считает, что скорострельный карабин заменяет начальника штаба.

Гуго Штейнгауз. Задачи и размышления

Глава 11. Высокопроизводительные вычисления с Mathematica

§ 1. Высокоэффективные возможности Mathematica

1.1. Высокопроизводительные вычисления (High-performance computing – HPC) необходимы для решения масштабных ресурсоемких технических проблем. Вместе со скоростью вычислений необходима точность результатов. Mathematica предоставляет все это и, более того, интегрирует важные технологии HPC в одну единую систему, в которой не приходится выбирать между скоростью и точностью. Со многими технологиями, автоматически применяемыми, Mathematica является высокопроизводительной вычислительной средой, которая позволяет быстро получать точные решения.

Высокоэффективные возможности Mathematica являются следствием использования большого множества алгоритмов, тщательно проанализированных и оптимизированных для лучшей производительности с точки зрения скорости, памяти и надежности. Во многих случаях функции Mathematica инкапсулируют более одного алгоритма и автоматически выбирают среди них тот алгоритм, который дает лучшую производительность для требуемых размеров и типов входных и выходных данных.

Многоядерные процессоры в современных компьютерах позволяют ускорить решения задач путем разделения потоков или процессов между разными вычислительными ядрами, так что они могут быть выполнены параллельно. Многие из наиболее часто выполняемых алгоритмов в Mathematica могут автоматически выполняться или на многоядерном компьютере или в многокомпьтерной сети. Распределение и управление задачами полностью автоматизировано и может быть расширено до любого количества процессоров.

Например, Mathematica поддерживает многопоточность решения задач линейной алгебры на многопроцессорных или многоядерных компьютерах. Другим примером является уже упоминаемый факт, что интерфейс системы Mathematica и ее вычислительное ядро работают независимо, так что даже при полной загрузке вычислительного ядра доступны интерфейсные операции.

Параллельные вычисления в настоящее время представляют собой стандартную часть Mathematica, облегчая тем самым использование параллельных методов программирования.

Возможности для параллельных вычислений почти полностью реализованы на языке Wolfram и поэтому являются машинно–независимыми.

Основа параллельных вычислений в системе Mathematica – среда MathLink. MathLink – это общий, гибкий интерфейс для взаимодействия сторонних программ с системой Mathematica, который также используется внутри самой системы. В MathLink платформа и архитектура разделены, что позволяет работать как локально, так и по сети. Данный интерфейс может пересылать все, что может представлять система Mathematica, а также предоставляет возможности для управления системой.

Перечислим способы подключения ядер в Mathematica:

- *локальные ядра* (*Local Kernels*). Используются для организации параллелизма на том же компьютере, где находится основной процесс системы Mathematica. Такой тип подключения подходит для многоядерной среды, и он является самым простым

способом работы с параллельными вычислениями;

- *легковесная сеть* (*The Lightweight Grid*). Метод, используемый для организации параллельных вычислений на множестве компьютеров с помощью основного процесса системы Mathematica. В методе используется технология *Wolfram Lightweight Grid* для запуска Mathematica на удаленных машинах. Такой способ подходит для неоднородных сетей и случаев, когда недоступны другие технологии управления;
- *кластерная интеграция* (*Cluster Integration*). Метод, используемый для организации параллельных вычислений на множестве компьютерах с помощью хозяйского процесса системы Mathematica, который интегрируется с большим числом сторонних кластерных технологий;
- *удаленные ядра* (*Remote Kernels*). Метод, используемый для организации параллельных вычислений на множестве компьютеров с помощью хозяйского процесса Mathematica, который использует технологию удаленного вызова оболочки для запуска и, как правило, сложнее для настройки и обслуживания.

1.2. Остановимся подробно на параллельных локальных ядрах. Метод соединения локальных ядер используется для выполнения параллельных процессов на том же самом компьютере, где выполняется и ведущий процесс системы Mathematica. Метод подходит для любой многоядерной среды и это самый простой способ для параллельных вычислений. Всякий раз, когда ваш процессор имеет более одного ядра, можно использовать локальные ядра для параллельных вычислений. Количество ядер определяется с помощью функции

```
$ProcessorCount
```

```
2
```

Первый шаг, который может просто показать, что система работает это вызвать функцию ParallelEvaluate. При этом происходит настройка параллельных ядер.

```
ParallelEvaluate[$ProcessID]
```

```
{5108, 1184}
```

Мы получили идентификатор процесса для каждого параллельного ядра.

Результат вызова

```
ParallelEvaluate[$MachineName]
```

```
{valzyuz-pc, valzyuz-pc}
```

говорит о том, что параллельные процессы выполняются на одном компьютере.

Полезно осуществлять мониторинг ядер во время параллельных вычислений. Для этого надо открыть пункт меню Evaluation/Parallel Kernel Status системы Mathematica.

1.3. Рассмотрим примеры параллельных вычислений в Mathematica. Простым примером является поиск в списке. Например, будем искать простые числа среди чисел вида $n! - 1$. Это делается путем распараллеливания повторяющихся вычислений Mathematica с помощью функции Parallelize. Проводим сначала последовательные вычисления, потом параллельные и сравниваем время вычислений.

```
AbsoluteTiming[Select[Table[n, {n, 500, 1000}], PrimeQ[#! – 1]&]]
```

```
{33.378909, {546, 974}}
```

```
AbsoluteTiming[Parallelize[Select[Table[n, {n, 500, 1000}], PrimeQ[#! – 1]&]]]
```

```
{17.574922, {546, 974}}
```

Функция AbsoluteTiming выдает реальное время вычисления своего аргумента в секундах и результат вычисления аргумента. В данном случае только два числа 546! – 1 и 974! – 1 являются простыми среди чисел вида $n! - 1$ в диапазоне от $n = 500$ до $n = 1000$.

Вот первые 23 степени 2 для чисел Мерсенна $2^n - 1$ (число $2^n - 1$ может быть простым только если n – простое: см. §2 главы 5):

```
AbsoluteTiming[Select[Range[15000], If[PrimeQ[#], PrimeQ[2^#-1], False]&]]
```

```
{536.250943, {2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281,
3217, 4253, 4423, 9689, 9941, 11213}}
```

```
AbsoluteTiming[Parallelize[Select[Range[15000], If[PrimeQ[#], PrimeQ[2^#-1],
False]&]]]]
```

```
{308.146082, {2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281,
3217, 4253, 4423, 9689, 9941, 11213}}
```

Рассмотрим параллельное вычисление функции, определенной пользователем.

```
mersenneQ[n_] := If[PrimeQ[n], PrimeQ[2^n - 1], False]
```

```
AbsoluteTiming[Select[Range[2000], mersenneQ]]
```

```
{1.130002, {2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279}}
```

```
AbsoluteTiming[Parallelize[Select[Range[2000], mersenneQ]]]
```

```
{0.752020, {2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279}}
```

Как видим, параллельное вычисление происходит автоматически и для функций, определенных пользователем.

1.4. Опция Method для функции Parallelize задает, какой метод для распараллеливания будет использоваться. Возможные значения:

"CoarsestGrained" – разбивает вычисление на столько частей, сколько доступных ядер имеется для вычисления;

"FinestGrained" – разбивает вычисление на наименьшие возможные подъединицы вычисления;

"EvaluationsPerKernel" → n – разбивает вычисление на не более чем n подъединиц вычисления на одно ядро;

"ItemsPerEvaluation" → m – разбивает вычисление для получения значения самое большое на m подъединиц.

Значение опции Automatic позволяет системе самой выбрать метод, осуществляя компромисс между балансировкой загрузки ядер и уменьшением используемой памяти и времени работы.

Значение опции Method → "CoarsestGrained" является подходящим для вычислений, включающих множество подъединиц, требующих одинакового времени для вычислений. Метод минимизирует время, но не обеспечивает баланса загрузки ядер.

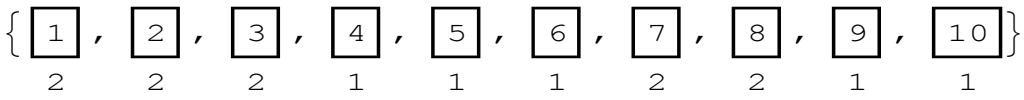
Значение опции Method → "FinestGrained" является подходящим для вычислений, включающих подъединицы, требующих заметно различного времени для вычислений. Метод приводит к повышению времени вычислений, но обеспечивает баланс загрузки ядер.

1.5. Продемонстрируем использование ядер при различных методах. Чтобы показывать номера ядер при вычислениях конкретного выражения используем функцию

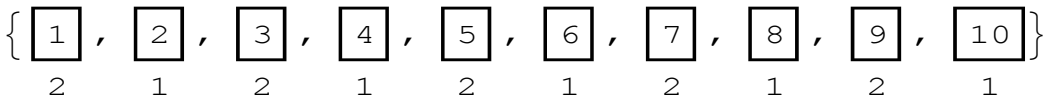
```
Labeled[Framed[#], $KernelID]&
```

Функция выдает вычисляемое выражение в рамке, а внизу помечает ядро номером. Серия примеров иллюстрирует применение методов:

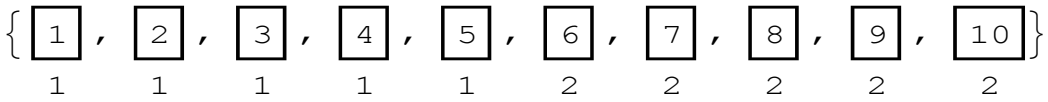
```
Parallelize[Map[Labeled[Framed[#], $KernelID]&, Range[10]],
                                                Method → Automatic]
```



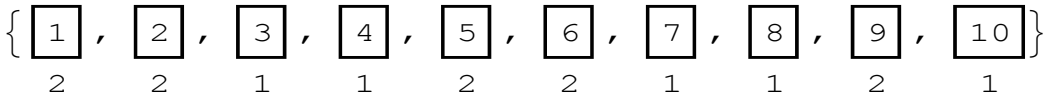
Parallelize[Map[Labeled[Framed[#], $KernelID]&, Range[10]],
Method → "FinestGrained"]



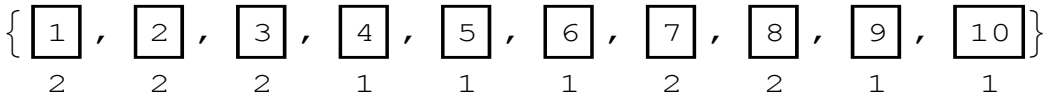
Parallelize[Map[Labeled[Framed[#], $KernelID]&, Range[10]],
Method → "CoarsestGrained"]



Parallelize[Map[Labeled[Framed[#], $KernelID]&, Range[10]],
Method → "EvaluationsPerKernel" → 3]



Parallelize[Map[Labeled[Framed[#], $KernelID]&, Range[10]],
Method → "ItemsPerEvaluation" → 3]



§ 2. Функциональное программирование и параллельные вычисления

2.1. Во всех рассмотренных примерах мы видели, что параллельные вычисления действительно происходят. Но всегда ли Mathematica может автоматически распараллелить вычисления – нет, конечно.

Вернемся к первому примеру из §6 главы 2. Требуется из списка пар составить новый список, переставляя элементы в парах. Запрограммируем процедурный подход с помощью функции *f*:

```
f[s_] := Module[{temp = s, i},
                Do[{temp[[i, 1]], temp[[i, 2]]} = {temp[[i, 2]], temp[[i, 1]]},
                {i, 1, Length[s]}]; temp]
```

```
Parallelize[f[{{a, 1}, {b, 2}, {c, 3}}]]
```

```
Parallelize :: nopar1: _f[{{a, 1}, {b, 2}, {c, 3}}]_ cannot be parallelized;
proceeding with sequential evaluation.

{{1, a}, {2, b}, {3, c}}
```

В данном случае функция вычисляется только последовательно.

Структурная итерация также выполняется только последовательно:

```
g[s_] := Module[{i}, Table[{s[[i, 2]], s[[i, 1]]}, {i, 1, Length[s]}]];
```

```
Parallelize[g[{{a, 1}, {b, 2}, {c, 3}}]]
```

```
Parallelize :: nopar1: _g[{{a, 1}, {b, 2}, {c, 3}}]_ cannot be parallelized;
proceeding with sequential evaluation.
{{1, a}, {2, b}, {3, c}}
```

И, наконец, рассмотрим функциональное программирование. В этом случае параллельное вычисление осуществляется.

Parallelize[Map[Reverse, {{a, 1}, {b, 2}, {c, 3}}]]

{{1, a}, {2, b}, {3, c}}

Возможность автоматического распараллеливания при функциональном программировании является следствием фундаментального принципа функционального программирования – «прозрачности по ссылкам» [22].

2.2. Выясним, какие типичные для функционального программирования функции могут вычисляться параллельно.

Задача: возвести в квадрат каждый элемент списка. Параллельные вычисления автоматизируются при использовании функции Map:

Parallelize[#^2&/@Range[20]]

{1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400}

Равносильные способы вычисления результирующего списка работают только при последовательном вычислении:

Parallelize[Range[20]^2]

Parallelize::nopar1: _Range[20]^2_ cannot be parallelized; proceeding with sequential evaluation.

{1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400}

Parallelize[Range[20] /. x_ → x^2]

Parallelize::nopar1: _Range[20] /. _x_ → x^2_ cannot be parallelized; proceeding with sequential evaluation.

{1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400}

Отметим некоторые тонкие моменты, когда последовательные вычисления заменяются параллельными вычислениями. Попытка выполнить одни и те же вычисления с одним выражением но в разной последовательности действий:

Parallelize[N[Sin /@ Range[10]]]

Parallelize::nopar1: _N[Sin /@ Range[10]]_ cannot be parallelized; proceeding with sequential evaluation.

{0.841471, 0.909297, 0.14112, -0.756802, -0.958924, -0.279415, 0.656987, 0.989358, 0.412118, -0.544021}

Parallelize[Sin /@ N[Range[10]]]

{0.841471, 0.909297, 0.14112, -0.756802, -0.958924, -0.279415, 0.656987, 0.989358, 0.412118, -0.544021}

N[Parallelize[Sin /@ Range[10]]]

{0.841471, 0.909297, 0.14112, -0.756802, -0.958924, -0.279415, 0.656987, 0.989358, 0.412118, -0.544021}

Предыдущий пример можно обобщить для двух произвольных функций f и g:

Parallelize[Map[f, #&[Range[10]]]]

{f[1], f[2], f[3], f[4], f[5], f[6], f[7], f[8], f[9], f[10]}

Parallelize[g[Map[f, Range[10]]]]

Parallelize::nopar1: _g[f /@ Range[10]]_ cannot be parallelized; proceeding with sequential evaluation.

g[{f[1], f[2], f[3], f[4], f[5], f[6], f[7], f[8], f[9], f[10]}]

Вывод: параллельные вычисления с Map осуществляются, когда последней операцией является операция Map.

Функция Select также вычисляется параллельно.

Parallelize[#&[Select[Range[10], True&]]]

Parallelize::nopar1: _(#1&)[Select[Range[10],True&]]_ cannot be parallelized; proceeding with sequential evaluation.

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Parallelize[Select[Range[10], True&]]

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Можно сделать заключение, что если функция Select является внешней в последовательности вычислений, то распараллеливание осуществляется автоматически.

Из двух функций Thread и MapThread распараллеливание возможно только для функции MapThread:

Parallelize[Thread[f[{a, b, c}, {x, y, z}]]]

Parallelize::nopar1: _Thread[f[{a, b, c},{x, y, z}]]_ cannot be parallelized; proceeding with sequential evaluation.

{f[a, x], f[b, y], f[c, z]}

Parallelize[MapThread[f, {{a, b, c}, {x, y, z}}]]

{f[a, x], f[b, y], f[c, z]}

Функция Apply всегда автоматически распараллеливается:

Parallelize[Apply[h, g[a, b, c]]]

h[a, b, c]

Отметим важный факт. Рекурсия является единственной реализацией итерации в чистом функциональном программировании. Но, по-видимому, рекурсию нельзя вычислить параллельно, используя фиксированное число ядер. Mathematica это не делает.

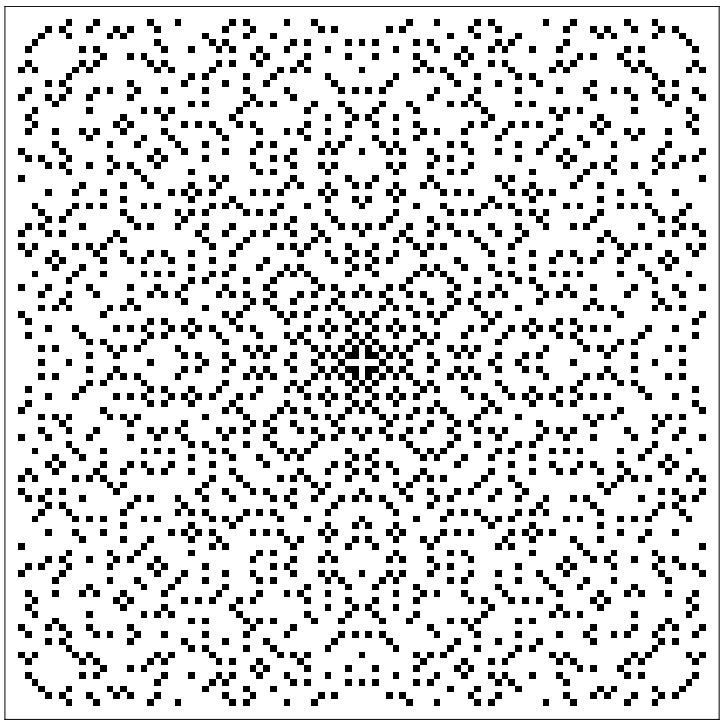
2.3. Ряд функций имеют параллельные аналоги. Начнем с функции Table. Эта функция вместе с Parallelize вычисляет параллельно.

Parallelize[Table[f[k], {k, 1, 7}]]

{f[1], f[2], f[3], f[4], f[5], f[6], f[7]}

Но вместо двух функций можно использовать одну функцию ParallelTable, которая делает тоже самое, что Table, но параллельно. Следующий красивый пример взят из справочной службы Mathematica и показывает распределение простых гауссовых чисел на плоскости. Используемая функция ArrayPlot[data] создает булевскую таблицу, где значения True изображаются в виде темных квадратиков.

```
data = ParallelTable[Boole[PrimeQ[x + y I, GaussianIntegers → True]],
      {x, -50, 50}, {y, -50, 50}];
ArrayPlot[data]
```



Функция ParallelMap[f, expr] делает то же самое, что комбинация функций Parallelize и Map. Следующий пример с параллельной работой 4-х процессоров взят также из справочной службы математики.

Map[(Pause[1]; f[#])&, {a, b, c, d}] // AbsoluteTiming

{4.000191, {f[a], f[b], f[c], f[d]}}

ParallelMap[(Pause[1]; f[#])&, {a, b, c, d}] // AbsoluteTiming

{1.013850, {f[a], f[b], f[c], f[d]}}

ParallelMap может работать с любой функцией:

ParallelMap[Composition[Framed, FactorInteger], Table[(10^n-1)/9, {n,3,7}]] //TableForm

```
{{3, 1}, {37, 1}}
```

```
{{11, 1}, {101, 1}}
```

```
{{41, 1}, {271, 1}}
```

```
{{3, 1}, {7, 1}, {11, 1}, {13, 1}, {37, 1}}
```

```
{{239, 1}, {4649, 1}}
```

Функция Composition[f, g, h, ...] создает композицию функций f, g, h, ... Результат виден из примера

Composition[1+#^#&, a #&, #/(#+1)&][x]

$$1+\left(\frac{a\,x}{1+x}\right)^{\frac{a\,x}{1+x}}$$

Функция ParallelDo работает подобно Do, но параллельно. Пример для двух процессоров:

Do[Pause[1];f[i], {i, 4}]//AbsoluteTiming

{4.000006, Null}

ParallelDo[Pause[1];f[i], {i, 4}]//AbsoluteTiming

{2.000003, Null}

ParallelSum является параллельной версией Sum, которая автоматически распределяет частичные суммирования между различными ядрами и процессорами. Выражение Parallelize[Sum[expr, iter …]] эквивалентно ParallelSum[expr, iter …].

ParallelSum[EulerPhi[i], {i, 10^6}]

303963552392

Функция ParallelProduct по своим свойствам подобна ParallelSum, но выполняет не суммирование, а умножение.

ParallelProduct[n!, {n, 10}]

665860658410473652224000000

Функция ParallelCombine [f, h[expr1, expr2, …], comb] вычисляет выражение f[h[expr1, expr2, …]] параллельно, распределяя части вычислений на всех параллельных ядрах, и объединяет частичные результаты с помощью comb. В примере функция f применяется параллельно к кускам списка (используется два ядра):

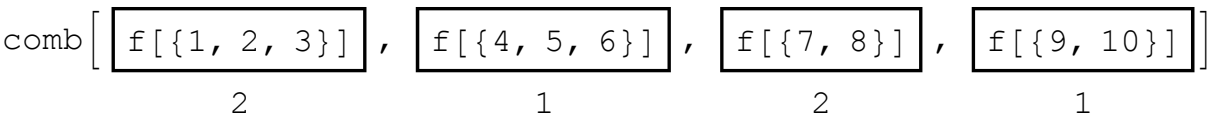
ParallelCombine[f[#]&, {1, 2, 3, 4, 5, 6, 7, 8, 9, 10},comb]

comb[f[{1, 2, 3}], f[{4, 5, 6}], f[{7, 8}], f[{9, 10}]]

Покажем, на каком ядре выполняется каждое вычисление:

ParallelCombine[Composition[Labeled[#, $KernelID]&, Framed, f],

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, comb]



Пример параллельной фильтрации списка (функция Join соединяет списки):

ParallelCombine[Select[#, PrimeQ]&, Range[20], Join]

{2, 3, 5, 7, 11, 13, 17, 19}

2.4. Mathematica на суперкомпьютерах. На любом многоядерном компьютере система Mathematica автоматически выполняет множественные части вычисления одновременно, делая параллельные вычисления обыденными. Параллельная инфрастуктура системы Mathematica легко справляется с масштабированием до размеров сети, кластера, виртуального суперкомпьютера или облака, а символьная природа системы Mathematica обеспечивает непосредственную поддержку многих парадигм программирования и моделей обмена данными.

В предыдущих примерах мы рассматривали параллельные вычисления на компьютерах с двумя или четырьмя ядрами. Теперь покажем примеры работы Mathematica на Вычислительном кластере СКИФ Cyberia (Томский государственный университет). Программы Mathematica могут выполняться на ядрах одного узла или на нескольких узлах. Один узел на кластере содержит 12 ядер:

$ProcessorCount

12

Настроим параллельные ядра.

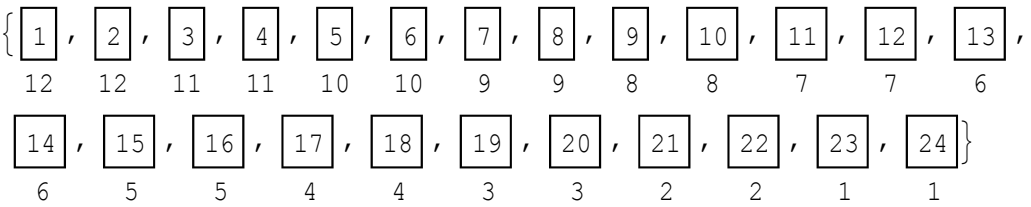
ParallelEvaluate[$ProcessID]

{6795, 7002, 7208, 7414, 7620, 7826, 8032, 8238, 8444, 8650, 8856, 9062}

Мы получили идентификатор процесса для каждого параллельного ядра.

Функции Parallelize по умолчанию использует метод Automatic для распараллеливания:

Parallelize[Map[Labeled[Frame[#], $KernelID]&, Range[24]]]



Находим показатели n чисел Мерсенна $2^n - 1$ в диапазоне от 1 до 10000:

AbsoluteTiming[Select[Range[10000], PrimeQ[2^# – 1]&]]

{145.794591, {2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941}}

AbsoluteTiming[Parallelize[Select[Range[10000], PrimeQ[2^# – 1]&]]]

{16.067398, {2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941}}

Как видим, при параллельных вычислениях на 12 ядрах в данном случае скорость вычисления выросла в 9 раз.

В отличие от традиционного программного обеспечения Mathematica дает возможность параллельных символьных вычислений и параллельных численных вычислений с неограниченной точностью, причем распараллеливание осуществляется просто.

Задачи и упражнения

4. Добавьте параллельность в функцию factory(z), осуществляющую факторизацию гауссова целого числа (реализация алгоритма 2.2 из главы 10).

5. Рассмотрите функции f(s) и g(s) из примера 2.1. Исследуйте, можно ли распараллелить эти функции с помощью параллельных аналогов Do и Table.

6. Определите параллельный аналог функции MapThread.

Литература

1. Agrawal M., Kayal H., Saxena N. PRIMES is in P. Ann. Of Math., 160:781–793, 2004.
2. Davenport J. H., Siret Y., Tournier Evelyne. Computer Algebra. Systems and Algorithms for Algebraic Computation. Academic Press, 1993.
3. Hardy G., Wright E. An Introduction to the Theory of Numbers. – 6th ed, Oxford University Press, 2008. – 621 p.
4. Koshy T. Elementary Number Theory with Application. – 2nd ed, Academic Press, 2007. – 774 p.
5. Mills W. A prime-representing function, Bull. Amer. Math. Soc. 53 (1947), 604.
6. Monier L. Evaluation and comparison of two efficient probabilistic primality testing algorithms. Theoret. Comput. Sci., 12:97-108, 1980.
7. Rabin M. Probabilistic algorithm for testing primality. J. Number Theory, 12:128-138, 1980.
8. Ribenboim P., The Little Book of Bigger Primes. – 2sd ed, Springer-Verlag, 2004. – 376 p.
9. Richardson D. Some Unsolvable Problems Involving Elementary Functions of a Real Variable. *J. Symbolic Logic* **33**, 514-520, 1968.
10. Sierpinski, W. Sur une formule donnant tous les nombres premiers. C.R. Acad. Sci. Paris, 235, 1952, 1078-1079.
11. Silverman J.H., Tate J. Rational Points on Elliptic Curves. Springer-Verlag, 1992.
12. The Haskell Programming Language. [Электронный ресурс]. – URL: http://www.haskell.org
13. Wolfram Mathematica [Электронный ресурс]. – URL: http://www.wolfram.com/mathematica/
14. Wolfram Mathematica. Русскоязычная поддержка. [Электронный ресурс]. – URL: http://www.wolframmathematica.ru/
15. WolframMathWorld URL: [Электронный ресурс:]. – URL: http://mathworld.wolfram.com/CarmichaelNumber.html
16. Wright, E.M. A prime-representing function. Amer.Math. Monthly, 58,1951, 616-618.
17. Андерсон Д. Дискретная математика и комбинаторика. – М.: Издательский дом «Вильямс», 2004. – 960 с.
18. Барендрегт Х. Ламбда-исчисление. Его синтаксис и семантика. – М.: Мир, 1985. – 606 с.
19. Воробьев Е.М. Введение в систему "МАТЕМАТИКА". // М.: "Финансы и статистика", 1998. – 262 с.
20. Дербишир Д. Простая одержимость. Бернхард Риман и величайшая нерешенная проблема в математике – М.: Астрель: CORPUS, 2010. – 463с.
21. Дэвенпорт Г. Высшая арифметика: введение в теорию чисел. Изд. 2-е. – М.: Книжный дом «ЛИБРОКОМ», 2010. – 176 с.
22. Зюзьков В.М. Ленивое функциональное программирование: Учебное пособие. – 2-е, перераб. и дополн. изд. – Томск: Изд-во Том. ун-та, 2008. – 294 с.
23. Катленд Н. Вычислимость. Введение в теорию рекурсивных функций. – М.: Мир, 1983. – 256 с.
24. Кнут Д. Искусство программирования для ЭВМ, т. 1: Основные алгоритмы. – М.: Мир, 1976.
25. Кнут Д. Искусство программирования для ЭВМ, т. 2: Получисленные алгоритмы. – М.: Мир, 1977.
26. Кострикин А. И. Введение в алгебру. Часть I. Основы алгебры. М,: Физико-математическая литература, 2000, – 272 с.
27. Кострикин А. И. Введение в алгебру. Часть III. Основные структуры. М,: ФИЗМАТЛИТ, 2004, – 272 с.

28. Крэндалл Р., Померанс К. Простые числа: Криптографические и вычислительные аспекты. – М.: УРСС: Книжный дом «ЛИБРОКОМ», 2011. – 664 с.
29. Матиясевич Ю. В., Десятая проблема Гильберта, М., Наука, 1993, 223 с.
30. Мендельсон Э. Введение в математическую логику. – М.: Наука, 1976. – 320 с.
31. Пойа Д. Математика и правдоподобные вычисления. – М.: «Наука», 1975. – 464 с.
32. Седов Е. Основы работы в системе компьютерной алгебры Mathematica. [Электронный ресурс]. – URL: http://www.intuit.ru/studies/courses/4765/1039/info
33. Цагир Д. Первые 50 миллионов простых чисел // В книге: Живые числа. Сборник статей 1981 г.: Пер. с нем. – М.: Мир, 1985. – 128 с.