МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Национальный исследовательский
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

В.М. Зюзьков

ЭКСПЕРИМЕНТЫ В ТЕОРИИ ЧИСЕЛ

ТОМСК
«Издательство НТЛ»
2019

УДК 510.2+510.5+510.6
ББК 22.12я73
З98

Зюзьков В.М. Эксперименты в теории чисел. – Томск: Изд-во
З98 НТЛ, 2019. – 348 с.

ISBN 978-5-89503-626-6

Экспериментальная математика – это тот раздел математики, который имеет дело, прежде всего, с кодированием и передачей идей в математическом сообществе с помощью экспериментальных исследований гипотез и менее формальных воззрений, а также с помощью анализа полученных данных. Как правило, математики не публикуют гипотезы. В этой книге, наоборот, читатель найдет много недоказанных утверждений, об истинности которых можно только предполагать. Но в этом и заключается цель книги – показать, как с помощью системы компьютерной алгебры Mathematica можно приходить к открытиям. Демонстрируется «кухня» компьютерных экспериментов. Предметной областью является элементарная теория чисел, так как эксперименты и исследования играют ключевую роль в изучении теории чисел. Автор стремился к «замкнутому» изложению материала.

Книга рассчитана на математиков, использующих системы компьютерной алгебры как инструмент в своих исследованиях. Также она будет полезна студентам и аспирантам математических направлений университетов, изучающих курс компьютерной алгебры.

УДК 510.2+510.5+510.6
ББК 22.12я73

Разработано и издано при поддержке
Регионального научно-образовательного
математического центра ММФ ТГУ
(государственное задание МИНОБРНАУКИ РФ
№ 1.12877.2018/12.1).

Рецензенты:

П.А. Крылов, профессор, доктор физико-математических наук, заведующий кафедрой алгебры НИ ТГУ;

М.И. Слободской, профессор, доктор физико-математических наук, заслуженный работник высшей школы Российской Федерации, профессор кафедры высшей математики ТГАСУ

ISBN 978-5-89503-626-6

© В.М. Зюзьков, 2019
© Оформление. Дизайн.
ООО «Издательство НТЛ», 2019

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ 6

ВВЕДЕНИЕ 9
§ 1. Экспериментальная математика 9
§ 2. Mathematica 15
Основные возможности и состав Mathematica 16
Способы выполнения приложений, написанных в системе Mathematica 18

Глава 1. **Основы языка Wolfram** 20
§ 1. Принципы языка 20
Понимание математической нотации 21
§ 2. Представление и упрощение данных в языке Wolfram 25
Представление данных 25
Автоматическое упрощение 27
Упрощение алгебраических выражений 30
§ 3. Процедурное программирование 35
§ 4. Функциональное программирование 40
§ 5. Программирование с правилами преобразований 46
§ 6. Параллельные вычисления и функциональное программирование 51
Евклидовы числа 58

Глава 2. **Вхождение в тему** 61
§ 1. Примитивные пифагоровы тройки 61
§ 2. Аликвотные дроби 64
§ 3. Гипотеза Коллатца 67
§ 4. Метазадача 75
§ 5. «Самоописательная последовательность» Голомба 80
§ 6. Три последовательности от Рональда Грэхэма 84
§ 7. Функция Аккермана 88
§ 8. Числа Улама 93
§ 9. Алгоритм Евклида 98
Линейные диофантовые уравнения 105

Глава 3. **Сравнения. Кольца классов вычетов** 111
§ 1. Определения и основные свойства 111
§ 2. Возведение в степень 115
§ 3. Нахождение обратного элемента по модулю 117
§ 4. Решение сравнений 120
Решение линейных сравнений 120
Китайская теорема об остатках 121
Сравнение любой степени 125
§ 5. Функция Эйлера φ 127
Прообраз функции EulerPhi 132
Теорема Эйлера 135
Итерация функции Эйлера 137
Задачи с функцией Эйлера 141
§ 6. Квадратичные вычеты 146
Квадратичные вычеты и невычеты 146
Когда $p - 1$ является квадратичным вычетом по модулю p? 150
Критерий Гаусса 151
Закон квадратичной взаимности 152
Символ Якоби 154
Извлечение квадратного корня по модулю 163
§ 7. Криптография с открытым ключом 167

Глава 4. **Мультипликативные функции** 175
§ 1. Первые понятия 175
§ 2. Число и сумма делителей 179
§ 3. Совершенные числа и числа Мерсенна 189
§ 4. Функция Мёбиуса 194
Задачи с функцией Мёбиуса 199
§ 5. Свёртка Дирихле 201

Глава 5. **Числа Фибоначчи** 216
§ 1. Определения и некоторые свойства 216
Многочлен для чисел Фибоначчи 221
Подстановка Фибоначчи 223
§ 2. Фибоначчиевая система счисления 224
§ 3. Последовательность Fibonacci(n) mod n 228
Когда в последовательности F_n mod n встречаются нули? 229

Подпоследовательность $F_p \bmod p$, p – простое.....231
Подпоследовательности $F_{q\times p} \bmod q\times p$, p – простое 233
Предположения без доказательств.....238

Глава 6. **Примитивные корни**.....242
§ 1. Порядок числа по модулю 242
§ 2. Примитивные корни – образующие в U(m).....246
§ 3. Периодические десятичные дроби.....254

Глава 7. **Производящие функции** 258
§ 1. Формальные степенные ряды.....258
§ 2. Производящие функции в математике и в языке Wolfram.....264
§ 3. Преобразования Дирихле.....276

Глава 8. **Вокруг теоремы Эйлера о суммах делителей** 281
§ 1. Последовательности Эйлера.....281
Разбиения 281
Последовательность с производящей функцией QPochhammer[x, x] 283
Последовательность 1, 2, 5, 7, 12, 15, 22, 26, 35, 40, 51, … 285
§ 2. Теоремы Эйлера.....290
Гипотеза Эйлера о законе чисел, относящемся к суммам их делителей 290
Рекуррентная формула о числе разбиений.....293

Глава 9. **Распознавание простых и составных чисел** 295
§ 1. Псевдопростые числа Ферма.....295
§ 2. Числа Кармайкла 298
§ 3. Сильно вероятные простые числа.....300
§ 4. Вероятные тесты на простоту, связанные с числами Фибоначчи.....306

Глава 10. **Сильно составные числа**.....311
§ 1. Вычисление сильно составных чисел.....311
§ 2. Итерация функции, определяющей число делителей 325

СПИСОК ЛИТЕРАТУРЫ.....339

ПРЕДМЕТНЫЙ И ПЕРСОНАЛЬНЫЙ УКАЗАТЕЛИ.....344

ПРЕДИСЛОВИЕ

Если вам угодно иметь характеристику научного метода в трех словах, то, по-моему, вот она:
догадывайтесь и испытывайте.

Дьёрдь Пойа

В настоящее время развивается экспериментальная математика: открытие новых математических закономерностей путем компьютерной обработки большого числа примеров. Такой подход не столь убедителен, как короткое доказательство, но может быть убедительнее длинного, сложного доказательства и в некоторых случаях вполне приемлем. Компьютеры иногда позволяют получить неформальные аргументы в пользу того или иного предположения, а иногда, наоборот, опровергнуть казавшиеся правдоподобными гипотезы. Компьютерные вычисления также поставляют первичную информацию, позволяющую обнаруживать новые свойства изучаемых объектов и выдвинуть новые гипотезы.

Экспериментальная математика – это тот раздел математики, который имеет дело, прежде всего, с кодированием и передачей идей в математическом сообществе с помощью экспериментальных исследований гипотез и менее формальных воззрений, а также с помощью анализа полученных данных [2, p. 243–265].

Как правило, математики не публикуют гипотезы. В этой книге, наоборот, читатель найдет много недоказанных утверждений, истинность которых можно только предполагать. Но в этом и заключается цель книги – показать, как с помощью системы компьютерной алгебры Mathematica можно приходить к открытиям. Демонстрируется «кухня» компьютерных экспериментов: вычисления дают набор исходных фактов, выдвигаются математические предположения, последующие эксперименты обнаруживают контрпримеры или дополнительно подтверждают предполагаемую закономерность. И в некоторых случаях догадки приводят к доказательствам.

Предметной областью является элементарная теория чисел, так как эксперименты и исследования играют ключевую роль в изучении тео-

рии чисел. Автор стремился к «замкнутому» изложению материала. Читателей книги знакомят с программированием на языке Wolfram, и с помощью этого языка дается необходимая математическая теория, в рамках которой формулируются различные задачи. Работа с примерами с помощью компьютера поможет вам изучить предмет и найти новые результаты самостоятельно!

Из интервью, данного математиком Юрием Маниным [1]:

«*Михаил Гельфанд:*

– Вы упомянули про компьютеры. Что изменилось в математике с появлением компьютера?

Юрий Манин:

– В чистой математике что изменилось? Появилась уникальная возможность делать физические эксперименты в ментальной реальности. Можно пробовать невероятные вещи. Точнее, невероятные нельзя, а то, что можно, Эйлер умел делать и без компьютера. Гаусс тоже умел. Но теперь то, что умели Эйлер и Гаусс, может делать любой математик, сидя за своим письменным столом. И если у него не хватает воображения, чтобы различить какие-то контуры в этой платоновской реальности, он может поэкспериментировать. Ему пришла в голову хорошая мысль, что что-то равно чему-то, – он сядет и посчитает одно значение, другое, третье, миллионное. Более того, появились люди с математическим, но компьютерно-ориентированным умом. Точнее сказать, это люди, которые были и раньше, но без компьютера им чего-то не хватало. Опять, к ним относился Эйлер, в той мере, в какой он был только математиком – он был гораздо больше, чем только математиком, – но Эйлер как математик сейчас бы работал с компьютерами со страстью. Еще Рамануджан, человек, который даже и математики толком не знал. Или вот, например, мой коллега по институту Дон Загир (Don Zagier). У него совершенно математический ум, который идеально приспособлен для работы с компьютером, ему компьютер помогает исследовать вот эту платоновскую реальность, чрезвычайно эффективно при этом».

В книге наряду с традиционным математическим текстом много фрагментов программ на языке Wolfram. Опишем используемый формат. Входные выражения, которые система Mathematica вычисляет, начинаются каждый раз на новой строке и выделяются полужирным шрифтом (как правило, *Arial*). Выходное выражение – значение вычисленного выражения – помещается сразу после входа без выделения жирности шрифтом *Times New Roman*. Функции языка Wolfram, введен-

ные пользователем, пишутся с маленькой буквы и курсивом, как в программных фрагментах, так и в остальном тексте. Встроенные функции языка Wolfram, в соответствии с синтаксисом, пишутся с большой буквы прямым шрифтом.

Используется версия 11.3 системы Mathematica.

О нумерации. На протяжении каждой главы идет сплошная нумерация, своя для теорем, примеров, алгоритмов, ссылок на формулы и др., начинающаяся с 1. Символ ■ обозначает конец доказательства теорем.

ВВЕДЕНИЕ

> Но этот способ мысли и проверки утверждений *<речь идет о доказательстве – В. З.>* помещен теперь в новый контекст. Это очень устойчивый, поддерживающий и стимулирующий контекст, который только помогает нашим целям. Он возвышает наше понимание. Он увеличивает наш оружейный арсенал. Он делает нас сильнее.
>
> *Стивен Кранц. Изменчивая природа математического доказательства. Доказать нельзя поверить*

§ 1. Экспериментальная математика

Экспериментальная математика – это тип математического исследования, в котором вычисление используется для исследования математических структур и определения их фундаментальных свойств и закономерностей. Как и в экспериментальной науке, экспериментальная математика может быть использована для составления математических предсказаний, которые затем могут быть подтверждены или опровергнуты на основе дополнительных вычислительных экспериментов. Эти исследования должны завершаться доказательством – «золотым стандартом» математической истины.

Но математика больше чем доказательство. В прошлом данную концепцию отстаивали и Дьёрдь Пойа [95, 96], и Имре Лакатос [87], убежденные сторонники эвристических методов и квазиэмпирической природы математики.

Общая идея поиска математических результатов путем проведения вычислительных экспериментов имеет продолжительную историю. Плодотворность такого поиска определяется, конечно, применяемыми инструментами, но даже в тех случаях, когда использовались только карандаш и бумага, математики часто приходили к правильным результатам. Пьер Ферма[1], Леонард Эйлер[2], Карл Гаусс[3] и Бернхард Риман[4] про-

[1] Последняя теорема Ферма, сформулированная, по-видимому, в 1637 г., доказана в 1994 г.

вели много часов своей жизни, выполняя (умственные) вычисления, чтобы установить «возможные истины» (многие, но не все, они впоследствии доказали).

Великий индийский математик Сриниваса Рамануджан был экспериментатором в математике: он свободно входил во вселенную математических возможностей и делал расчеты для того, чтобы найти интересные и значимые факты – и только затем получал свои удивительные тождества и строил теории, основанные на них [100]. Явление хаоса – знаменитый аттрактор Лоренца – было замечено в компьютерном эксперименте Эдварда Лоренца в 1962 году [38]. Универсальное поведение при итерации одномерных отображений, наиболее известным из которых является логистическое отображение $f\colon x \to \lambda x(1 - x)$, было открыто Митчеллом Фейгенбаумом в 1975 году с помощью электронного калькулятора [36]. Многие аспекты фракталов были найдены Бенуа Мандельбротом в 1970-х годах с использованием компьютерной графики [92]. Отметим, что понятия «фрактал» и «хаос» возникли на основании визуализации без предшествующего теоретического обоснования. Методы экспериментальной математики при изучении простых математических систем, таких как клеточные автоматы, простые программы, бесконечные последовательности и др., показывающие сложное поведение, приведены в книге «Новый вид науки» Стивена Вольфрама [49].

Современная экспериментальная математика возникла после появления программных систем символьных вычислений (синонимом является «система компьютерной алгебры»). Наиболее популярные из таких систем – Maple (сайт: https://www.maplesoft.com/) и Mathematica. Современной экспериментальной математике посвящены книги [2, 3, 4].

В данной книге используется система Mathematica с языком программирования Wolfram. Как с помощью Mathematica проводить математические исследования см. в [29]. В математическом аппарате, реали-

---

[2] Эйлер был непревзойдённым мастером формальных выкладок и преобразований. Наглядными примерами мастерства Эйлера служат его вычисление суммы обратных квадратов и получение необычайной формулы, связывающей суммы делителей натуральных чисел (см. [95. Глава VI] и главу 8 данной книги).

[3] Гаусс начиная с 1792 г., – когда ему было всего 15 лет! – искал асимптотический закон простых чисел. Закон им был найден, но строгое доказательство было получено математиками только в 1896 г.

[4] Гипотеза Римана, несмотря на все усилия математиков, остается недоказанной с 1859 г.

зованном в Mathematica и Wolfram Language, сегодня сделан акцент на «практическую», вычислительную математику. На данный момент она покрывает практически всю математику, существовавшую в 19 веке и ранее. Но компания Wolfram Research планирует использовать все виды новых встроенных функций и стандартизованных объектов данных для того, чтобы реализовать конструкции, необходимые для представления современной чистой математики. Разумеется, создание системы, позволяющей работать с чистой математикой, которая имела бы широкие возможности, была бы удобной и понятной, потребует многих лет тщательной разработки [51].

То, что делает современную экспериментальную математику отличной (как деятельность) от классической концепции и практики математики, заключается в том, что экспериментальный процесс рассматривается не только как предшественник доказательства. Скорее, экспериментирование рассматривается как значительная часть математики в своем собственном праве, которая будет опубликована, рассмотрена другими и (особенно важно) будет способствовать нашим общим математическим знаниям. В частности, это дает эпистемологический статус утверждениям, которые, несмотря на значительный объем экспериментальных результатов, до сих пор не были формально доказаны[5], а в некоторых случаях никогда не могут быть доказаны.

Как известно, метод математического исследования и, с другой стороны, изложение и опубликование его результатов – это разные вещи. Приведем две цитаты об этом.

«Гаусс, прежде чем опубликовать какой бы то ни было труд, подвергал свое изложение самой тщательной обработке, прилагая крайнюю заботливость о краткости изложения, изяществе методов и языка, не оставляя при этом следов той черновой работы, которой он до этих методов достиг. Он говаривал, что когда здание построено, то не оставляют тех лесов, которые для постройки служили; поэтому он не только не торопился с опубликованием своих работ, но оставлял их вылеживаться не то что годами, а десятками лет, часто к этой работе по временам возвращаясь, чтобы довести её до совершенства. <…> Свои исследования по эллиптическим функциям, главные свойства которых он открыл за

[5] В качестве примера. На настоящее время проверено, что более 10^{13} первых нетривиальных нулей дзета-функции Римана удовлетворяют гипотезе Римана. В предположении истинности гипотезы Римана написано около 10 тысяч математических работ [43].

34 года до Абеля и Якоби, он не удосужился опубликовать в течение 61 года, и они были опубликованы в его «Наследии» примерно еще через 60 лет после его смерти. Эйлер поступал как раз обратно Гауссу. Он не только не разбирал лесов вокруг своего здания, но иногда даже как бы загромождал его ими. Зато у него видны все подробности самого способа его работы, что у Гаусса так тщательно скрыто. За отделкой Эйлер не гнался, работал сразу вчистую и публиковал в том виде, как работа получилась; но он далеко опередил печатные средства Академии, так что сам сказал, что академическим изданиям хватит его работ на 40 лет после его смерти; но здесь он ошибся – их хватило больше чем на 80 лет» [85, с. 331, 332].

«… Гаусс был очень замкнутым человеком и вёл затворнический образ жизни. Он не опубликовал массу своих открытий, и многие из них были заново сделаны другими математиками. В публикациях он уделял больше внимания результатам, не придавая особого значения методам их получения и часто заставляя других математиков тратить массу сил на доказательство его выводов. Эрик Темпл Белл, один из биографов Гаусса, считает, что его необщительность задержала развитие математики по меньшей мере на пятьдесят лет; полдюжины математиков могли бы прославиться, если бы получили результаты, годами, а то и десятилетиями хранившиеся у него в архиве» [61, с. 156].

Современной экспериментальной математике более свойственно изложение результатов, следуя Эйлеру. Возможно, потому, что синтаксис языков программирования современных систем символьных вычислений очень близок к математической нотации (см. §1 главы 1).

Борвейн и Бейли [4] под экспериментальной математикой подразумевают методологию математических исследований с компьютером для следующих видов познания:

1. Возникновение инсайта (интуитивного прозрения).
2. Обнаружение новых образцов и отношений.
3. Использование графической визуализации для обнаружения математических фактов.
4. Тестирование и особенно фальсификация догадок.
5. Изучение возможного результата, чтобы увидеть, стоит ли для него проводить формальное доказательство.
6. Предложение подходов для формального доказательства.
7. Замена длинных «ручных» выводов на компьютерные выводы.
8. Подтверждение аналитически полученных результатов.

Типичным сценарием использования этой экспериментальной методологии является следующее. Обратите внимание на «диалог» между человеком и компьютером, что очень характерно для этого подхода к математическим исследованиям:

1. Изучение математической проблемы для определения аспектов, которые необходимо лучше понять.
2. Использование компьютера для изучения этих аспектов, путем разработки конкретных примеров, создания графиков и т. д.
3. Отмечая шаблоны или другие явления, очевидные в компьютерных результатах, которые относятся к исследуемой проблеме.
4. Использование компьютерных средств для определения или «объяснения» этих шаблонов.
5. Формулирование цепочки достоверных догадок, которые, если это так, разрешат изучаемый вопрос.
6. Решение о том, указывает ли потенциальный результат в желаемом направлении и стоит ли полноценная попытка формального доказательства.
7. Выполнение дополнительных компьютерных экспериментов для получения большей уверенности в ключевых гипотезах.
8. Подтверждение этих гипотез строгими доказательствами.
9. Использование символического вычислительного программного обеспечения для двойной проверки аналитических выводов.

Многие примеры в этой книге включают в себя вычисления, которые приводят к численным или символьным результатам, а также к визуализации. Mathematica часто используется таким образом, чтобы проверить гипотезы и решить, продолжать ли дальнейшее изучение проекта или идеи. Интерпретирующий язык Wolfram и другие черты Mathematica позволяют этому типу исследования эффективно и немедленно выполнить программу, чтобы увидеть результаты.

Пример 1. (Amer. Math. Monthly. 1977. V. 84, № 6). Известна задача об определении числа R_n областей, образуемых $n(n-1)/2$ хордами, которые соединяют n фиксированных точек на окружности при предположении, что никакие три хорды не пересекаются внутри круга (рис. 1).

Результаты при $n = 2$, 3, 4 и 5 наводят на мысль, что $R_n = 2^{n-1}$, но практические попытки получить 32 области для $n = 6$ безуспешны. Удается построить только 31 область. Следовательно, R_6 должно быть равно 31? (рис. 2).

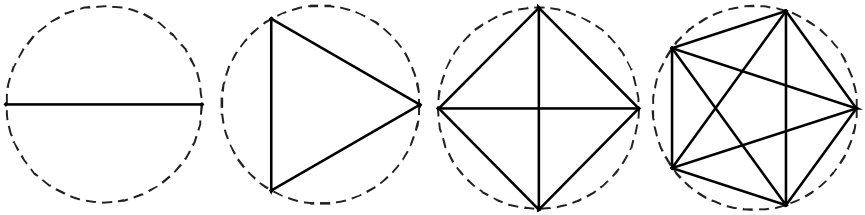


Рис. 1. Хорды в круге

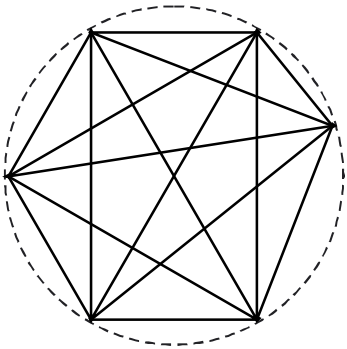


Рис. 2. Контрпример: $R_6 = 31$

Но какова тогда зависимость R_n от n? Mathematica может определить закономерность этой последовательности:

```
FindSequenceFunction[{1, 2, 4, 8, 16, 31}, n
```

$$\frac{1}{24}(24 - 18n + 23n^2 - 6n^3 + n^4)$$

Каким образом сам С. Вольфрам использует Mathematica в качестве инструмента экспериментальной математики см. в [54].

С. Вольфрам в статье о Рамануджане [56] говорит о значении экспериментального подхода в математике следующее[6]:

[6] Цитируется по переводу на русский язык.
См. https://habrahabr.ru/company/wolfram/blog/306250/

«Доказать многие из результатов Рамануджана оказалось сложной задачей. Отчасти потому, что создание своего рода контекста, необходимого для доказательства, требует наращивания гораздо более абстрактных и концептуально сложных структур. <…> Рамануджан обладал замечательными навыками. Но мне кажется, что для того, чтобы пойти по его стопам, нужно быть смелым: не оставаться в комфорте хорошо зарекомендовавших себя математических теорий, а вместо этого выходить в более широкую математическую вселенную и начинать экспериментировать. Для того чтобы поместить многие открытия Рамануджана в более широкий и более абстрактный контекст, понадобился почти век. Рамануджан вдохновляет нас сделать большой шаг вперед – даже до того, как был понят более широкий контекст. И я надеюсь, что гораздо больше людей воспользуется теми инструментами, которые мы имеем сегодня, чтобы последовать примеру Рамануджана и сделать великие открытия в экспериментальной математике».

§ 2. Mathematica

Устройство нашего мира непостижимо
без знания Mathematica.

Приписывается Роджеру Бэкону

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, 11.3 версия, – 2018 г. [48]. Об истории Mathematica см. [53].

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.

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, функциональное 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, выравнивание 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 данным Wolfram|Alpha и вычислениям.

Wolfram|Alpha – база знаний и набор вычислительных алгоритмов. Wolfram|Alpha в ответ на запрос вычисляет ответ, основываясь на собственной базе знаний, содержащей данные о математике, физике, астрономии, химии, биологии, медицине, истории, географии, политике, музыке, кинематографе, а также информацию об известных людях и интернет-сайтах. Модуль Wolfram|Alpha способен переводить данные между различными единицами измерения, системами счисления, подбирать общую формулу последовательности, находить возможные замк-

нутые 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 генерацию 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.

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 программой 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 файлы Mathematica, 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 поддерживается 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.

Глава 1

ОСНОВЫ ЯЗЫКА WOLFRAM

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, поэтому с июня 2013 г. он получил официальное название язык Wolfram [55]. Этот язык с несколькими парадигмами 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. На языке Wolfram программу можно написать так, что разобраться в ней можно без инструкции, просто понимая язык математики.

§ 1. Принципы языка

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 нумерует ввод выражения надписью

In[*n*]:=

и после этого ядро вычисляет значение выражения. Соответствующий вывод результата располагается под вводом и отмечается надписью

Out[n]= [7].

[7] Во всех примерах в этой книге нумерация ввода/вывода отсутствует; это можно сделать при настройке параметров системы. Все вводимые выражения в тексте напечатаны полужирным шрифтом и вывод, если он показывается, всегда следует сразу после ввода.

Пример 1. Вычисляем факториал:

100!

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
```

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[*a*^4 – *b*^4]

$$(a-b)(a+b)\left(a^2+b^2\right)$$

Понимание математической нотации

В язык Wolfram были внедрены возможности понимания синтаксиса, очень близкого к стандартной математической нотации [52].

Но синтаксис языка не совпадает полностью с математической нотацией. Имеются важные особенности.

- Прописные и строчные буквы считаются разными знаками.
- Все встроенные символы, в частности имена Wolfram-функций, начинаются с большой буквы (функции и переменные, которые вы определяете, не имеют такого ограничения[8]).
- 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}.

[8] И в тексте этой книги имена, вводимые пользователем, дополнительно отмечаются курсивом.

• Двойные квадратные скобки, [[ ]], используются для индексации подвыражений в выражениях. Например, запись {1, *a*, Cos[x + y]}[[3]] дает третий элемент списка, т.е. Cos[$x + y$].

• Запятые используются для разделения аргументов (список – это тоже функция с именем List и с любым количеством аргументов).

Большие математические выражения – в отличии от фрагментов обычного текста – часто представляют собой результаты вычислений и создаются автоматически. Большие усилия разработчиков языка Wolfram были направлены на обработку подобных выражений и на то, чтобы сделать их более понятными для людей. Так как традиционная математическая нотация представляет математические объекты, а не математические процессы, то была разработана нотация для агоритмов и реализована в языке Wolfam. Математическая нотация часто содержит двумерные структуры, поэтому наряду со стандартной текстовой формой ввода, Wolfram поддерживает общую, двумерную форму ввода, графику и управление интерактивными моделями, произвольно смешиваемую с вводом текста.

С. Вольфрам пишет в [52] о том, что наряду с созданием Mathematica одной из его главных целей было распространить вычислительную мощь на все виды технической и математической работы. Эта задача имеет две части: одна, как вычисления происходят внутри, и другая, – как люди направляют эти вычисления для получения того, что они хотят.

Одно из самых больших достижений Mathematica заключается в сочетании высокой общности вычислений изнутри и сохранении практичности, основанной на преобразованиях символьных выражений, где символьные выражения могут представлять данные, графику, документы, формулы – да что угодно. Язык Wolfram дает возможность людям сообщать Mathematica о том, какие вычисления они хотят произвести.

«*Вот в чём заключалась для меня эта работа в отношении Mathematica*: *я попробовал представить, какие вообще вычисления люди будут производить, какие фрагменты в этой вычислительной работе повторяются снова и снова. А затем, собственно, я дал имена этим фрагментам и внедрил в качестве встроенных функций в Mathematica*» [52].

Как оказалось, в отличие от естественных человеческих языков, для обычной математической нотации можно сделать очень хорошее приближение, которое компьютер сможет понимать.

К сожалению, математическая нотация развивалась в результате непродуманных случайных исторических процессов. Только в середине XX века Бурбаки акцентировали внимание на использовании обозначений там, где это только возможно, любым способом сводя использование потенциально неточного текста к минимуму [62].

Математика 20 века восприняла влияние формалистких взглядов Н. Бурбаки [99]:

«…стиль практически всех научных работ по математике в период от пятидесятых по семидесятые годы постепенно изменился в сторону формализации, стал в той или иной степени походить на формально-бурбакистскую манеру, притом, как правило, этот процесс происходил неосознанно».

В области чистой математики произошли серьёзные изменения, которые заключались в переходе от работ, полных текста и лишь с основными алгебраическими и вычислительными выкладками к работам, насыщенным обозначениями.

По-видимому, язык Wolfram – следующий этап в формализации, который дает возможность вычислять некоторые математические тексты.

Много усилий было потрачено разработчиками на представлении входных данных для вычисления на языке Wolfram. При синтаксическом разборе вычисляемого текста необходимо различать операнды и операторы и знать приоритеты операторов – какие действуют сильнее, а какие слабее в отношении операндов. Математики, как оказалось, имеют в основном одинаковые мнения об упорядоченности приоритетов математических операций. Этими общепринятыми представлениями руководствовались при создании языка Wolfram, в частности, при решении задачи интерпретации вводимых математических обозначений.

В языке принят ввод строковый, большей частью одноуровневый, или использующий шаблоны из меню Palletes (палитры). Например, имеется шаблон для интеграла, в котором заполняются слоты подынтегрального выражения, переменной и так далее. Примерами шаблонов являются также обозначения предела или «двухэтажной» дроби. И когда шаблон вставляется в документ, то всё выглядит как надо, однако всё ещё содержится информация о том, что это за шаблон, и программа понимает, как это интерпретировать.

Более сложная задача – возможность ввода в свободной форме на ангийском языке. Это также частично было решено в модуле Wolfram|Alpha.

Серьезной проблемой является реализация необходимого требования при вычислениях – однозначная семантика, которая, с одной стороны, обеспечивается синтаксисом языка программирования, а с другой – используется знакомая математическая нотация. Например, символы *i* и *e* могут в математических текстах быть именами переменных, а могут обозначать соответственно мнимую единицу и основание натуральных алгоритмов.

Чтобы ликвидировать данную неоднозначность используются символы с двойным начертанием. Таким образом, символы с двойным начертанием, вводимые с помощью меню Palletes (палитры),

$$\mathbb{i},\ \mathbb{e},\ \mathbb{d}$$

суть специфические объекты, обозначающие мнимую единицу, число *e* и знак дифференциала в интеграле соответственно. Преимущество введения символа дифференциала в интеграле, отличного от простого символа *d*, показывает следующий пример:

$$\int x \operatorname{Cos}[d + x]\, \mathbb{d}x$$

$$\operatorname{Cos}[d + x] + x \operatorname{Sin}[d + x]$$

В обычном текстовом вводе в Wolfram подобные неоднозначности решены простым путём: *все встроенные объекты Wolfram начинаются с заглавной буквы.* Например, мнимая единица и основание натуральных алгоритмов также обозначаются I и E, соответственно.

В языке Wolfram результаты выполнения (Out) – объекты той же природы, что и входные данные (In), то есть их можно редактировать, использовать их части по отдельности, использовать их фрагменты в качестве входных данных и так далее.

Для многих математических выражений имеется несколько форм представления. Во-первых, все выражения имеют строковую входную форму (InputForm). Возможно, иногда InputForm не совсем похожа на традиционную математическую нотацию. Но она очень близка. И, несомненно, содержит все особенности структуры и форм записи обычной математической нотации. Важно то, что ни у кого, владеющего

обычной математической нотацией, не возникнет трудностей в интерпретации этого выражения.

Стандартная форма (StandartForm) выражения – двумерное изображение выражения, представленного во входной форме. Между InputForm и StandartForm имеется взаимно-однозначное соответствие.

Традиционная форма (TraditionalForm) является изображением выражения в виде, наиболее близком к традиционной математической нотации. Следующий пример показывает три формы выражения:

```
Integrate[x Sin[x]^2, x]
```

$$\int x \sin[x]^2 \, dx$$

$$\int x \sin^2(x) \, dx$$

С. Wolfram считает, что StandartForm в языке Wolfram лучше и яснее для представления математических выражений [52].

Данное ниже описание языка Wolfram недостаточно, чтобы программировать без затруднений, но в какой-то мере позволяет понимать приводимые программы. Для изучения языка полезна литература [7, 27, 47, 50, 78].

§ 2. Представление и упрощение данных в языке Wolfram

Представление данных

Язык Wolfram имеет дело со многими различными видами вещей: математическими формулами, списками, графиками и т.д. Хотя они выглядят различными, но представляются в языке одним способом – все они есть выражения.

Типичным примером выражения в языке 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 используете запись, подобную $f[x, y, z, \ldots]$. Например, $x + y$ – 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 (таблица).

Исходное представление и полное представление

«Естественное» (общепринятое) представление	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]
$a \to b$ (правило для подстановки)	Rule[a, b]
$a = b$ (присваивание)	Set[a, b]
$a == b$ (равенство логическое)	Equal[a, b]

Вы можете получить 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]
```

Отметим, что возведение в степень имеет два «естественных» представления: символ "^" между основанием и показателем, например x^n, или используется надстрочный знак x^n.

Объект f в выражении $f[x, y, z, \ldots]$ называется **головной частью** выражения. Головную часть выражения можно получить, используя функцию Head.

```
Head[g[a, b]]
g

Head[a b]
Times

Head[876]
Integer

Head[3.14]
Real

Head[π]
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[9].

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

Тогда его стандартная полная форма есть

$$\text{List}[a, \text{List}[3, b], \text{List}[6, \text{Plus}[x, y]]].$$

Поэтому подвыражение $x + y$ можно извлечь из выражения *list* с помощью индексирования:

```
list[[3, 2]]
```
$x + y$

Точно также доступ к подвыражению $x + y$ в выражение

$$a + 3\text{^}b + f[4, x + y]$$

получаем с помощью такого же индексирования

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

поскольку имеем

```
FullForm[a + 3^b + f[4, x + y]]
```
Plus[Power[3, *b*], *a*, *f*[4, Plus[*x*, *y*]]]

Редко кто пользуется FullForm. Разве что в случаях, когда работа Mathematica непонятна. Однако с этой формой весьма утомительно работать. Возможно, именно поэтому синтаксис языков наподобие Lisp кажется столь трудным – по сути это синтаксис FullForm в Wolfram.

Автоматическое упрощение

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.

[9] Если вводимое выражение оканчивается символом ‘;’, как ниже, то вывод не показывается, хотя значение вычисляется.

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 другие 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.

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

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

```
2x + y + (3/2)x
```

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

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:

```
2x + √2x + 3x
```

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

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.

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?

```
√I^4
1

√((-1)(-1))
1
```

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.

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:

$$\int \frac{1}{x^4 - 1}\,dx$$

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

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

Доказательство.

$$\text{Simplify}\left[\left(\frac{a^3 - 729}{9a^2+81a+729}\right)^3+\left(\frac{-a^3+243a+729}{9a^2+81a+729}\right)^3+\left(\frac{27a^2+243a}{9a^2+81a+729}\right)^3\right]$$

a ■

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]
```

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

```
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[(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 его записать 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*]

Упрощение с допущениями. Преобразования выражений функциями, подобными Expand и Factor, всегда корректны, независимо от значений входящих переменных. Но в общем случае для других функций 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]]
```
$2a + 2\sqrt{a - \sqrt{-b}}\sqrt{a + \sqrt{-b}}$

```
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. гл. 3, следствие из теоремы 10)[10]:

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

[10] Функция Mod[x, y] вычисляет остаток от деления целых чисел x на y.

Сейчас используется тот 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]]}
```

Пример 2. С. Рамануджану принадлежит следующее утверждение.

Для вещественных чисел a, b и c таких, что $a + b + c = 0$ выполнено равенство

$$2(ab + b\,c + ca)^4 = a^4(b - c)^4 + b^4(c - a)^4 + c^4(a - b)^4. \qquad (1)$$

Это легко доказать, применяя упрощение:

```
FullSimplify[2(a b + b c + c a)^4 == a^4(b – c)^4 + b^4(c – a)^4 + c^4(a – b)^4,
             a + b + c == 0]
True
```

Формула (1) справедлива и для комплексных чисел:

```
FullSimplify[2(a b + b c + c a)^4 == a^4(b – c)^4 + b^4(c – a)^4 + c^4(a – b)^4,
             (a + b + c == 0) && Element[{a, b, c}, Complexes]]
True
```

Дело в том, что FullSimplify в (1) использует только свойства умножения и сложения, справедливые и для кольца комплексных чисел.

§ 3. 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 литературе, указанной в начале данной главы, или в help'е Mathematica – интерактивном электронном учебнике, содержащем более чем 100000 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]
```
$$\frac{47}{12} + \mathrm{Sin}[c]$$

Атомарные выражения суть числа и символы. Числа делятся на четыре класса: целые, рациональные, вещественные и комплексные. Ко-

личество цифр в числе не ограничивается. Символы – это последовательность букв, цифр и знака $, не начинающаяся с цифры. К символам относятся и именованные константы, имеющие предопределенные 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 – это удобная форма 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} в этом контексте называется **итератор.** 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, т.е. 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 относительно функции List: 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 четные 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[11].

```
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 Wolfram 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 = 0;
If[x ≠ 0, 1/x, Indeterminate]
Indeterminate
```

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

[11] Булевская функция PrimeQ[*n*] проверяет, является ли целое число *n* простым. Лексема && обозначает операцию конъюнкции.

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
```

Конечно, возможно определять рекурсивные функции. Например, пусть дана последовательность $f_1 = 1, f_2 = 1$ и $f_n = (n - 1) f_{n-1} + (n - 2) f_{n-2}$ для $n > 2$. Определяем рекурсию обычным образом, но добавим глобальный счетчик m для подчета количества сложений при рекурсивном вызове:

```
f[1] = 1;
f[2] = 1;
f[n_] := (m = m + 1;(n – 1) f[n – 1] + (n – 2) f[n – 2])
```

Вычислим значения первых 30 членов последовательности f.

```
m = 0;
xs = f /@ Range[30];
```

Выясним, чему равны значения f_{30} и m.

```
xs[[–1]]
1008337762980636369901233425099 97

m
2178278
```

Количество рекурсивных вызовов очень велико, но небольшое изменение синтаксиса определения функции дает возможность сохранять промежуточные значения функции при рекурсивном вызове.

```
ff[1] = 1;
ff[2] = 1;

ff[n_] := ff[n] = (m = m + 1;(n – 1) ff[n – 1] + (n – 2) ff[n – 2])
m = 0; ys = ff /@ Range[30];
xs = ys
True

m
28
```

Рекурсия стала эффективной.

§ 4. 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 Wolfram – выражения – имеют как раз рекурсивную структуру.

До этого мы рассматривали встроенные функции Wolfram и сразу применяли их для решения задачи. Сейчас перейдем к изучению мощных и полезных конструкций функционального программирования и

будем программировать 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[12]. 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.

---

[12] 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 [26, 77]) используют ламбда-исчисление как теоретическую модель и как промежуточный код при трансляции. В нашем изложении функционального программирования в Wolfram мы не будем явно использовать ламбда-исчисление; желающие могут познакомиться с ним по книгам [60, 72].

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 является сущностью функционального программирования в Mathematica.

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

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

Можем оперировать с неопределенной одноместной функцией *f* и одноуровневым списком:

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

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]
```

Функции **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}}
```

Функция **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 головную часть 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]
```

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]]
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
```

Применение функций из списка к соответствующим аргументам:

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

Суперпозиция функций. Важнейшей особенностью функционального программирования является последовательное применение, или суперпозиция функций. Встроенные функции Nest и Fold автоматизи-

руют 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 последовательных 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

Функции **Fold** и **FoldList** являются аналогами Nest и NestList применительно к функциям от двух аргументов. Выражение

```
FoldList[f, x,{a, b, …}]
```
порождает список

$\{x,\ f[x, a],\ f[f[a, x], b], \ldots\}$,

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}]
```
$\{0, a, a + b, a + b + c, a + b + c + d\}$

§ 5. 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. Они 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.

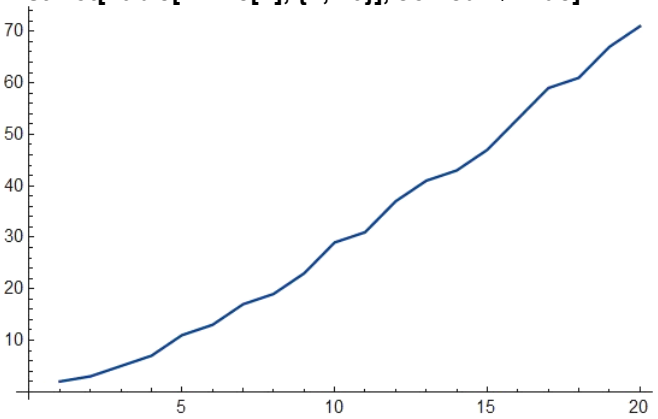
Пользователь Mathematica обычно встречает правила, когда использует некоторые встроенные функции. Например, правила появляются в выводе функции Solve, применяемой для решения уравнения.

```
solve = Solve[a x^2 + b x + c == 0, {x}]
```
$$\left\{\left\{x \to \frac{-b - \sqrt{b^2 - 4ac}}{2a}\right\}, \left\{x \to \frac{-b + \sqrt{b^2 - 4ac}}{2a}\right\}\right\}$$

Правила также используются для указания значений опций встроенных функций, например, на следующем графике первых 20 простых чи-

сел правило Joined → True указывает, что точки графика должны быть соединены.

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



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 + 3x^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 $f[x_] \to x$^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 головной частью 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]
{100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110}
```

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

В языке Wolfram 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. 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 выраже-

нию, 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 1. 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[0] := 0;
f[n_; n > 0] := "0" <>
ToString[FromDigits[Nest[Flatten[# /. {0 → {0, 1}, 1 → {1, 0}}]&, {0}, n]]]
f[3]
01101001
```

Для каждого *n* результат является строкой символов, начинающихся с 0. Встроенная функция FromDigits преобразует список цифр в соответствующее число, но при этом отбрасывается самый левый 0 в списке цифр. Чтобы результат получался с 0 слева, полученное функцией

FromDigits число преобразуется в строку символов (это делает функция ToString), а потом в начало строки добавляется символ ″0″ (операция <>). Функция Flatten преобразует вложенный список в одноуровневый список. Функция *f* определена для случаев: $n = 0$ и $n > 0$. Детали смотрите в справочной службе Mathematica.

§ 6. Параллельные вычисления и функциональное программирование

Высокопроизводительные вычисления необходимы для решения масштабных ресурсоемких технических проблем. Вместе со скоростью вычислений необходима точность результатов. Mathematica предоставляет все это и, более того, интегрирует важные технологии вычислений 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 языка Wolfram являются следствием использования большого множества алгоритмов, тщательно проанализированных и оптимизированных для лучшей производительности с точки зрения скорости, памяти и надежности. Во многих случаях функции языка 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 автоматизированы и могут быть расширены 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 Wolfram, 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 языком Wolfram, 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.

Имеются различные способы подключения ядер в Mathematica, но остановимся только 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.

Рассмотрим примеры параллельных вычислений в языке Wolfram. 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]&]]
{25.250863, {546, 974}}

AbsoluteTiming[Parallelize[Select[Table[n, {n, 500, 1000}],
PrimeQ[#! – 1]&]]]
{7.851218, {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. теорему 9, §3 главы 4):

```
AbsoluteTiming[Select[Range[15000], If[PrimeQ[#], PrimeQ[2^# – 1],
False]&]]
{345.932258, {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]&]]]
{99.222824, {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]]
{0.573752, {2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279}}

AbsoluteTiming[Parallelize[Select[Range[2000], mersenneQ]]]
{0.231016, {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.

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 §4. Требуется из списка пар составить новый список, переставляя элементы в парах. Запрограммируем процедурный подход:

```
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»[13].

Выясним, какие типичные для функционального программирования функции могут вычисляться параллельно.

Задача 2. Возвести в квадрат каждый элемент списка. Параллельные вычисления автоматизируются при использовании функции Map[14]:

```
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}

---

[13] Принцип **прозрачности по ссылкам** утверждает: «Значение целого не меняется, когда какая-то часть целого меняется на равную часть». Подробности см. в [77].

[14] Кроме полной формы функции Map[*f*, *xs*] используется также синтаксис *f* /@ *xs*.

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.

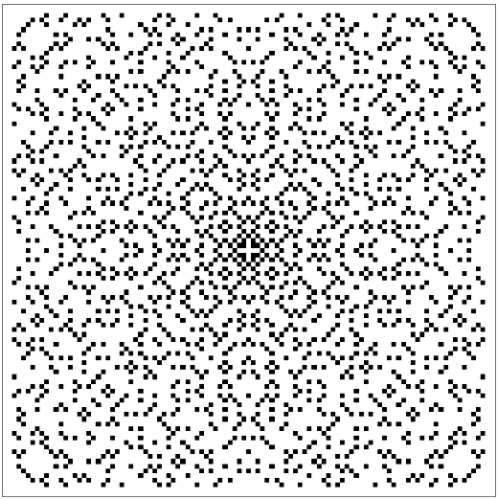
Ряд функций имеют параллельные аналоги. Начнем с функции Table. Эта функция вместе с Parallelize вычисляет параллельно:

```
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 то же 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[15].

```
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 процессоров взят также из справочной службы математики[16].

```
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]}}
```

[15] Дополнительный параметр (опция) GaussianIntegers → True функции PrimeQ говорит о том, что тестируются гауссовы простые числа.

[16] Pause[*n*] совершает паузу в вычислениях продолжительностью, как минимум, *n* секунд.

ParallelMap может работать с любой функцией[17]:

```
ParallelMap[Composition[Framed, FactorInteger], Table[(10^n – 1)/9,
{n, 3, 7}]] // Column
```

```
{{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}}
```

Используемая функция Composition[*f*, *g*, *h*, …] создает композицию функций *f*, *g*, *h*, … . Результат виден из примера

```
Composition[1 + #^#&, a #&, #/(# + 1)&][x]
```

$$1+\left(\frac{a\,x}{1+x}\right)^{\frac{a\,x}{1+x}}$$

Предложенного описания языка Wolfram, на наш взгляд, достаточно, чтобы понять приведенные в книге программы. Некоторые программы будут дополнительно комментироваться.

Евклидовы числа

Продемонстрируем использования языка Wolfram при рассмотрении евклидовых чисел. Евклидово доказательство бесконечности множества простых чисел использует числа

$$E_n = \prod_{i=1}^{n} p_i + 1,$$

где p_i есть *i*-е по порядку простое число. Эти числа называются *евклидовыми*. Исследуем, как часто среди евклидовых чисел встречаются простые числа.

---

[17] Функция FactorInteger[*n*] выдает список простых делителей вместе с их кратностью для целого числа *n*.

Программа для вычисления евклидовых чисел:

```
euclid[n_] := 1 + Apply[Times,Prime[Range[n]]]
```

Номера простых евклидовых чисел E_n, когда $n \le 2000$:

```
Parallelize[Select[Range[2000], PrimeQ[euclid[#]]&]]
```

{1, 2, 3, 4, 5, 11, 75, 171, 172, 384, 457, 616, 643, 1391, 1613}

Первые семь простых евклидовых чисел:

```
euclid /@ {1, 2, 3, 4, 5, 11, 75}
```

{3, 7, 31, 211, 2311, 200560490131,
171962010545840643348334056831754301958457563589574256043877110505832165523856261308397965147955578800999455782202456522693290629520826275682227566369411 1}

Мы можем оценить количество десятичных цифр у найденных чисел:

```
IntegerLength[euclid[#]]&/@{1, 2, 3, 4, 5, 11, 75, 171, 172, 384, 457,
616, 643, 1391, 1613}
```

{1, 1, 2, 3, 4, 12, 154, 425, 428, 1115, 1368, 1939, 2038, 4951, 5862}

Последнее найденное простое евклидово число E_{1613} имеет 5862 цифр, последнее рассмотренное число E_{2000} имеет 7483 цифры:

```
IntegerLength[euclid[2000]]
```

7483

Являются ли евклидовы числа свободными от квадратов[18]?

```
AllTrue[Parallelize[SquareFreeQ /@ (euclid /@ Range[44])], #&]
```

True

Функция AllTrue[*xs*] выдает истину только в том случае, когда все элементы списка *xs* есть True.

Первые 44 евклидовых числа свободны от квадратов, но проверка уже следующего числа требует слишком много времени. Ниже напечатано 45-е евклидово число.

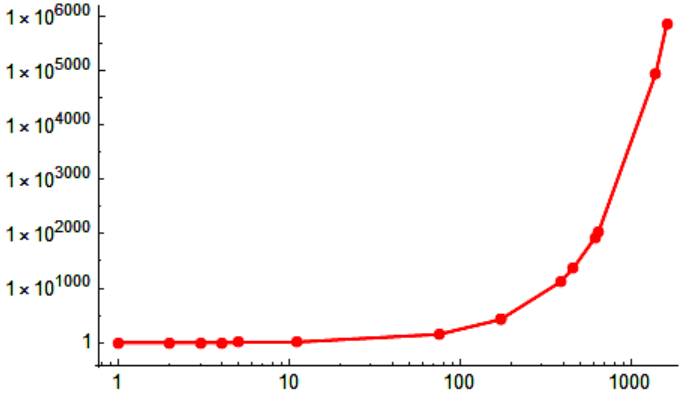
```
euclid[45]
```

391955881491631233831618045544211752597386773361987484678041832907965403827371 91

[18] Целое число свободно от квадратов, если оно не делится ни на какой квадрат.

Диаграмма в логарифмическом масштабе по обеим осям наглядно представляет разброс найденных простых евклидовых чисел:

```
ListLogLogPlot[{#, euclid[#]}& /@ {1, 2, 3, 4, 5, 11, 75, 171, 172, 384, 457,
616, 643, 1391, 1613}, PlotStyle → Red, Mesh → All, Joined → True]]
```



Неизвестно, бесконечно ли множество простых евклидовых чисел [35]?

Глава 2

ВХОЖДЕНИЕ В ТЕМУ

§ 1. Примитивные пифагоровы тройки

В этом параграфе рассмотрим программы, связанные с **пифагоровыми тройками**[19] – тройками натуральных чисел, удовлетворяющих соотношению $x^2 + y^2 = z^2$. Такие тройки чисел, хотя и названы по имени Пифагора, открыты были значительно раньше. Если числа x, y и z взаимно просты, то пифагорова тройка (x, y, z) называется **примитивной тройкой**. Для того чтобы найти все пифагоровы тройки достаточно сначала найти все примитивные тройки, а все остальные тройки могут быть получены из примитивных умножением на то или иное натуральное число. Следующая теорема, принадлежащая Евклиду, показывает, как генерировать все примитивные тройки.

Теорема 1 [69, с. 161]. Для любой пары взаимно простых натуральных чисел (m, n), таких, что одно из них есть нечетное, а другое четное, и $m > n > 0$, три числа $(m^2 - n^2, 2mn, m^2 + n^2)$ образуют примитивную тройку. Более того, каждая примитивная тройка имеет такой вид.

Пример 1.

{*m*, *n*} = {$10^9 + 1$, 10^9}

{$m^2 - n^2$, $2mn$, $m^2 + n^2$}

{2000000001, 2000000002000000000, 2000000002000000001}

Напишем программу для получения всех примитивных пифагоровых троек с гипотенузой, не превосходящей данного натурального числа h. Для этого будем перебирать натуральные числа m и n, для которых $m^2 + n^2 < h$. Решим это неравенство[20].

---

[19] Используют также терминологию «пифагоровы треугольники».

[20] Reduce – одна из функций для решения систем уравнений или неравенств; выдает ответ в виде булевского выражения – ограничения на значения переменных.

```
Reduce[{m^2 + n^2 < h, m > n > 0, h > 0}, {m, n}]
```

$$h > 0 \;\&\&\left(\left(0 < m \le \frac{\sqrt{h}}{\sqrt{2}} \;\&\&\, 0 < n < m\right) \| \left(\frac{\sqrt{h}}{\sqrt{2}} < m < \sqrt{h} \;\&\&\, 0 < n < \sqrt{h - m^2}\right)\right)$$

Лексемы && и || обозначают конъюнкцию и дизъюнкцию. В привычной математической записи m и n должны удовлетворять соотношению

$$\left(0 < m \le \sqrt{h/2} \wedge 0 < n < m\right) \vee \left(\sqrt{h/2} < m < \sqrt{h} \wedge 0 < n < \sqrt{h - m^2}\right). \qquad (1)$$

Определим функцию *pairs*[h, m], которая для заданного m, меньшего $\sqrt{h}$, находит все пары $\{n, m\}$, где числа n и m взаимно просты, различной четности, $n < m$ и выполнено соотношение (1). Рассматривая по отдельности два диапазона для m из соотношения (1) и комбинируя с различной четностью m, получаем четыре варианта. Полностью определение функции *pairs* состоит из четырех правил. Для каждого m список пар $\{n, m\}$ создается только с помощью одного правила. Функция Ceiling[x] выдает наименьшее целое число, не меньшее значения x.

```
pairs[h_, m_] := {#, m} & /@
  Select[Range[1, m - 1, 2], GCD[m, #] == 1 &] /;
    EvenQ[m] && (0 < m <= Sqrt[h/2])

pairs[h_, m_] := {#, m} & /@
    Select[Range[1, Ceiling[Sqrt[h - m^2]] - 1, 2], GCD[m, #] == 1 &] /;
      EvenQ[m] && (Sqrt[h/2] < m < Sqrt[h])

pairs[h_, m_] := {#, m} & /@
  Select[Range[1, m - 1, 2], GCD[m, #] == 1 &] /;
    OddQ[m] && (0 < m <= Sqrt[h/2])

pairs[h_, m_] := {#, m} & /@
    Select[Range[1, Ceiling[Sqrt[h - m^2]] - 1, 2], GCD[m, #] == 1 &] /;
      OddQ[m] && (Sqrt[h/2] < m < Sqrt[h])
```

Например,

```
pairs[100, 8]
```

{{1, 8}, {3, 8}, {5, 8}}

Следующая функция *mns*[*h*] соединяет в один список все пары {*n m*} для данного числа *h*. Второй аргумент функции *pairs* (это значение *m*) пробегает все натуральные числа, меньшие $\sqrt{h}$:

```
mns[h_] := Flatten[pairs[h,#] & /@Range[Ceiling[√h] - 1],1]]
```

Например,

```
pairs[100, 8]
```

{{1, 2}, {2, 3}, {1, 4}, {3, 4}, {2, 5}, {4, 5}, {1, 6}, {5, 6}, {2, 7}, {4, 7}, {6, 7}, {1, 8}, {3, 8}, {5, 8}, {2, 9}, {4, 9}}

И наконец, функция *threeps*[*h*] из полученных пар {*n*, *m*} создает список примитивных пифагоровых троек, у которых гипотенуза меньше *h*:

```
threeps[h_] := {#[[2]]^2 – #[[1]]^2, 2#[[2]]  #[[1]], #[[2]]^2 + #[[1]]^2}&
/@ mns[h]
```

Например,

```
threeps[100]
```

{{3, 4, 5}, {5, 12, 13}, {15, 8, 17}, {7, 24, 25}, {21, 20, 29}, {9, 40, 41}, {35, 12, 37}, {11, 60, 61}, {45, 28, 53}, {33, 56, 65}, {13, 84, 85}, {63, 16, 65}, {55, 48, 73}, {39, 80, 89}, {77, 36, 85}, {65, 72, 97}}

Количество примитивных пифагоровых троек с гипотенузой, меньшей 10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, равно соответственно

```
Length[threeps[#]]&/@ Table[10^k, {k, 1, 7}]
```

{1, 16, 158, 1593, 15919, 159139, 1591579}

Известно [69, с. 163], что число $\Delta(N)$ примитивных пифагоровых троек с гипотенузой меньшей *N*, удовлетворяет соотношению

$$\lim_{N\to\infty} \frac{\Delta(N)}{N} = \frac{1}{2\pi}.$$

Сравним полученные результаты (из каждого вычисленного значения $\Delta(N)/N$ вычитаем число $1/(2\pi)$):

```
N[Table[Length[threeps[10^k]]/10^k, {k, 1, 7}] – 1/(2π)]
```

{–0.0591549, 0.000845057, –0.00115494, 0.000145057, 0.0000350569, –0.0000159431, 0.00000295691}.

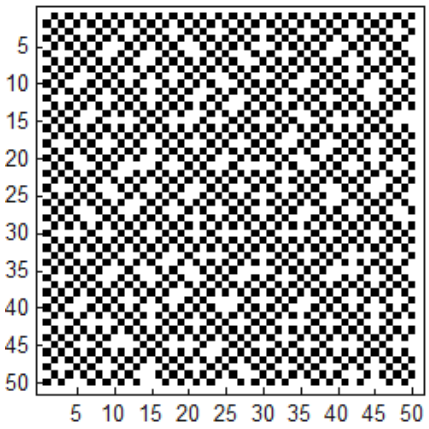
Для больших *h* получение значения *threeps*[*h*] требует уже заметного времени, поэтому можно воспользоваться параллельным программиро-

ванием. Для реализации параллельного программирования в данном случае достаточно *mns* определить с использованием встроенной функции Parallelize:

```
mns[h_] := Flatten[Parallelize[pairs[h,#] & /@ Range[Ceiling[√h] - 1]],1]
pqQ[n_,m_]:=(OddQ[n]∧EvenQ[m]∨OddQ[m]∧EvenQ[n])∧GCD[n,m]==1
```

Изобразим на декартовой плоскости множество всех пар $\{n, m\}$, для которых $\max(n, m) \le 50$.

```
ArrayPlot[Table[Boole[pqQ[i, j]], {i, 50}, {j, 50}],
          FrameTicks → {Range[5, 50, 5], Range[5, 50, 5]}]
```



§ 2. Аликвотные дроби

Аликвотная дробь [90, с. 67] есть дробь вида $1/n$, где n – положительное целое. Так как древние египтяне представляли дроби в виде суммы различных аликвотных дробей, то такие суммы называются **египетскими дробями**. Известно несколько алгоритмов для представления рациональных чисел p/q, где p и q – целые числа с условием $0 < p < q$, в виде суммы различных аликвотных дробей, т.е. как египетская дробь. Но неизвестен алгоритм, который давал бы минимальное количество слагаемых или наименьших по величине знаменателей [34]. Любую правильную дробь можно представить в виде египетской дроби бесконечным числом способов с различным количеством слагаемых и с как угодно большими знаменателями у некоторых аликвотных дробей.

Для этого достаточно воспользоваться тождеством

$$\frac{1}{a}=\frac{1}{a+1}+\frac{1}{a(a+1)}.$$

Покажем, что *жадный* алгоритм, который на каждом шаге добавляет наибольшую возможную аликвотную дробь, всегда заканчивается.

Доказывать будем с помощью математической индукции по p. Базис индукции, когда $p = 1$, выполняется тривиально.

Индуктивный шаг. Пусть дано рациональное число p/q, для которого $(p, q) = 1$, $p > 1$ и $0 < p/q < 1$. Предположим, что любую дробь с числителем, меньшим p, можно с помощью жадного алгоритма представить в виде конечной суммы аликвотных дробей. Применив один шаг алгоритма к p/q, найдем аликвотную дробь $1/s$, для которой $1/(s-1) > p/q > 1/s$ (равенство $p/q = 1/s$ невозможно, так как $(p, q) = 1$). Имеем $p/q - 1/s = (ps - q)/(qs)$. С другой стороны, если мы умножим обе стороны неравенства $1/(s-1) > p/q$ на $q(s - 1)$, то получим $q > p(s - 1)$, откуда следует, что $p > ps - q$. Это показывает, что числитель дроби p/q строго больше числителя дроби $(ps - q)/(qs)$ – разности, полученной после первого шага алгоритма. По индуктивной гипотезе $(ps - q)/(qs)$ выражается как сумма аликвотных дробей $1/u_1 + 1/u_2 + \ldots + 1/u_k$. Поэтому $p/q = 1/s + 1/u_1 + 1/u_2 + \ldots + 1/u_k$, что заканчивает доказательство индуктивного шага.

Напишем программу, реализующую жадный алгоритм для разложения рационального числа p/q, $p > 1$ и $0 < p/q < 1$ в сумму аликвотных дробей. Функция $g(x)$ выдает $\{1/q, x - 1/q\}$, где $1/q$ – наибольшая из всех аликвотных дробей, меньших x $(0 < x < 1)$:

```
g[x_] := Module[{q = 2}, While[1/q > x, q++]; {1/q, x – 1/q}]
```

Проверим

```
g[2/13]
```

$$\left\{\frac{1}{7},\frac{1}{91}\right\}$$

Функция g[#[[2]]]&, в отличие от g, не только выдает результат вида $\{1/q, x - 1/q\}$, но и имеет аргумент такого же вида. В основной функции *greedy*, которая выдает список слагаемых – аликвотных дробей, повторяется вызов g[#[[2]]]& до тех пор, пока второй элемент аргумента-пары не станет равен нулю (для этого используется функция NestWhileList):

```
greedy[x_] := #[[1]]& /@ Rest[NestWhileList[g[#[[2]]]&, {0, x}, #[[2]] > 0&]]
greedy[4/5]
```

$$\left\{\frac{1}{2}, \frac{1}{4}, \frac{1}{20}\right\}$$

```
4/5 == 1/2 + 1/4 + 1/20
```

True

Жадный алгоритм часто выдает сумму аликвотных дробей не в оптимальном виде – не минимальное количество слагаемых и/или большие по величине знаменатели дробей. Например, для дробей p/q, $1 < p < q \le 30$, самыми «громоздкими» из полученных египетских дробей являются дроби:

$$\frac{4}{17} = \frac{1}{5} + \frac{1}{29} + \frac{1}{1233} + \frac{1}{3039345};$$

$$\frac{14}{19} = \frac{1}{2} + \frac{1}{5} + \frac{1}{28} + \frac{1}{887} + \frac{1}{2359420};$$

$$\frac{7}{23} = \frac{1}{4} + \frac{1}{19} + \frac{1}{583} + \frac{1}{1019084};$$

$$\frac{4}{25} = \frac{1}{7} + \frac{1}{59} + \frac{1}{5163} + \frac{1}{53307975};$$

$$\frac{27}{29} = \frac{1}{2} + \frac{1}{3} + \frac{1}{11} + \frac{1}{148} + \frac{1}{28328} + \frac{1}{1003066152}.$$

Но, оказывается, «ручной» выбор одного из слагаемых и применение жадного алгоритма дает приемлемые египетские дроби.

```
greedy[4/17 – 1/6]
```

$$\left\{\frac{1}{15}, \frac{1}{510}\right\}$$

```
greedy[14/19 – 1/6]
```

$$\left\{\frac{1}{2}, \frac{1}{15}, \frac{1}{285}\right\}$$

```
greedy[4/25 – 1/10]
```

$$\left\{\frac{1}{17}, \frac{1}{850}\right\}$$

***greedy*[27/29 – 1/29]**
$\left\{\frac{1}{2},\frac{1}{3},\frac{1}{16},\frac{1}{1392}\right\}$

Таким образом, имеем

$$\frac{4}{17}=\frac{1}{6}+\frac{1}{15}+\frac{1}{510};$$

$$\frac{14}{19}=\frac{1}{2}+\frac{1}{6}+\frac{1}{15}+\frac{1}{285};$$

$$\frac{4}{25}=\frac{1}{10}+\frac{1}{17}+\frac{1}{850};$$

$$\frac{27}{29}=\frac{1}{2}+\frac{1}{3}+\frac{1}{16}+\frac{1}{29}+\frac{1}{1392}.$$

Египетские дроби ставят ряд трудных и по сей день нерешенных математических проблем. О египетских дробях смотри сайт [8].

§ 3. Гипотеза Коллатца

Рассмотрим некоторые вопросы, связанные с **последовательностями Коллатца**. Проблема Коллатца поставлена немецким математиком Лотаром Коллатцем (1910–1990) в 1937 г.

Пусть a_0 – натуральное число. Образуем последовательность по правилу

$$a_n=\begin{cases}a_{n-1}/2, \text{ если } a_{n-1} \text{ – четное число,}\\ 3a_{n-1}+1, \text{ если } a_{n-1} \text{ – нечетное число.}\end{cases}$$

Для любого ли a_0 последовательность a_n достигает 1? Если $a_0 = 1$, то образуется цикл (4, 2, 1).

До сих пор не найден способ доказательства этой гипотезы, хотя она было проверена для всех $a_0 \le 19 \times 2^{58} \approx 5.48 \times 10^{18}$ и каждый раз соответствующая последовательность достигала 1 [33].

Эта задача также называется «$(3x + 1)$-отображение», «$(3n + 1)$-проблема», «проблема Улама» и др. Математик Пал Эрдёш так прокомментировал сложность этой гипотезы: «Математика еще не готова к реше-

нию таких задач» [94, с. 370]. Эта гипотеза популярна и литература по ней многочисленна [21, 22].

Будем предполагать, что гипотеза Коллатца справедлива, и обозначать через $cs(a_0)$ последовательность Коллатца, начинающуюся с a_0 и заканчивающуюся первой единицей. Запрограммируем вычисление $cs(a_0)$ и, в качестве примеров, вычислим последовательности $cs(37)$ и $cs(27)$:

```
f[n_Integer?EvenQ] := n/2
f[n_] := 3n + 1
cs[a0_] := NestWhileList[f, a0, # ≠ 1&]
cs[37]
{37, 112, 56, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1}

cs[27]
{27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242,
121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350,
175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167,
502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479,
1438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911,
2734, 1367, 4102, 2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732,
866, 433, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35,
106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1}
```

Последовательность $cs(27)$ содержит 112 чисел.

```
Length[%]
112
```

Для любого a_0 последовательность $cs(a_0)$ можно изобразить в виде линейного ориентированного графа $g(a_0)$ с ребрами $a_0 \to a_1$, $a_1 \to a_2$, …, $a_k \to a_{k+1}$, …, $8 \to 4$, $4 \to 2$, $2 \to 1$. Если A есть некоторое множество натуральных чисел, то объединение линейных ориентированных графов $g(a_0)$, $a_0 \in A$, без повторения одинаковых вершин, превращается в связный ориентированный граф $g(A)$, в котором из любой вершины есть единственный путь в вершину 1. По сути, $g(A)$ – дерево, но направления ребер противоположны традиционному обозначению. Напишем программу для изображения $g(A)$.

---

[21] Collatz's Sequence. § E16 в [6], с. 330–336.
[22] The $3x+1$ Problem. Ch. 7 в [28], с. 129–137.

Функция $g[a_0]$ выдает список ребер при представлении $cs(a_0)$ в виде ориентированного графа.

```
g[a0_Integer] := (Rule@@#)& /@ Partition[cs[a0], 2, 1]
g[37]
```

{37→112, 112→56, 56→28, 28→14, 14→7, 7→22, 22→11, 11→34, 34→17, 17→52, 52→26, 26→13, 13→40, 40→20, 20→10, 10→5, 5→16, 16→8, 8→4, 4→2, 2→1}

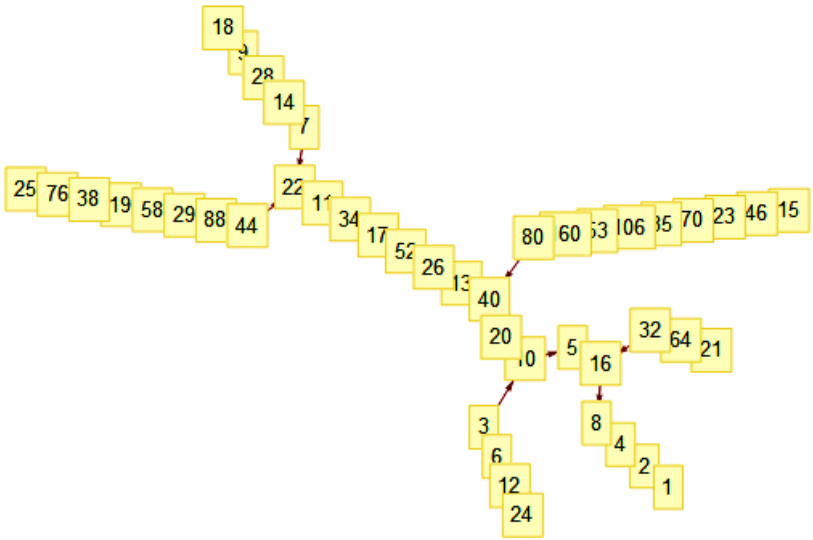
Функция $g[as]$, где as есть список начальных вершин a_0, выдает список ребер графа $g(as)$.

```
g[as_List] := Union[Flatten[g /@ as]]
g[Range[10]]
```

{2→1, 3→10, 4→2, 5→16, 6→3, 7→22, 8→4, 9→28, 10→5, 11→34, 13→40, 14→7, 16→8, 17→52, 20→10, 22→11, 26→13, 28→14, 34→17, 40→20, 52→26}

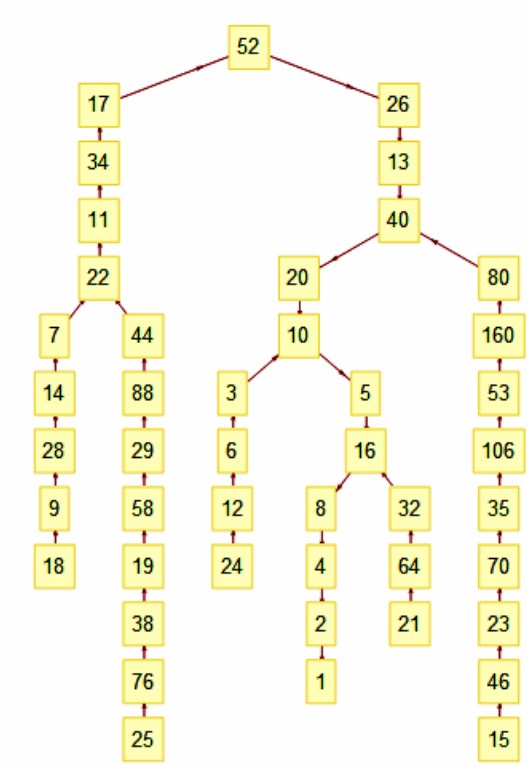
Следующий вызов функции GraphPlot изображает ориентированный граф для объединения $g(n)$, $1 \le n \le 26$.

```
GraphPlot[g[Range[26]], VertexLabeling → True, DirectedEdges → True]
```



Тот же граф изображен в виде дерева. Отметим, что, добавляя к исходному множеству вершину $n = 27$, получаем еще одну длинную ветвь с 95 вершинами.

```
TreePlot[g[Range[26]], VertexLabeling → True, DirectedEdges → True]
```



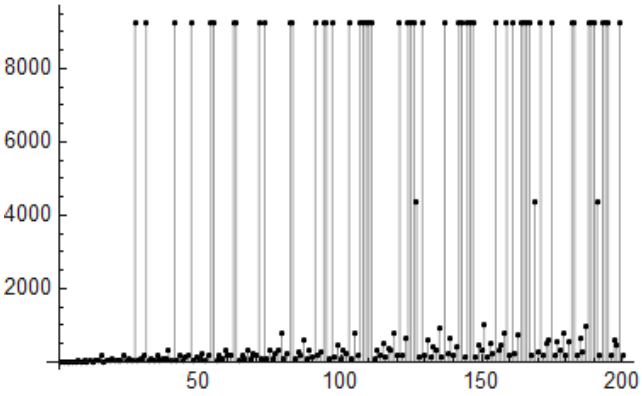
Остановимся на одном аспекте поведения последовательностей Коллатца, который, по-видимому, не встречался в литературе.

Рассмотрим наибольшие элементы последовательностей Коллатца.

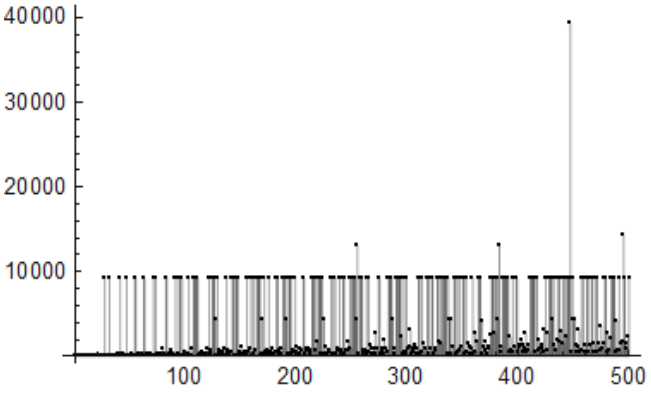
```
maxCollatz[n_] := Max[cs[n]]
```

Некоторое представление о наибольших элементах последовательностей Коллатца дают следующие графики.

```
ListPlot[maxCollatz /@ Range[200], Filling → Axis, PlotStyle →
Black, ImageSize → 300, PlotRange → All]
```




```
ListPlot[maxCollatz /@ Range[500], Filling → Axis, PlotStyle →
Black, ImageSize → 300, PlotRange → All]
```



Какие значения не может принимать функция *maxCollatz*? Ни одно нечетное число не может быть наибольшим в последовательности, так как после нечетного элемента a следует больший элемент $3a + 1$. Но и некоторые четные числа также не являются наибольшими элементами

всех последовательностей Коллатца. Об этом говорит следующее наблюдение. Наибольший элемент b последовательности, начинающейся с a_0, очевидно удовлетворяет условию $b \geq a_0$. Поэтому если все последовательности Коллатца для $a_0 = 1, 2, 3, \ldots, n$ не содержат в качестве наибольшего элемента число m, не превосходящее n, то тогда вообще все последовательности Коллатца не имеют m в качестве наибольшего элемента. В частности, найдем наибольшие элементы последовательностей Коллатца для $a_0 \leq 10$.

```
maxCollatz /@ Range[10]
{1, 2, 16, 4, 16, 16, 52, 8, 52, 16}
```

Четные числа, не превосходящие 10 и не встречающиеся в этом списке, суть 6 и 10. Поэтому 6 и 10 не могут быть наибольшими элементами любой последовательности Коллатца.

Обозначим через Y множество натуральных четных чисел, которые не являются максимальными элементами для любой последовательности Коллатца. Пусть n – натуральное число и положим $Y_n = Y \cap \{y \mid y \leq n\}$. Определим функцию $ys[n]$ на языке Wolfram для получения списка Y_n.

```
ys[n_] :=
  Module[{t = Table[{i, False}, {i, n}], m},
    Map[(m = maxCollatz[#]; If[m ≤ n, t[[m]][[2]] = True])&, Range[n]];
      #[[1]]& /@ Select[t, EvenQ[#[[1]]] && Not[#[[2]]]&]]
```

Пример вычисления Y_{20}:

```
ys[20]
{6, 10, 12, 14, 18}
```

Найдем Y_{20000} и количество чисел в этом множестве.

```
y20000 = ys[20000];
Length[y20000]
7434
```

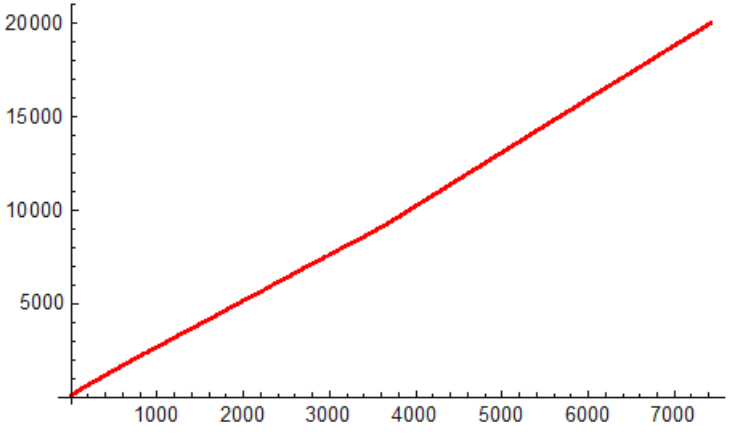
Посмотрим начало и конец списка $y20000$.

```
Short[y20000]
{6, 10, 12, 14, 18, 22, 26, 28, 30, 34, 36, 38, <<7410>>, 19974, 19978,
19980, 19982, 19984, 19986, 19988, 19990, 19992, 19994, 19996, 19998}
```

Пропущено 7410 чисел в списке из 7434 элементов.

На графике числа из списка располагаются почти на прямой.

```
gr1 =ListPlot[y20000, PlotStyle → {Red, Large}]
```

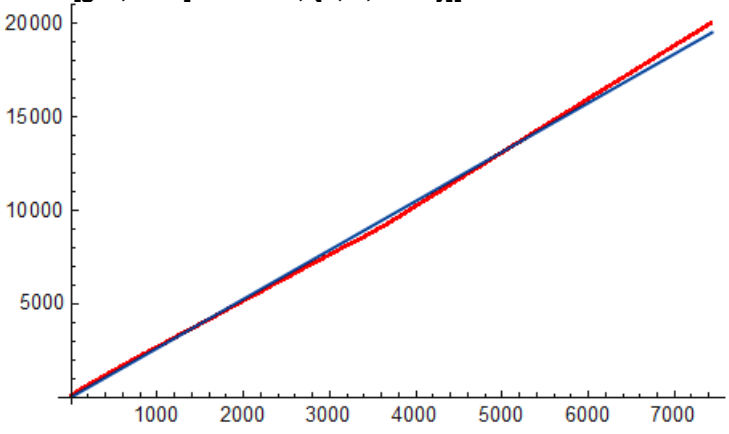


Используя функцию Fit, определим уравнение аппроксимирующей прямой, которая по номеру выдает приближенное значение n-го по возрастанию числа из Y.

```
Fit[y20000, {n}, n]
2.62829 n
```

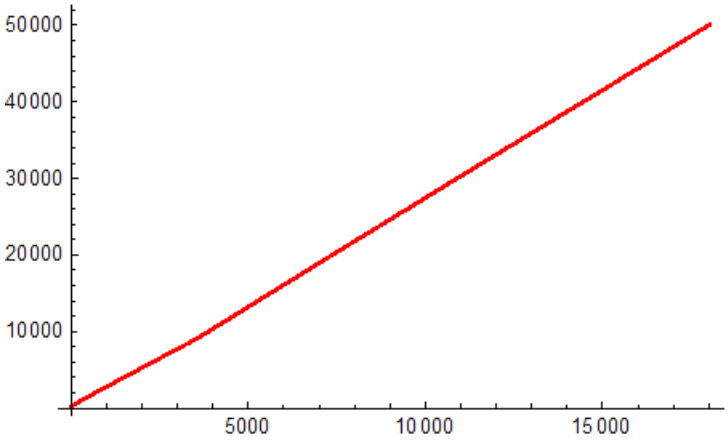
На следующем графике числа из списка y20000 и аппроксимирующая кривая $y = 2.62829n$ располагаются вместе.

```
Show[gr1, Plot[2.62829n, {n, 1, 7434}]]
```

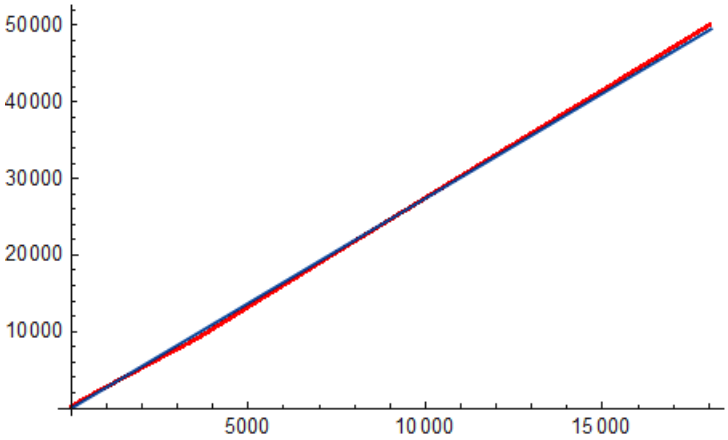


Найдем аппроксимирующую кривую для Y_{50000}.

```
Y50000 = ys[50000];
Length[y50000]
```
18060
```
gr2 =ListPlot[y50000, PlotStyle → {Red, Large}]
```




```
Fit[y50000, {n}, n]
2.74281 n
Show[gr2, Plot[2.74281n, {n, 1, 18060}]]
```



Прямая $y = 2.74281n$ визуально лучше аппроксимирует возрастание чисел из множества Y от порядкового номера.

§ 4. Метазадача

Ты знаешь, что я знаю, что ты знаешь, что я знаю.

Том Ганн. Плотское знание

Приведем пример решения метазадачи, который иллюстрирует некоторые особенности программирования со списками на языке Wolfram. Формулировка задачи взята из книги [59].

Господин *S* и господин *P* [23]

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*. (M1)

S: *Я знаю, что вам это и не удалось бы*. (M2)

P: *Ах, так… . Но тогда я их знаю!* (M3)

S: *Ну, тогда и я тоже их знаю!* (M4)

Найдите эти числа.

Приступим к решению задачи. Нам требуется найти два натуральных числа x и y, больших 1 и меньших 100. Обозначим через n и m неизвестные нам сумму и произведение соответственно чисел x и y. Будем ограничивать исходное множество пар чисел $M = \{2..99\}\times\{2..99\}$ постепенно шаг за шагом, учитывая информацию (M1) – (M4). Определим множества $S_0 = \{x + y \mid (x, y) \in M\}$ и $P_0 = \{x \cdot y \mid (x, y) \in M\}$.

1. Пусть $P_1 = \{n \in P_0 \mid$ существуют $(x_1, y_1), (x_2, y_2) \in M$ и $(x_1, y_1) \neq (x_2, y_2), n = x_1 . y_1 = x_2 \cdot y_2\}$. Когда господин *P* утверждает (M1), он тем самым говорит, что известное ему произведение n принадлежит множеству P_1.

2. Прежде чем господин *S* сказал (M2), он рассмотрел все варианты представления известной ему суммы $m = x + y$, $(x, y) \in M$, в виде двух слагаемых и убедился, что во всех вариантах $x \cdot y \in P_1$. Отсюда мы можем заключить, что сумма неизвестных чисел принадлежит множеству $S_1 = \{m \in S_0 \mid$ для любой пары $(x, y) \in M$, такой, что $x + y = m$, имеем $x \cdot y \in P_1\}$.

3. Теперь господин *P* знает следующее: произведение n можно представить в виде произведения неизвестных чисел x и y и при этом $x + y \in S_1$. Он перебирает все варианты представления n в виде произведения двух сомножителей x и y и только в одном варианте получается

[23] *S* и *P* – первые буквы слов «somme» (фр. сумма) и «produit» (фр. произведение) соответственно.

$x + y \in S_1$. После этого господин P заявляет (M3). Мы же можем определить, что произведение принадлежит множеству $P_2 = \{n \in P_1 \mid$ существует единственная пара (x, y), такая, что $n = x \cdot y$ и $x + y \in S_1\}$.

4. После (3) господин S знает, что неизвестные числа x и y таковы, что $x \cdot y \in P_2$. Он раскладывает известную ему сумму на два слагаемых $m = x + y$ и обнаруживает, что только в одном случае $x \cdot y \in P_2$. Он сообщает (M4), а мы, в свою очередь, получаем, что m принадлежит множеству $S_2 = \{m \in S_1 \mid$ существует единственная пара (x, y), такая, что $m = x + y$ и $x \cdot y \in P_2\}$.

Теперь вы можем вычислить пару (x, y) – она такова, что одновременно $x + y \in S_2$ и $x \cdot y \in P_2$. Если проделать это, то оказывается такая пара чисел единственна.

Напишем программу для вычисления x и y. Будем использовать списки для представления множеств S_0, S_1, S_2, P_0, P_1 и P_2. Удобно для каждого элемента n множества S_0 (и точно так же для элементов множеств S_1, S_2) одновременно хранить все представления n в виде пары двух слагаемых из M. Точно так же будем хранить и представления чисел в виде произведений (это касается множеств P_0, P_1 и P_2). Эти представления будем задавать в виде списков пар. Эти списки будем именовать, добавляя суффикс *Full*.

```
s0Full = Split[
            Sort[Cases[Flatten[Table[{i, j}, {i, 2, 99}, {j, 2, 99}], 1],
                 {i_, j_} /; i ≤ j],
                 (#1[[1]] + #1[[2]] ≥ #2[[1]] + #2[[2]])&],
            (#1[[1]] + #1[[2]] == #2[[1]] + #2[[2]])&];
```

Покажем, как выглядит начало и конец списка *s*0*Full*:

```
Short[s0Ful]
{{{99, 99}}, {{98, 99}}, {{97, 99}, {98, 98}}, <<189>>,
{{2, 4}, {3, 3}}, {{2, 3}}, {{2, 2}}}
```

Каждый подсписок содержит списки двух слагаемых, которые в сумме дают одинаковые числа.

Кратко опишем действие применяемых функций. Функция Table в данном случае создает список списков пар. Функция Flatten делает из этого просто список пар. Cases оставляет в этом списке только пары, у которых первый компонент не больше второго компонента, тем самым

исключаются пары, отличающиеся только порядком компонентов. Функция Sort, используя предикат

$$(\#1\ [[1]] + \#1\ [[2]] \geq \#2\ [[1]] + \#2\ [[2]])\&,$$

сортирует пары этого списка по невозрастанию суммы. Наконец, функция Split с предикатом

$$(\#1\ [[1]] + \#1\ [[2]] = = \#2\ [[1]] + \#2\ [[2]])\&$$

собирает все пары с одинаковой суммой в подсписки.

Множество различных сумм $s0$ имеет 195 элементов:

```
s0 = #[[1, 1]] + #[[1, 2]]& /@ s0Full;
Length[s0]
195
```

Аналогично создадим множества $p0$ и $p0Full$.

```
p0Full = Split[
                Sort[Cases[Flatten[Table[{i, j}, {i, 2, 99}, {j, 2, 99}], 1],
                    {i_, j_} /; i ≤ j],
                    (#1[[1]] × #1[[2]] ≥ #2[[1]] × #2[[2]])&],
                (#1[[1]] × #1[[2]] == #2[[1]] × #2[[2]])&];
Short[p0Full, 3]
{{{99, 99}}, {{98, 99}}, {{98, 98}}, {{97, 99}}, {{97, 98}},
{{96, 99}}, {{97, 97}}, {{96, 98}}, {{95, 99}}, {{96, 97}}, <<2824>>,
{{2, 8}, {4, 4}}, {{3, 5}}, {{2, 7}}, {{2, 6}, {3, 4}}, {{2, 5}}, {{3, 3}},
{{2, 4}}, {{2, 3}}, {{2, 2}}}
```

Множество различных произведений $p0$ имеет 195 элементов:

```
p0 = #[[1, 1]] × #[[1, 2]]& /@ p0Full;
Length[p0]
2843
```

Чтобы получить множества $p1Full$ и $p1$, мы из множества $p0Full$ отбираем те числа, которые имеют, по крайней мере, два различных представления в виде произведения.

```
p1Full = Cases[p0Full, a_/; Length[a] > 1];
Short [p1Full, 2]
{{{80, 99}, {88, 90}}, {{78, 98}, {84, 91}}, {{77, 96}, {84, 88}},
{{75, 96}, {80, 90}}, <<1061>>, {{2, 9}, {3, 6}}, {{2, 8}, {4, 4}},
{{2, 6}, {3, 4}}}
```

```
p1 = #[[1, 1]] * #[[1, 2]] &  /@ p1Full;
Length[p1]
```
1068

Чтобы получить множества $s1Full$ и $s1$, мы из множества $s0Full$ отбираем те числа n, для которых все пары слагаемых $\{x, y\}$, $n = x + y$, удовлетворяют условию $x\,y \in p1$.

```
s1Full = Select[s0Full, And@@((MemberQ[p1, #[[1]] × #[[2]]]&)/@#)&];
s1 = #[[1,1]] + #[[1,2]]& /@ s1Full
```
{53, 47, 41, 37, 35, 29, 27, 23, 17, 11}

Чтобы получить множества $p2Full$ и $p2$, мы из множества $p1$ отбираем те элементы n, у которых существует только единственная пара сомножителей $\{x, y\}$, $xy = n$, такая, что $x + y \in s1$.

```
p2Full = Select[p1Full, (Count[ #, x_ /; MemberQ[s1, x[[1]] + x[[2]]]] ==1)&];
p2 = #[[1, 1]] × #[[1, 2]] & /@ p2Full
```

{702, 700, 696, 690, 682, 672, 660, 646, 630, 612, 592, 570, 552, 550, 540, 532, 522, 520, 510, 496, 492, 480, 442, 430, 418, 414, 408, 400, 390, 378, 370, 364, 360, 348, 340, 336, 310, 306, 304, 294, 288, 282, 280, 276, 270, 252, 250, 246, 240, 238, 234, 232, 216, 208, 204, 198, 190, 186, 182, 176, 174, 172, 170, 168, 162, 160, 154, 152, 148, 140, 138, 130, 124, 114, 112, 110, 100, 96, 92, 76, 54, 52, 50, 28, 24, 18}

Точно так же определяем множества $s2Full$ и $s2$; мы из множества $s1$ отбираем те элементы n, у которых существует только единственная пара слагаемых $\{x, y\}$, $x + y = n$, такая, что $x\,y \in p2$.

```
s2Full = Select[s1Full, (Count[#, x_ /; MemberQ[p2, x[[1]] × x[[2]]]] ==1)&]
```
{{{2, 15}, {3, 14}, {4, 13}, {5, 12}, {6, 11}, {7, 10}, {8, 9}}}

```
s2 = # [[1, 1]] + #[[1, 2]] & /@ s2Full
```
{17}

Сумма загаданных чисел равна 17. Используя $p2$, теперь определим сами эти числа. Оказываются искомыми числами являются 4 и 13.

```
Select [s2Full [[1]], MemberQ[p2, #[[1]] × #[[2]]]&]
```
{{4, 13}}

Обсудим эту задачу. В данном решении предполагается, что искомая пара чисел существует и она единственна. Очевидно, что если изменить

условие «x и y меньше 100», то требуемой пары чисел может не существовать или таких пар будет несколько. Введем параметр *max* для задачи – верхняя граница для x и y. Будем искать числа, удовлетворяющие условию $x < max$ и $y < max$. Соберем предыдущие шаги вычислений в одну функцию *sp* с параметром *max*.

```
sp[max_] := Module[{s0Full, s1Full, s2Full, s0, s1, s2, p0Full, p1Full,
p2Full, p0, p1, p2},
s0Full = Split[Sort[Cases[Flatten[Table[{i, j}, {i, 2, max – 1},
{j, 2, max –1}], 1], {i_, j_}/;i ≤ j],
    (#1[[1]] + #1[[2]] ≥ #2[[1]] + #2[[2]])&],
    (#1[[1]] + #1[[2]] == #2[[1]] + #2[[2]])&];
s0 = (#[[1, 1]] + #[[1, 2]])&  /@ s0Full;
p0Full = Split[Sort[Cases[Flatten[Table[{i, j}, {i, 2, max – 1},
{j, 2, max –1}], 1], {i_, j_}/;i ≤ j],
    (#1[[1]] × #1[[2]] ≥ #2[[1]] × #2[[2]])&],
    (#1[[1]] × #1[[2]] == #2[[1]] × #2[[2]])&];
p0 = #[[1, 1]] × #[[1, 2]]&  /@ p0Full;
p1Full = Cases[p0Full, a_/;Length[a] > 1];
p1 = #[[1, 1]] × #[[1, 2]]& /@ p1Full;
s1Full = Parallelize[Select[s0Full, And@@((MemberQ[p1, #[[1]] ×
#[[2]]]&) /@ #)&]];
s1 = #[[1, 1]] + #[[1, 2]]&  /@ s1Full;
p2Full = Parallelize[Select[p1Full, (Count[#, a_/;MemberQ[s1,
a[[1]] + a[[2]]]] == 1)&]];
p2 = #[[1, 1]] × #[[1, 2]]& /@ p2Full;
s2Full = Parallelize[Select[s1Full, (Count[#, a_/;MemberQ[p2,
a[[1]] × a[[2]]]] == 1)&]];
s2 = #[[1, 1]] + #[[1, 2]]& /@ s2Full;
Parallelize[Select[s2Full[[1]], MemberQ[p2, #[[1]] × #[[2]]]&]]]
```

Функция Parallelize добавлена, чтобы ускорить вычисления для больших *max*. Созданная функция *sp* будет правильно работать, если решение задачи для данного *max* существует и единственно.

```
sp[100]
{{4, 13}}
```

Для использования *sp* в общем случае необходимо проделать небольшие изменения в теле функции. Если решение не существует, то списки *s*2*Full* и *s*2 будут пустыми и функция *sp* должна выдавать соот-

ветствующее сообщение. Если же решений несколько, то список *s*2*Full* будет содержать несколько пар и последняя строка в коде функции должна быть модифицирована. Соответствующая модификация функции *sp* была проделана. И вызовы функции *sp*[*max*] с различными значениями *max* показали следующее.

Наименьшее значение *max*, при котором решение существует, равно 63. При ограничении $63 \le max \le 866$ решение единственно: пара чисел 4 и 13. При условии $867 \le max \le 1000$ имеется два решения {4, 13} и {4, 61}. Вызов функции *sp*[*max*] для $max > 1000$ не осуществлялся.

§ 5. «Самоописательная последовательность» Голомба

«Самоописательная последовательность» [67, с. 87] Соломона Голомба $f_1, f_2, f_3, \ldots$ – единственная неубывающая последовательность целых чисел, обладающая тем свойством, что она содержит ровно f_k вхождений каждого k. Эта последовательность должна начинаться так:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
f_n	1	2	2	3	3	4	4	4	5	5	5	6	6	6	6	7

Для последовательности выполняется свойство $f_n \le n$ для всех n. Напишем программу для вычисления f_n. Последовательность строится поэлементно, начиная с исходного списка {1}, и на каждом шаге добавляется очередное число. Функция *h*[*g*] добавляет очередной член в список *g*. На каждом шаге надо принять решение: очередной элемент последовательности должен совпадать с предыдущим или же должен на 1 быть больше.

```
h[g_] := Appendn[g, Last[g]]/;Count[g, Last[g]] < g[[Last[g]]]
h[g_] := Append[g, Last[g] + 1]
```

Функция Last[*g*] выдает последний элемент в *g*. Функция Append[*s*, *x*] добавляет в конец списка *s* элемент *x* и удлиненный список возвращается как результат.

Если число вхождений последнего элемента *m* в список *g* меньше f_m, то первое правило для *h* удлиняет на один член список *g*, повторяя *m*. В противном случае выполняется второе правило для *h*, которое удлиняет на один член список *g*, добавляя значение $m + 1$.

Итерируя функцию *h* 99 раз, получаем первые 100 элементов последовательности Голомба.

```
Nest[h, {1}, 99]
```
{1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 9,
10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 12, 12, 12, 12, 12, 12, 13, 13, 13,
13, 13, 13, 14, 14, 14, 14, 14, 14, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16,
16, 16, 16, 17, 17, 17, 17, 17, 17, 17, 18, 18, 18, 18, 18, 18, 18, 19, 19,
19, 19, 19, 19, 19, 20, 20, 20, 20, 20, 20, 20, 20, 21, 21}

Теперь мы можем явно вычислить f_n, положив

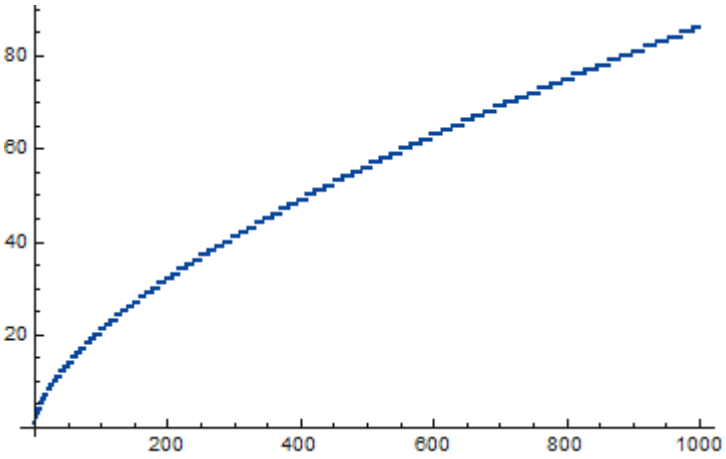
```
f[k_] := Last[Nest[h, {1}, k – 1]]
```

Имеем

```
f /@ Range[10]
```
{1, 2, 2, 3, 3, 4, 4, 4, 5, 5}

Посмотрим на график первой тысячи членов последовательности:

```
g1000 = Nest[h, {1}, 999];
ListPlot[g1000]
```



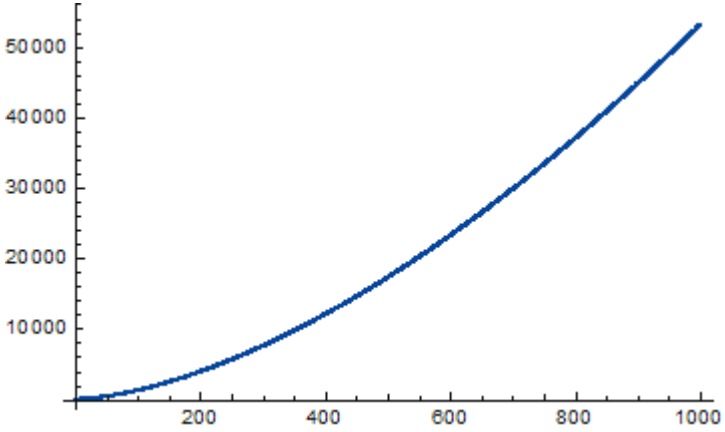
Определим новую последовательность r_n, члены которой вычисляются как наименьшее целое m, такое, что $f_m = n$. Нетрудно видеть, что $r_{n+1} - r_n = f_n$. Определим функцию $r[n]$, используя рекурсию с запоминанием промежуточных значений:

```
r[1] = 1;
r[n_] := r[n] = r[n – 1] + g1000[[n – 1]]
```

Функция $r[n]$ в данном виде позволяет вычислить значения r_n для $1 \le n \le 1000$. Если потребуется значения $r[n]$ для $n > 1000$, скажем, для $1000 < n \le 10000$, надо вычислить $g10000$, подобно тому, как это было сделано для $g1000$, и переопределить функцию r. Мы избегаем явно использовать $f[n]$, чтобы избежать повторных вычислений.

Находим первую тысячу значений последовательности r_n и получаем график этой последовательности.

```
r1000 = r /@ Range[1000];
ListPlot[r1000]
```



Какая математическая функция аппроксимирует поведение последовательности r_n? Для нахождения аппроксимации будем использовать функцию Fit языка Wolfram. Но надо сначала выбрать, какого вида функции наиболее подходящие для аппроксимации. После нескольких экспериментов были выбраны функции $x(\ln x)^4$ с точностью до числового коэффициента. Представим список $r1000$ в виде списка пар вида $\{n, r[n]\}$ с помощью функции MapIndexed:

```
data = MapIndexed[{First[#2], #1}&, r1000];
```

Список *data* выглядит следующим образом:

```
Short[data]
```

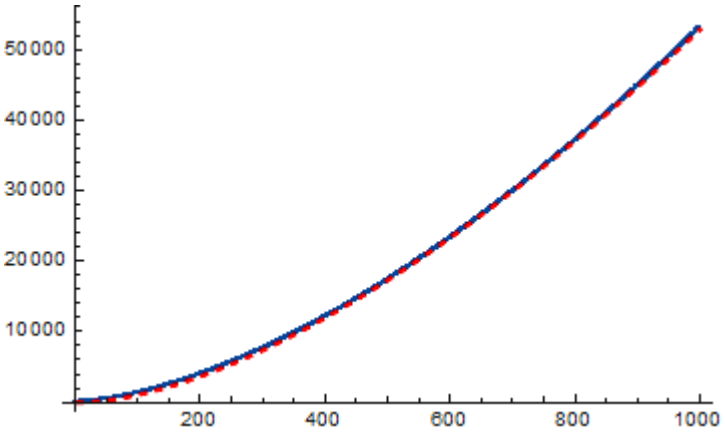
{{1, 1}, {2, 2}, {3, 4}, {4, 6}, <<992>>, {997, 52817}, {998, 52903}, {999, 52989}, {1000, 53075}}

Вызовем функцию Fit:

```
Fit[data, {x Log[x]^4}, x]
```
$0.0232034\ x\ \mathrm{Log}[x]^4$

Простроим вместе два графика, где значения функции $0.0232034x\ \mathrm{Log}[x]^4$ показаны пунктиром.

```
Show[{ListPlot[data], Plot[0.0232034 x Log[x]^4,{x, 1, 1000},
                           PlotStyle → {Red, Dashed, Thick}]}]
```



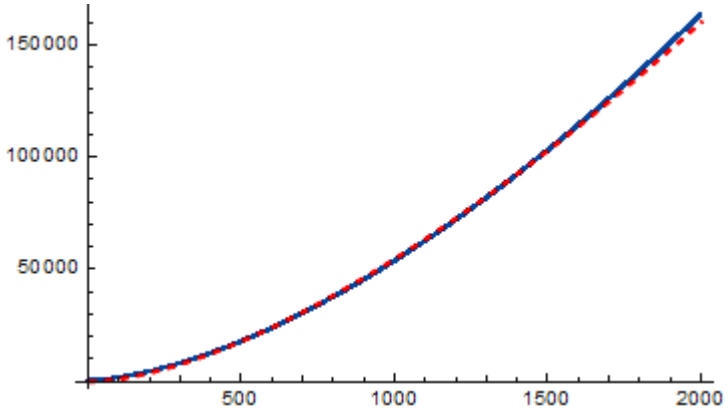
Насколько хороша будет полученная аппроксимации в диапазоне $1 < n \le 2000$? Для ответа вычислим $g2000$, увеличим параметр для глубины рекурсии, переопределим функцию $r[n]$ и вычислим $r2000$:

```
g2000 = Nest[h, {1}, 1999];
Clear[r]
$RecursionLimit = 5000;
r[1] = 1;
r[n_] := r[n] = r[n – 1] + g2000[[n – 1]]
r[2000];
r2000 = r /@ Range[2000];
```

Повторим аппроксимацию и построим новые графики:

```
data = MapIndexed[{First[#2], #1}&, r2000];
Fit[data,{x Log[x]^4}, x]
```
$0.0239782\ x\ \mathrm{Log}[x]^4$

```
Show[{ListPlot[data], Plot[0.0239782 x Log[x]^4,{x, 1, 2000},
                           PlotStyle → {Red, Dashed, Thick}]}]
```



Несмотря на удовлетворительный вид графиков, у нас нет оснований считать, что при увеличении диапазона значение числового множитель при функции $x(\ln x)^4$ стабилизируется и даже возможно аппроксимирующую функцию придется изменить.

§ 6. Три последовательности от Рональда Грэхэма

Источником служит интервью с профессором Рональдом Грэхемом [81].

Последовательность 1. Пусть a и b – вещественные числа. Определим последовательность $x_1 = a$, $x_2 = b$ и при $n > 2$

$$x_n = \frac{1 + x_{n-1}}{x_{n-2}}.$$

Оказывается, за исключением некоторых значений a и b, последовательность x_n является периодической с длиной периода 5.

Убедимся в этом. Определим последовательность на языке Wolfram:

```
x[1] := a; x[2] := b
x[n_] := x[n] = (1 + x[n – 1]) / x[n – 2]
```

Зададим случайные начальные значения и вычислим 7 членов последовательности.

```
{a, b} = {RandomReal[], RandomReal[]}
{0.229672, 0.478249}

x /@ Range[7]
{0.229672, 0.478249, 6.43633, 15.5491, 2.5712, 0.229672, 0.478249}
```

Убедимся в периодичности последовательности x_n в общем случае и найдем исключительные значения для a и b. В [1] утверждается, что исключительными значениями являются $a = 0$ и $b = -1$. Но на самом деле исключительных значений бесконечно много.

Вычислим первые семь элементов последовательности, не задавая численные значения для a и b. Предварительно необходимо очистить значения a, b, x и заново переопределить функцию $x[n]$ перед вызовом:

```
Clear[a, b, x]
x /@ Range[7]
```

$$\left\{a,\ b,\ \frac{1+b}{a},\ \frac{1+\frac{1+b}{a}}{b},\ \frac{a\left(1+\frac{1+\frac{1+b}{a}}{b}\right)}{1+b},\ \frac{b\left(1+\frac{a\left(1+\frac{1+\frac{1+b}{a}}{b}\right)}{1+b}\right)}{1+\frac{1+b}{a}},\right.$$

$$\left.\frac{(1+b)\left(1+\frac{b\left(1+\frac{a\left(1+\frac{1+\frac{1+b}{a}}{b}\right)}{1+b}\right)}{1+\frac{1+b}{a}}\right)}{a\left(1+\frac{1+\frac{1+b}{a}}{b}\right)}\right\}$$

Упростим эти элементы:

```
xs = Simplify[x /@ Range[7]]
```

$$\left\{a,b,\frac{1+b}{a},\frac{1+a+b}{ab},\frac{1+a}{b},a,b\right\}$$

Действительно, последовательность имеет период длиною 5. Но при некоторых значениях a и b при вычислении элементов последовательности происходит деление на 0. Это происходит каждый раз, когда предыдущий элемент последовательности равен 0. Полагая числители и знаменатели дробей из списка xs не равные нулю, находим допустимые значения a и b. Итак, необходимым условием для периодичности последовательности x_n является неравенство $ab(1 + a)(1 + b)(1 + a + b) \neq 0$.

Последовательность 2. Пусть a, b и c – вещественные числа. Определим последовательность $y_1 = a$, $y_2 = b$, $y_3 = c$ и при $n > 3$

$$y_n = \frac{1 + y_{n-1} + y_{n-2}}{y_{n-3}}.$$

Последовательность y_n также является периодической с длиной периода 8 для тех значений a, b и c, при которых никакой член последовательности не равен 0.

Определим последовательность на языке Wolfram:

```
y[1] := a; y[2] := b; y[3] := c
y[n_] := y[n] = (1 + y[n – 1] + y[n – 2]) / y[n – 3]
```

Зададим случайные начальные значения и вычислим 11 членов последовательности.

```
{a, b, c} = {RandomReal[], RandomReal[], RandomReal[]}
{0.69792, 0.71644, 0.80826}
y /@ Range[11]
{0.69792, 0.71644, 0.80826, 3.61746, 7.57317, 15.0826, 6.53932,
2.98711, 0.69792, 0.71644, 0.80826}
```

Убедимся в периодичности последовательности y_n в общем случае и найдем исключительные значения для a, b и c. Вычислим первые 11 элементов последовательности, не задавая численные значения для a, b и c. Предварительно необходимо очистить значения a, b, c, y и заново переопределить функцию $y[n]$ перед вызовом. Поскольку элементы последовательности y_n в символьном виде получаются очень громоздкие, то применяем функцию упрощения Simplify:

```
Clear[a, b, c, y]
Simplify[y[#]]& /@ Range[11]
```

$$\{a, b, c, \frac{1+b+c}{a}, \frac{1+a+b+c+ac}{ab}, \frac{(1+a+b)(1+b+c)}{abc},$$
$$\frac{1+a+b+c+ac}{bc}, \frac{1+a+b}{c}, a, b, c\}$$

Действительно, последовательность имеет период длиною 8. Но при некоторых значениях a, b и c при вычислении элементов последовательности происходит деление на 0. Полагая числители и знаменатели дробей не равные нулю, находим допустимые значения a, b и c. Итак,

необходимым условием для периодичности последовательности y_n является неравенство $abc(1 + a + b)(1 + b + c)(1 + a + b + c + ac) \neq 0$.

Последовательность 3. Эта последовательность определяется просто: $z_1 = 2$ и полагаем $z_n = z_{n-1} - 1 / z_{n-1}$ при $n > 1$. Соответствующее определение на языке Wolfram есть

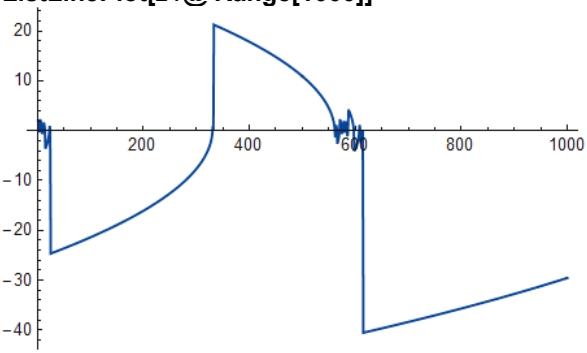
```
z[1] = 2;
z[n_] := z[n] = N[z[n – 1] – 1 / z[n – 1]]
```

Для вычисления элементов последовательности z_n зададим глубину рекурсии не меньше 2 000 000:

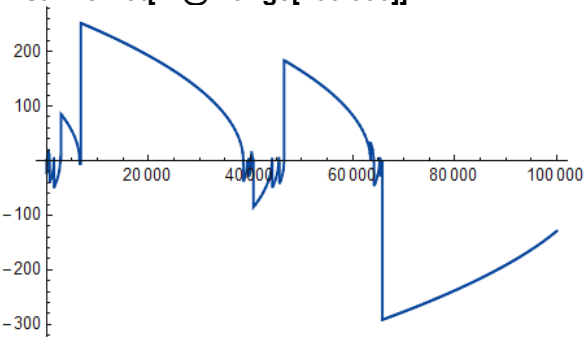
```
$RecursionLimit = 2000000;
```

Поведение последовательности наглядно представлено на следующих графиках. Цитируем [81]: «Будет ли z_n неограниченным? Ответ почти наверняка положительный, но никто это не умеет доказать».

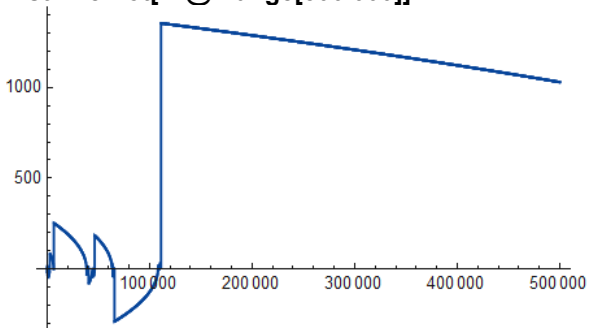
```
ListLinePlot[z /@ Range[1000]]
```



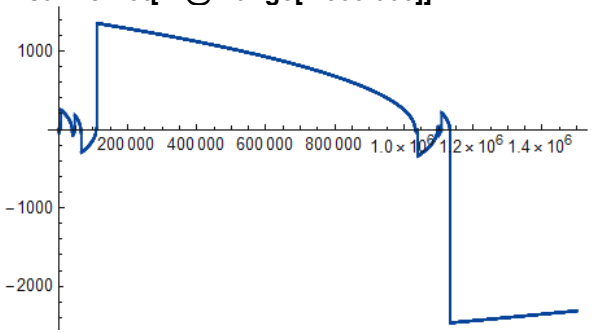

```
ListLinePlot[z /@ Range[100 000]]
```



ListLinePlot[*z* /@ Range[500 000]]



ListLinePlot[*z* /@ Range[1 500 000]]



§ 7. Функция Аккермана

Возникновение теории алгоритмов неразрывно связано с появлением понятия «вычислимая функция». К наиболее часто вычисляемым функциям относятся примитивно-рекурсивные функции (см., например, [82]). Функции, которые не являются примитивно-рекурсивными, растут быстрее любой экспоненты и даже кратной экспоненты.

Рассмотрим пример функции, не являющейся примитивно-рекурсивной, хотя и вычислимой.

Определим последовательность одноместных функций F_n: $\mathbb{N} \to \mathbb{N}$, $n \in \mathbb{N}$, следующим образом:

$$F_0(x) = x + 1,$$

$$F_{n+1}(x) = F_n(F_n(\ldots F_n(1)\ldots))$$

(функция F_n применяется $x + 1$ раз). (2)

Теорема 2. Для функций $F_n(x)$ справедливы следующие утверждения:

a) $F_1(x+1) = F_0(F_1(x))$;

b) $F_{n+1}(x+1) = F_n(F_{n+1}(x))$;

c) $F_1(x) = x + 2$;

d) $F_2(x) = 2x + 3$;

e) $F_3(x) = 2^{x+3} - 3$;

f) $F_4(x) = \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{x+3 \text{ раза}} - 3$.

Доказательство. *a*) Имеем в силу (2) равенства

$$F_1(x) = \underbrace{F_0(F_0(...F_0(1)..))}_{F_0 \text{ повторяется } x+1 \text{ раз}},$$

$$F_1(x+1) = \underbrace{F_0(F_0(...F_0(1)..))}_{F_0 \text{ повторяется } x+2 \text{ раза}}.$$

Поэтому получаем $F_1(x+1) = F_0(F_1(x))$.

b) Проводим математическую индукцию по n. Базис индукции для $n = 0$ доказан в пункте *a*). Для доказательства индуктивного перехода предположим, что для $n = k$ выполнено $F_{k+1}(x+1) = F_k(F_{k+1}(x))$. Имеем в силу (2)

$$F_{k+2}(x+1) = \underbrace{F_{k+1}(F_{k+1}(...F_{k+1}(1)..))}_{F_{k+1} \text{ повторяется } x+2 \text{ раза}},$$

$$\underbrace{F_{k+1}(F_{k+1}(...F_{k+1}(1)..))}_{F_{k+1} \text{ повторяется } x+1 \text{ раз}} = F_{k+2}(x).$$

Отсюда получаем $F_{k+2}(x+1) = F_{k+1}(F_{k+2}(x))$.

c) Проводим математическую индукцию по x. Базис индукции для $x = 0$: $F_1(0) = F_0(1) = 2$. Для доказательства индуктивного перехода предположим, что для $x = y$ выполнено $F_1(y) = y + 2$. Имеем по свойству *b*) $F_1(y+1) = F_0(F_1(y)) = F_0(y+2) = y + 2 + 1 = (y+1) + 2$. Что и требовалось доказать.

d) Проводим математическую индукцию по x. Базис индукции для $x = 0$: $F_2(0) = F_1(1) =$ (используем *c*)) $= 3 = 2 \cdot 0 + 3$. Для доказательства индуктивного перехода предположим, что для $x = y$ выполнено

$F_2(y) = 2y + 3$. Имеем по свойству *b*) $F_2(y + 1) = F_1(F_2(y)) = F_1(2y + 3) = 2y + 3 + 2 = 2(y + 1) + 3$. Что и требовалось доказать.

e) Проводим математическую индукцию по x. Базис индукции для $x = 0$: $F_3(0)$ = (используем (2)) $F_2(1)$ = (используем *d*)) = $5 = 2^{0+3} - 3$. Для доказательства индуктивного перехода предположим, что для $x = y$ выполнено $F_3(y) = 2^{y+3} - 3$. Имеем по свойству *b*) $F_3(y + 1) = F_2(F_3(y)) = F_2(2^{y+3} - 3) = 2(2^{y+3} - 3) + 3 = 2^{(y+1)+3} - 3$. Что и требовалось доказать.

f) Проводим математическую индукцию по x. Базис индукции для $x = 0$: $F_4(0)$ = (используем (2)) $F_3(1)$ = (используем свойство *e*)) = $= 13 = 2^{2^2} - 3$. Для доказательства индуктивного перехода предположим, что для $x = y$ выполнено $F_4(y) = \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{y+3 \text{ раза}} - 3$. Имеем по свойству *b*) $F_4(y + 1) = F_3(F_4(y)) = 2^{F_4(y)+3} - 3 = 2^{(\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{y+3 \text{ раза}} - 3)+3} - 3 = \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{y+4 \text{ раза}} - 3$. Что и требовалось доказать.

Выполнены также следующие свойства [93, с. 91–97]:

1) для каждого n функция $x \to F_n(x)$ является примитивно-рекурсивной;

2) $F_n(x) > 0$;

3) $F_n(x + 1) > F_n(x)$;

4) $F_n(x) > x$;

5) $F_{n+1}(x) \geq F_n(x + 1)$;

6) для каждой k-местной примитивно-рекурсивной функции $f(x_1, x_2, \ldots, x_k)$ существует такое n, что F_n мажорирует f, т.е. $f(x_1, x_2, \ldots, x_k) \leq \leq F_n(\max(x_1, x_2, \ldots, x_k))$ для всех $x_1, x_2, \ldots, x_k$.

Функция $A(n, x) = F_n(x)$ известна как **функция Аккермана**[24]. Её можно определить и в традиционной записи:

$$
\begin{aligned}
&f(0, y) = y + 1,\\
&f(x + 1, 0) = f(x, 1),\\
&f(x + 1, y + 1) = f(x, f(x + 1, y)).
\end{aligned}
$$

[24] Вильгельм Фридрих Аккерман (1896–1962 гг.) – немецкий математик и логик.

Можно думать, что при каких-то аргументах функция Аккермана уходит в бесконечную рекурсию, но известно [93, с. 91–97], что она всюду определена, не является примитивно-рекурсивной и $f(x, x)$ растет быстрее любой одноместной примитивной функции.

Определение этой функции на языке Wolfram:

```
f[0, y_] := y + 1
f[x_, 0] /; x > 0 := f[x – 1, 1]
f[x_, y_ ] /; x > 0 && y > 0 := f[x – 1, f[x, y – 1]]
```

В отличие от примитивной рекурсии последовательность значений аргументов при вычислении функции Аккермана не обнаруживает признаков упорядочивания. Рассмотрим подробно поведение функции.

Перепишем определение функции так, чтобы при вычислении запоминались промежуточные значения рекурсивных вызовов. Пусть $g(x, y)$ – новое имя функции Аккермана:

```
g[0, y_] := g[0, y] = y + 1
g[x_, 0] /; x > 0 := g[x, 0] = g[x – 1, 1]
g[x_, y_] /; x > 0 && y > 0 := g[x, y] = g[x – 1, g[x, y – 1]]
```

Функция g вычисляется быстрее, чем функции f, поскольку она совершает меньше рекурсивных вызовов. Это можно обнаружить, если использовать глобальную переменную в качестве счетчика рекурсивных вызовов. Напишем вариант функции f с переменной $c1$ в качестве счетчика:

```
fc[0, y_] := (c1 = c1 + 1; y + 1)
fc[x_, 0] /; x > 0 := (c1 = c1 + 1; fc[x – 1, 1])
fc[x_, y_ ] /; x > 0 && y > 0 := (c1 = c1 + 1; fc[x – 1, fc[x, y – 1]])
```

Вычислим значение функции Аккермана для аргументов 3 и 7 с помощью функции *fc* и попутно посчитаем количество рекурсивных вызовов:

```
c1 = 0; fc[3, 7]
1021
c1
693964
```

Как видим, функция вызывает себя 693964 раз, прежде чем получить окончательно значение 1021.

Теперь напишем вариант функции g с переменной $c2$ в качестве счетчика и проведем такие же вычисления:

```
gc[0, y_] := gc[0, y] = (c2 = c2 + 1; y + 1)
gc[x_, 0] /; x > 0 := gc[x, 0] = (c2 = c2 + 1; gc[x – 1, 1])
gc[x_, y_] /; x > 0 && y > 0 := gc[x, y] = (c2 = c2 + 1; gc[x – 1, gc[x, y – 1]])
c2 = 0; gc[3, 7]
1021

c2
2558
```

В этом случае количество рекурсивных вызовов всего 2558. Таким образом, $f(3,7) = 1021$, но прежде, чем получить это значение, функция Аккермана вызывает себя 693964 раза, а различных пар параметров во время вычисления всего 2558.

Попробуем проследить конкретный путь вычисления значения функции $f(2, 2)$. Для этого перепишем функцию так, чтобы она сохраняла в списке *path* пары аргументов при каждом вызове. Новый вариант определения функции носит название *h*. Список *path* перед вычислением функции *h* пуст.

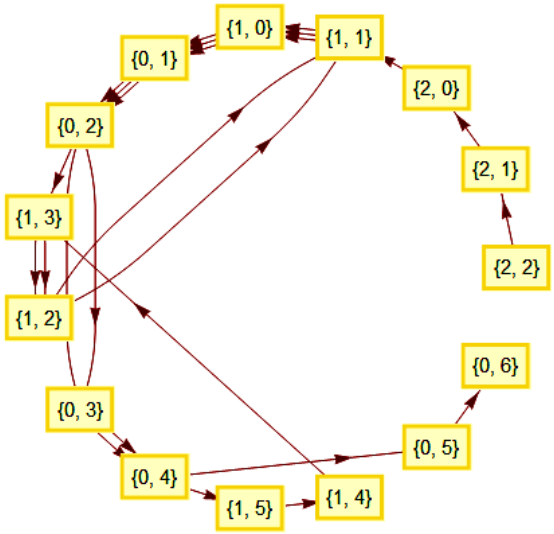
```
h[0, y_] := (AppendTo[path, {0, y}]; y + 1)
h[x_, 0] /; x > 0 := (AppendTo[path, {x, 0}]; h[x – 1, 1])
h[x_, y_] /; x > 0 && y > 0 := (AppendTo[path, {x, y}]; h[x – 1, h[x, y – 1]])
path = { };
h(2, 2)
7

path
{{2, 2}, {2, 1}, {2, 0}, {1, 1}, {1, 0}, {0, 1}, {0, 2}, {1, 3}, {1, 2},
{1, 1}, {1, 0}, {0, 1}, {0, 2}, {0, 3}, {0, 4}, {1, 5}, {1, 4}, {1, 3}, {1, 2},
{1, 1}, {1, 0}, {0, 1}, {0, 2}, {0, 3}, {0, 4}, {0, 5}, {0, 6}}
```

Всего вызовов 27, но различных только 15.

В отличие от примитивной рекурсии последовательность значений аргументов при вычислении функции Аккермана не обнаруживает признаков упорядочивания: прежде чем закончить вычисление, функция многократно вызывает себя, причем второй параметр как больше, так и меньше исходного значения. Причем вызовы функции Аккермана для некоторых пар аргументов повторяются многократно.

Следующая программа чертит граф вызовов при вычислении $f(2, 2)$:

```
path := {}; h[2,2];
GraphPlot[Partition[path, 2, 1] /. List[{a_, b_}, {c_,d_}] → Rule[{a, b}, {c, d}],
          Method → "CircularEmbedding", ImageSize → 300,
          DirectedEdges → True, VertexLabeling → True]
```



§ 8. Числа Улама

Определение. **Числа Улама** u_n, $n = 1, 2, 3, \ldots$, определяются рекуррентно: $u_1 = 1$, $u_2 = 2$ и для каждого следующего натурального $n > 2$ число u_n есть наименьшее натуральное число, которое можно единственным образом представить в виде суммы двух различных предшествующих чисел Улама.

Числа Улама названы в честь Станислава Улама[25], который впервые описал их в 1964 году. Найдем несколько первых чисел Улама. Очевидно, $u_3 = u_1 + u_2 = 3$. Число 4 является четвертым числом Улама: $4 = u_1 + u_3$ (представление $4 = u_2 + u_2$ не учитывается, так как требуется, чтобы слагаемые были различными). Число 5 не является числом Улама, так

[25] Станислав Улам (Stanislav M. Ulam), 1909–1988, – польский и американский математик.

5 можно двумя способами представить: $5 = u_2 + u_3 = u_1 + u_4$. Имеем $6 = u_2 + u_4$, и другого представления 6 в виде суммы различных чисел Улама не существует, поэтому $u_5 = 6$. Число 7 не является числом Улама, так как $7 = u_3 + u_4 = u_1 + u_5$. Далее видим $u_6 = 8$, так как 8 можно единственным образом представить в виде суммы двух различных чисел Улама: $8 = u_2 + u_5$. Числа 9 и 10 пропускаем, так как $9 = 1 + 8 = 3 + 6$ и $10 = 2 + 8 = 4 + 6$. Седьмое число есть $u_7 = 11$, так как представление $11 = 3 + 8 = u_3 + u_6$. Таким образом, первые семь чисел Улама – 1, 2, 3, 4, 6, 8, 11.

Чисел Улама бесконечно много. Действительно, от противного, пусть u_n и u_{n-1} – два наибольших числа Улама. Тогда число $u_{n+1} = u_n + u_{n-1}$ есть число Улама. Это представление u_{n+1} в виде суммы единственно, потому что $u_j + u_i < u_n + u_{n-1}$, если $j < n$ или $j = n$ и $i < n - 1$.

Выбирая в качестве первых двух чисел Улама другие натуральные числа, мы получаем обобщение – последовательности Улама [45; 6, с. 166, 167].

Напишем программу для получения первых n чисел Улама.

Описание алгоритма. Пусть $n > 2$ – вход алгоритма. Выход алгоритма: список u чисел Улама u_k, $k = 1, 2, 3, \ldots, n$. Список u создается в цикле, когда k меняется от 2 до n. Начальное значение списка $u = \{u_1, u_2\} = \{1, 2\}$. Используется вспомогательный список s, который перед началом каждого шага цикла состоит из возрастающей конечной последовательности чисел $s_1, s_2, \ldots, s_m$, где все s_t ($t = 1, 2, \ldots, m$) больше u_k и однозначно представимы в виде суммы $u_j + u_i$ ($i, j \le k$) двух различных чисел Улама. Количество элементов m списка s, вообще говоря, меняется на каждом шаге. На каждом шаге в список u добавляется первый элемент a списка s (он является очередным числом Улама) и список s возможных претендентов в числа Улама модифицируется. Перед первым шагом $s = \{3\}$.

Этот алгоритм не очень быстр, но для наших целей достаточен. Программа на языке Wolfram:

```
f[n_] :=
    Module[{u = {1, 2} , s = {3}, p, d = {}, a, k = 2},
        While [k < n, a = s[[1]]; p = Map[a + # &, u]; s = Rest[s];
                d = Union[d, Intersection[s, p]];
                s = Complement[Union[s, p], d];
                AppendTo[u, a]; k++];
        u]
```

Посмотрим на первые 50 чисел:

```
f[50]
```

{1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, 28, 36, 38, 47, 48, 53, 57, 62, 69, 72, 77, 82, 87, 97, 99, 102, 106, 114, 126, 131, 138, 145, 148, 155, 175, 177, 180, 182, 189, 197, 206, 209, 219, 221, 236, 238, 241, 243, 253}

```
ListPlot[f[50], Joined → True]
```

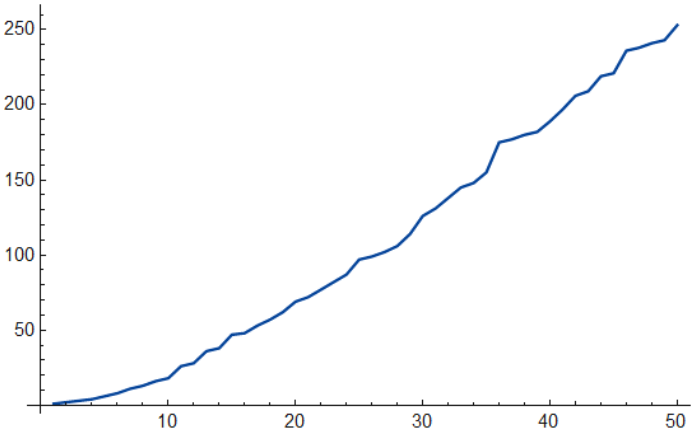
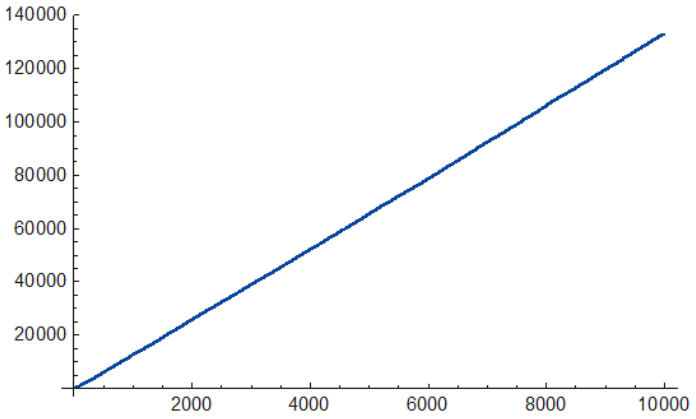


График роста чисел Улама на большом диапазоне выглядит прямолинейным отрезком.

```
us = f[10000];
ListPlot[us]
```



Найдем уравнение аппроксимирующей прямой. С помощью функции MapIndexed последовательность $\{u_1, u_2, u_3, \ldots, u_{10000}\}$ преобразуем в список пар $\{\{1, u_1\}, \{2, u_2\}, \{3, u_3\}, \ldots, \{10000, u_{10000}\}\}$:

```
data = MapIndexed[{First[#2], #1}&, us];
```

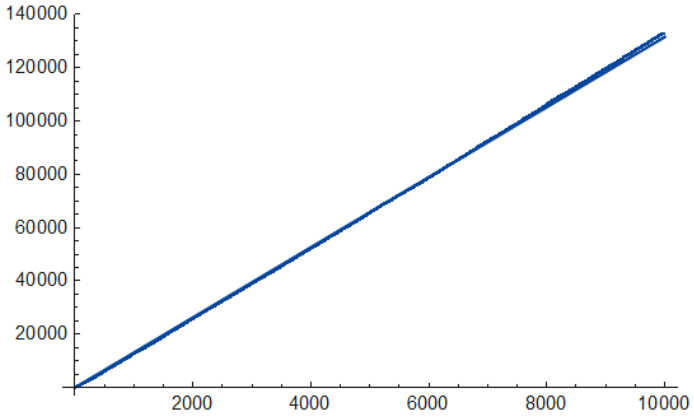
Используя функцию Fit, определим уравнение аппроксимирующей прямой, проходящей через начало координат:

```
Fit[data, {x}, x]
```

$13.1635\,x$

Графики ListPlot[*us*] и Plot[$13.1635\,x$, {*x*, 1, 5000}] визуально отличаются только на концах диапазона.

```
Show[ListPlot[us], Plot[13.1635 x, {x, 1, 10000}]]
```



Изучим, как меняется расстояние между последовательными числами Улама. Найдем *ds* – список пар $\{n, u_{n+1} - u_n\}$, $n = 1, 2, \ldots, 9999$, для 10000 последовательных чисел Улама:

```
ds = Table[{n, us[[n + 1]] – us[[n]]}, {n, 9999}];
```

Как мы уже обнаружили, первые три разности есть 1. Есть ли еще последовательные числа Улама, отличающиеся на 1?

```
Select[ds, #[[2]] == 1&]
```

{{1, 1}, {2, 1}, {3, 1}, {15, 1}}

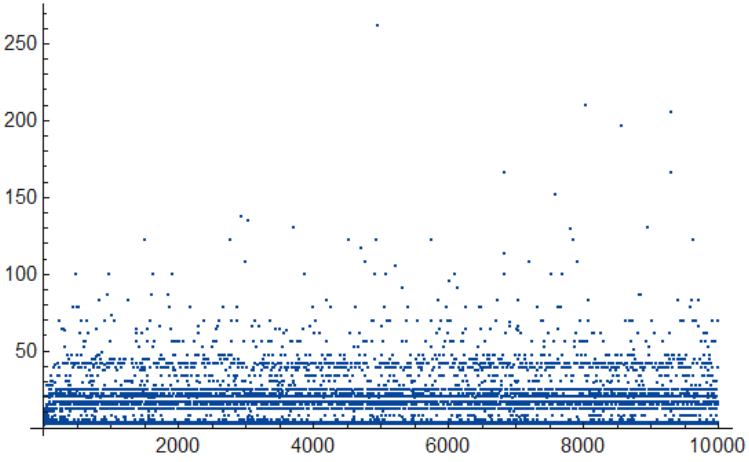
```
{us[[15]], us[[16]]}
```

{47, 48}

Встретилась еще одна пара последовательных чисел Улама, отличающихся на единицу: 47 и 48. В [6, с. 166, 167] сообщается, что были проверены первые 25 000 чисел Улама, но других пар последовательных чисел Улама с единичной разностью между ними не обнаружено.

Закономерности расстояния между соседними числами Улама не наблюдаются.

```
ListPlot[ds, PlotRange → All]
```



Исследуем, какие значения принимают разности $u_{n+1} - u_n$, $n = 1, 2, \ldots, 9999$ между соседними числами Улама. Для этого сначала упорядочим все разности (функция Sort), потом список разностей разобьем на подсписки с одинаковыми значениями (функция Split) и подсчитаем, сколько раз встречалась каждая разность.

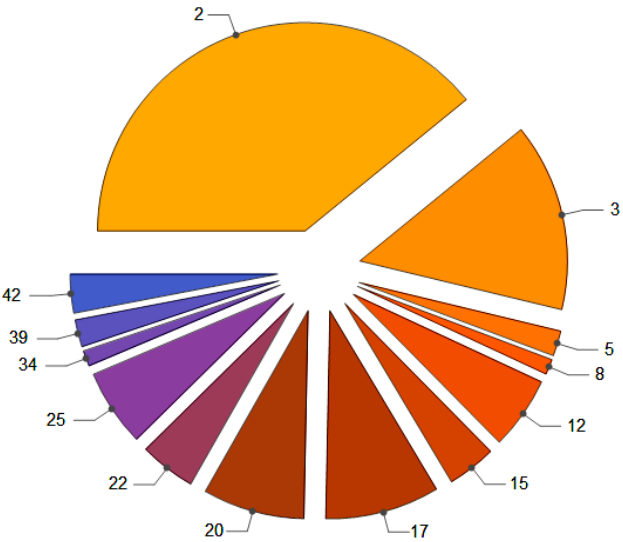
```
rs = {#[[1]], Length[#]}& /@ Split[Sort[#[[2]]& /@ ds]]
```

{{1, 4}, {2, 3630}, {3, 1356}, {4, 2}, {5, 182}, {7, 36}, {8, 111}, {9, 3}, {10, 29}, {12, 526}, {13, 1}, {15, 354}, {17, 827}, {19, 29}, {20, 736}, {22, 407}, {24, 25}, {25, 557}, {27, 51}, {29, 16}, {30, 72}, {32, 1}, {34, 113}, {37, 44}, {39, 197}, {41, 13}, {42, 279}, {44, 62}, {46, 1}, {47, 89}, {49, 1}, {51, 1}, {52, 17}, {56, 57}, {61, 25}, {63, 3}, {64, 16}, {66, 10}, {68, 1}, {69, 39}, {73, 1}, {78, 20}, {83, 10}, {86, 5}, {91, 2}, {95, 1}, {100, 11}, {105, 1}, {108, 4}, {113, 1}, {117, 1}, {122, 7}, {129, 1}, {130, 2}, {135, 1}, {137, 1}, {152, 1}, {166, 2}, {196, 1}, {205, 1}, {210, 1}, {262, 1}}

В каждой паре первое число есть значение разности, а второе – сколько раз эта разность между последовательными числами Улама повторяется. Как мы видим, разность 2 составляет 36.3 % всех значений. Максимальная разность – 262 и встречается один раз.

На круговой диаграмме увидим значения, которые встречаются не менее 100 раз.

```
b = Select[rs, #[[2]] > 99&];
data = #[[2]]&/@b
{3630, 1356, 182, 111, 526, 354, 827, 736, 407, 557, 113, 197, 279}
PieChart[data, ChartLabels → Placed[#[[1]]&/@b,"RadialCallout"]]
```



§ 9. Алгоритм Евклида

Рассмотрим свойства алгоритма Евклида 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

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

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 $5\log_{10} m$ [10, с. 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 1. 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: $\{x, y, d\}$ = EEA(m, n):
If m == 0 then return({0, 1, n})
else begin
  $r = n \bmod m$;
  $\{r_1, m_1, d\}$ = EEA(r, m); (* рекурсивный вызов *)
  return($\{m_1 - \lfloor n/m \rfloor r_1, r_1, d\}$)
end

Текст соответствующей программы:

```
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 языке Wolfram 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 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, т.е. 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.

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

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

Получаем последовательность остатков:

r_1 = PolynomialRemainder[*u*, *v*, *x*]

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

r_2 = PolynomialRemainder[*v*, r_1, *x*]

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

r_3 = PolynomialRemainder[r_1, r_2, *x*]

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

r_4 = PolynomialRemainder[r_2, r_3, x]

$$\frac{17428125}{802816} - \frac{8166587x}{802816} - \frac{36161099x^2}{802816} - \frac{3461627x^3}{200704} - \frac{3280121x^4}{802816}$$

r_5 = PolynomialRemainder[r_3, r_4, x]

$$\frac{2198164799488}{63663868489} - \frac{2030741536768x}{63663868489} - \frac{4783122333696x^2}{63663868489} - \frac{554215997440x^3}{63663868489}$$

r_6 = PolynomialRemainder[r_4, r_5, x]

$$\frac{1272943070806564261}{13664195416883200} - \frac{103051366981906031x}{1115444523827200} - \frac{101412892653396525 63x^2}{54656781667532800}$$

r_7 = PolynomialRemainder[r_5, r_6, x]

$$-\frac{1265016663855632442130432 00}{124265326150557573927519717} - \frac{1265016663855632442130432 00x}{124265326150557573927519717}$$

r_8 = PolynomialRemainder[r_6, r_7, x]

0

Simplify[r_7]

$$-\frac{1265016663855632442130432 00(1+x)}{124265326150557573927519717}$$

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:

```
r[k+2] = (p = PolynomialRemaider[r[k], r[k+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

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

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

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

$$-\frac{103125}{19409}+\frac{48323x}{19409}+\frac{213971x^2}{19409}+\frac{81932x^3}{19409}+x^4$$

$$-\frac{684517}{172585}+\frac{2529523x}{690340}+\frac{851133x^2}{98620}+x^3$$

$$-\frac{79978996}{159294267}+\frac{79315271x}{159294267}+x^2$$

$$1+x$$

$$0$$

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 $\mathbb{Z}[x]$.

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 для этого имеется соответствующая функция PolynomialExtendedGCD.

Расширенный gcd многочленов $f(x)$ и $g(x)$ относительно переменной x это такой список $\{d,\{r, s\}\}$, где $d = \gcd(f, g)$ 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]
```

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

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

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

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

```
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 1. 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]
```

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

Решение существует, если c делится на d.

```
h = Expand[c / d]
```
$$-7+x^2$$
```
{f, g} = h {r, s}
```
$$\left\{\frac{1}{11}\left(-7+x^2\right)\left(-7-2x-x^2-x^3\right),\frac{1}{11}\left(-7+x^2\right)\right\}$$

Проверка:

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

Задача 2. Имеется дробь

$$e1=\frac{\left(1+\sqrt{2}\right)^{2/3}}{1+(1+\sqrt{2})^{1/3}+\left(1+\sqrt{2}\right)^{2/3}},$$

содержащая алгебраическое иррациональное число

$$\alpha=\left(1+\sqrt{2}\right)^{1/3}.$$

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 $fu + gv = 1$ – 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:

```
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], 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

Линейные диофантовые уравнения

Рассмотрим линейное диофантово уравнение с двумя неизвестными

$$a\,x + b\,y = c, \tag{3}$$

где a и b не равны 0.

Лемма 1. Пусть для целых чисел $d = \gcd(a, b)$. Тогда $\gcd(a/d, b/d) = 1$.

Лемма 2. Если d делит $\gcd(m, n)$ и $\gcd(d, m) = 1$, то d делит n.

Теорема 3. Пусть $d = \gcd(a, b)$. Если d не делит c, то уравнение (3) не имеет решения в целых числах. Если d делит c, то имеется бесконечно много решений, которые находятся следующим образом. Частное решение есть $(x_0, y_0) = (cs/d, ct/d)$, где s и t суть коэффициенты из соотношения Безу – результата работы расширенного алгоритма Евклида на числах a, b. Общее решение имеет вид $x = x_0 + (b/d)n$, $y = y_0 - (a/d)n$, где n – любое целое число.

Доказательство. Для существования решения d должно делить c, так как d делит a и b. Пусть d делит c, тогда расширенный алгоритм Евклида дает целые числа s и t такие, что $sa + tb = d$. Так как c/d есть целое, то умножая на c/d, получаем $a(c/d)s + b(c/d)t = (c/d)d = c$, и поэтому мы имеем первое решение – $(x_0, y_0) = (cs/d, ct/d)$.

Для общего случая мы применяем традиционный алгебраический способ – правую часть уравнения делаем 0 (полученное уравнение называется *однородным*). Тогда одно из решений есть $(bn, -an)$, но лучше взять решение $((b/d)n, -(a/d)n)$, потому что любое решение кратно этому решению. Пусть имеем $a\,x_1 + b\,y_1 = 0$, тогда $(a/d)\,x_1 + (b/d)\,y_1 = 0$, так что $(a/d)x_1 = -(b/d)\,y_1$. Но числа a/d и b/d – относительно простые (лемма 1), так что a/d делит $-y_1$ (лемма 2), скажем $-y_1 = n(a/d)$. Подставляя y_1 в однородное уравнение, получаем $x_1 = nb/d$. Таким образом, однородное уравнение решено.

Сейчас предположим, что (x_2, y_2) есть решение уравнения (3). Тогда легко проверить, что $(x_2 - x_0, y_2 - y_0)$ удовлетворяет однородному уравнению и, следовательно, существует такое целое n, что $x_2 - x_0 = bn\ /d$ и $y_2 - y_0 + - na\ /d$. ■

Соответствующая программа получается короткой, если использовать матричные операции:

```
diophSolve[{a_, b_}, c_, n_] :=
    Module[{d, s, t},{d, {s, t}} = ExtendedGCD[a, b];
        If[Mod[c, d] == 0, Transpose[{c {s, t}, {b, –a}}/d].{1, n}, {}]]
```

Следующие вычисления выражений показывают результаты отдельных операций в теле функции *diophSolve*:

```
{c {s, t}, {b, –a}}/d  // MatrixForm
```

$$\begin{pmatrix} \frac{cs}{d} & \frac{ct}{d} \\ \frac{b}{d} & -\frac{a}{d} \end{pmatrix}$$

Транспозиция предыдущей матрицы:

Transpose[%]

$$\left\{\left\{\frac{cs}{d}, \frac{b}{d}\right\}, \left\{\frac{ct}{d}, -\frac{a}{d}\right\}\right\}$$

Умножение матрицы на вектор:

% . {1, *n*}

$$\left\{\frac{bn}{d}+\frac{cs}{d}, -\frac{an}{d}+\frac{ct}{d}\right\}$$

Приведем пример решения диофантова уравнения с двумя неизвестными:

***xy* = *diophSolve*[{999, 47}, 500, *n*]**

$\{2000 + 47\,n, -42500 - 999\,n\}$

Мы можем получить целый ряд частных решений:

Table[*xy* , {*n*, –1, 4}]

{{1953, – 41501}, {2000, – 42500}, {2047, – 43499}, {2094, – 44498}, {2141, – 45497}, {2188, – 46496}}

Решение линейных диофантовых уравнений и даже систем таких уравнений хорошо изучено (см., например, [15, с. 219–231]). Для нелинейных уравнений общего метода нет (эта задача алгоритмически неразрешима), но в языке Wolfram на практике успешно решаются многие диофантовы уравнения – автоматически применяются различные оригинальные методы, многие из которых базируются на новейших достижениях в теории чисел. Рассмотрим функции, дающие решения диофантовых уравнений.

Функция FindInstance[*expr, vars*, Integers] находит частное решение системы диофантовых уравнений. Рассмотрим примеры:

FindInstance[999 *x* + 47 *y* == 500, {*x*, *y*}, Integers]

$\{\{x \to 26, y \to -542\}\}$

Если добавить еще один параметр – целое положительное число, то функция находит заданное число решений:

FindInstance[999 *x* + 47 *y* == 500, {*x*, *y*}, Integers, 4]

$\{\{x \to -7870, y \to 167290\}, \{x \to 3128, y \to -66476\}, \{x \to 6324, y \to -134408\}, \{x \to 9379, y \to -199343\}\}$

Примеры решения системы нелинейных уравнений с ограничением в виде неравенства:

```
FindInstance[x^2 + y + z == 10 && 2x + 3y^2 + 4z == 100 && z < 0,
             {x, y, z}, Integers]
{{x → –10, y → –12, z → –78}}

FindInstance[x^2 + y + z == 10 && 2x + 3y^2 + 4z == 100 && z > 0,
             {x, y, z}, Integers]
{}
```

Функция Reduce применима во многих ситуациях, и, в частности, может давать общее решение системы линейных диофантовых уравнений:

```
Reduce[x + 2y – 3z = = 5 && 2x – 2y + 3z == 10, {x, y, z}, Integers]
C[1] ∈ Integers && x = = 5 && y = = 3 C[1] && z = = 2 C[1]

Reduce[999 x + 47 y == 500, {x, y}, Integers]
C[1] ∈ Integers && x = = 26 + 47 C[1] && y = = –542 – 999 C[1]
```

Решение уравнения $999x + 47y = 500$ функция *diophSolve* выдает с бóльшими числами $\{2000 + 47n,\ -42500 - 999n\}$, чем функция FindInstance: $\{\{x \to 26, y \to -542\}\}$.

Можно уменьшить числа, получаемые с помощью *diophSolve*, выполнив подстановку

$$\{2000 + 47n, -42500 - 999n\} /. n \to n - 42$$

Перепишем программу для *diophSolve*:

```
diophSolve1[{a_, b_}, c_, n_] :=
    Module[{d, s, t, k, g}, {d, {s, t}} = ExtendedGCD[a, b];
           If[Mod[c, d] == 0, g = Transpose[{c {s, t}, {b, –a}}/d].{1, n};
           k = Floor[Coefficient[g[[1]], n ,0] / Coefficient[g[[1]], n, 1]];
           Expand[g /. n → n – k], {}]]
diophSolve1 [{999, 47}, 500, n] /. n → 0
{26, –542}
```

Задача 3. Мартышка и кокосовые орехи.

Шесть матросов и мартышка потерпели кораблекрушение и высадились на необитаемом острове. Весь первый день они занимались сбором кокосовых орехов. Вечером они сложили все орехи в кучу и легли спать.

Ночью, когда все заснули, один из матросов встал. Он подумал, что утром при разделе орехов может вспыхнуть ссора, и решил взять свою долю орехов немедля. Поэтому он разделил все кокосовые орехи на

пять равных кучек, а один оставшийся орех отдал мартышке. Затем он спрятал свою долю, а все остальные орехи снова сложил в одну кучу.

Через некоторое время проснулся другой «робинзон» и сделал то же самое. У него тоже остался один лишний орех, и он отдал его мартышке. И так один за другим поступили все пятеро потерпевших кораблекрушение. Каждый из них взял себе одну пятую орехов из той кучи, которую он нашёл при пробуждении, и каждый отдал один орех мартышке. Утром все проснулись и, бросив один кокос ожидающей обезьяне, они поделили оставшиеся орехи, и каждому досталось поровну – по одной пятой. Разумеется, каждый из матросов не мог не знать, что часть орехов не хватает, но так как у каждого из них совесть была одинаково нечиста, то никто ничего не сказал. Сколько кокосовых орехов было первоначально?

В книге М. Гарднера [66, с. 209] приводится текст этой задачи немного в другом виде: когда утром моряки стали делить кокосовые орехи, то мартышке ничего не досталось.

Покажем, как условия могут быть объединены, чтобы получилось уравнение вида $ax + by = c$, где x является общее количество кокосовых орехов, а y – количество орехов, полученных каждым моряком утром при делении. Решим уравнение и найдите наименьшее возможное значение для x.

Нетрудно видеть, что мы можем записать исходное уравнение в следующем виде:

```
y == (Nest[4(# – 1)/5&, x, 5] – 1)/5
```

$$y == \frac{1}{5}\left(-1+\frac{4}{5}\left(-1+\frac{4}{5}\left(-1+\frac{4}{5}\left(-1+\frac{4}{5}\left(-1+\frac{4}{5}(-1+x)\right)\right)\right)\right)\right)$$

Раскроем скобки и перемножим:

```
Expand[%]
```

$$y == -\frac{11529}{15625}+\frac{1024x}{15625}$$

Получаем линейное диофантово уравнение $1024x - 15625y = 11529$, которое и решаем:

```
Reduce[1024x – 15625y == 11529 && x > 0 && y > 0, {x, y}, Integers]
C[1] ∈ Integers && C[1] ≥ 0 && x = = 15621 + 15625 C[1] && y = =
1023 + 1024 C[1]
```

Наименьшее положительное значение x получаем при C[1] = 0. Таким образом, 15621 – исходное количество кокосовых орехов.

Задача 4. Найти двузначное число, у которого количество единиц, умноженное на 8, на 13 меньше количества десятков, умноженного на 3.

Решение. Положим неизвестное двузначное число x равно $10a + b$. Тогда имеем диофантово уравнение $3a - 8b = 13$. Решая его, получаем $x = 71$:

```
Reduce[3a – 8b == 13 && 9 ≥ a > 0 && 9 ≥ b ≥ 0, {a, b}, Integers]
a == 7 && b == 1
```

Глава 3

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.

Два целых числа a, b называются **сравнимыми по модулю *m***, если при делении на m они дают одинаковые остатки. При этом пишут $a \equiv b (\mathrm{mod}\ m)$, 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 2). Точнее, если $a \equiv b \pmod m$ и $c \equiv d \pmod m$, то выполнены сравнения $a \pm c \equiv b \pm d \pmod m$ 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 $\mathbb{Z}_m = m/m\mathbb{Z}$

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 $\{1\}_m = 1 + \mathbb{Z}_m$. 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 $2(m-1) = -2 = m - 2$.

Таблицу умножения в $\mathbb{Z}_m$ можно построить с помощью функции *timesMod*:

```
timesMod[m_] :=
  Grid[
    ReplaceAll[ Table[Mod[a b,  m], {a, 1, m – 1}, {b, 1, m – 1}],
      {1 → Style[1, Larger, Bold], 0 → Style[0, Larger, Bold]}],
  Frame → All]
```

Вот как выглядят таблицы умножения по модулям 5, 6 и 7 соответственно:

```
timesMod[5]
```

1	2	3	4
2	4	**1**	3
3	**1**	4	2
4	3	2	**1**

***timesMod*[6]**

1	2	3	4	5
2	4	**0**	2	4
3	**0**	3	**0**	3
4	2	**0**	4	2
5	4	3	2	**1**

***timesMod*[7]**

1	2	3	4	5	6
2	4	6	**1**	3	5
3	6	2	5	**1**	4
4	**1**	5	2	6	3
5	3	**1**	6	4	2
6	5	4	3	2	**1**

В этих таблицах сомножители (отличные от 0) выбираются из самого левого столбца и самой верхней строки.

Теорема 1. 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 $\varepsilon\mu + km = 1$, это означает, что ε и m взаимно просты. Обратное также очевидно. ■

Легко написать программу, которая показывает, какие сомножители из $\mathbb{Z}_m$ при умножении дают единицу. Это делает функция *inversesMod*; с ее помощью мы обнаруживаем делители единицы для модулей 6, 7 и 8:

```
inversesMod[m_] :=
  TableForm[
   Select[Flatten[Table[{x, y, Mod[x y, m]}, {x, 1, m – 1},
                        {y, x, m – 1}], 1], #[[3]] = = 1 &] /.
         {a_Integer, b_Integer, c_Integer} :>
         ToString[a] <> " × " <> ToString[b] <> " = " <> ToString[c]]
inversesMod[6]
```

$1 \times 1 = 1$
$5 \times 5 = 1$

***inversesMod*[7]**
$1 \times 1 = 1$
$2 \times 4 = 1$
$3 \times 5 = 1$
$6 \times 6 = 1$

***inversesMod*[8]**
$1 \times 1 = 1$
$3 \times 3 = 1$
$5 \times 5 = 1$
$7 \times 7 = 1$

Следствие. 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. Если $\gcd(a, m) = d > 1$, то класс вычетов a в $\mathbb{Z}_m$ являются делителем нуля.

Следующие свойства сравнений легко доказываются.

Теорема 2.

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 $p(x) = p_n x^n + p_{n-1}x^{n-1} + ... + p_1 x + p_0$ и $q(y) = q_n y^n + q_{n-1}y^{n-1} + ... + q_1 y + q_0$, 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 3 (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, …, $p-1$ 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, …, $p-1$ 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, …, $p-2$ (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 делит $(p-1)!$. 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, далее теорему 9 (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, т.е. множеству, на котором определено произведение $x \cdot x \cdot \ldots \cdot x$.

Алгоритм 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:

$$pow(5,15) = 5\,pow(5,7)^2 = 5(5\,pow(5,3)^2)^2$$
$$= 5(5(5\,pow(5,1)^2)^2)^2 = 5(5(5\times 5^2)^2)^2.$$

В теории чисел, как мы увидим позднее, часто приходится вычислять $x^n \bmod m$. Если x и/или n – большие числа, то последовательное выполнение операций – сначала возведение в степень, а потом нахождение остатка – является неэффективным. Лучше на каждом шаге алгоритма 1 (четном или нечетном) выполнять приведение по модулю, так как тогда вычисления идут с небольшими числами и поэтому быстрее.

Например, вычисляя полученную выше цепочку степеней по модулю $m = 8$, легко получить ответ

$$5^{15} (\bmod 8) = 5(5(5\times 5^2)^2)^2 (\bmod 8)$$
$$= 5(5(5\times 1)^2)^2 (\bmod 8) = 5(5\times 25)^2 (\bmod 8)$$
$$= 5(5\times 1)^2 (\bmod 8) = 5\times 25 (\bmod 8)$$
$$= 5\times 1 (\bmod 8) = 5.$$

Вместо рекурсии можно использовать цикл по двоичным цифрам показателя степени n слева направо (первая единица игнорируется).

Алгоритм 2 (алгоритм для возведения в степень с итерацией). Алгоритм вычисляет $y = x^n \bmod m$, где x принадлежит некоторой полугруппе, а n – положительное целое число:

Определяем *digs* как список двоичных цифр числа n;
k = количество цифр в *digs*;
$y = x$;
for i = 2 to k do
begin

$y = y^2 \bmod m$;
if *i*-я цифра == 1 then $y = (y\ x) \bmod m$
end;
return(y)

Но для возведения в степень с приведением по модулю более короткий и быстрый код дает функциональное программирование в языке Wolfram:

```
powerMod[x_, n_, m_] :=
Fold[Mod[Mod[#1 #1, m] * If[#2 == 1, x, 1], m]&, x,
Rest[IntegerDigits[n, 2]]]
```

Встроенная функция PowerMod в языке Wolfram реализует алгоритм 2.

Пример 1. 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[71500000008, 12] // Timing
{28.125,1}

powerMod[7, 1500000008, 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 1, глава 2), сейчас предлагается версия с циклом.

Алгоритм 3 (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 $x = n^{-1} = a \bmod m$.

В языке Wolfram алгоритм 3 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 2. Рассмотрим два достаточно больших числа

```
m = 18496 × 2^18496 + 1;
n = Quotient[2m, 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 языке Wolfram используется тот же алгоритм, что и для описанной функции *inversion*, и он работает при вызове PowerMod[n, –1, m]. Возьмем предыдущие числа n и m. Тогда

```
Mod[n PowerMod[n, –1, m], m]
1
```

Пример 3. Обратный элемент можно вычислить с помощью функции Solve. Вычислим обратный элемент для 9876543210 по модулю 541:

```
Solve[9876543210x == 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 2, нахождение обратного элемента с помощью функции PowerMod происходит быстрее, чем с помощью Solve.

§ 4. Решение сравнений

Решение линейных сравнений

Теорема 4. 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 $a^{-1}ax \equiv \equiv a^{-1}b \pmod m$ 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 2(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. 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.

Можно сформулировать следующую задачу для общего случая.

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

$$x \equiv r_1 \pmod{m_1}, \ldots, x \equiv r_n \pmod{m_n}.$$

Теорема 5 (китайская теорема об остатках). Пусть $m_1, \ldots, m_n > 1$ – целые попарно взаимно простые числа и $a_1, \ldots, a_n$ – произвольные вычеты по модулям $m_1, \ldots, m_n$ соответственно. Положим $M = m_1 \cdot m_2 \cdots m_n$. Тогда существует единственный класс вычетов x по модулю M, который является решением системы сравнений

$$x \equiv a_1 \pmod{m_1}, \ldots, x \equiv a_n \pmod{m_n}. \qquad (2)$$

Для доказательства нам потребуется легко доказываемая

Лемма 1. 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* теоремы**. Рассмотрим $M/m_1 = m_2 \cdot m_3 \cdots m_n$; это число взаимно просто с m_1, поэтому существует b_1 – обратный элемент для M/m_1 по модулю m_1. Возьмем $y_1 = a_1 b_1 (M/m_1)$; имеем $y_1 \equiv 0 \pmod{m_i}$ для каждого $i \neq 1$ и $y_1 \equiv a_1 \pmod{m_1}$. Другими словами, y_1 есть решение первого сравнения системы и никак не влияет на решения других уравнений системы. Точно так же мы строим $y_2, y_3, \ldots, y_n$ – соответствующие решения остальных уравнений. Следовательно, для каждого $k = 1, 2, \ldots, n$

выполнены сравнения $y_k \equiv 0 \pmod{m_i}$ для всех $i \neq k$ и $y_k \equiv a_k \pmod{m_k}$. Определим число

$$x^* = \sum_{k=1}^{n} y_k; \tag{3}$$

имеем $x^* - a_k \equiv 0 \pmod{m_k}$ для каждого k. Из (2) следует $x - x^* \equiv 0 \pmod{m_k}$ для каждого k, поэтому в силу леммы 1 имеем $x - x^* \equiv 0 \pmod{M}$, т.е. x^* – решение системы сравнений (2).

Осталось доказать, что решение единственно по модулю M. Предположим, x и z – два различных решения. Тогда $x \equiv z \pmod{m_k}$ для каждого k и в силу леммы 1 имеем $x \equiv z \pmod{M}$. ■

При реализации программы, находящей решение задачи, будем представлять решение с помощью формулы (3). Сначала определим вспомогательную функцию, выясняющую, являются ли модули m_1, …, m_n попарно взаимно просты

```
pairwiseCoprimeQ[m_List] := LCM@@m == Times@@m
```

Данное определение избегает очевидного, но медленного перебора всех пар модулей. LCM – встроенная функция языка Wolfram, вычисляющая наименьшее общее кратное списка целых чисел.

Проверим:

```
{pairwiseCoprimeQ[{3, 5, 7}], pairwiseCoprimeQ[{2, 5, 4}]}
{True, False}
```

Теперь определим главную функцию *crt*[26] для задачи:

```
crt[a_, m_?pairwiseCoprimeQ]:=
    Mod[a . (PowerMod[#/m, –1, m] #/m), #]& [Times@@m]
```

Код функции небольшой, но эффективный, поэтому объясним некоторые особенности, с которыми до этого в книге не встречались. При объяснении используются обозначения из доказательства теоремы.

Параметры a и m – списки остатков и модулей соответственно. Поскольку используется чисто функциональное программирование, то отдельные элементы из списков явно не упоминаются. Все арифметические операции выполняются сразу для списков. Выражение Times@@m является краткой записью Apply[Times, m], что в качестве результата выдает M – произведение всех модулей из списка m. Чистая функция в

[26] *Crt* – Chinese Remainder Theorem

теле функции *crt* использует *M* как аргумент (все три вхождения символа # заменяются на *M*). Выражение #/*m* после вычисления есть список, состоящий из элементов M/m_k. Функция PowerMod обладает атрибутом[27] Listable. Любая функции *f* с таким атрибутом при применении к списку *s* равносильна вызову Map[*f*, *s*]. В нашем случае значением выражения PowerMod[#/*m*, –1, *m*] будет список, состоящий из элементов

$$\text{PowerMod}[M/m_k, -1, m_k],$$

а поскольку операция умножения[28] имеет атрибут Listable, то значением выражения (PowerMod[#/*m*, –1, *m*] #/*m*) будет список, состоящий из элементов

$$\text{PowerMod}[M/m_k, -1, m_k]\ M/m_k.$$

Операция Dot, обозначаемая точкой, дает скалярное произведение двух векторов (двух списков).

Найдем решение для китайской задачи:

```
crt[{1, 2, 3}, {3, 5, 7}]
52
```

В языке Wolfram имеется встроенная функция ChineseRemaider[*a*, *m*], которая выдает наименьшее неотрицательное число *x*, удовлетворяющее системе сравнений (2), но в отличии от *crt*, эта функция не требует, чтобы модули были попарно взаимно простыми и выдает ответ, когда решение существует:

```
ChineseRemainder[{1, 0, 2}, {3, 2, 4}]
10

ChineseRemainder[{1, 1, 2}, {3, 2, 4}]
ChineseRemainder[{1, 1, 2}, {3, 2, 4}]
```

Задача 1. Классическим применением ChineseRemainder является решение следующей задачи: найти 10 последовательных целых чисел, каждое из которых делится на квадрат. Мы можем искать такое *x*, что $x \equiv 0 \pmod{2^2}$, $x + 1 \equiv 0 \pmod{3^2}$, $x + 2 \equiv 0 \pmod{5^2}$ и т.д., до $x + 9 \equiv 0 \pmod{29^2}$. Мы выбираем квадраты небольших простых чисел, чтобы модули были попарно взаимно простыми.

---

[27] Чтобы узнать атрибуты любой встроенной функции *f* достаточно вызвать Attributes[*f*].

[28] Так же как и большинство арифметических операций.

Таким образом, имеем систему сравнений

$$x \equiv 0 \pmod{2^2},\ x \equiv -1 \pmod{3^2},\ x \equiv -2 \pmod{5^2},\ x \equiv -3 \pmod{7^2},$$
$$x \equiv -4 \pmod{11^2},\ x \equiv -5 \pmod{13^2},\ x \equiv -6 \pmod{17^2},$$
$$x \equiv -7 \pmod{19^2},\ x \equiv -8 \pmod{23^2},\ x \equiv -9 \pmod{29^2}.$$

```
x = ChineseRemainder[Range[0, –9, –1],{2, 3, 5, 7, 11, 13, 17, 19, 23, 29}^2]
```
4 368 761 145 612 023 948

Задача 2. Найти 5 последовательных целых чисел, каждое из которых делится на 100-ю степень какого-то числа.

```
x = ChineseRemainder[Range[0, –4, –1], {2, 3, 5, 7, 11}^100]
```
```
22790836016329060826239551178768284705563950178897504778722
29004884795649075491756155767392858151458417718029450778768
74529462059636399087601877157716847118357994205698247857656
72875335835387728502559643097255163952145861259556411510751
79060146382163396058770197801243330191617276220011148468464
617922180082132399547845125198364257812 48
```

```
IntegerLength [x]
```
336

Искомыми числами являются x, $x + 1$, $x + 2$, $x + 3$, $x + 4$; κ-е число в списке $\{2^{100}, 3^{100}, 5^{100}, 7^{100}, 11^{100}\}$ (нумерация начинается с 1) делит $x + k - 1$. Проверим

```
Mod[x + #, Prime[# + 1]^100]& /@ Range[0, 4]
```
{0, 0, 0, 0, 0}

Задача 3. Пусть p_k обозначает k-е простое число. Реализуем алгоритм, который выдаёт $2p_k + 1$ последовательное составное число. Используя китайскую теорему об остатках, находим такое b, что b делится на произведение первых k простых, а также $b \equiv -1 \pmod{p_{k+1}}$ и $b \equiv (\mathrm{mod}\ p_{k+2})$. Тогда числа $\{b - p_k, \ldots, b - 1, b, b + 1, \ldots, b + p_k\}$ – все составные.

```
b[k_] := ChineseRemainder[Table[0, {k}] ~Join~ {–1, 1},
Prime[Range[k + 2]]]
b[10]
```
4 696 997 284 980

```
ps[k_, b_] := Table[b – i, {i, Prime[k], 0, –1}] ~Join~ Table[b + i,{i, 1,
Prime[k]}]
AnyTrue[ps[10, b[10]], PrimeQ]
```
False

Так как $p_{10} = 29$, то мы получаем 59 составных последовательных чисел начиная с 4696997284951 и заканчивая 4696997285009. Этот алгоритм выдает меньшие последовательные составные числа, чем можно получить с помощью факториала известным способом: 59!, 59! + 1, 59! +2, …, 59! + 59, поскольку 59! имеет 81 цифру.

Сравнение любой степени

Пусть левая часть сравнения есть многочлен относительно неизвестного x. Выясним, сколько может быть решений у такого сравнения.

Лемма 2. 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

$$h(x) = q(x)(x - a) + r(x), \tag{4}$$

причем либо $r(x) = 0$, либо его степень меньше степени $x - a$ [83, с. 187–188]. Таким образом, $r(x) = c$ – целое число. Заменим в уравнении (4) x на a:

$$h(a) = q(a)(a - a) + c = c.$$

Таким образом, мы можем переписать (4) в виде

$$h(x) = (x - a)q(x) + h(a),$$

что завершает доказательство. ■

Теорема 6. 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). 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 до-

казывать нечего, ибо $0 \le k$. (Пример 5 такого многочлена приведен после доказательства.) Следовательно, мы можем предполагать, что $f(x)$ имеет корень a в Z_p, т.е. $f(a) \equiv 0 \pmod p$. По лемме 2

$$f(x) = (x - a)q(x) + f(a), \tag{5}$$

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 (5) по модулю p, получаем

$$f(x) \equiv (x - a)q(x) \pmod p. \tag{6}$$

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 (6), 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 $\mathbb{Z}_p$. Следовательно, $f(x)$ имеет не больше k различных корней. Доказательство теоремы закончено. ■

Пример 5. Найдем все возможные вычеты у многочлена $f(x) = x^2 + 3$ по модулю 5.

```
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 6. 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

Функцией Эйлера φ: $\mathbb{N} \to \mathbb{N}$ 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

Перечислим простейшие свойства функции Эйлера:

1) Если $n \ge 2$, то $1 \le \varphi(n) \le n - 1$;

2) Если $n > 2$, то $\varphi(n)$ – четно;

3) Если n – простое число, то $\varphi(n) = n - 1$;

4) Если $\varphi(n) = n - 1$, то n – простое число.

Свойства 1 и 3 следуют сразу из определения. Свойство 2) справедливо потому, что, если $\gcd(k, n) = 1$, то $\gcd(n - k, n) = 1$ и $n - k \ne k$, при $n > 2$. Утверждение 4) имеет место потому, что, если n имеет делитель $1 < d < n$, то тогда значение $\varphi(n)$ меньше максимального $n - 1$.

Теорема 7. Функция Эйлера есть *мультипликативная* функция, что означает

$$\forall m, n \in \mathbf{Z}, \gcd(m, n) = 1 \text{ влечет } \varphi(mn) = \varphi(m)\varphi(n).$$

Для доказательства нам потребуется

Лемма 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:

$$
\begin{array}{lllll}
1 & m+1 & 2m+1 & \ldots & (n-1)m+1 \\
2 & m+2 & 2m+2 & \ldots & (n-1)m+2 \\
3 & m+3 & 2m+3 & \ldots & (n-1)m+3 \\
\ldots & & & & \\
r & m+r & 2m+r & \ldots & (n-1)m+r \quad \leftarrow \text{строка с номером } r \\
\ldots & & & & \\
m & 2m & 3m & \ldots & nm
\end{array}
$$

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 $\gcd(r, m) = 1$:

$$r, m + r, 2m + r, \ldots, (n-1)m + r.$$

Так как $\gcd(m, n) = 1$, то по лемме 3 числа этой строки – полная система вычетов по модулю n, т.е. остатки этих чисел при делении на n образуют перестановку чисел $0, 1, 2, \ldots, n-1$, среди которых $\varphi(n)$ штук взаимно простых с n чисел. Поэтому ровно $\varphi(n)$ элементов строки с номером r являются взаимно простыми с n и, следовательно, с mn.

Поэтому $\varphi(m)$ строк содержат натуральные числа, взаимно простые с mn, и в каждой такой строке содержится $\varphi(n)$ элементов, взаимно простых с mn. Таким образом, таблица содержит $\varphi(m)\varphi(n)$ натуральных чисел $\le mn$ и взаимно простых с mn; т.е. $\varphi(mn) = \varphi(m)\varphi(n)$. ■

Свойство мультипликативности сводит вычисление $\varphi(n)$ к вычислению $\varphi(p^\alpha)$, где p – простое и $\alpha \ge 1$. Чтобы вычислить значение $\varphi(p^\alpha)$, 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

$$h = pq, \qquad 1 \le q \le p^{\alpha-1}.$$

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 $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$ есть разложение числа n на простые множители, то получаем

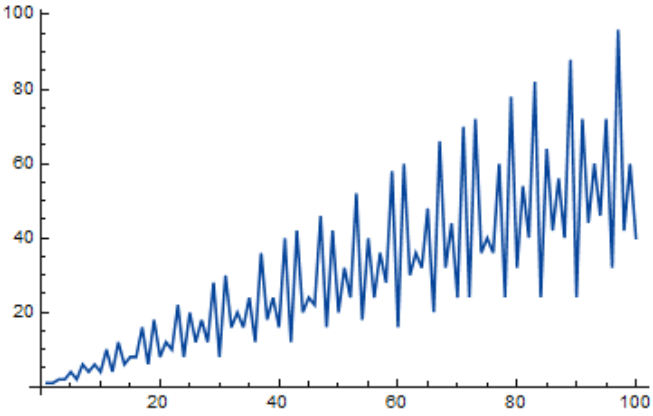
$$\varphi(n) = (p_1^{\alpha_1} - p_1^{\alpha_1-1}) \cdots (p_1^{\alpha_k} - p_1^{\alpha_k-1}) = n(1-1/p_1) \cdots (1-1/p_k). \qquad (7)$$

Таким образом, полученная формула для вычисления $\varphi(n)$ требует факторизация, а известные алгоритмы факторизации не являются полиномиальными относительно количества цифр числа n, поэтому задача нахождения $\varphi(n)$ вычислительно трудная.

В языке Wolfram функция Эйлера $\varphi(n)$ вычисляется с помощью EulerPhi[n].

График функции Эйлера для первых ста чисел:

```
ListPlot[EulerPhi[Range[100]], Joined → True]
```



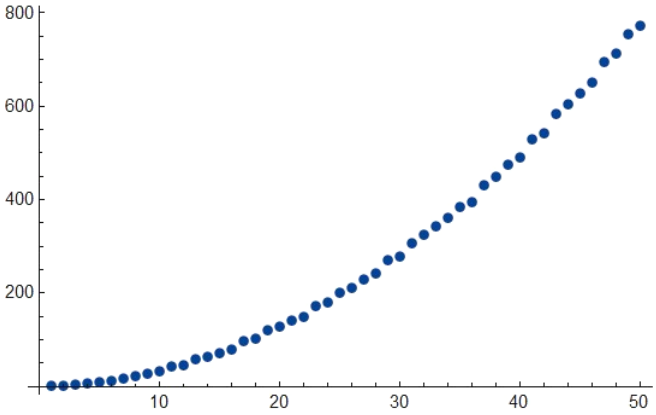
В справочной службе системы Mathematica приведено несколько интересных численных значений функции Эйлера.

Функция Эйлера для степеней 10:

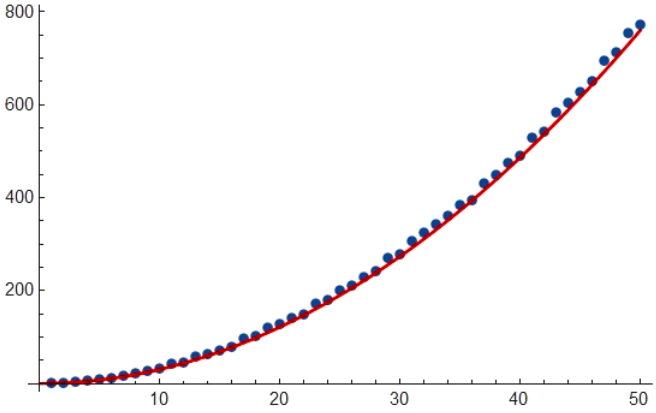
```
Table[EulerPhi[10^k], {k, 0, 10}]
{1, 4, 40, 400, 4000, 40000, 400000, 4000000, 40000000, 40000000,
4000000000}
```

Накопленные суммы функции Эйлера аппроксимируются функцией $(3n^2)/\pi^2$, см. [44],

```
ListPlot[Table[Sum[EulerPhi[k], {k, 1, n}], {n, 50}]]
```




```
Show[%, Plot[3n^2/π^2, {n, 0, 50}, PlotStyle → Red]]
```



О встроенной функции Show см. справочную службу Mathematica.

Докажем еще несколько свойств Эйлера.

Теорема 8. Функция Эйлера обладает следующими свойствами:

(a) $\varphi(mn) = \varphi(m)\varphi(n)(d/\varphi(m))$, где $d = (m, n)$.

(b) Если $a \mid b$, то $\varphi(a) \mid \varphi(b)$.

(c) $\varphi(n)$ – четно для $n \geq 3$. Более того, если n имеет r различных нечетных простых делителей, то $2^r \mid \varphi(n)$.

(d) Если n имеет r различных простых множителей, то

$$\varphi(n) \geq n\prod_{i=1}^{r}(1-1/p_i),$$

где $p_1, p_2, \ldots, p_r$ – первые простые числа в порядке возрастания.

Доказательство. (a) Отношение (7) можно записать в виде

$$\frac{\varphi(n)}{n} = \prod_{p|n}(1-\frac{1}{p}),$$

где запись $p|n$ обозначает, что в произведении p пробегает по всем простым делителям n. Далее заметим, что каждый простой делитель p произведения mn делит m или n и если делит оба числа m и n, то p делит также (m, n). Следовательно,

$$\frac{\varphi(nm)}{nm} = \prod_{p|nm}(1-\frac{1}{p}) = \frac{\prod_{p|n}(1-\frac{1}{p})\prod_{p|m}(1-\frac{1}{p})}{\prod_{p|(n,m)}(1-\frac{1}{p})} = \frac{\frac{\varphi(n)}{n}\frac{\varphi(m)}{m}}{\frac{\varphi(d)}{d}},$$

т.е. получаем (a).

Выведем (b) из (a). Так как $a|b$, то $b = ac$, где $1 \leq c \leq b$. Если $c = b$, то $a = 1$ и часть (b) тривиально выполняется. Поэтому допустим $c < b$. Из (a) имеем

$$\varphi(b) = \varphi(ac) = \varphi(a)\varphi(c)\frac{d}{\varphi(d)} = d\varphi(a)\frac{\varphi(c)}{\varphi(d)}, \tag{8}$$

где $d = (a, c)$. Нужный результат следует по математической индукции по b. Базис индукции для $b = 1$ тривиально выполняется. Предположим, что часть (b) имеет место для всех натуральных чисел меньших b. Поэтому утверждение (b) справедливо для c, т.е. $\varphi(d) \mid \varphi(c)$, так как $d|c$. Следовательно, правая часть в (8) кратна $\varphi(a)$, что означает $\varphi(a) \mid \varphi(b)$. Это доказывает часть (b).

Сейчас докажем (c). Если $n = 2^a$, $a \geq 2$, то $\varphi(n) = 2^a - 2^{a-1}$ влечет четность $\varphi(n)$. Если n имеет хотя бы один нечетный простой делитель, то запишем

$$\varphi(n) = n\prod_{p|n}\frac{p-1}{p} = \frac{n}{\prod_{p|n}p}\prod_{p|n}(p-1) = c\prod_{p|n}(p-1),$$

где c – какое-то целое число. Произведение, которое умножаем на c

четно, поэтому и $\varphi(n)$ четно. Более того, каждый простой нечетный делитель дает множитель 2 в это произведение, поэтому получаем $2^r \mid \varphi(n)$, если n имеет r различных нечетных простых делителей.

(d) Пусть $q_1, q_2, \ldots, q_r$ – все различные простые делители числа n в возрастающем порядке. Тогда в силу (7) имеем

$$\varphi(n) = n\prod_{i=1}^{r}(1-1/q_i) \geq n\prod_{i=1}^{r}(1-1/p_i),$$

где последнее неравенство следует из $q_i \geq p_i$. ■

Прообраз функции EulerPhi

Нижняя граница для $\varphi(n)$ поможет нам вычислить прообраз для функции φ. Предположим, что требуется найти все числа x, такие, что $\varphi(x) = 1000$. Пусть $q_1, q_2, \ldots, q_r$ – все различные простые делители числа x. Тогда по теореме 8 (d) имеем

$$\varphi(x) = x\prod_{i=1}^{r}(1-1/q_i) \geq x\prod_{i=1}^{r}(1-1/p_i) \geq \prod_{i=1}^{r}p_i\prod_{i=1}^{r}(1-1/p_i) = \prod_{i=1}^{r}(p_i-1).$$

Оценим последнее произведение для $r = 4, 5, 6, 7$:

```
Table[∏_{i=1}^{r} (Prime[i] - 1), {r, 4, 7}]
```

{48, 480, 5760, 92160}

Мы видим, что $r \leq 5$. Это означает, что $\varphi(x) \geq x\prod_{i=1}^{5}(1-1/p_i)$ так что, если $\varphi(x) = 1000$, то $x \leq 1000/\prod_{i=1}^{5}(1-1/p_i)$. Имеем

```
N[1000 / ∏_{i=1}^{5} (EulerPhi[i] - 1)]
```

4812.5

Поэтому мы можем просто проверить интервал [1001, 4812], чтобы найди подходящее значение x:

```
Select[Range[1001, 4812], EulerPhi[#] = = 1000&]
```

{1111, 1255, 1375, 1875, 2008, 2222, 2500, 2510, 2750, 3012, 3750}

Руководствуясь действиями в данном примере, легко написать функцию *phiInverse*, которая является обратной для функции Эйлера; точнее, *phiInverse*[*n*] находит список всех *x*, для которых EulerPhi[*x*] равен *n*.

Удобно предварительно создать список значений $\prod_{i=1}^{k}(Prime[i]-1)$, для $1 \le k \le 20$. Этот список будет использоваться каждый раз при вызове *phiInverse*. При необходимости этот список можно расширить и для больших значений *k*:

```
cs = Table[Product[Prime[i] - 1, {i, 6}], {i, 20}]
```

{1, 2, 8, 48, 480, 5760, 92160, 1658880, 36495360, 1021870080, 30656102400, 1103619686400, 44144787456000, 1854081073152000, 85287729364992000, 4434961926979584000, 257227791764815872000, 15433667505888952320000, 1018622055388670853120000, 71303543877206959718400000}

```
phiInverse[n_] :=
  Module[{k = 1, r}, While[cs[[k]] ≤ n, k++]; r = k – 1;
    Select[Range[n + 1, ⌊n / Product[1 – 1 / Prime[i], {i, 1, r}]⌋],
   EulerPhi[#] == n&]]
```

Никогда не лишне проверить:

```
phiInverse[1000]
```

{1111, 1255, 1375, 1875, 2008, 2222, 2500, 2510, 2750, 3012, 3750}

Функция работает и для больших значений аргумента:

```
phiInverse [1000000]
```

{1004251, 1039391, 1062517, 1107161, 1253125, 1286375, 1328125, 1394305, 1409375, 1901325, 1921875, 2000032, 2008502, 2078782, 2083125, 2125034, 2214322, 2230888, 2500000, 2500040, 2506250, 2510000, 2525000, 2535100, 2562500, 2572750, 2656250, 2750000, 2761000, 2777500, 2788610, 2818750, 3000048, 3346332, 3750000, 3750060, 3765000, 3787500, 3802650, 3843750, 4125000, 4141500, 4166250}

```
AllTrue[EulerPhi[%], # == 1000000&]
```

True

Функция Эйлера $\varphi(n)$ принимает только четные значения, но, оказывается, не все. Рассмотрим прообразы функции Эйлера для четных чисел, не превосходящих 100. Некоторые четные числа никогда не совпадают со значениями функции $\varphi(n)$:

```
Select[Range[2, 100, 2], phiInverse[#] == {}&]
{14, 26, 34, 38, 50, 62, 68, 74, 76, 86, 90, 94, 98}
```

Все значения функции phiInverse[m] для четных $m \le 36$:

```
{#, phiInverse[#]}& /@ Range[2, 36, 2] // Column
{2, {3, 4, 6}}
{4, {5, 8, 10, 12}}
{6, {7, 9, 14, 18}}
{8, {15, 16, 20, 24, 30}}
{10, {11, 22}}
{12, {13, 21, 26, 28, 36, 42}}
{14, {}}
{16, {17, 32, 34, 40, 48, 60}}
{18, {19, 27, 38, 54}}
{20, {25, 33, 44, 50, 66}}
{22, {23, 46}}
{24, {35, 39, 45, 52, 56, 70, 72, 78, 84, 90}}
{26, {}}
{28, {29, 58}}
{30, {31, 62}}
{32, {51, 64, 68, 80, 96, 102, 120}}
{34, {}}
{36, {37, 57, 63, 74, 76, 108, 114, 126}}
```

Функция валентности $v(n)$ определяется как число целых положительных чисел k, таких, что $\varphi(k) = n$. Известна гипотеза Кармайкла: валентность любого числа не равна 1 [32]. Используя обращение функции Эйлера, легко обнаружить отсутствия контрпримера при $n \le 5000$:

```
Parallelize[Select[{#, phiInverse[#]}& /@ Range[2, 5000, 2],
Length[#[[2]]] == 1&]]
{}
```

Если контрпример существует, то он должен иметь более 10 000 000 цифр [30].

Теорема Эйлера

Прежде чем сформулировать теорему Эйлера, приведем необходимое определение. **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 4. 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 не сравнимы (доказательство как в лемме 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 9 (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 4). 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 $a^{\varphi(m)} \equiv 1 (\mathrm{mod}\, m)$. ■

Следствие. Пусть $m > 1$, $\gcd(a, m) = 1$ и $a \equiv b \pmod{\varphi(m)}$. Тогда $s^a \equiv s^b \pmod m$.

Доказательство. Запишем a в виде суммы $k\,\varphi(m) + b$. Тогда $s^a = s^{k\,\varphi(m)+b} = s^{k\,\varphi(m)} s^b \equiv s^b \pmod m$. ■

Доказанное свойство часто используется для вычисления больших степеней вручную. Например, если мы хотим узнать значение $10^{12340} \pmod{21}$, то можем использовать тот факт, что $\varphi(21) = 12$ и $10^{12} \equiv 1 \pmod{21}$. Имеем $10^{12340} = 10^{1028 \cdot 12 + 4} = (10^{12})^{1028} 10^9 \equiv 10^4 \equiv 100^2 \equiv \equiv (-4)^2 \equiv 16 \pmod{21}$.

Теорема Эйлера может быть переформулирована следующим образом: если $\gcd(a, m) = 1$, то последовательность $a^k \pmod{m}$, где $k = 0, 1, 2, \ldots$, является периодической с периодом $\varphi(m)$.

Частный случай этой теоремы принадлежит П. Ферма.

Теорема 10 (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 1. 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 2. Пусть p – простое число и p не делит a. Тогда

$$a^{-1} = a^{p-2} \bmod p. \tag{9}$$

Доказательство. По малой теореме Ферма $a^{p-1} \equiv 1 \pmod{p}$. То есть $a \cdot a^{p-2} \equiv 1 \pmod{p}$, 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 (9):

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

Итерация функции Эйлера

Пусть n – натуральное число. Определим последовательность n, n_1, n_2, … рекурсивно: $n_1 = \varphi(n)$ и $n_{k+1} = \varphi(n_k)$ для $k = 1, 2, 3, \ldots$. Заметим, что $1 \le \varphi(m) \le m - 1$ для $m > 1$. Следовательно, если $n \ge 2$, то $n > n_1 > n_2 > \ldots \ge 1$. Так как n, n_1, n_2, … – убывающая последовательность чисел, то существует такое целое число r, что $n_r = 1$.

Пусть $r(n)$ – функция от натурального числа n, дающая число итераций r функции Эйлера, такое, что $n > n_1 > n_2 > \ldots \ge n_r = 1$. Напишем первую реализацию в виде функции f на языке Wolfram, вычисляющую значения $r(n)$. Это легко сделать, используя функцию NestWhileList:

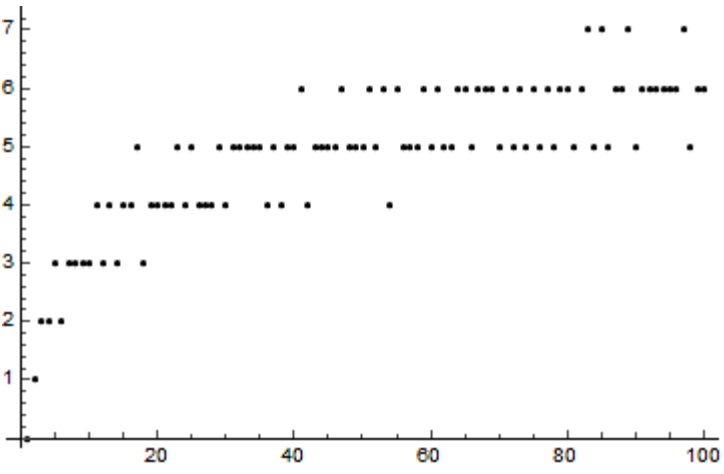
```
f[n_] := Length[NestWhileList[EulerPhi, n, # > 1&]] – 1
```

Значения этой функции для первых 20 натуральных чисел таковы

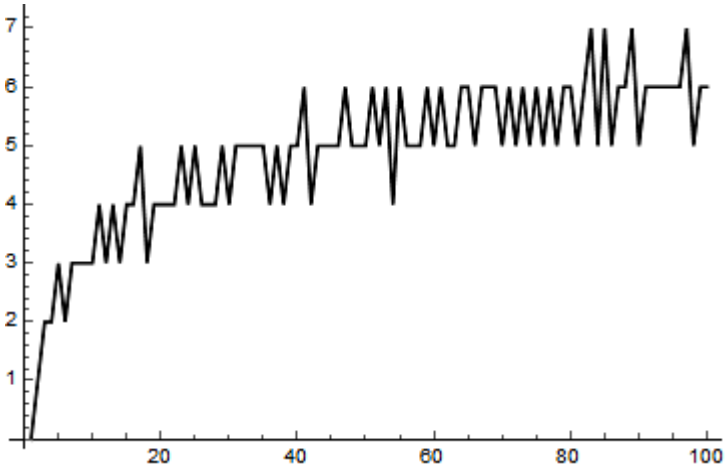
```
f /@ Range[20]
```
{0, 1, 2, 2, 3, 2, 3, 3, 3, 3, 4, 3, 4, 3, 4, 4, 5, 3, 4, 4}

Для чисел, не превосходящих 100, изобразим два графика (на втором графике соседние значения функции $r(n)$ соединим отрезками):

```
ListPlot[f /@ Range[100]]
```

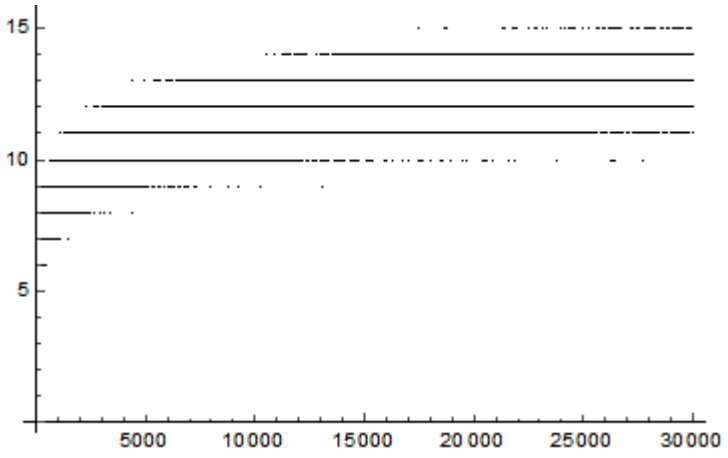


```
ListPlot[f /@ Range[100], Joined → True]
```



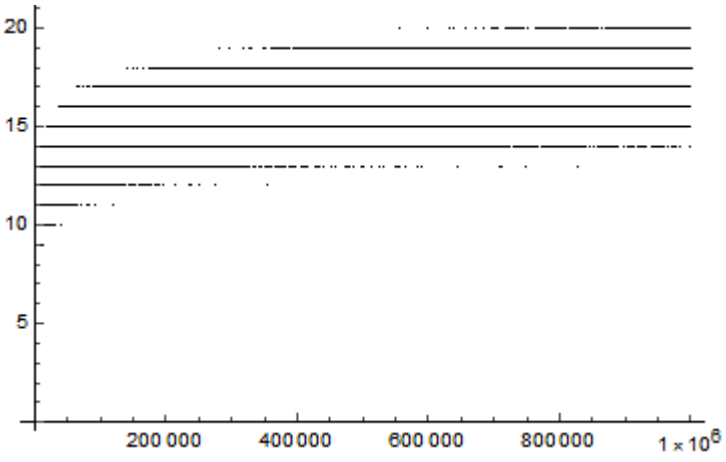
Поведение функции $r(n)$ будет более понятно, если рассматривать график на большом диапазоне:

```
ListPlot[f /@ Range[30000]]
```



И еще увеличим диапазон:

```
ListPlot[Parallelize[f /@ Range[1000000]]]
```



Значения функции $r(n)$ растут не монотонно, меняясь в некотором диапазоне, верхняя и нижняя границы которого монотонно возрастают. Как узнать, в каких пределах колеблется $r(n)$ при увеличении n? Границу сверху задает функция $maxr(n) = \max\{r(k) | \ 1 \le k \le n\}$, где max обозначает наибольшее число во множестве. Хотя вычисления значений $maxr(n)$ можно проделывать непосредственно, исходя из определения функции f, но это будет неэффективно, так как одни и те же значения функции $\varphi(n)$ придется вычислять многократно.

Мы напишем новую реализацию $r(n)$ с помощью рекурсивной функции g, которая будет запоминать предыдущие вычисленные значения $r(n)$:

```
g[1] = 0;
g[n] := g[n] = 1 + g[EulerPhi[n]]
```

Результаты функций f и g равны, но g более быстро вычисляется для больших n, хотя это и требует больше памяти.

Вызов функции g

```
g[30000] ;
```

определяет значения $g[n]$ для всех n от 1 до 30000 и сохраняет их в памяти. При любом дальнейшем обращение к g значение функции не вычисляется, а извлекается из памяти.

Что касается функции *maxr*, то ее значения зададим, используя конструкцию Array языка Wolfram. Во-первых, создадим символьный массив из 30000 элементов вида *maxg*[*i*]:

```
Array[maxg, 30000];
```

А теперь в цикле по возрастанию *i* присвоим значения *maxg*[*i*]:

```
For[i = 1; maxg[1] = 0, i ≤ 30000, i++,
    maxg[i + 1] = If[g[i + 1] > maxg[i], g[i + 1], maxg[i]]]
```

Нижнюю границу для значений $r(n)$ определим с помощью функции $minr(n) = \min\{r(k) | \ n \le k \le 30000\}$, где min означает наименьшее число множества. Программируем таким же образом, как и для верхней границы. Создаем символьный массив *ming*[*i*]:

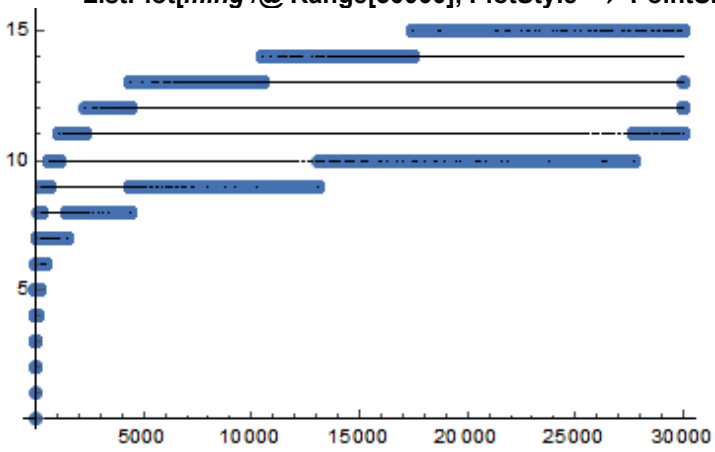
```
Array[ming, 30000];
```

Но теперь в цикле по убыванию *i* присваиваем значения *ming*[*i*]:

```
For[i = 30000; ming[30000] = g[30000], i > 1, i – –,
    ming[i – 1] = If[g[i – 1] < ming[i], g[i – 1], ming[i]]]
```

Значения *maxg*[*i*] и *ming*[*i*] являются соответственно «верхней огибающей» и «нижней огибающей» для графика функции *g*, поэтому покажем сразу три графика вместе (точки для *maxg* и *ming* изобразим крупнее чем точки *g*):

```
Show[ListPlot[g /@ Range[30000]],
     ListPlot[maxg /@ Range[30000], PlotStyle → PointSize[0.02]],
     ListPlot[ming /@ Range[30000], PlotStyle → PointSize[0.02]]]
```



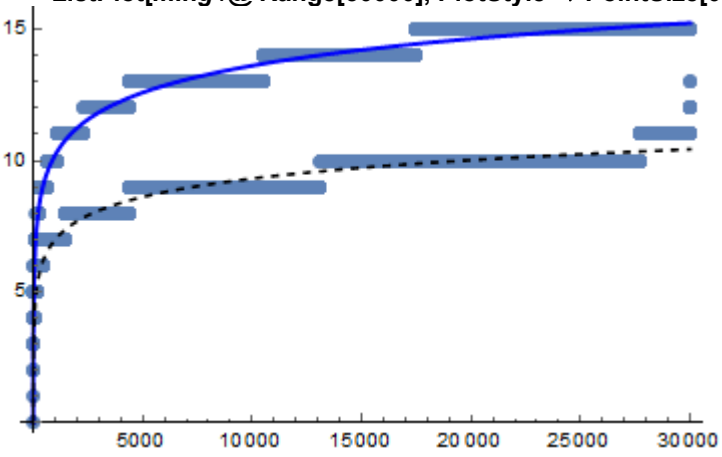
Функции *maxr* и *minr* аппроксимируются функцией натуральный логарифм. Аппроксимацию проводим с помощью функции Fit:

```
data1 = {#, maxg[#]}& /@ Range[30000];
Fit[data1, Log[x], x]
```
1.4787 Log[x]

```
data2 = {#, ming[#]}& /@ Range[30000];
Fit[data2, Log[x], x]
```
1.0117 Log[x]

Чтобы оценить аппроксимации, изобразим четыре графика вместе:

```
Show[Plot[{1.4787 Log[x], 1.0117 Log[x]}, {x, 1, 30000},
      PlotLegends → "Expressions", PlotStyle → {Thick, Dashed}],
  ListPlot[maxg /@ Range[30000], PlotStyle → PointSize[0.02]],
  ListPlot[ming /@ Range[30000], PlotStyle → PointSize[0.02]]
```



Если взять диапазон чисел от 1 до 10^6 и повторить предыдущие вычисления с этим диапазоном, то «верхняя огибающая» и «нижняя огибающая» аппроксимируются соответственно функциями 1.47317 Log[x] и 0.986594 Log[x].

Задачи с функцией Эйлера

Задача 4. Для каких натуральных n выполнено $\varphi(3n) = 3\varphi(n)$?

Выясним сначала, для каких чисел n, не больших 100, выполнено данное равенство:

```
Select[Range[100], EulerPhi[3#] == 3EulerPhi[#]&]
{3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60,
63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99}
```

Возможно, равенство выполняется тогда и только тогда, когда n делится на 3. Докажем это. Пусть $n = 3^k m$, где $\gcd(3, m) = 1$. Если $k = 0$, то $\varphi(3n) = 2\varphi(n) \neq 3\varphi(n)$. С другой стороны, если $k \geq 1$, то $\varphi(3n) = \varphi(3^{k+1} m) = (3^{k+1} - 3^k)\,\varphi(m) = 3(3^k - 3^{k-1})\,\varphi(m) = 3\varphi(3^k m) = 3\varphi(n)$. Поэтому $\varphi(3n) = 3\varphi(n)$ тогда и только тогда, когда $3|n$.

Задача 5. Для каких натуральных n выполнено $\varphi(n)|n$?

Проведем эксперименты. Проверим, для каких чисел n, не больших 100, выполнено $\varphi(n)|n$:

```
Select[Range[100], Divisible[#, EulerPhi[#]]&]
{1, 2, 4, 6, 8, 12, 16, 18, 24, 32, 36, 48, 54, 64, 72, 96}
```

Мы видим, что необходимым свойством обладают степени числа 2 и еще некоторые четные числа. Чтобы определить, какие это числа, факторизуем их:

```
FactorInteger[%] // Column
{{1, 1}}
{{2, 1}}
{{2, 2}}
{{2, 1}, {3, 1}}
{{2, 3}}
{{2, 2}, {3, 1}}
{{2, 4}}
{{2, 1}, {3, 2}}
{{2, 3}, {3, 1}}
{{2, 5}}
{{2, 2}, {3, 2}}
{{2, 4}, {3, 1}}
{{2, 1}, {3, 3}}
{{2, 6}}
{{2, 3}, {3, 2}}
{{2, 5}, {3, 1}}
```

Можно предположить, что n имеет вид

$$2^\alpha \text{ или } 2^\beta 3^\gamma, \text{ где } \alpha \geq 0 \text{ и } \beta > 0,\ \gamma > 0. \qquad (10)$$

Проверим гипотезу в диапазоне от 1000 до 2000:

```
Select[Range[1000, 2000], Divisible[#, EulerPhi[#]]&]
{1024, 1152, 1296, 1458, 1536, 1728, 1944}

FactorInteger[%] // Column
{{2, 10}}
{{2, 7}, {3, 2}}
{{2, 4}, {3, 4}}
{{2, 1}, {3, 6}}
{{2, 9}, {3, 1}}
{{2, 6}, {3, 3}}
{{2, 3}, {3, 5}}
```

Проверим, не пропускаются ли какие-нибудь числа вида (10):

```
s1 = Select[Range[10000], Not[Divisible[#, EulerPhi[#]]&]];
```

Список $s1$ содержит числа $n \le 1000$, для которых $\varphi(n)$ не делит n. А есть ли там числа вида (10)?

```
Intersection[s1,  Table[2^k, {k, 0, 13}]]
{}
```

Степеней числа 2 там нет. Проверим наличие чисел вида $2^\beta 3^\gamma$, где $\beta > 0$, $\gamma > 0$:

```
Select[s1, Divisible[#, 6]&]
{}
```

Как видим, в $s1$ нет чисел вида (10).

Докажем наше предположение. Если n имеет факторизацию $n = p_1^{a_1} \cdots p_r^{a_r}$ и $\varphi(n)|n$, то

$$k\,\varphi(n) = kn(p_1 - 1)/p_1 \times \cdots \times (p_r - 1)/p_r = n,$$

так что $k = p_1/(p_1 - 1) \times \cdots \times p_r/(p_r - 1)$ – целое число. Поэтому p_i не могут быть все нечетными, следовательно, $p_1 = 2$. Поэтому в знаменателе дроби для k может быть самым большим только одно четное выражение вида $p_i - 1$, где p_i – простое нечетное число. Таким образом, $n = 2^{a_1}$ и $\varphi(n) = n/2$ или $n = 2^{a_1} p^{a_2}$ и $\varphi(n) = n\,((2–1)/2)((p - 1)/p)$, т.е. $k = 2p\,(p - 1)$. В этом случае получаем $p = 2$ или $p = 3$ и тогда $n = 2^{a_1}3^{a_2}$. Поэтому n может принимать только значения $1, 2^{a_1}, 2^{a_1}3^{a_2}$, где $a_1, a_2 \ge 1$.

Задача 6. Как связаны $\varphi(n)$ и $\varphi(2n)$?

Вычислим $\varphi(n)$ и $\varphi(2n)$ для первых двадцати чисел:

```
{#, EulerPhi[#], EulerPhi[2#]}& /@ Range[20]//Column
{1, 1, 1}
{2, 1, 2}
{3, 2, 2}
{4, 2, 4}
{5, 4, 4}
{6, 2, 4}
{7, 6, 6}
{8, 4, 8}
{9, 6, 6}
{10, 4, 8}
{11, 10, 10}
{12, 4, 8}
{13, 12, 12}
{14, 6, 12}
{15, 8, 8}
{16, 8, 16}
{17, 16, 16}
{18, 6, 12}
{19, 18, 18}
{20, 8, 16}
```

Докажем, что если n – натуральное число, то

$$\varphi(2n) = \begin{cases} \varphi(n), \text{ если } n \text{ нечетно;} \\ 2\varphi(n), \text{ если } n \text{ четно.} \end{cases}$$

Если n нечетно, то $(2, n) = 1$ и $\varphi(2n) = \varphi(2)\varphi(n) = 1 \cdot \varphi(n) = \varphi(n)$. Если n четно, то, скажем, $n = 2^s t$, где t нечетно. Тогда $\varphi(2n) = \varphi(2^{s+1}t) = \varphi(2^{s+1})\varphi(t) = 2^s\varphi(t) = 2(2^{s-1}\varphi(t)) = 2\varphi(2^s)\varphi(t) = 2\varphi(2^s t) = 2\varphi(n)$.

Задача 7. Для каких чисел выполнено $n = 2\varphi(n)$?

Найдем числа, не бóльшие 100, для которых выполнено указанное свойство:

```
Select[Range[100], # == 2EulerPhi[#]&]
{2, 4, 8, 16, 32, 64}
```

Полученные числа – натуральные степени числа 2. Докажем, что это справедливо в общем случае. Пусть n имеет факторизацию

$n = p_1^{a_1} \cdots p_r^{a_r}$. Тогда

$$p_1^{a_1} \cdots p_r^{a_r} = 2\prod_{j=1}^{r} p_j^{a_j - 1}(p_j - 1).$$

Сокращая степени всех p_j, получаем $p_1 \cdots p_r = 2(p_1 - 1) \cdots (p_r - 1)$. Если какое-нибудь p_j – нечетное простое число, то множитель $p_j - 1$ – четное число и поэтому должен делить произведение слева. Но слева в качестве множителя может быть только одна двойка, которая соответствует множителю 2 в правом произведении в последнем равенстве. Поэтому нечетных простых чисел в факторизации n нет, т.е. $n = 2^j$ для некоторого натурального числа j.

Задача 8. Для каких натуральных чисел $n > 1$ выполнено $\varphi(n)|(n-1)$?

Найдем числа, не большие 100, для которых выполнено указанное свойство:

```
Select[Range[100], Divisible[# – 1, EulerPhi[#]]&]
```
{1, 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}

Если диапазон увеличить, то снова будут только простые числа. Так как для простого n имеем $\varphi(n) = n - 1$, то возникает вопрос, есть ли составное число n, для которого $\varphi(n)$ делит $n - 1$?

Известно, что если n – составное натуральное число и $\varphi(n)|(n-1)$, то n должно быть произведением не менее трех различных простых чисел и свободно от квадратов.

Действительно, если $n = p^k m$, где p – простое, то $\varphi(p^k m) = (p^k - p^{k-1}) \times \varphi(m) \mid (p^k m - 1)$, то $p|1$ или $k = 1$. Таким образом, n свободно от квадратов. Если $n = pq$ – произведение двух простых чисел, то $\varphi(pq) = (p-1)\times (q-1) \mid (pq - 1)$. Тогда $(p-1)$ делит $(pq-1) - (p-1)q = q - 1$. Точно так же заключаем, что $q - 1$ делит $p - 1$. Отсюда получаем противоречие $p = q$.

Попробуем найти такое составное число:

```
Parallelize[Select[Range[2, 1000000000], Divisible[# – 1, EulerPhi[#]]
&& Not[PrimeQ[#]]&]]
```
{}

Получили, что если составное число n со свойством $\varphi(n)|(n-1)$ существует, то n должно быть больше 109.

§ 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. Содержание данного параграфа отвечает на вопрос: когда целое число a является точным квадратом по модулю простого числа p? Работы Эйлера, Лежандра и Гаусса по этому и связанным вопросам привели к развитию большей части современной теории чисел. Сначала мы определим понятие квадратичного вычета – целого числа a, являющегося квадратом по модулю p, и установим основные свойства квадратичных вычетов. Введем символ Лежандра, обозначение, которое говорит нам, является ли данное целое число квадратичным вычетом по указанному модулю и развивает его основные свойства. Сформулируем и докажем два важных критерия, открытых Эйлером и Гауссом, для определения того, является ли данное число квадратичным вычетом. Будет рассмотрен закон квадратичной взаимности Гаусса и символ Якоби, который является обобщением символа Лежандра. И изучим, как при рассмотрении этих вопросов применяется Wolfram Mathematica.

Квадратичные вычеты и невычеты

Начнем с определения. Если положительное целое число m и целое a взаимно простые, то a называется **квадратичным вычетом по модулю *m*** тогда и только тогда, когда сравнение $x^2 \equiv a \pmod{m}$ имеет решение. Если такого x не существует, то a называется **квадратичным невычетом по модулю *m***.

Пример 7. Найдем все квадратичные вычеты для чисел $m = 24$, $m = 19$ и $m = 23$. Для этого просто будем искать числа a, являющиеся квадратами, перебирая x в диапазоне от 1 до $m - 1$ включительно.

```
Mod[#^2, 24]& /@ Range[23]
{1, 4, 9, 16, 1, 12, 1, 16, 9, 4, 1, 0, 1, 4, 9, 16, 1, 12, 1, 16, 9, 4, 1}
Union[%]
{0, 1, 4, 9, 12, 16}
Mod[#^2, 19]& /@ Range[18]
{1, 4, 9, 16, 6, 17, 11, 7, 5, 5, 7, 11, 17, 6, 16, 9, 4, 1}
Union[%]
{1, 4, 5, 6, 7, 9, 11, 16, 17}
```

```
Mod[#^2, 23]& /@ Range[22]
{1, 4, 9, 16, 2, 13, 3, 18, 12, 8, 6, 6, 8, 12, 18, 3, 13, 2, 16, 9, 4, 1}

Union[%]
{1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18}
```

Как видим, для составного $m = 24$ имеется 6 вычетов: 0, 1, 4, 9, 12, 16 и 17 невычетов. Для простых модулей $m = 19$ и $m = 23$ соответственно вычетов 9 и 11. И количество невычетов для этих значений m равно количеству вычетов. Это не случайно.

Теорема 11. 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 $x_3 \equiv -x_1 \equiv x_2 \pmod p$. То есть на самом деле все равно имеется два решения. ■

Теорема 12. 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. ■

Если p – нечетное простое число, то **символ Лежандра** $\left(\frac{a}{p}\right)$ для целого a определяется как

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

Таким образом, символ Лежандра показывает, является ли ненулевой вычет a по модулю p квадратом по модулю p. Символ называется в честь французского математика Анри Мари Лежандра (1752–1833), который ввел это обозначение. Мы можем провести аналогию символа Лежандра со знаком вещественного числа: если вещественное число a отрицательное, т.е. имеет вид $(-1)|a|$, то оно не имеет квадратного корня в $\mathbb{R}$, и признаком отсутствия квадратного корня из $a \in \mathbb{Z}_p$ является значение $\left(\frac{a}{p}\right) = -1$.

Пример 8. Предыдущий пример показывает, что символ Лежандра $\left(\frac{a}{19}\right)$ при $a = 1, 2, 3, \ldots, 18$ равен

$$\left(\frac{1}{19}\right)=\left(\frac{4}{19}\right)=\left(\frac{5}{19}\right)=\left(\frac{6}{19}\right)=\left(\frac{7}{19}\right)=\left(\frac{9}{19}\right)=\left(\frac{11}{19}\right)=\left(\frac{16}{19}\right)=\left(\frac{17}{19}\right)=1,$$

$$\left(\frac{2}{19}\right)=\left(\frac{3}{19}\right)=\left(\frac{8}{19}\right)=\left(\frac{10}{19}\right)=\left(\frac{12}{19}\right)=\left(\frac{13}{19}\right)=\left(\frac{14}{19}\right)=\left(\frac{15}{19}\right)=\left(\frac{18}{19}\right)=-1.$$

Мы сейчас предоставим критерий, который позволяет определить, является ли целое число квадратичным вычетом простого числа. Этот критерий будет полезен для демонстрации свойств символа Лежандра.

Теорема 13 (критерий Эйлера). Пусть a не кратно простому нечетному 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 $a^{(p-1)/2} \equiv 1 \pmod p$, так как, будучи сравнением степени $(p-1)/2$, оно не может иметь более $(p-1)/2$ решений (теорема 6). Это означает, что квадратичные невычеты удовлетворяют сравнению $a^{(p-1)/2} \equiv -1 \pmod p$. ■

Заметим, что из критерия Эйлера сразу следует $\left(\frac{1}{p}\right) = 1$ для любого нечетного простого p.

Пример 9. Применим критерий Эйлера для нахождения всех квадратичных вычетов a по модулю p, где $a \in [1, 36]$.

```
Map[#[[1]]&, Select[{#, Mod[#^18, 37]}& /@ Range[36], #[[2]] == 1&]]
```
{1, 3, 4, 7, 9, 10, 11, 12, 16, 21, 25, 26, 27, 28, 30, 33, 34, 36}

Докажем сейчас некоторые свойства символа Лежандра.

Теорема 14. Пусть p – нечетное простое число и $a, b \in \mathbb{N}$ не делятся на p. Тогда имеем

(i) если $a \equiv b \pmod p$, то $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$;

(ii) $\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right)$;

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

Доказательство. (i) Если $a \equiv b \pmod p$, то $x^2 \equiv a \pmod p$ имеет решение тогда и только тогда, когда $x^2 \equiv b \pmod p$ имеет решение. Следовательно, $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$.

(ii) Мы знаем, что по критерию Эйлера

$$a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right) \pmod p,\ b^{\frac{p-1}{2}} \equiv \left(\frac{b}{p}\right) \pmod p \text{ и } (ab)^{\frac{p-1}{2}} \equiv \left(\frac{ab}{p}\right) \pmod p.$$

Поэтому

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

Так как возможные значения для символа Лежандра есть ± 1, то мы заключаем, что

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

(iii) Так как $\left(\frac{a}{p}\right)=\pm 1$, то из (ii) следует, что

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

Пункт (ii) теоремы имеет интересное следствие: произведение двух квадратичных вычетов или двух квадратичных невычетов является квадратичным вычетом, а произведение квадратичного вычета и невычета является квадратичным невычетом.

Когда $p-1$ является квадратичным вычетом по модулю p?

Выясним, для каких простых p число -1 является квадратичным вычетом. На языке Wolfram это нетрудно проверить для первых 50-ти нечетных простых чисел, используя критерий Эйлера.

```
Select[Prime[Range[2, 51]], PowerMod[ – 1, (# – 1)/2, #] == 1&]
{5, 13, 17, 29, 37, 41, 53, 61, 73, 89, 97, 101, 109, 113, 137, 149, 157,
173, 181, 193, 197, 229, 233}
```

Мы видим, что все такие простые числа p удовлетворяют сравнению $p \equiv 1 (\mathrm{mod}\ 4)$.

```
Times@@Mod[%, 4]
1
```

Теорема 15. Если p – нечетное простое число, то

$$\left(\frac{-1}{p}\right)=\begin{cases}1, \text{ если } p\equiv 1(\mathrm{mod}\,4);\\ -1, \text{ если } p\equiv -1(\mathrm{mod}\,4).\end{cases}$$

Доказательство. По критерию Эйлера имеем

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

Если $p \equiv 1 \pmod 4$, то $p = 4k + 1$ для некоторого положительного целого k. Поэтому

$$(-1)^{(p-1)/2}=(-1)^{2k}=1,$$

так что $\left(\frac{-1}{p}\right) = 1$. Если $p \equiv 3 \pmod 4$, то $p = 4k + 3$ для некоторого неотрицательного целого k. Поэтому

$$(-1)^{(p-1)/2} = (-1)^{2k+1} = -1,$$

так что $\left(\frac{-1}{p}\right) = -1$. ■

Критерий Гаусса

Следующий результат Гаусса дает другой критерий того, является ли целое число квадратичным вычетом простого числа.

Теорема 16 (критерий Гаусса). Пусть $p > 2$ – простое число и a – положительное целое число, не делящееся на p. Для каждого положительного нечетного числа $2i - 1$, меньшего p, обозначим через r_i вычет

$$r_i \equiv a(2i - 1) \pmod p, \quad 0 < r_i < p.$$

Пусть t обозначает количество четных вычетов r_i. Тогда $\left(\frac{a}{p}\right) = (-1)^t$.

Напишем программу, реализующую критерий Гаусса:

```
criterionG[a_, p_] := If[EvenQ[Length[Select[Mod[a Range[1, p – 1, 2], p], EvenQ]]], 1, –1]
```

Функция *criterionG* выдает соответствующее значение $(-1)^t$. В примере 7 было найдено 11 квадратичных вычетов {1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18} для $p = 23$. Такие же вычеты находим с помощью *criterionG*.

```
p = 23; {#, criterionG[#, p]}& /@ Range[p – 1]
{{1, 1}, {2, 1}, {3, 1}, {4, 1}, {5, –1}, {6, 1}, {7, –1}, {8, 1}, {9, 1},
{10, –1}, {11, –1}, {12, 1}, {13, 1}, {14, –1}, {15, –1}, {16, 1}, {17, –
1}, {18, 1}, {19, –1}, {20, –1}, {21, –1}, {22, –1}}
```

***Доказательство* теоремы 16.** Пусть $p = 2m + 1$. Таким образом, имеется всего m положительных нечетных чисел, меньших p. Перенумеруем вычеты r_i так, чтобы $r_1, r_2, \ldots, r_t$ были все четные вычеты и $r_{t+1}, r_{t+2}, \ldots, r_m$ – все нечетные вычеты. Пусть $b_1, b_2, \ldots, b_m$ – все положительные нечетные числа, меньшие p и упорядоченные так, что $r_i \equiv a\, b_i \pmod p$.

Рассмотрим целые числа

$$p - r_1, p - r_2, \ldots, p - r_t, r_{t+1}, \ldots, r_m. \tag{11}$$

Эти числа – нечетные положительные и меньшие p. Мы утверждаем, что все они различные. Так как все b_i различные, то в списке (11) среди первых t чисел и среди последних $m - t$ чисел нет повторений. Поэтому достаточно доказать, что не может быть равенства $p - r_i = r_j$, если $i \le t$ и $j > t$. От противного, пусть равенство выполнено, тогда имеем

$$p = r_i + r_j \equiv ab_i + ab_j = a(b_i + b_j) \pmod{p}.$$

Так как p не делит a, то $b_i + b_j$ делится на p. Но $0 < b_i + b_j < 2p$, следовательно, $b_i + b_j = p$, что приводит к противоречию: сумма двух нечетных чисел b_i и b_j нечетна.

Таким образом, числа из списка (11) попарно совпадают с числами $b_1, b_2, \ldots, b_m$, но возможно записаны в другом порядке. Поэтому

$$b_1 b_2 \cdots b_m = (p - r_1)\cdots(p - r_t)\, r_{t+1} \cdots r_m \equiv (-1)^t r_1 r_2 \cdots r_m \pmod{p}.$$

Но p не делит никакое целое число, большее 0 и меньшее p. Следовательно, мы можем разделить обе части сравнения на $b_1 b_2 \cdots b_m$. Получаем сравнение

$$1 \equiv (-1)^t a^{(p-1)/2} \pmod{p},$$

и, используя критерий Эйлера, приходим к

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

Следствие. Для простого модуля p, число 2 является квадратичным вычетом тогда и только тогда, когда $p \equiv \pm 1 \pmod 8$.

Доказательство. Если $1 \le i \le (p-1)/2$, то $r_i \equiv 2(2i - 1) \pmod{p}$ будет четным тогда и только тогда, когда $2i - 1 < p/2$ или равносильно $i < p/4 + 1/2$. Если $p = 4m + 1$, то (используя обозначения в критерии Гаусса) это означает, что $t = m$, так что 2 будет квадратичным вычетом при $p \equiv 1 \pmod 8$ и невычетом – при $p \equiv 5 \pmod 8$. Если $p = 4m + 3$, то $t = m + 1$, так что 2 будет квадратичным вычетом при $p \equiv 7 \pmod 8$ и невычетом – при $p \equiv 3 \pmod 8$. ■

Закон квадратичной взаимности

Предположим, p и q – различные нечетные простые числа. Если мы знаем о том, является ли q квадратичным вычетом по модулю p, то можем ли мы сделать из этого какое-то заключение о том, является ли p квадратичным вычетом по модулю q? Ответ на этот вопрос был известен еще Эйлеру, который получил его в результате численных экспери-

ментов, но Эйлер не смог доказать обнаруженный им результат. Позднее Лежандр, используя введенные им символы, сформулировал результат Эйлера в следующем элегантном виде:

Теорема 17 (**закон квадратичной взаимности**). Пусть p и q – различные нечетные простые числа, тогда

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

Доказательство смотрите, например, в [58]. Имеется много различных доказательств теоремы. Гаусс первым получил полное доказательство закона квадратичной взаимности в 1796 г., но он продолжал искать другие доказательства. Гаусс старался найти такое доказательство квадратичного закона взаимности, чтобы это доказательство можно было обобщить на высшие степени. В частности он интересовался кубическими и биквадратными вычетами простых чисел, т.е. когда для простого числа p и целого a, не делящегося на p, сравнения $x^3 \equiv a \pmod{p}$ и $x^4 \equiv a \pmod{p}$ разрешимы. В конце концов, Гаусс нашел такое доказательство. Последнее восьмое доказательство было получено в 1863 г.

На 2013 г. имелось 246 различных доказательств; хронологические и библиографические описания доказательств смотрите в [13, 17]. Конечно, не все они различаются по существу. Многие отличаются друг от друга лишь небольшими деталями. Одно из самых простых доказательств было предложено русским математиком Егором Ивановичем Золотарёвым (1847–1878) в 1872 году [97].

Прежде чем начать использовать закон квадратичной взаимности, обсудим некоторые следствия из него. Сначала заметим, что число $(p-1)/2$ – четно, если $p \equiv 1 \pmod 4$ и $(p-1)/2$ – нечетно, если $p \equiv 3 \pmod 4$. Следовательно, мы видим что $\frac{p-1}{2}\cdot\frac{q-1}{2}$ – четно, если $p \equiv 1 \pmod 4$ или $q \equiv 1 \pmod 4$, в то время как $\frac{p-1}{2}\cdot\frac{q-1}{2}$ – нечетно, если $p \equiv q \equiv 3 \pmod 4$. Из-за того, что числа $p \neq q$, символ Лежандра имеет значение только ± 1, и поэтому имеем

$$\left(\frac{p}{q}\right) = \begin{cases} \left(\frac{q}{p}\right), & \text{если } p \equiv 1 \pmod 4 \text{ или } q \equiv 1 \pmod 4; \\ -\left(\frac{q}{p}\right), & \text{если } p \equiv q \equiv 3 \pmod 4. \end{cases}$$

Символ Якоби

Следующие примеры 10–12 показывают, как можно вычислять символы Лежандра (в этих примерах p является простым числом).

Пример 10. Пусть $p = 269$ и $a = 3949$. Тогда имеем

$$\left(\frac{3949}{269}\right) = \left(\frac{183}{269}\right) = \left(\frac{3 \cdot 61}{269}\right) = \left(\frac{3}{269}\right)\left(\frac{61}{269}\right).$$

На первом шаге применили пункт (i) теоремы 14, на втором шаге – пункт (ii) теоремы 14. Далее каждый символ $\left(\frac{3}{269}\right)$ и $\left(\frac{61}{269}\right)$ вычисляем по отдельности. Сначала к символу $\left(\frac{3}{269}\right)$ применяем закон квадратичной взаимности, затем теорему 14(i) и теорему 15:

$$\left(\frac{3}{269}\right) = \left(\frac{269}{3}\right) = \left(\frac{2}{3}\right) = \left(\frac{-1}{3}\right) = -1.$$

К символу $\left(\frac{61}{269}\right)$ применяем также закон взаимности, потом теорему 14(i) и 14(iii):

$$\left(\frac{61}{269}\right) = \left(\frac{269}{61}\right) = \left(\frac{25}{61}\right) = \left(\frac{5 \cdot 5}{3}\right) = 1.$$

И окончательный результат $(-1) \cdot 1 = -1$ говорит о том, что 3949 является квадратичным невычетом по модулю 269.

Пример 11. Пусть $p = 3823$ и $a = 1157$. Тогда получаем

$$\left(\frac{1157}{3823}\right) = \left(\frac{13 \cdot 89}{3823}\right) = \left(\frac{13}{3823}\right)\left(\frac{89}{3823}\right),$$

используя теорему 14(ii). Вычисляем символ $\left(\frac{13}{3823}\right)$, сначала применяя закон квадратичной взаимности, а потом теорему 14(i) и тождество $\left(\frac{1}{p}\right) = 1$:

$$\left(\frac{13}{3823}\right) = \left(\frac{3823}{13}\right) = \left(\frac{1}{13}\right) = 1.$$

Вычисления символа $\left(\frac{89}{3823}\right)$ требует больше шагов:

$$\left(\frac{89}{3823}\right) = \left(\frac{3823}{89}\right) = \left(\frac{85}{89}\right) = \left(\frac{5 \cdot 17}{89}\right) = \left(\frac{5}{89}\right)\left(\frac{17}{89}\right).$$

Сначала применили закон взаимности, потом пункты (i) и (ii) теоремы 14. Далее мы применяем к каждому из двух последних символов закон квадратичной взаимности, потом пункты (i) и (iii) теоремы 14:

$$\left(\frac{5}{89}\right)\left(\frac{17}{89}\right) = \left(\frac{89}{5}\right)\left(\frac{89}{17}\right) = \left(\frac{4}{5}\right)\left(\frac{4}{17}\right) = \left(\frac{2 \cdot 2}{5}\right)\left(\frac{2 \cdot 2}{17}\right) = 1.$$

Следовательно, $\left(\frac{13}{3823}\right)\left(\frac{89}{3823}\right) = 1\cdot 1 = 1$, т. е. 1157 является квадратичным вычетом по модулю 3823.

Пример 12. Пусть $p = 1009$ и $a = -713$. Выделяем множитель $\left(\frac{-1}{p}\right)$:

$$\left(\frac{-713}{1009}\right) = \left(\frac{-1}{1009}\right)\left(\frac{713}{1009}\right) = \left(\frac{713}{1009}\right),$$

так как $1009 \equiv 1 (\mathrm{mod}\ 4)$ и теорема 15. Далее, раскладывая 713 на простые множители и применяя пункт (ii) теоремы 14, имеем

$$\left(\frac{713}{1009}\right) = \left(\frac{23\cdot 31}{1009}\right) = \left(\frac{23}{1009}\right)\left(\frac{31}{1009}\right).$$

Чтобы оценить два последних символа Лежандра, мы используем закон квадратичной взаимности. Получаем

$$\left(\frac{23}{1009}\right) = \left(\frac{1009}{23}\right), \left(\frac{31}{1009}\right) = \left(\frac{1009}{31}\right).$$

Используя теорему 14, пункт (i), имеем

$$\left(\frac{1009}{23}\right) = \left(\frac{20}{23}\right), \left(\frac{1009}{31}\right) = \left(\frac{17}{31}\right).$$

Применяя пункт (ii) и (iii) теоремы 14, получаем

$$\left(\frac{20}{23}\right) = \left(\frac{4\cdot 5}{23}\right) = \left(\frac{4}{23}\right)\left(\frac{5}{23}\right) = \left(\frac{5}{23}\right).$$

Закон квадратичной взаимности, пункт (i) теоремы 14 и следствие из критерия Гаусса дают

$$\left(\frac{5}{23}\right) = \left(\frac{23}{5}\right) = \left(\frac{3}{5}\right) = \left(\frac{5}{3}\right) = \left(\frac{2}{3}\right) = -1.$$

Поэтому $\left(\frac{23}{1009}\right) = -1$.

Подобным образом, используя закон квадратичной взаимности, теорему 14 и следствие из критерия Гаусса, находим, что

$$\left(\frac{17}{31}\right) = \left(\frac{31}{17}\right) = \left(\frac{14}{17}\right) = \left(\frac{2}{17}\right)\left(\frac{7}{17}\right) = \left(\frac{7}{17}\right) = \left(\frac{17}{7}\right) = \left(\frac{3}{7}\right) =$$

$$= -\left(\frac{7}{3}\right) = -\left(\frac{4}{3}\right) = -\left(\frac{2\cdot 2}{3}\right) = -1.$$

Следовательно, $\left(\frac{31}{1009}\right) = -1$.

Поэтому имеем $\left(\frac{-713}{1009}\right) = (-1)(-1) = 1$.

Рассмотренные примеры подсказывают алгоритм для вычисления символов Лежандра (см., например, [102, с. 35–36]). Но этот алгоритм

требует выполнение факторизации натуральных чисел, а в настоящее время не известен ни один алгоритм, который проводил бы разложение на простые множители за полиномиальное время. Но здесь на помощь приходит символ Якоби.

Пусть n – нечетное натуральное число с факторизацией $n = p_1^{t_1} p_2^{t_2} \cdots p_m^{t_m}$ и a – целое число, взаимно простое с n. Тогда **символ Якоби** $\left(\frac{a}{n}\right)$ определяется как

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1^{t_1} p_2^{t_2} \cdots p_m^{t_m}}\right) = \left(\frac{a}{p_1}\right)^{t_1} \left(\frac{a}{p_2}\right)^{t_2} \cdots \left(\frac{a}{p_m}\right)^{t_m},$$

где символы правой стороны равенства являются символами Лежандра.

Когда $(a, n) = 1$, то символ Якоби $\left(\frac{a}{n}\right) = \pm 1$, так как каждый символ Лежандра в определении равен ± 1. Если $(a, n) \neq 1$, то $\left(\frac{a}{n}\right) = 0$. Чтобы увидеть это, заметим, что $(a, n) \neq 1$ влечет существование простого числа p, которое является делителем и a и n. Поэтому в определении $\left(\frac{a}{n}\right)$ справа в равенстве в качестве множителя встречается символ Лежандра $\left(\frac{a}{p}\right) = 0$.

Пример 13. Из определения символа Якоби получаем

$$\left(\frac{2}{75}\right) = \left(\frac{2}{3 \cdot 5^2}\right) = \left(\frac{2}{3}\right)\left(\frac{2}{5}\right)^2 = (-1)(-1)^2 = -1$$

и

$$\left(\frac{13}{105}\right) = \left(\frac{13}{3 \cdot 5 \cdot 7}\right) = \left(\frac{13}{3}\right)\left(\frac{13}{5}\right)\left(\frac{13}{7}\right) = \left(\frac{1}{3}\right)\left(\frac{3}{5}\right)\left(\frac{-1}{7}\right) =$$
$$= 1 \cdot \left(\frac{5}{3}\right) \cdot (-1) = -\left(\frac{2}{3}\right) = -(-1) = 1.$$

Когда n – простое число, то символ Якоби значит то же самое, что и символ Лежандра. Однако, если n – составное число, то значение 1 для символа Якоби не гарантирует, что сравнение $x^2 \equiv a \pmod{n}$ разрешимо. Действительно, пусть $n = 15$ и $a = 8$. Вычислим символ Якоби

$$\left(\frac{8}{15}\right) = \left(\frac{8}{3 \cdot 5}\right) = \left(\frac{8}{3}\right)\left(\frac{8}{5}\right) = \left(\frac{2}{3}\right)\left(\frac{3}{5}\right) = \left(\frac{2}{3}\right)\left(\frac{5}{3}\right) = \left(\frac{2}{3}\right)\left(\frac{2}{3}\right) = 1.$$

Проверим,

```
Union[PowerMod[Range[14],2,15]]
{1, 4, 6, 9, 10}
```

Как видим, 8 не является квадратичным вычетом по модулю 15.

Символ Якоби обладает некоторыми свойствами подобными свойствам Лежандра [102, с. 36–37]:

1. Если $a \equiv b \pmod n$, то $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$.
2. $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$.
3. Если $(a, n) = 1$, то $\left(\frac{a^2 b}{n}\right) = \left(\frac{b}{n}\right)$.
4. $\left(\frac{1}{n}\right) = 1, \left(\frac{-1}{n}\right) = (-1)^{(p-1)/2}$.
5. $\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}$.

Имеется обобщение закона квадратичной взаимности. Для любых нечетных m и n справедливо равенство

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

Пример 14. Применяя данные свойства, вычислим символ Якоби для таких же $p = 1009$ и $a = -713$, как в примере 5.

$$\left(\frac{-713}{1009}\right) = \left(\frac{-1}{1009}\right)\left(\frac{713}{1009}\right) = \left(\frac{713}{1009}\right) = \left(\frac{1009}{713}\right) = \left(\frac{296}{713}\right) = \left(\frac{2^2\cdot 2\cdot 37}{713}\right) =$$
$$= \left(\frac{2}{713}\right)\left(\frac{37}{713}\right) = \left(\frac{713}{37}\right) = \left(\frac{10}{37}\right) = \left(\frac{2}{37}\right)\left(\frac{5}{37}\right) = -\left(\frac{37}{5}\right) = -\left(\frac{2}{5}\right) = 1.$$

Можно заметить, что разложение на простые множители не требуется, за исключениием определения наибольшей степени 2 у четных чисел.

Свойства символов Якоби превращают их вычисление в элементарную задачу, сопоставимую по сложности с нахождением наибольшего общего делителя. Из критерия Эйлера следует [86, с. 118], что сложность вычисления символа Лежандра не превосходит $O(\ln^3 p)$. Но вычисления символа Якоби (значение которого в случае простого m совпадает со значение символа Лежандра) можно выполнить с помощью алгоритма со сложностью $O(\ln^2 m)$.

Алгоритм 4 (вычисление символа Лежандра / Якоби) [86, с. 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. Проверим правильность значений функции *jacobiSymbol*, вычисляя 100 000 различных символов $\left(\frac{a}{m}\right)$, где составное число $m = 25! + 1 = 401\times 38681321803817920159601$ и a пробегает числа из списка $as = 1234567890987654321\times$Range[100 000]:

```
jacobiSymbol[#, m] & /@ as == JacobiSymbol[as, m]
True
```

Возможно, алгоритм вычисления функции JacobiSymbol аналогичен алгоритму 4, но сравнение работы по времени показывает преимущество встроенной функции JacobiSymbol.

```
Timing[JacobiSymbol[as, m]; Null]
{0.390625, Null}

Timing[jacobiSymbol[as, m]& /@ as; Null]
{17.8906, Null}
```

Пример 15. Покажем, что простое $p = 30\,059\,924\,764\,123$ имеет $\left(\frac{q}{p}\right) = -1$ для всех простых q с условием $2 \le q \le 181$.

```
p = 30059924764123;
PrimeQ[p]
True

JacobiSymbol[Select[Range[2, 181], PrimeQ], p]
{–1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1,
–1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1, –1,
–1, –1, –1, –1}
```

Задача 9. Для каких простых модулей p число 2 является квадратичным вычетом? Хотелось бы получить свойство более простое, чем $\left(\frac{2}{p}\right) = 1$.

Начнем с экспериментов. Найдем среди первых 100 простых чисел p те, для которых 2 является квадратичным вычетом по модулю p.

```
as = Select[Prime[Range[2, 100]], JacobiSymbol[2, #] == 1&]
{7, 17, 23, 31, 41, 47, 71, 73, 79, 89, 97, 103, 113, 127, 137, 151, 167,
191, 193, 199, 223, 233, 239, 241, 257, 263, 271, 281, 311, 313, 337,
353, 359, 367, 383, 401, 409, 431, 433, 439, 449, 457, 463, 479, 487,
503, 521}
```

Изучая полученные числа, при некотором везении можно заметить, что для всех этих чисел остаток от делении на 8 равен 1 или 7.

```
Mod[as, 8]
{7, 1, 7, 7, 1, 7, 7, 1, 7, 1, 1, 7, 1, 7, 1, 7, 7, 7, 1, 7, 7, 1, 7, 1, 1, 7, 7, 1, 7,
1, 1, 1, 7, 7, 7, 1, 1, 7, 1, 7, 1, 1, 7, 7, 7, 7, 1}
```

Является ли это условие достаточным? Проверим.

```
bs = Select[Prime[Range[2, 100]], Mod[#, 8] == 1||Mod[#, 8] == 7&];
as == bs
True
```

Теперь осуществим более серьезную проверку. Рассмотрим первый миллион нечетных простых чисел.

```
primes = Prime[Range[2, 1000001]];
Parallelize[Select[primes, JacobiSymbol[2, #] == 1&]] ==
   Parallelize[Select[primes, Mod[#, 8]==1||Mod[#, 8]==7&]]
True
```

Теперь осталось доказать утверждение: «Для того чтобы 2 было квадратичным вычетом по нечетному простому модулю p, необходимо и достаточно, чтобы $p \bmod 8 = \pm 1$».

Задача 10. При определении символа Якоби было замечено, что $\left(\frac{8}{15}\right) = 1$, но 8 не является квадратичным вычетом по модулю 15. И таких пар чисел для составного модуля бесконечно много. Найдем еще несколько пар $\{a, m\}$, для которых $\left(\frac{a}{m}\right) = 1$, но сравнение $x^2 \equiv a \pmod{m}$ не имеет решения.

Создадим 100 пар случайных натуральных чисел в диапазоне от 2 до 20 и выберем среди них подходящие. Будем использовать функцию RandomInteger языка Wolfram.

```
ams = RandomInteger[{2, 20}, {100, 2}];
Select[ams, Not[PrimeQ[#[[2]]]] && JacobiSymbol[#[[1]], #[[2]]] == 1
&& (Solve[x^2 == #[[1]], Modulus → #[[2]]] == {})&]
{{3, 16}, {13, 14}, {7, 8}, {7, 8}, {5, 9}, {2, 9}, {19, 16}, {20, 9}, {8,
15}, {17, 18}, {19, 14}, {11, 9}}
```

Кроме пары {8, 15} получили еще 11 пар чисел $\{a, m\}$.

Задача 11. Найти наименьшие квадратичные невычеты для всех нечетных простых чисел меньших 1000.

Определим функцию, которая для простого числа p ищет наименьший квадратичный невычет.

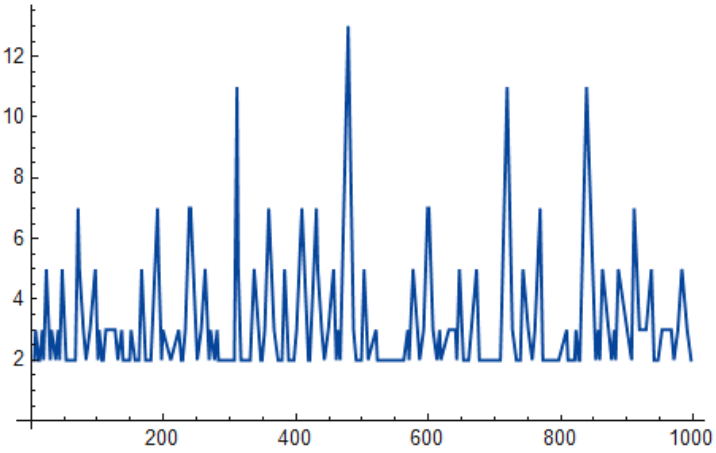
```
minNonresidue[p_] := Min[Select[Range[2, (p + 1)/2],
JacobiSymbol[#, p] == –1&]]
```

Посмотрим на ее значения для $3 \le p \le 97$.

```
{#, minNonresidue[#]}& /@ Prime[Range[2, 25]]
```
{{3, 2}, {5, 2}, {7, 3}, {11, 2}, {13, 2}, {17, 3}, {19, 2}, {23, 5}, {29, 2}, {31, 3}, {37, 2}, {41, 3}, {43, 2}, {47, 5}, {53, 2}, {59, 2}, {61, 2}, {67, 2}, {71, 7}, {73, 5}, {79, 3}, {83, 2}, {89, 3}, {97, 5}}

А для простых чисел p, $3 \le p < 1000$, получим визуальную картину.

```
ListLinePlot[{#, minNonresidue[#]}& /@ Prime[Range[2,
PrimePi[1000]]], PlotRange → All]
```



Задача 12. Рассмотрим два диапазона простых чисел. В первом диапазоне простые числа находятся между 100 000 и 1 000 000, во втором диапазоне – между 100 000 000 и 1 000 000 000. Если простые числа использовать как модули, то отличаются ли заметно данные диапазоны по величине наименьших квадратичных невычетов?

Функцию *minNonresidue* из предыдущей задачи модифицируем, чтобы сделать ее более быстрой.

```
minNonresidue1[p_] := Module[{n}, n = 2;
                      While[JacobiSymbol[n, p] ≠ –1, n = n + 1]; n]
```

Проверим, что функции minNonresidue и minNonresidue1 выдают одинаковые результаты:

```
minNonresidue /@ Prime[Range[2, PrimePi[1000]]] ==
minNonresidue1 /@ Prime[Range[2, PrimePi[1000]]]
```
True

Возьмем в каждом диапазоне 100 случайных простых чисел (будем использовать функцию RandomPrime) и вычислим, используя их как модули, наименьшие квадратичные невычеты.

```
minNonresidue1 /@ RandomPrime[{100000, 1000000}, 100]
```
{5, 5, 2, 2, 2, 5, 13, 7, 11, 2, 29, 2, 2, 2, 3, 3, 2, 13, 2, 2, 3, 2, 3, 5, 2, 2, 3, 2, 2, 2, 2, 2, 13, 3, 3, 2, 3, 3, 3, 2, 13, 2, 2, 5, 5, 2, 5, 2, 2, 19, 7, 3, 2, 2, 3, 2, 2, 7, 2, 2, 3, 5, 5, 5, 2, 13, 2, 3, 3, 2, 5, 2, 3, 5, 19, 7, 3, 5, 2, 29, 2, 7, 2, 3, 2, 2, 5, 2, 2, 2, 2, 2, 3, 3, 5, 2, 2, 2, 2, 3, 2}

```
minNonresidue1 /@ RandomPrime[{100000000, 1000000000}, 100]
```
{5, 7, 3, 5, 2, 29, 7, 11, 2, 2, 3, 3, 2, 2, 3, 2, 2, 5, 2, 2, 3, 2, 2, 5, 2, 3, 2, 3, 2, 2, 3, 2, 2, 3, 2, 5, 11, 2, 2, 2, 2, 7, 5, 5, 5, 5, 2, 17, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 5, 3, 2, 3, 5, 2, 3, 3, 11, 3, 5, 2, 13, 3, 3, 5, 17, 3, 5, 7, 3, 2, 3, 2, 7, 5, 2, 13, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 7, 3, 5, 3, 2, 2, 3}

Как видим, незаметно, чтобы увеличение модулей привело к увеличению наименьших квадратичных невычетов.

Задача 13. Найдем экспериментальные свидетельства, чтобы ответить на следующий вопрос. Для каких нечетных простых p имеется больше квадратичных вычетов a из диапазона $1 \le a \le (p-1)/2$, чем из диапазона $(p+1)/2 \le a \le p-1$?

Определим функцию moreResiduesQ[p], которая возвращает истину, если при вызове с аргументом p имеется больше квадратичных вычетов a из диапазона $1 \le a \le (p-1)/2$, чем из диапазона $(p+1)/2 \le a \le p-1$.

```
moreResiduesQ[p_] := Length[Select[Range[1, (p – 1)/2],
                       JacobiSymbol[#, p] == 1&]] >
                       Length[Select[Range[(p + 1)/2, p – 1],
                       JacobiSymbol[#, p] == 1&]]
```

Создадим список тех чисел из первой сотни нечетных простых, для которых *moreResiduesQ* выдает истину.

```
ns100 = Select[Prime[Range[2, 100]], moreResiduesQ]
```
{3, 7, 11, 19, 23, 31, 43, 47, 59, 67, 71, 79, 83, 103, 107, 127, 131, 139, 151, 163, 167, 179, 191, 199, 211, 223, 227, 239, 251, 263, 271, 283, 307, 311, 331, 347, 359, 367, 379, 383, 419, 431, 439, 443, 463, 467, 479, 487, 491, 499, 503, 523}

Каким свойством обладают все эти числа? Найдем для чисел из списка *ns*100 остаток от деления на 4.

```
Mod[ns100, 4]
{3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3}
```

Можно будет сформулировать гипотезу, если оставшиеся простые числа из списка Prime[Range[2, 100] при делении на 4 дают в остатке 1.

```
Mod[Complement[Prime[Range[2, 100]], ns100], 4]
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
```

Проведем теперь аналогичные вычисления для первых трех тысяч нечетных простых чисел. Для более быстрого вычисления используем параллельные вычисления.

```
ns3000 = Parallelize[Select[Prime[Range[2, 3000]], moreResiduesQ]];
```

Далее для проверки чисел из списков *ns*3000 и его дополнения применяем функцию AllTrue.

```
AllTrue[Mod[#, 4] == 3& /@ ns3000, #&]
True

AllTrue[Mod[#, 4] == 1& /@ Complement[Prime[Range[2, 3000]],
ns3000], #&]
True
```

Теперь имеется больше основания высказать гипотезу: нечетное простое число p имеет больше квадратичных вычетов a из диапазона $1 \le a \le (p-1)/2$, чем из диапазона $(p+1)/2 \le a \le p-1$, тогда и только тогда, когда $p \equiv 3 \pmod 4$.

Извлечение квадратного корня по модулю

Мы видели, что для нечетного простого p и a – не кратного p, разрешимость сравнения $x^2 \equiv a \pmod p$ определяется значением символа Лежандра $(\frac{a}{p})$. Если $(\frac{a}{p}) = 1$, 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 корень по модулю простого числа p. Этот алгоритм достаточный сложный и его обоснование приведено в [86, с. 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 5 (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 $d \pmod p$ *)
   Найти случайное целое $d \in [2, p-1]$, для которого $\left(\frac{a}{p}\right) = -1$;
   (* Вычислить символ Якоби, алгоритм 4 *)
   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 [86, с. 122].

Вычисление квадратного корня по простому модулю запрограммируем в Mathematica в виде функции

```
sqrtMod[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 16. Вычисление квадратного корня. Возьмем наибольшее простое число, известное на начало 1951 г. [101, с. 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 17. С помощью функции Solve в Mathematica можно извлечь и квадратный корень по модулю.

Найдем еще раз случайный квадратичный вычет a по модулю p в диапазоне $(p/2, p-1]$:

b = –1; While[$b \neq 1$, a = RandomInteger[{$\lceil p/2 \rceil$, 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}

§ 7. Криптография с открытым ключом

Математики – подобно французам:
независимо от того, что Вы говорите им,
они переводят это на их собственный язык,
и немедленно это означает что-то совершенно другое.

И.В. Гете

Как вы можете сделать сообщение тайным, чтобы только предполагаемый получатель сообщения смог его прочесть? Эта проблема интересовала людей с древних времен, особенно в дипломатии, военном деле и торговле. Это стало еще более важно в современном мире с появлением электронной почты и Интернета. Криптография – наука и искусство прятать содержание конфиденциального сообщения.

Некоторые термины. Прежде чем обсуждать конкретные системы секретности, мы представляем соответствующую основную терминологию. Дисциплина, посвященная системам секретности, называется **криптологией**. **Криптография** – это часть криптологии, которая занимается разработкой и внедрением систем секретности, в то время как **криптоанализ** нацелен на «взлом» (поражение) этих систем. Сообщение, которое должно быть представлено в секретной форме, называется **открытым текстом**. **Шифрование** (метод шифрования) – процедуры для изменения открытого текста в **шифрованный текст** путем изменения символов открытого текста с использованием преобразования. **Ключ** определяет конкретное преобразование из множества возможных преобразований. Обратный процесс изменения шифрованного текста в открытый текст получателем, обладающим методом, как это сделать, называется **дешифрованием** или **расшифрованием**. Дешифрование, конечно, отличается от процесса криптоанализа, с помощью которого какая-то третья сторона пытается прочесть сообщение.

Формально говоря, криптосистема есть система, которая состоит из конечного множества P возможных открытых текстов, конечного множества C возможных шифрованных текстов, множества K возможных ключей, и для каждого ключа $k \in K$ имеются шифрующая функция E_k: $P \to C$ и дешифрующая функция D_k: $C \to X$. Для этих функций выполняется свойство $D_k(E_k(x)) = x$, где x – любой открытый текст.

За два последних столетия были развиты классические методы для обеспечения секретности, базирующие на модулярной арифметике.

Во всех этих классических системах два человека, которые хотят общаться в частном порядке, должны иметь общий секретный ключ.

С 1970-х годов было введено и развито понятие криптографии с открытым ключом. В криптографии с открытым ключом двум людям, которые хотят общаться, нет необходимости иметь общий ключ. Вместо этого каждый человек имеет свои два ключа: закрытый и открытый. Закрытый ключ знает только он один, а открытый ключ является публичным, этот ключ знают все. Используя такую криптографическую систему (систему с открытым ключом), человек X посылает своему корреспонденту Y сообщение, зашифрованное с помощью открытого ключа человека Y. Прочесть это сообщение может только Y, используя свой закрытый ключ.

Алгоритм *RSA*. Мы рассмотрим систему *RSA* – наиболее используемую систему с открытым ключом, чья безопасность основана на трудностях факторизации целых чисел. Название системы есть аббревиатура имён ее создателей (Ronald Riverst, Adi Shamir и Leonard Adleman [21]). *RSA* является криптосистемой с открытым ключом, связанная с возведением в степень по некоторому модулю. Ключ представлен парой натуральных чисел (e, n), где e – показатель степени, n – модуль, являющийся произведением двух больших простых чисел $n = pq$ и $(e, \varphi(n)) = 1$. Чтобы зашифровать сообщение, мы сначала переводим письмо в последовательность цифр и потом формируем блоки открытого текста возможно большего размера. Каждый блок будем рассматривать как отдельное натуральное число P. К числу P применяем преобразование шифрования $E(P)$, получая шифрованный блок – число C, где

$$E(P) = C \equiv P^e (\mathrm{mod}\ n),\ \ 0 \le C < n.$$

Процедура дешифрования требует знания обратноего к e числа d по модулю $\varphi(n)$, которое существует, потому что $(e, \varphi(n)) = 1$. Процедура дешифрования числа C вычисляет

$$D(C) \equiv C^d (\mathrm{mod}\ n),\ \ 0 \le D(C) < n.$$

Чтобы увидеть, что $D(C) = (P^e)^d \equiv P (\mathrm{mod}\ n)$ для всех возможных открытых текстов P, заметим, что

$$D(C) = C^d \equiv (P^e)^d = P^{ed} \equiv P^{k\varphi(n)+1} \equiv P^{k\varphi(n)} P\ (\mathrm{mod}\ n),$$

где $ed = k\ \varphi(n)+1$ для некоторого целого числа k, потому что $ed = 1\ (\mathrm{mod}\ \varphi(n))$. Когда $(P, n) = 1$, то по теореме Эйлера мы знаем, что $P^{\varphi(n)} \equiv 1 (\mathrm{mod}\ n)$. Вследствие этого,

$$P^{k\varphi(n)} P\ (\mathrm{mod}\ n) \equiv (P^{\varphi(n)})^k P \equiv P (\mathrm{mod}\ n).$$

Следовательно,

$$D(C) \equiv P(\mathrm{mod}\ n).$$

Далее рассмотрим случай, когда $(P, n) > 1$. Чтобы показать, что и в этом случае преобразование дешифровки восстанавливает открытый текст сообщения, нам сначала необходимо рассматривать отдельно сравнения по модулю p и по модулю q и затем применить китайскую теорему об остатках. (Наши рассуждения будут также применимы и для $(P, n) = 1$, хотя и будут сложнее, чем предыдущее рассуждение.) Итак, предположим, что P не сравнимо с 0 по модулю p. Тогда мы имеем $D(C) \equiv P^{k\varphi(n)}P \equiv P^{(p-1)(q-1)k}P \equiv (P^{(p-1)})^{(q-1)k}P \equiv P(\mathrm{mod}\ p)$. На последнем шаге мы используем сравнение $P^{(p-1)} \equiv 1(\mathrm{mod}\ p)$ из малой теоремы Ферма. Если же $P \equiv 0(\mathrm{mod}\ p)$, то $C = P^e \equiv 0(\mathrm{mod}\ p)$, так что $D(C) \equiv P(\mathrm{mod}\ p)$ и в этом случае. Подобное рассуждение имеет место и для простого q, поэтому $D(C) \equiv P(\mathrm{mod}\ q)$. Применяя теперь китайскую теорему об остатках, получаем, что два сравнения по модулю p и q влекут сравнение $D(C) \equiv P(\mathrm{mod}\ n)$ для всех P, включая и те P, для которых $(P, n) > 1$.

Мы увидели, что для криптосистемы *RSA* пара (d, n) является ключом дешифрования, соответствующим ключу шифрования (e, n), d – обратное число для e по модулю n.

Программа. Опишем реализацию криптосистемы *RSA* на языке Wolfram. Начнем с рассмотрения того, как находить большие простые числа. Будем искать простые числа среди нечетных чисел, имеющих от 250 до 300 цифр. По теореме о распределении простых чисел вероятность того, что нечетное число с 300 цифрами будет простое, приблизительно равна $2/\ln(10^{300})$. Следовательно, мы ожидаем найти простое число после проверки в среднем $1/(2/\ln(10^{300})) \approx 345$ чисел. Для тестирования на простоту случайно выбранного нечетного числа можно использовать вероятностный тест Рабина – Миллера (см. главу 9). По-видимому, подобный подход используется в языке Wolfram для реализации встроенной функции RandomPrime[{a, b}], которая выдает вероятное простое число из диапазона {a, b}.

Следующие вычисления создают открытый и секретный ключ. Вначале находим два больших простых числа (второй аргумент функции RandomPrime задает количество генерируемых простых чисел):

```
{p, q} = RandomPrime[{10^250, 10^300}, 2];
N[Abs[p – q]]
9.11898 × 10^299
n = p q;
```

Полученные числа p и q не являются близкими, иначе можно было взломать шифр, факторизуя n методом Ферма [86 , с. 255–256].

```
φ = (p – 1)(q – 1);
e = RandomPrime[{φ + 1, n}];
```

Так как e – простое число, большее $p - 1$ и $q - 1$, то e и φ получаются взаимно простыми.

```
GCD[e, φ]
1
```

Обратное к e по модулю φ число d определяется с помощью модулярного возведения в степень.

```
d = PowerMod[e, –1, φ];
```

Проверим:

```
Mod[e d , φ]
1
```

Используемые числа являются достаточно большими, поэтому их точные значения не печатаются, но мы можем посмотреть, сколько десятичных цифр имеют числа p, q, n, φ, e, d соответственно:

```
IntegerLength[{p, q, n, φ, e, d}]
{300, 299, 599, 599, 599, 599}
```

Функция Short[a] выдает начальные и последние цифры числа a, указывая в двойных угловых скобках количество пропущенных цифр:

```
Short /@{p, q, n, φ, e, d}
{9 901 276 601 963 777 887 405 901 <<250>> 7 873 932 218 029 904 441 280 841,
 7 822 947 486 354 434 596 492 319 <<249>> 9 334 208 994 541 632 563 335 593,
 7 745 716 690 503 251 386 452 026 <<549>> 9 430 105 288 634 788 244 273 713,
 7 745 716 690 503 251 386 452 026 <<549>> 2 221 964 076 063 251 239 657 280,
 7 745 716 690 503 251 386 452 026 <<549>> 4 037 045 060 213 834 252 413 747,
 6 993 232 015 208 652 800 326 908 <<549>> 6 359 317 491 162 914 947 174 843}
```

В языке Wolfram открытый текст может содержать любой символ из набора Unicode, содержащего $2^{16} = 65536$ значений. Для кодирования строк символов в последовательность Unicode-кодов и обратного преобразования используются функции языка Wolfram:

```
s = ToCharacterCode["ЯЁяёYZyz αβ, +!"]
{1071, 1025, 1103, 1105, 89, 90, 121, 122, 32, 945, 946, 44, 32, 43, 33}
```

FromCharacterCode[*s*]
ЯЁяёYZyz αβ, +!

В качестве открытого текста возьмем для демонстрации криптосистемы достаточно большой текст на русском языке (R. Kipling “If –”, перевод С. Маршака):

***openText* =**

"О, если ты покоен, не растерян,
Когда теряют головы вокруг,
И если ты себе остался верен,
Когда в тебя не верит лучший друг,
И если ждать умеешь без волненья,
Не станешь ложью отвечать на ложь,
Не будешь злобен, став для всех мишенью,
Но и святым себя не назовешь,

И если ты своей владеешь страстью,
А не тобою властвует она,
И будешь тверд в удаче и в несчастье,
Которым, в сущности, цена одна,
И если ты готов к тому, что слово
Твое в ловушку превращает плут,
И, потерпев крушенье, можешь снова –
Без прежних сил – возобновить свой труд,

И если ты способен все, что стало
Тебе привычным, выложить на стол,
Все проиграть и вновь начать сначала,
Не пожалев того, что приобрел,
И если можешь сердце, нервы, жилы
Так завести, чтобы вперед нестись,
Когда с годами изменяют силы
И только воля говорит: Держись! –

И если можешь быть в толпе собою,
При короле с народом связь хранить
И, уважая мнение любое,
Главы перед молвою не клонить,
И если будешь мерить расстоянье
Секундами, пускаясь в дальний бег, –
Земля – твое, мой мальчик, достоянье!
И более того, ты – человек!";

Текст содержит 1411 символов.

```
StringLength[openText]
1411
```

Для шифрования используется функция *encrypt* с открытыми ключами *e* и *n*.

```
encript[openText_String, e_, n_] :=
   Module[{tcc, f, id}, tcc = ToCharacterCode[openText];
                        f  = Fold[#1 2^16 + #2&, 0, tcc];
                        id = IntegerDigits[f, n]; PowerMod[id, e, n] ];
```

Функция ToCharacterCode[*openText*] преобразует открытый текст в список *tcc* Unicode-кодов символов текста. Каждый элемент списка *tcc* – натуральное число в пределах от 0 до 2^{16}. Функция Fold преобразует список в большое число *f* в системе счисления по основанию 2^{16}. Например,

```
Fold[#1 2^16 + #2&, 0, {2, 7, 13}]
8590393357
```

Число 8590393357 равно $13 + 7\times2^{16} + 2 \times (2^{16})^2$.

Список *id* состоит из чисел представляющих «цифры» числа *f* в системе счисления по основанию *n*. Таким образом, элементы *id* являются блоками открытого текста (в цифровом виде), о которых шла речь в описании алгоритма *RSA*. И наконец, функция PowerMod, используя открытый ключ *e*, выдает список чисел – зашифрованных блоков открытого текста.

Зашифруем текст стихотворения «Если»:

```
cipherText = encript[openText, e, n];
```

Проследим работу функции *encrypt* в нашем случае.

```
tcc = ToCharacterCode[openText];
Length[tcc]
1411
f = Fold[#1 2^16 + #2&, 0, tcc];
IntegerLength[f]
6795
```

Список кодов *tcc*, содержащий 1411 чисел, символов открытого текста с помощью функции Fold «сворачивается» в одно число *f* с 6795 цифрами.

```
id = IntegerDigits[f, n];
Length[id]
12

IntegerLength/@id
{207, 599, 599, 599, 599, 599, 599, 599, 599, 599, 599, 599}
```

Число *f* преобразуется в список *id* блоков – чисел, не превосходящих *n*. Всего получается 12 блоков. Первый блок состоит из 207 цифр, а остальные блоки – из 599 цифр. На этом этапе любой человек, знающий *n*, может восстановить из списка *id* исходный текст. Но вызов PowerMod[*id*, *e*, *n*] последней внутренней в *encript* функции выдает зашифрованный результат *cipherText*, для расшифровки которого нужен секретный ключ *d*.

Опишем теперь функцию дешифрования *decript*:

```
decript[cipherText_List, d_, n_] :=
  Module[{pm, f, id}, pm = PowerMod[cipherText, d, n];
                      f = Fold[#1 n + #2&, 0, pm];
                      id = IntegerDigits[f, 2^16]; FromCharacterCode[id]];
```

На вход функции подается список натуральных чисел – блоков шифрованного сообщения. С помощью секретного ключа *d* функция PowerMod выдает список *pm* натуральных чисел. На этом этапе из списка *pm* любой человек, знающий открытый ключ *n*, уже может получить исходный текст. Это делается с помощью оставшихся шагов в *decript*. Функция Fold, рассматривая каждое число списка *pm* как «цифру» в системе счисления по основанию *n*, создает большое число *f*. Рассматривая *f* как число в системе счисления по основанию 2^{16}, получаем список кодов *id* открытого сообщения. Теперь достаточно воспользоваться функцией FromCharacterCode, которая из списка кодов символов получает строку, представляющую открытый текст. Мы можем убедиться в этом:

```
decript[cipherText, d, n] == openText
True
```

Проследим выполнение *decript*[*cipherText*, *d*, *n*]. Двенадцать блоков зашифрованного текста преобразуется в список *pm* блоков открытого текста.

```
pm = PowerMod[cipherText, d, n];
```

Мы можем проверить их количество и длину:

```
Length[pm]
12

IntegerLength/@pm
{207, 599, 599, 599, 599, 599, 599, 599, 599, 599, 599, 599}
```

Получаемое число *f* содержит 6795 цифр:

```
f = Fold[#1 n + #2&, 0, pm];
IntegerLength[f]
6795
```

Список кодов *id* кодирует 1411 символов открытого сообщения:

```
id = IntegerDigits[f, 2^16];
Length[id]
1411
```

Дополнительные сведения математического характера об алгоритме *RSA* и его стойкости см. в [86, с. 438–440; 102, с. 87–98].

Глава 4

МУЛЬТИПЛИКАТИВНЫЕ ФУНКЦИИ

Важную роль в изучении делимости целых чисел и распределении простых чисел играют мультипликативные функции.

§ 1. Первые понятия

Рассмотрим общие свойства мультипликативных функций, операции с ними, и познакомимся с важными примерами.

Будем рассматривать **арифметические** функции, под которыми понимаются функции с множеством натуральных чисел в качестве области определения.

Арифметическая (в общем случае комплекснозначная) функция f называется **мультипликативной**, если f – не тождественный ноль и $f(nm) = f(n)\,f(m)$ для любых взаимно простых m и n. Функция называется **вполне мультипликативной**, если равенство $f(nm) = f(n)\,f(m)$ имеет место для произвольных натуральных чисел.

Похожее определение имеют аддитивные функции. Арифметическая функция f, удовлетворяющая условию $f(nm) = f(n) + f(m)$ для всех взаимно простых натуральных чисел m и n, называется **аддитивной** и, если данное выше равенство справедливо для всех натуральных чисел, то f называется **вполне аддитивной** функцией.

Пример 1. Пусть $f_a(n) = n^a$, где a – фиксированное вещественное или комплексное число. Эта функция для любого a – вполне мультипликативна и, следовательно, просто мультипликативна. В частности, единичная функция $u = f_0$ является вполне мультипликативной.

Пример 2. Функция Эйлера $\varphi(x)$, как доказано в теореме 7 из главы 3, является мультипликативной, но она не вполне мультипликативна, так как имеем $\varphi(9) = 6$, а $\varphi(3)\varphi(3) = 4$.

Пример 3. Произведением fg двух арифметических функций f и g называется арифметическая функция, определяемая равенством

$(fg)(n) = f(n)g(n)$. Подобным образом определяется частное двух функций $(f/g)(n) = f(n)/g(n)$, если $g(n) \neq 0$ для всех n. Если f и g – мультипликативные функции, то таковыми являются и функции fg и f/g. Если f и g – вполне мультипликативные функции, то такими же будут функции fg и f/g.

Очевидны следующие два утверждения:

1. Если $f(n)$ – аддитивная (вполне аддитивная функция), то $g(n) = c^{f(n)}$ для любого числа c является мультипликативной (вполне мультипликативной) функцией.

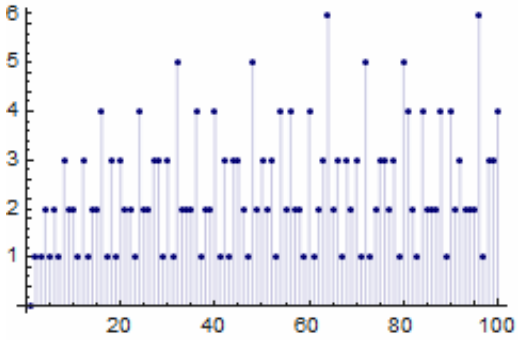
2. Если $f(n)$ – мультипликативная (вполне мультипликативная) функция, то $g(n) = \log_b f(n)$ для любого основания b является аддитивной (вполне аддитивной) функцией.

Пример 4. Пусть функция $\Omega(n)$ вычисляет число различных простых делителей n с учетом их кратности. В частности, $\Omega(2^3 \times 2^2) = \Omega(2^3) + \Omega(2^2)$. Эта функция является вполне аддитивной. В языке Wolfram значение $\Omega(n)$ вычисляется функцией PrimeOmega[n], которая используется как в символических, так и в численных преобразованиях. Имеем

```
PrimeOmega[5^2×3^3×15^4]
13
PrimeOmega[5^2] + PrimeOmega[3^3] + PrimeOmega[15^4]
13
{PrimeOmega[2], PrimeOmega[2^2], PrimeOmega[2^2×3]}
{1, 2, 3}
```

График наглядно показывает поведение функции PrimeOmega[n] на натуральных n.

```
DiscretePlot[PrimeOmega[n], {n, 100}]
```



Пример 5. Пусть функция $\nu(n)$ вычисляет число различных простых делителей n. Имеем

$$\nu(2^3\times3^2) = \nu(2^3) + \nu(3^2) \text{ и } \nu(2^3\times2^2) \neq \nu(2^3) + \nu(2^2).$$

Эта функция является аддитивной и в языке Wolfram описана как PrimeNu. Имеем

```
{PrimeNu[2], PrimeNu[2×3^5], PrimeNu[2×3×5]}
{1, 2, 3}
```

Докажем некоторые свойства мультипликативных функций.

Теорема 1. Пусть f – мультипликативная функция. Тогда $f(1) = 1$.

Доказательство. Пусть для некоторого n значение $f(n) \neq 0$. Тогда $f(n) = f(1\times n) = f(1)f(n)$, поэтому, сокращая на $f(n)$, получаем $f(1) = 1$. ■

Теорема 2. Если для арифметической функции f выполнено $f(1) \neq 0$, то имеем:

(a) функция f мультипликативна тогда и только тогда, когда

$$f(p_1^{a_1}\cdots p_k^{a_k}) = f(p_1^{a_1})\cdots f(p_k^{a_k}),$$

для всех простых p_i и всех целых $a_i \geq 1$;

(b) если f – мультипликативная функция, то f вполне мультипликативна тогда и только тогда, когда $f(p^a) = (f(p))^a$ для всех простых p и целых $a \geq 1$.

Доказательство легко следует из определений.

В теории чисел часто используется следующее сигма-обозначение

$$\sum_{d|n} f(d),$$

где f – арифметическая функция, а суммирование осуществляется по всем положительным делителям числа n. Функция

$$F(n) = \sum_{d|n} f(d)$$

называется сумматорной функцией для f. Рассмотрим значения сумматорной функции для функции Эйлера $\varphi(x)$ (функция Divisors[n] выдает список всех положительных делителей числа n):

```
Sum[EulerPhi[d], {d, Divisors[#]}]& /@ Range[20]
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
```

Вычисления подсказывают справедливость следующей теоремы:

Теорема 3. Пусть n – натуральное число. Тогда имеем

$$n = \sum_{d|n} \varphi(d).$$

Доказательство. Введем на множестве $C = \{1, 2, \ldots, n\}$ отношение эквивалентности $a\rho b$ тогда и только тогда, когда $(a, n) = (b, n)$. Получаем разбиение множества C на множества $C_d = \{m \mid (m, n) = d\}$. Очевидно, если $m \in C_d$, то $(m/d, n/d) = 1$, и наоборот. Следовательно, количество целых чисел в C_d равно количеству чисел, не превышающих m/d и взаимно простых с m/d. Поэтому во множестве C_d содержится $\varphi(n/d)$ чисел. Так как любое число из C принадлежит только одному множеству C_d, то имеем

$$n = \sum_{d|n} \varphi(n/\mathrm{d}).$$

Когда d пробегает все положительные делители n, то n/d также пробегает эти делители. Следовательно,

$$n = \sum_{d|n} \varphi(n/d) = \sum_{d|n} \varphi(d),$$

что доказывает теорему. ■

Очень простое доказательство этой теоремы с помощью Mathematica получим в примере 9(9).

Лемма 1. Для натуральных чисел имеем, если $r|mn$, где $(m, n) = 1$, то r можно представить в виде произведения cd, где $c|m$, $d|n$ и $(c, d) = 1$.

Доказательство. Рассмотрим факторизацию mn и положим $c = (r, m)$ и $d = (r, n)$. Любой простой делитель r является делителем c или d, но не одновременно. Точно так же любой простой делитель c или d (а общих делителей у c и d нет) является делителем r. Поэтому факторизации чисел r и cd совпадают, следовательно, $r = cd$. ■

Теорема 4. Если f – мультипликативная функция, то сумматорная функция

$$F(n) = \sum_{d|n} f(d)$$

является также мультипликативной.

Доказательство. Если $(m, n) = 1$, то мы должны доказать, что $F(mn) = F(m)F(n)$ или

$$\sum_{r|mn} f(r) = (\sum_{c|m} f(c))(\sum_{d|n} f(d)).$$

Но это следует непосредственно после раскрытия скобок в правой части равенства при учете леммы 1. Если $c_i|m$ и $d_j|n$, то правая часть есть

$$(f(c_1)+f(c_2)+f(c_3)+\cdots)(f(d_1)+f(d_2)+f(d_3)+\cdots)=$$
$$=\sum_{i,j} f(c_i)f(d_j)=\sum_{i,j} f(c_i d_j)=\sum_{d|mn} f(d).$$

Что и требовалось доказать. ■

§ 2. Число и сумма делителей

Рассмотрим функции, которые вычисляют число и сумму делителей натурального числа. Эти полезные функции являются мультипликативными.

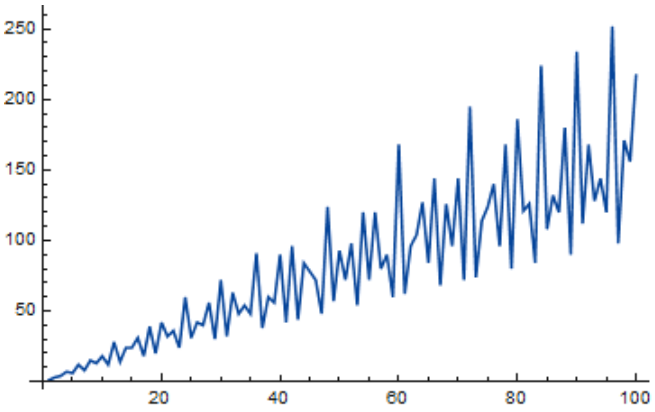
Функция **сумма делителей**, обозначаемая σ, определяется следующим образом: σ(n) равно сумме всех положительных делителей натурального числа n.

Мы можем, используя Mathematica, вычислить суммы делителей для первых 20 натуральных чисел:

```
σ[n_] := Plus @@ Divisors[n]
σ /@ Range[20]
```
{1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28, 14, 24, 24, 31, 18, 39, 20, 42}

График функции σ(n) показывает поведение функции более наглядно.

```
ListPlot[σ /@ Range[100], Joined → True]
```



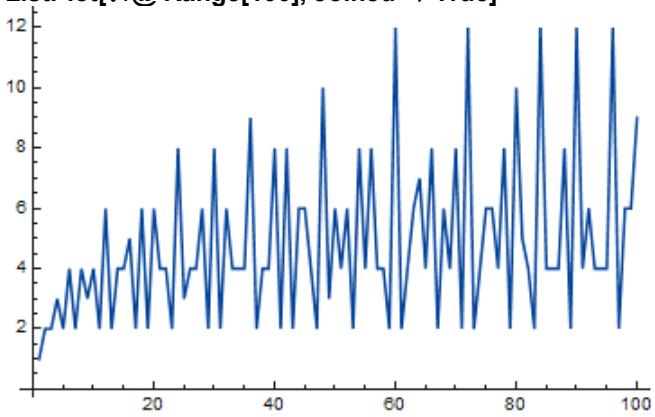
Функция **число делителей**, обозначаемая τ, определяется следующим образом: $\tau(n)$ равно числу всех положительных делителей натурального числа n.

Вычислим количества делителей для первых 20 натуральных чисел.

```
τ[n_] := Plus @@ (1& /@ Divisors[n])
τ /@ Range[20]
{1, 2, 2, 3, 2, 4, 2, 4, 3, 4, 2, 6, 2, 4, 4, 5, 2, 6, 2, 6}
```

График функции $\tau(n)$:

```
ListPlot[τ /@ Range[100], Joined → True]
```



Теорема 5. Функции сумма делителей $\sigma(n)$ и число делителей $\tau(n)$ являются мультипликативными.

Доказательство. Очевидно, функции $f_1(n) = n$ и $u(n) = 1$ мультипликативны. По теореме 4 мы видим, что функции

$$\sigma(n) = \sum_{d|n} f_1(d) \quad \text{и} \quad \tau(n) = \sum_{d|n} u(d)$$

также мультипликативны. ■

Так как мы знаем, что функции $\sigma(n)$ и $\tau(n)$ мультипликативны, то мы можем вывести формулы для их значений, используя факторизацию n на простые числа. Сначала найдем формулу для случая, когда n есть степень простого числа.

Лемма 2. Пусть p – простое и a – натуральное число. Тогда

$$\sigma(p^a) = 1 + p + p^2 + \cdots + p^a = \frac{p^{a+1} - 1}{p - 1} \quad \text{и} \quad \tau(p^a) = a + 1.$$

Доказательство. Делителями p^a являются 1, p, p^2,…, p^a. Следовательно, p^a имеет $a + 1$ делитель, так что $\tau(p^a) = a + 1$. Формулу для $\sigma(p^a)$ можно получить прямо с помощью *Mathematica*:

$$\sum_{i=0}^{a} p^i$$

$$\frac{-1 + p^{1+a}}{-1 + p}$$ ■

Теорема 6. Пусть натуральное число n имеет факторизацию $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$. Тогда

$$\sigma(n) = \prod_{i=1}^{k} \frac{p_i^{a_i+1} - 1}{p_i - 1} \quad \text{и} \quad \tau(n) = \prod_{i=1}^{k} (a_i + 1).$$

Доказательство. Так как обе функции $\sigma(n)$ и $\tau(n)$ мультипликативны, имеем равенства

$$\sigma(n) = \sigma(p_1^{a_1})\sigma(p_2^{a_2}) \cdots \sigma(p_k^{a_k})$$

и

$$\tau(n) = \tau(p_1^{a_1})\tau(p_2^{a_2}) \cdots \tau(p_k^{a_k}).$$

Вставив величины для $\sigma(p_i^{a_i})$ и $\tau(p_i^{a_i})$, найденные в лемме 2, получаем желаемые формулы. ■

В языке Wolfram сумму делителей и число делителей числа n можно вычислить с помощью функции DivisorSigma, вызывая с аргументами соответственно DivisorSigma[1, n] и DivisorSigma[0, n]. О других значениях первого аргумента в DivisorSigma будет сказано несколько ниже. В частности, значения $\tau(30!)$ и $\sigma(30!)$ вычисляются следующим образом:

```
DivisorSigma[0, 30!]
2 332 800
DivisorSigma[1, 30!]
1 661 684 017 916 976 317 002 970 027 596 800
```

Заметим, что если целое $k > 1$, то уравнение $\sigma(n) = k$ может иметь только конечное множество решений n. Действительно, если $n > k$, то $\sigma(n) \geq n > k$. Следовательно, решение должно быть натуральным n, меньшим или равным k. То есть число решений может быть только конечным. Но оказывается, решений для некоторых k может не быть совсем.

Теорема 7 [23, с. 176]. Существует бесконечно много натуральных чисел, которые не являются значениями функции $\sigma(x)$ для натуральных x.

Доказательство. Пусть $n > 9$ – натуральное число и натуральное число k, такое, что

$$n/3 - 1 < k \le n/2. \quad (1)$$

Количество чисел k, удовлетворяющих (1), очевидно, больше $n/2 - n/3 = n/6$. В силу (1) выполнено

$$2k \le n \text{ и } 3k + 3 > n, \quad (2)$$

и, так как $n > 9$, то $3k > 6$ и, окончательно, $k \ge 3$. Число $2k$ имеет, по крайней мере, четыре различных делителя: 1, 2, k, $2k$. Поэтому $\sigma(2k) \ge 1 + 2 + k + 2k$, и, в силу (2), имеем $\sigma(2k) > n$.

Так как количество различных натуральных k, для которых выполнено (1), (2) и, следовательно, неравенство $\sigma(2k) > n$, больше $n/6$, то среди чисел $\sigma(1)$, $\sigma(2)$, …, $\sigma(n)$ имеется более чем $n/6$ чисел, больших чем n. Следовательно, в последовательности 1, 2, …, n имеются натуральные числа $y_1, y_2, \ldots, y_m$, $m \ge n/6$, которые не могут быть значениями функции $\sigma(x)$ для $x \le n$. Но числа $y_1, y_2, \ldots, y_m$ не могут быть и значениями $\sigma(x)$ для $x > n$, так как все $y_i \le n$, а для $x > n$ имеем $\sigma(x) \ge 1 + x > n$.

Поэтому для любого натурального числа $n > 9$ имеется в количестве более чем $n/6$ таких натуральных чисел в последовательности 1, 2, …, n, которые не могут быть значениями функции $\sigma(x)$ для натуральных x. Что и требовалось доказать. ■

Задача 1. Найти числа $x \le 500000$, которые не могут быть значениями функции σ.

Очевидно, множество S, состоящее из всех таких чисел, есть разность множеств $S = \{x \mid 1 \le x \le 500000\} \setminus \{\sigma(x) \mid 1 \le x \le 500000\}$. Сначала вычислим список $\{\sigma(x) \mid 1 \le x \le 500000\}$, оставляя в нем все значения в единственном экземпляре:

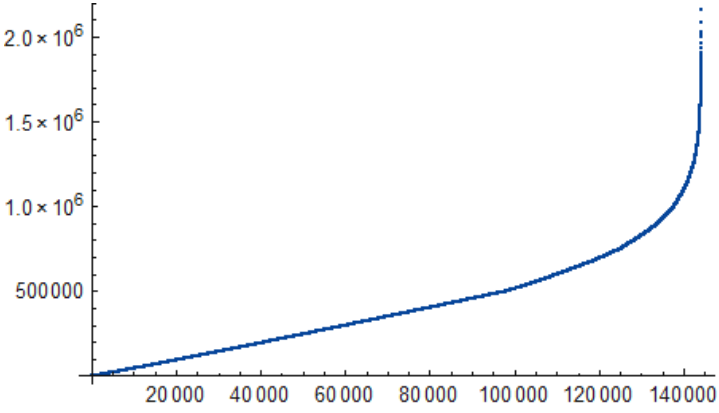
```
ys = Union[Parallelize[DivisorSigma[1, #]& /@ Range[500000]]];
```

Функция Union не только удаляет повторяющие элементы из списка, но и упорядочивает результат по возрастанию. Определим количество элементов и наибольшее значение в *ys*:

```
{Length[ys], Max[ys]}
{144145, 2160576}
```

Посмотрим диаграмму разброса данных из *ys* (опция PlotRange → All гарантирует, что все значения списка будут показаны):

```
ListPlot[ys, PlotRange → All]
```



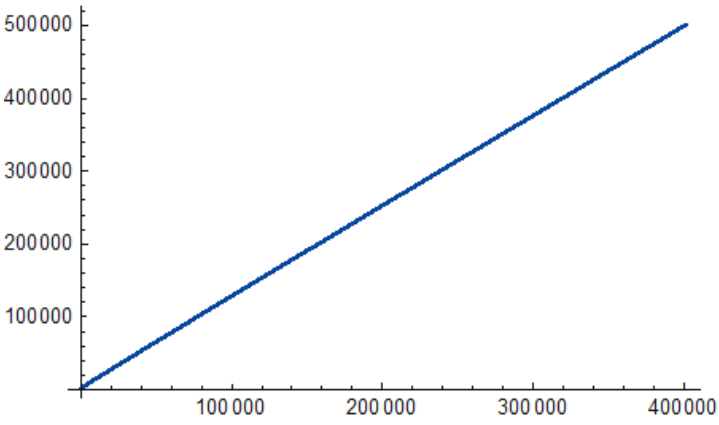
Теперь получим множество S:

```
s = Complement[Range[500000], ys];
{Length[s], Max[s]}
```

{401924, 499999}

Мы можем увидеть распределение чисел, которые не являются значениями функции σ:

```
ListPlot[s]
```



Определим линейную функцию, которая аппроксимирует распределение элементов S. Для этого список s преобразуем в список пар вида

<порядковый номер, соответствующий элемент списка *s*>:

```
data = MapIndexed[{First[#2], #1}&, s];
```

Для нахождения аппроксимирующей функции используем встроенную функцию Fit:

```
Fit[data, x, x]
1.24923x
```

Как мы видим, числа, которые не являются значениями функции σ, распределяются «равномерно» в натуральном ряду и количество их значительно. Учитывая найденную аппроксимирующую функцию, обнаруживаем, что когда n возрастает до 500 000, количество натуральных чисел, которые не могут быть значениями функции σ, приближенно равно $0.8n$ ($1/1.24923 = 0.8$), что значительно больше используемого отношения $n/6$ в доказательстве теоремы 7.

Функции σ_k, где k – в общем случае комплексное число, является обобщением функций σ и τ:

$$\sigma_k(n) = \sum_{d|n} d^k.$$

Так как все функции вида $d \to d^k$ – мутипликативны, то по теореме 4 все функции σ_k – также мультипликативны. Имеем $\sigma = \sigma_1$ и $\tau = \sigma_0$. Для того чтобы найти формулы для вычисления σ_k достаточно определить формулу для степени простого числа $\sigma_k(p^a)$. Получить сумму членов геометрической последовательности можно на языке Wolfram:

```
Sum[(p^i)^k, {i, 0, a}]
(-1 + (p^(1 + a))^k)/(-1 + p^k)
```

Следовательно, если n имеет факторизацию $n = p_1^{a_1} p_2^{a_2} \cdots p_r^{a_r}$, то получаем

$$\sigma_k(n) = \prod_{i=0}^{r} \frac{-1 + (p^{1+a_i})^k}{-1 + p^k}.$$

В языке Wolfram функция $\sigma_k(n)$ вычисляется с помощью функции DivisorSigma[k, n]; функцию можно вызывать с символьным показателем степени:

```
DivisorSigma[k, 36]
```

$$\frac{\left(-1+2^{3k}\right)\left(-1+3^{3k}\right)}{\left(-1+2^{k}\right)\left(-1+3^{k}\right)}$$

Рассмотрим поведение функции $\tau(n)$ «в среднем», т.е. как растет функция

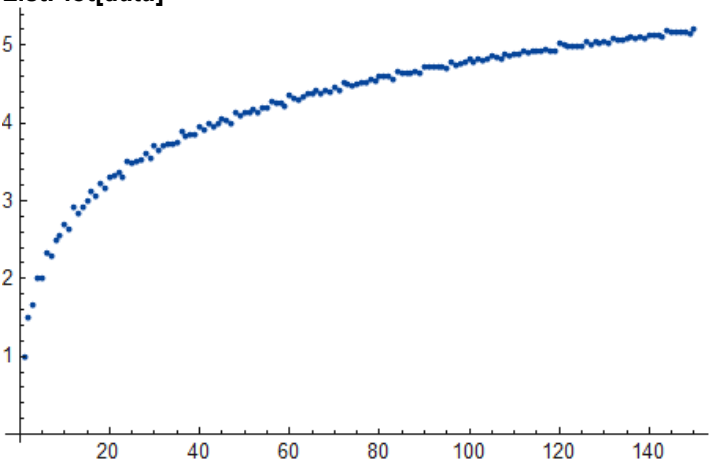
$$average(n) = (\sum_{k=1}^{n} \tau(k)) / n,$$

когда n стремится к бесконечности. Чтобы эту функцию можно было быстро вычислять для больших n, применим рекурсию с запоминанием предшествующих значений. Рекурсивной будет вспомогательная функция *sumTau*.

```
sumTau[1] = 1;
sumTau[n_] := sumTau[n] = sumTau[n – 1] + DivisorSigma[0, n]
average[n_] := sumTau[n]/n
```

Изобразим график функции *average*:

```
data = {#, average[#]}& /@ Range[150];
ListPlot[data]
```

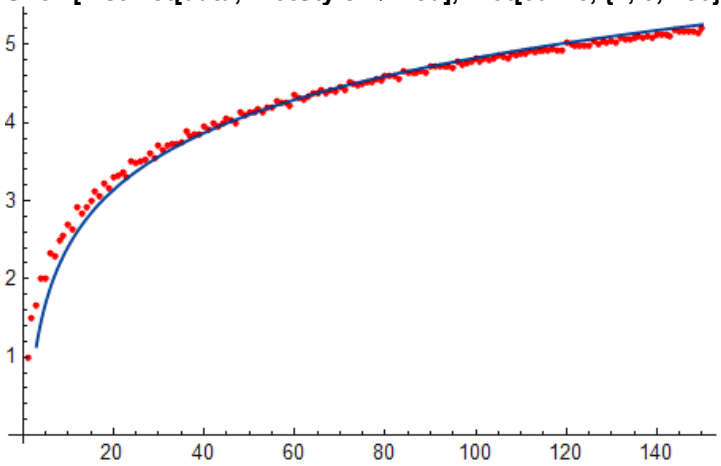


Аппроксимируем *average*(n) функцией ln n.

```
curve = Fit[data, Log[x], x]
```

1.04853 Log[x]

```
Show[ListPlot[data, PlotStyle → Red], Plot[curve, {x, 0, 150}]]
```

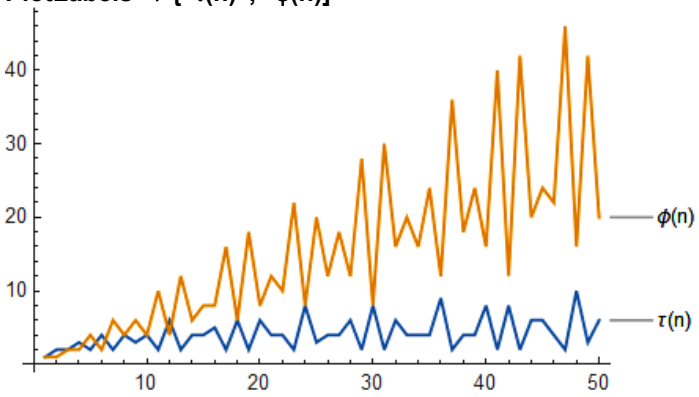


Аналитическое исследование функции *average*(n) показывает [25, с. 70], что

$$\frac{1}{n}\sum_{k=1}^{n}\tau(k) = \ln n + O(1).$$

Задача 2. Сравним величины функций $\tau(n)$ и $\varphi(n)$ для $n < 200$. Построим график:

```
ListLinePlot[{DivisorSigma[0, Range[50]], EulerPhi[Range[50]]},
PlotLabels → {"τ(n)", "φ(n)}
```



Докажем, используя Mathematica, что для $n > 30$ имеем неравенство $\tau(n) < \varphi(n)$.

Решение из справочной службы языка Wolfram:

```
FullSimplify[DivisorSigma[0, n] < EulerPhi[n], n ∈ Integers && n > 30]
True
```

Задача 3. Исследуем поведение бесконечных последовательностей, создаваемых следующим образом. Первый элемент – произвольное натуральное число. Далее, пусть n – текущий элемент последовательности. Следующий элемент определяется по правилу: если $n = 0$, то следующий элемент также 0, иначе новый элемент последовательности есть $\sigma(n) - n$.

Возникают два вопроса.

1. Любая ли последовательность становится периодической?

2. Если последовательность периодическая, то каковы возможные периоды?

Пусть S_n обозначает последовательность, начинающуюся с числа n. Выражение

FixedPointList[If[# == 0, 0, DivisorSigma[1, #] – #]&, n, k]

вычисляет первые k членов последовательности S_n, если в ней не появляются два нуля подряд, в противном случае вычисление прекращаются раньше. Например, пусть $n = 138$ и $k = 200$, тогда за 180 шагов последовательность достигает двух нулей подряд.

```
s = FixedPointList[If[# == 0, 0, DivisorSigma[1, #] – #]&, 138, 200];
Length[s]
180
```

Можем убедиться, что список s заканчивается двумя нулями.

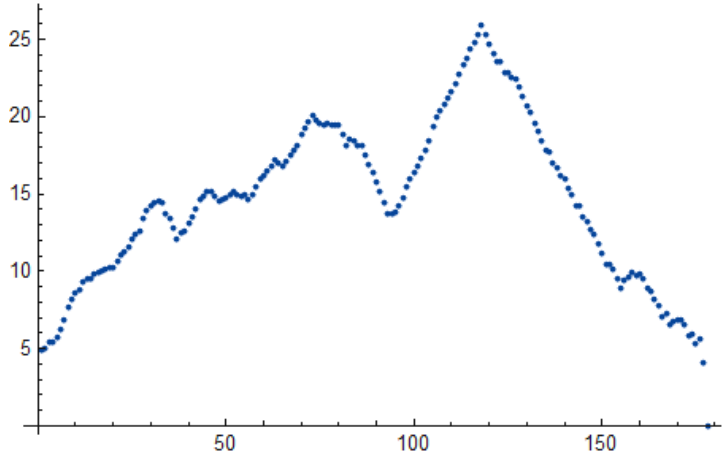
```
Take[s, –3]
{1, 0, 0}
```

Вычислим наибольший член в этой последовательности:

```
Max[s]
179931895322
```

График, где на оси ординат отмечены логарифмы значений элементов, иллюстрирует поведение последовательности S_{138}.

```
ListPlot[Log[s]]
```



Множество последовательностей M_{1000} определим, как $M_{1000} = \{S_n \mid 1 \le n \le 1000\}$. Если рассмотреть 250 элементов для каждой последовательности из M_{1000}, то периодическими становятся все последовательности, за исключением тех S_n, у которых

$$n \in \{276, 306, 396, 552, 564, 660, 696, 702, 780, 828, 840, 888, 966, 978, 990, 996\}.$$

Появляющиеся периоды сводятся либо к повторяющимся числам 0 (это для большинства последовательностей), 6, 28, 496 (это совершенные числа), либо к двухэлементному периоду {220, 284} (это дружественные числа).

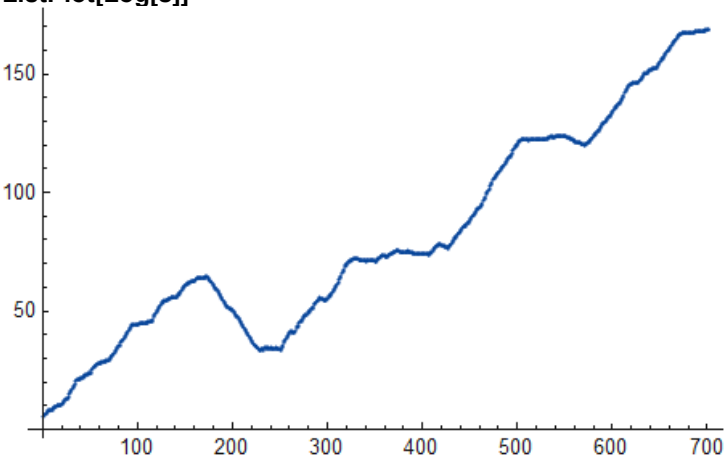
Рассмотрим последовательность S_{276}. Эта последовательность все еще не периодическая и после вычисления 700 элементов.

```
s = FixedPointList[If[# == 0, 0, DivisorSigma[1, #] – #]&, 276, 700];
//Timing
{6783.73, Null}
```

Последний элемент списка *s*, по-видимому, является наибольшим среди первых 700 членов последовательности (десятичное значение представлено 74 цифрами).

```
Max[s]
196720824905057327598072899837672081753417241644335345818959
43000246457396
```

```
IntegerLength[%]
74

ListPlot[Log[s]]
```



Поскольку числа последовательности S_{276} почти все время возрастают, то для вычисления значений $\sigma(n)$ требуется все больше времени – для процессора Intel® Core™ i5-4590 CPU@3.30GHz потребовалось на вычисления более 101 минуты.

§ 3. Совершенные числа и числа Мерсенна

Математики античной Греции интересовались числами, у которых сумма собственных делителей равнялась самому числу. Если n – натуральное число и $\sigma(n) = 2n$, то n называется **совершенным** числом. Найдем совершенные числа меньшие 10 000.

```
Select[Range[10000], DivisorSigma[1, #] == 2# &]
{6, 28, 496, 8128}
```

Теорема 8. Для того чтобы натуральное число n являлось четным совершенным числом необходимо и достаточно, чтобы $n = 2^{m-1}(2^m - 1)$, где m – целое со свойствами $m \geq 2$ и $2^m - 1$ – простое число.

Доказательство. *Достаточность*. Заметим сначала, что из нечетности $2^m - 1$ следует, что $(2^{m-1}, 2^m - 1) = 1$. Так как σ – мультипликативная функция, то $\sigma(n) = \sigma(2^{m-1})\sigma(2^m - 1)$. В силу леммы 1 имеем

$\sigma(2^{m-1}) = 2^m - 1$ и, так как $2^m - 1$ есть простое число, то $\sigma(2^m - 1) = 2^m$. Поэтому

$$\sigma(n) = (2^m - 1)\, 2^m = 2n,$$

т.е. n – совершенное число.

То, что число n – совершенное, можно было бы проверить с Mathematica:

```
n = 2^p - 1 2^p - 1;
FullSimplify[DivisorSigma[1, n], Element[2^p - 1, Primes]]
- 2^p + 4^p

Simplify[% == 2n]
True
```

Необходимость. Пусть n – четное совершенное число. Представим n как $n = 2^s t$, где s и t являются натуральными числами и t нечетно. Так как $(2^s, t) = 1$, то, по лемме 1, имеем

$$\sigma(n) = \sigma(2^s t) = \sigma(2^s)\sigma(t) = (2^{s+1} - 1)\sigma(t). \tag{3}$$

Так как n – совершенное число, то имеем

$$\sigma(n) = 2n = 2^{s+1} t. \tag{4}$$

Комбинируя (3) и (4), видим, что

$$(2^{s+1} - 1)\sigma(t) = 2^{s+1} t. \tag{5}$$

Поскольку $(2^{s+1} - 1, 2^{s+1}) = 1$, то 2^{s+1} делит $\sigma(t)$. Поэтому существует такое целое q, что $\sigma(t) = 2^{s+1} q$. Вставляя это выражение для $\sigma(t)$ в (5), получаем

$$(2^{s+1} - 1)2^{s+1} q = 2^{s+1} t,$$

и поэтому

$$(2^{s+1} - 1)q = t. \tag{6}$$

Следовательно, $q|t$ и $q \neq t$.

Сложив почленно равенство (6) и тождество $q = q$, получаем

$$t + q = (2^{s+1} - 1)q + q = 2^{s+1} q = \sigma(t). \tag{7}$$

Докажем, что $q = 1$. Заметим, что если $q \neq 1$, то имеется три различных положительных делителей t, а именно, 1, q и t. Следовательно, $\sigma(t) \geq 1 + q + t$, но это противоречит (7). Поэтому $q = 1$, и из (6) заключаем, что $t = 2^{s+1} - 1$. Также из (7) получаем $\sigma(t) = t + 1$, откуда следует, что t – простое число, так как единственные делители t – это 1 и t. Окончательно имеем $n = 2^s(2^{s+1} - 1)$, где $2^{s+1} - 1$ есть простое число. ■

Теорема 8 указывает нам, что для поиска четных совершенных чисел необходимо искать простые числа вида $2^m - 1$. Поиск последних чисел можно облегчить, если заметить, что m должно быть также простым числом.

Теорема 9. Если m – натуральное число и $2^m - 1$ есть простое число, тогда m – простое число.

Доказательство. Заметим, что $m > 1$. Теперь, от противного, пусть $m = ab$, где $1 < a < m$ и $1 < b < m$. Легко проверить, что $2^m - 1 = 2^{ab} - 1$ есть составное число. Например, это можно сделать с помощью *Mathematica*:

```
Simplify[(2^a - 1) Sum[2^(a (b - i)), {i, 1, b}]]
```

$$-1 + 2^{ab}$$

Так как оба сомножителя $2^{ab} - 1$ больше 1, то это число составное. ■

Пример 6. Если рассмотреть первые 20 простых m, то мы получаем первые 9 четных совершенных чисел. Используем *Mathematica*:

```
Select[{#, 2^(#-1) (2^# - 1)} & /@ Prime[Range[20]], PrimeQ[2^#[[1]] - 1] &] // Column
```

```
{2, 6}
{3, 28}
{5, 496}
{7, 8 128}
{13, 33 550 336}
{17, 8 589 869 056}
{19, 137 438 691 328}
{31, 2 305 843 008 139 952 128}
{61, 26 584 559 9156 983 174 465 469 261 595 3842 176}
```

Неизвестно, существуют ли нечетные совершенные числа. Доказано, что если нечетные совершенные числа существуют, то они должны быть больше 10^{15000} [16]. В языке Wolfram имеется функция PerfectNumber[n], которая выдает n-е по порядку совершенное число. Функция PerfectNumber[n] будет пытаться найти совершенные числа для любого n, но нельзя ожидать, чтобы она получила результат в разумное время при $n > 18$. Но вызов функции PerfectNumber[n, "Even"] выдаст значения для всех натуральных $n \le 45$. Также имеется функция

PerfectNumberQ[n], которая проверяет, является ли n совершенным числом.

Нахождение простых чисел вида $2^m - 1$ является исследовательской проблемой, которой занимаются с начала семнадцатого века. Много усилий потратил на их изучение Марен Мерсенн (1588–1648), французский священник, математик и философ.

Если m – натуральное число, $2^m - 1$ называется **числом Мерсенна**; если p – простое число и $2^p - 1$ – также простое, то $2^p - 1$ называется **простым числом Мерсенна**.

Числа Мерсенна замечательны в первую очередь тем, что для них существует эффективный тест на простоту.

Теорема 10. Тест Люка – Лемера[29] (доказательство см. в [86, с. 209, 210]). Пусть p – нечетное простое число и $M_p = 2^p - 1$ – число Мерсенна. Определим последовательность целых чисел рекурсивно, положив $r_1 = 4$, и для $k \geq 2$ определяем

$$r_k \equiv r_{k-1}^2 - 2 (\mathrm{mod}\, M_p),\ 0 \leq r_k < M_p.$$

Число M_p – простое тогда и только тогда, когда $r_{p-1} \equiv 0 \pmod{M_p}$.

Определим функцию, реализующую этот тест:

```
testLL[p_] :=
    Module[{k = 1, ms = 2^p – 1, r = 4},
        While[k < p – 1, r = Mod[r^2 – 2, ms]; k++];
        r = = 0]
```

Среди первых двух тысяч нечетных простых чисел имеется 22 простых числа Мерсенна. Определим индексы этих чисел:

```
Parallelize[Select[Prime[Range[2, 2000]], testLL]]
{3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281,
3217, 4253, 4423, 9689, 9941, 11213}
```

Самое большое из этих чисел M_{11213} имеет 3376 цифр:

```
Length[IntegerDigits[2^11213 – 1]]
3376
```

[29] Эдуард Люка (1842–1891) – французский математик, теоретически обосновал тест в семидесятых годах XIX века; Деррик Лемер (1867–1938) – американский математик, получил простейшую версию теста.

В языке Wolfram реализована функция MersennePrimeExponent[n], которая по номеру n в порядке возрастания выдает показатель степени m простого числа Мерсенна $2^m - 1$. Например, получаем первые 7 показателей степени:

```
MersennePrimeExponent[Range[7]]
{2, 3, 5, 7, 13, 17, 19}
```

и вычисляем соответствующие простые числа Мерсенна:

```
2^% – 1
{3, 7, 31, 127, 8191, 131071, 524287}
```

В настоящее время известно 50 простых чисел Мерсенна [39]. Для 45 из них известны их порядковые последовательные номера. Функция MersennePrimeExponent[n] выдает значение для n от 1 до 45 включительно. Для пяти простых чисел Мерсенна $M_{42643801}$, $M_{43112609}$, $M_{57885161}$, $M_{74207281}$, $M_{77\,232\,917}$ порядковые номера неизвестны.

Посмотрим рост простых чисел Мерсенна на графике:

```
{data1, data2} = Transpose[Table[{n, MersennePrimeExponent[n]},
{n, 45}]];
ListPlot[{data1, data2}, Joined → True, PlotRange → All,
    PlotLabels → Placed[{n, "MersennePrimeExponent[n]"}, Above]]
```

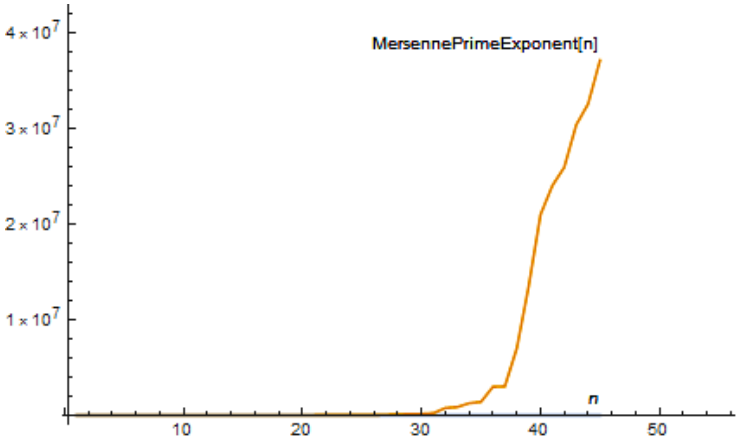
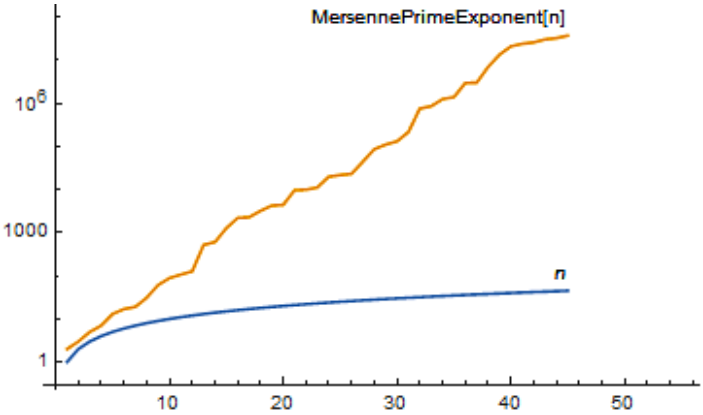


График становится более наглядным, если использовать логарифмический масштаб на оси ординат.

```
ListLogPlot[{data1, data2}, Joined → True,
    PlotLabels → Placed[{n, "MersennePrimeExponent[n]"}, Above]]
```



Число Мерсенна $M_{77232917}$ является не только наибольшим известным на сегодняшний день простым числом Мерсенна, но и наибольшим из найденных явно простых чисел вообще. Это число открыто в декабре 2017 г. и имеет 23 249 425 десятичных цифр. За редкими исключениями рекорд максимального открытого простого числа всегда принадлежал именно простым числам Мерсенна. О поисках больших простых чисел см в [86, с. 36–38; 39].

§ 4. Функция Мёбиуса

Пусть f – арифметическая функция. Формула $F(n) = \sum_{d|n} f(d)$ выражает значения сумматорной функции F через сумму значений функции f. Можно ли эту зависимость инвертировать (обратить)? То есть, возможно ли выразить значения функции f через значения функции F? Оказывается, можно, и в этом параграфе будет показано, как это сделать. Начнем с некоторых экспериментов, которые должны подсказать нам, какой вид формулы для обратной зависимости следует ожидать.

Следующие символьные преобразования можно было бы проделать вручную, но система Mathematica облегчит наш труд. Начнем с того, что явно выразим $F(n)$ через значения функции f для чисел $n = 1, 2, \ldots, 10$. Для этого определим вспомогательную функцию $r[n]$, которая для конкретного числа n выдает тождество с $F[n]$ в левой стороне, а в пра-

вой стороне – сумма значений $f[d]$, где d пробегает все делители n. И сразу посмотрим, что выдает $r[6]$:

```
r[n_] := F[n] == Plus @@ (f /@ Divisors[n])
r[6]
```

$$F[6] == f[1] + f[2] + f[3] + f[6]$$

Получим тождества для $n = 1, 2, \ldots, 10$:

```
r /@ Range[10] // Column
```

$$F[1] == f[1]$$
$$F[2] == f[1] + f[2]$$
$$F[3] == f[1] + f[3]$$
$$F[4] == f[1] + f[2] + f[4]$$
$$F[5] == f[1] + f[5]$$
$$F[6] == f[1] + f[2] + f[3] + f[6]$$
$$F[7] == f[1] + f[7]$$
$$F[8] == f[1] + f[2] + f[4] + f[8]$$
$$F[9] == f[1] + f[3] + f[9]$$
$$F[10] == f[1] + f[2] + f[5] + f[10]$$

Теперь можно выразить $f(1)$, $f(2)$, …, $f(10)$ через значения $F(1)$, $F(2)$, …, $F(10)$. Определим rs как список десяти тождеств, а потом применим функцию Reduce, которая выразит из этих уравнений $f(1), f(2), \ldots, f(10)$.

```
rs = r /@ Range[10];

Reduce[rs, {f}]
```

$$f[10] == F[1] - F[2] - F[5] + F[10] \;\&\&\; f[9] == -F[3] + F[9] \;\&\&\; f[8] == -F[4] + F[8] \;\&\&\; f[7] == -F[1] + F[7] \;\&\&\; f[6] == F[1] - F[2] - F[3] + F[6] \;\&\&\; f[5] == -F[1] + F[5] \;\&\&\; f[4] == -F[2] + F[4] \;\&\&\; f[3] == -F[1] + F[3] \;\&\&\; f[2] == -F[1] + F[2] \;\&\&\; f[1] == F[1]$$

Получили конъюнкции десяти новых тождеств. Для наглядности, превратим эту конъюнкцию в список.

```
List @@ % // Column
```

$$f[10] == F[1] - F[2] - F[5] + F[10]$$
$$f[9] == -F[3] + F[9]$$
$$f[8] == -F[4] + F[8]$$
$$f[7] == -F[1] + F[7]$$
$$f[6] == F[1] - F[2] - F[3] + F[6]$$
$$f[5] == -F[1] + F[5]$$

$f[4] = = -F[2] + F[4]$
$f[3] = = -F[1] + F[3]$
$f[2] = = -F[1] + F[2]$
$f[1] = = F[1]$

Заметим, что $f(n)$ равно сумме слагаемых вида $\pm F(n/d)$, где $d|n$. Этот факт приводит к догадке, что полезно рассмотреть тождество вида

$$f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d}),$$

где μ есть некоторая арифметическая функция. Изучая полученные тождества, определяем значения функции μ на числах 1, 2, …, 10. Имеем, $\mu(1) = 1$, $\mu(2) = -1$, $\mu(3) = -1$, $\mu(4) = 0$, $\mu(5) = -1$, $\mu(6) = 1$, $\mu(7) = -1$, $\mu(8) = 0$, $\mu(9) = 0$, $\mu(10) = 1$.

Далее, если p – простое число, то $F(p) = f(1) + f(p)$, что влечет $f(p) = F(p) - F(1)$. Последнее равенство и предполагаемое тождество $f(p) = \mu(1)F(p) + \mu(p)F(1)$ требуют $\mu(p) = -1$. Более того, так как

$$F(p^2) = f(1) + f(\text{p}) + f(p^2),$$

то имеем

$$f(p^2) = F(p^2) - (f(1) + f(\text{p})) = F(p^2) - F(p).$$

Последнее равенство вместе с предполагаемым тождеством

$$f(p^2) = \mu(1)F(p^2) + \mu(p)F(p) + \mu(p^2)F(1)$$

приводит к заключению, что $\mu(p^2) = 0$ для простого p, если дополнительно потребовать, что $f(1) \neq 0$. Подобное рассуждение можно использовать, чтобы доказать, что $\mu(p^k) = 0$ для любого целого $k > 1$.

Если мы предположим, что функция μ должна быть мультипликативной, то значения μ определяются через значения на степенях простых чисел. Это приводит к следующему определению.

Функцией Мёбиуса называется функция со следующими значениями:

$$\mu(n) = \begin{cases} 1, \text{ если } n = 1; \\ (-1)^r, \text{ если } n = p_1 p_2 \cdots p_r, \\ \qquad \text{где все } p_i - \text{различные простые числа}; \\ 0, \text{ иначе.} \end{cases}$$

Функция названа в честь Августа Мёбиуса[30]. Из этого определения мы видим, что $\mu(n) = 0$, если n делится на квадрат какого-нибудь числа.

[30] Август Мёбиус (1790–1868) – немецкий математик.

Убедимся, что функция Мёбиуса является мультипликативной.

Теорема 11. Функция Мёбиуса $\mu(n)$ – мультипликативная.

Доказательство. Пусть m и n – произвольные взаимно простые числа, докажем, что $\mu(n\ m) = \mu(n)\mu(n)$. Доказывать будем, используя разбор случаев. Пусть хотя бы один из сомножителей m и n равен 1. Так как $\mu(1) = 1$, то $\mu(n\ m) = \mu(n)\mu(n)$ выполнено.

Предположим сейчас, что хотя бы один из сомножителей делится на квадрат простого числа. Тогда $\mu(n\ m)$ и $\mu(n)\mu(n\ )$ одновременно равны 0. Наконец рассмотрим случай, когда m и n свободны от квадратов и больше 1. Предположим, что $n = p_1 p_2 \cdots p_r$ и $m = q_1 q_2 \cdots q_s$, где все сомножители p_i и q_j есть простые числа. Так как $(n, m) = 1$, то нет простого числа, которое одновременно присутствовало бы в факторизациях m и n. Следовательно, $m\ n$ есть произведение различных простых чисел в количестве $r + s$. Поэтому $\mu(n\ m) = (-1)^{r+s} = (-1)^r (-1)^s = \mu(n)\mu(n)$. ■

Заметим, что функция Мёбиуса не является вполне мультипликативной, так как $\mu(9) = 0$, но $\mu(3)\mu(3) = 1$.

Сумматорная функция для функции Мёбиуса оказывается очень важной и будет часто использоваться в дальнейшем. Введем для нее специальное обозначение – $\varepsilon(n)$.

Теорема 12. Пусть $\varepsilon(n) = \sum_{d|n} \mu(d)$, тогда

$$\varepsilon(n) = \begin{cases} 1, \text{если } n = 1; \\ 0, \text{если } n > 1. \end{cases}$$

Доказательство. Рассмотрим сначала случай $n = 1$. Имеем

$$\varepsilon(1) = \sum_{d|1} \mu(d) = \mu(1) = 1.$$

Пусть $n > 1$. По теореме 4, так как μ – мультипликативная функция, то сумматорная функция ε также мультипликативна. Сейчас предположим, что p – простое натуральное число и k – натуральное число. Тогда

$$\varepsilon(p^k) = \sum_{d|p^k} \mu(d) = \mu(1) + \mu(p) + \mu(p^2) + \ldots + \mu(p^k) =$$

$$= 1 + (-1) + 0 + \ldots + 0 = 0,$$

потому что $\mu(p^i) = 0$, при $i \geq 2$. Наконец, пусть факторизация $n > 1$ есть $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$. В силу мультипликативности ε получаем $\varepsilon(n) = \varepsilon(p_1^{a_1})\varepsilon(p_2^{a_2}) \cdots \varepsilon(p_r^{a_r}) = 0$. ■

Формула обращения Мёбиуса дает ответ, поставленный в начале параграфа. Она дает возможность выразить значения функции f через значения сумматорной функции F. Это формула интенсивно используется в изучении мультипликативных функций, в частности, с ее помощью получают новые тождества, включающие эти функции.

Теорема 13. **Формула обращения Мёбиуса.** Пусть f – арифметическая функция, а

$$F(n) = \sum_{d|n} f(d)$$

– сумматорная функция для f. Тогда для всех натуральных n имеет место

$$f(n) = \sum_{d|n} \mu(d) F(n/d).$$

Доказательство. Мы будем преобразовывать двойные суммы, начиная с правой стороны предыдущей формулы, после подстановки вместо $F(n/d)$ выражения

$$\sum_{e|(n/d)} f(e),$$

которое возникает из определения функции F как сумматорной функции для f. Мы имеем

$$\sum_{d|n} \mu(d) F(n/d) = \sum_{d|n} \mu(d) \Big(\sum_{e|(n/d)} f(e) \Big) = \sum_{d|n} \Big(\sum_{e|(n/d)} \mu(d) f(e) \Big).$$

Заметим, что множество пар чисел (d, e) с условиями $d|n$ и $e|(n/d)$ остается таким же, если выполнены условия $e|n$ и $d|(n/e)$. Это влечет равенства

$$\sum_{d|n} \Big(\sum_{e|(n/d)} \mu(d) f(e) \Big) = \sum_{e|n} \Big(\sum_{d|(n/e)} f(e) \mu(d) \Big) = \sum_{e|n} f(e) \Big(\sum_{d|(n/e)} \mu(d) \Big).$$

По теореме 12, $\sum_{d|(n/e)} \mu(d) = 0$, если $n/e \neq 1$. Если $n/e = 1$, т.е $n = e$, то сумма равна 1. Следовательно,

$$\sum_{e|n} \Big(f(e) \sum_{d|(n/e)} \mu(d) \Big) = f(n) \cdot 1 = f(n).$$

Что и требовалось доказать. ■

Пример 7. Функции $\tau(n)$ и $\sigma(n)$ являются сумматорными функциями для функций $u(n) = 1$ и $f_1(n) = n$. То есть,

$$\sigma(n) = \sum_{d|n} f_1(d) \text{ и } \tau(n) = \sum_{d|n} u(d).$$

Используя обращение Мёбиуса, получаем справедливость следующих тождеств для всех натуральных n:

$$n = \sum_{d|n} \mu(n/d)\sigma(d) \text{ и } 1 = \sum_{d|n} \mu(n/d)\tau(d).$$

Пример 8. Функция $f_1(n) = n$ являются сумматорной функцией для функции $\varphi(n)$ (теорема 3). Тогда при помощи формулы обращения Мёбиуса получаем

$$\varphi(n) = \sum_{d|n} \mu(n/d)d = n\sum_{d|n} \mu(d)/d.$$

Задачи с функцией Мёбиуса

Задача 4. Исследуйте, чему равно $\mu(n)\mu(n+1)\mu(n+2)\mu(n+3)$ для различных натуральных n. Предложите гипотезу и докажите.

Решение. Вычислим для первых 10 чисел:

```
MoebiusMu[#] MoebiusMu[# + 1] MoebiusMu[# + 2]
MoebiusMu[# + 3]& /@ Range[10]
```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

Докажем, что для любого натурального n соответствующее произведение равно 0. Действительно, каждое четвертое по порядку натуральное число делится на 4, Следовательно, для такого числа значение функции Мёбиуса равно 0. ■

Задача 5. Докажите или опровергните, что существует бесконечно много натуральных чисел n, таких что

$$\mu(n) + \mu(n+1) = 0. \tag{8}$$

Решение. Сначала просто подсчитаем количество чисел, удовлетворяющих равенству (8) и не превосходящих 10^2, 10^3, 10^4, 10^5 и 10^6, соответственно.

```
Length[Select[Range[#], MoebiusMu[#] + MoebiusMu[#+1] == 0 &]]& /@
      {100, 1000, 10000, 100000, 1000000}
```
{26, 274, 2673, 26909, 267872}

Частота таких чисел, по-видимому, не уменьшается с увеличением диапазона. Поэтому, возможно, натуральных чисел, для которых выполнено равенство (8) бесконечно много. Для доказательства рассмотрим две ситуации: 1) $\mu(n) = \mu(n + 1) = 0$; 2) $\mu(n) \times \mu(n + 1) = -1$.

Чисел n, для которых выполнено свойство $\mu(n) = \mu(n + 1) = 0$, бесконечно много. Действительно, для любого натурального числа $n = 36k + 8$ и $n + 1 = 36k + 9$ делятся соответственно на $2^2 = 4$ и на $3^2 = 9$. Поэтому $\mu(n) = \mu(n + 1) = 0$.

Условие $\mu(n) \times \mu(n + 1) = -1$ выполнено, в частности, и для простые чисел n, для которых $\mu(n + 1) = 1$. Снова посчитаем количество таких чисел в различных диапазонах, но уже только среди простых чисел

```
Length[Select[Prime[Range[#]], MoebiusMu[# + 1] == 1 &]]& /@
      {100, 1000, 10000, 100000, 1000000}
```
{16, 179, 1891, 18583, 186551}

Возможно, что простых чисел p, для которых $\mu(p) \times \mu(p + 1) = -1$, бесконечно много, но доказать это предоставляется читателю. ■

Задача 6. Докажите или опровергните, что существует бесконечно много натуральных чисел n, для которых $\mu(n - 1) + \mu(n) + \mu(n + 1) = 0$.

Решение. Если n есть решение системы сравнений $n \equiv -1 \pmod 4$, $n \equiv 0 \pmod 9$ и $n \equiv +1 \pmod{25}$, то $4 \mid (n + 1)$, $9 \mid n$ и $25 \mid (n - 1)$, и ни одно из чисел $n + 1$, n и $n - 1$ не свободно от квадратов. Поэтому для такого n имеем $\mu(n - 1) + \mu(n) + \mu(n + 1) = 0$. Китайская теорема об остатках говорит, что существует бесконечно много таких n. Найдем наименьшее:

```
ChineseRemainder[{3, 0, 1}, {2, 3, 5}^2]
```
351 ■

Задача 7. Как много последовательных натуральных чисел могут иметь ненулевые значения функции Мёбиуса?

Решение. Так как каждое четвертое по порядку натуральное число делится на 4, то может существовать самое большое три последовательных натуральных числа с ненулевыми значениями функции Мёбиуса.

Несколько первых троек таких чисел:

```
{# – 1, #, # + 1}& /@ Select[Range[50], MoebiusMu[# – 1] ≠ 0 &&
MoebiusMu[#] ≠ 0 && MoebiusMu[# + 1] ≠ 0&]
```
{{1, 2, 3}, {5, 6, 7}, {13, 14, 15}, {21, 22, 23}, {29, 30, 31},
{33, 34, 35}, {37, 38, 39}, {41, 42, 43}}

Задача 8. Как много может быть последовательных натуральных чисел, для которых функция Мёбиуса имеет значение 0.

Решение. Таких чисел может быть как угодно много. Покажем это. Пусть k – натуральное число и p_i – i-е простое число. По Китайской теореме об остатках существует решение n системы сравнений

$$n \equiv 0 (\mathrm{mod}\ p_1^2), n \equiv -1 (\mathrm{mod}\ p_2^2), n \equiv -2 (\mathrm{mod}\ p_3^2), \ldots, n \equiv -k+1 (\mathrm{mod}\ p_k^2).$$

Тогда для $i = 0, 1, \ldots, k-1$ мы имеем $p_{i+1}^2 \mid n+i$ и ни одно из чисел $n, n+1, \ldots, n+k-1$ не свободно от квадратов. Следовательно, μ имеет значение 0 для всех этих чисел.

Найдем 5 наименьших таких чисел.

```
ChineseRemainder[{0, 3^2 – 1, 5^2 – 2, 7^2 – 3, 11^2 – 4}, {2, 3, 5, 7, 11}^2] +
{0, 1, 2, 3, 4}
```
{1308248, 1308249, 1308250, 1308251, 1308252}

```
MoebiusMu /@ %
```
{0, 0, 0, 0, 0}

§ 5. Свёртка Дирихле

В конце предыдущего параграфа мы встретились с суммами вида

$$\sum_{d|n} f(d)g(n/d).$$

Суммы такого рода часто встречаются в аналитической теории функций и полезны при изучении свойств арифметических функций. Полезно трактовать эти суммы как некоторый новый вид умножения арифметических функций.

Пусть f и g – две арифметические функции. Определим **свертку Дирихле** функций f и g, обозначаемую $f * g$, как

$$(f * g)(n) = \sum_{d|n} f(d)g(n/d).$$

Эту функцию называют также произведением Дирихле или конволюцией Дирихле[31].

Теорема 14. Свёртка Дирихле – коммутативная и ассоциативная операция, т.е. для любых арифметических функций f, g и h имеем $f * g = g * f$ и $(f * g) * h = f * (g * h)$.

[31] Петер Густав Лежён Дирихле (1805 – 1859) – немецкий математик.

Доказательство. Сначала заметим, что множества $\{d \mid d$ – делитель $n\}$ и $\{n / d \mid d$ – делитель $n\}$ равны. И отсюда сразу получаем коммутативность свертки (d_1 обозначает n/d):

$$(f * g)(n) = \sum_{d|n} f(d)g(n/d) = \sum_{d_1|n} f(n/d_1)g(d_1) = (g * f)(n).$$

Для доказательства ассоциативности положим $A = g * h$. Тогда

$$(f * A)(n) = \sum_{a|n} f(a)A(n/a) =$$

$$= \sum_{a\cdot d=n} f(a) \sum_{b\cdot c=d} g(b)h(c) = \sum_{a\cdot b\cdot c=n} f(a)g(b)h(c).$$

Подобным образом, если положим $B = f * g$, то

$$(B * h)(n) = \sum_{d\cdot c=n} B(d)h(c) =$$

$$= \sum_{d\cdot c=n} \sum_{a\cdot b=d} f(a)g(b)h(c) = \sum_{a\cdot b\cdot c=d} f(a)g(b)h(c).$$

Поэтому $(f * (g * h))(n) = ((f * g) * h)(n)$. ■

Примеры 7 и 8 из предыдущего параграфа показывают, что

$$f_1 = \sigma * \mu = \mu * \sigma,\ u = \tau * \mu = \mu * \tau \text{ и } \varphi = \mu * f_1 = f_1 * \mu.$$

Функция $\varepsilon(n)$, введенная в теореме 12, является нейтральным элементом для операции свёртки Дирихле.

Теорема 15. Для любой арифметической функции f имеем

$$\varepsilon * f = f * \varepsilon = f.$$

Доказательство. По определению $\varepsilon(n)$ имеем

$$(\varepsilon * f)(n) = \sum_{d|n} \varepsilon(d) f(n/d) = f(n).$$

В силу коммутативности свертки, мы заключаем, что $f * \varepsilon = f$. ■

Теорема 16. Если f – арифметическая функция со свойством $f(1) \neq 0$, то существует единственная арифметическая функция g, для которой

$$f * g = \varepsilon. \tag{9}$$

Доказательство. Будем доказывать существования функции g с помощью математической индукции по натуральным числам, определяя для каждого числа m единственное значение $g(m)$. Базис индукции: $m = 1$. Для того чтобы выполнялось (9), необходимо, в частности, чтобы g удовлетворяло условию $f(1)g(1) = 1$. Так как $f(1) \neq 0$, то получаем $g(1) = 1/f(1)$.

Индуктивный переход. Предположим, что $m > 1$ и для всех целых k, $1 \le k \le m-1$, уже определено значение $g(k)$. Из отношения (9) находим

$$(f * g)(m) = \sum_{d|m} f(d)g(m/d) =$$
$$= f(1)g(m) + \sum_{d|m,\, d>1} f(d)g(m/d) = 0.$$

Поэтому имеем

$$g(m) = -\frac{1}{f(1)} \sum_{d|m,\, d>1} f(d)g(m/d).$$

И, тем самым, $g(m)$ определено однозначно. Следовательно, по математической индукции, определена функция g, такая, что выполнено (9). ■

Пусть f – арифметическая функция со свойством $f(1) \neq 0$. Тогда единственная арифметическая функция g, для которой выполнено отношение $f * g = \varepsilon$, называется **Дирихле обратная функция для функции** f и обозначается f^{-1}.

Оказывается, что Дирихле обратная функция для μ есть u. Из-за коммутативности свертки также имеем $u^{-1} = \mu$. Действительно, получаем

$$(\mu * u)(n) = \sum_{d|n} \mu(d)u(n/d) = \sum_{d|n} \mu(d) = \varepsilon(n),$$

где последнее равенство доказано в теореме 12.

Теорема 17. Множество арифметических функций со свойством $f(1) \neq 0$ образует абелеву группу с операцией свертки Дирихле и нейтральным элементом ε.

Доказательство сразу следует из теорем 14–16. ■

Теорема 18 (формула обращения Дирихле). Равенство $f = g * h$ выполнено тогда и только тогда, когда $g = f * h^{-1}$.

Доказательство. Пусть $f = g * h$. Тогда $f * h^{-1} = (g * h) * h^{-1} = g * (h * h^{-1}) = g * \varepsilon = g$. Обратно аналогично. ■

Введение свертки Дирихле позволяет переформулировать формулу обращения Мебиуса.

Следствие (формула обращения Мёбиуса через свертку Дирихле). Равенство $f = g * u$ выполнено тогда и только тогда, когда $g = f * \mu$.

В частности, следствие не только переформулирует теорему 4, но является и обратным утверждение для теоремы 4.

Покажем, что операция свертки замкнута относительно мультипликативности.

Теорема 19. Если функции f и g – мультипликативные функции, то их свертка $h = f * g$ также мультипликативна.

Доказательство. Заметим, что

$$h(1) = \sum_{d|1} f(d)g(1/d) = f(1)g(1) = 1.$$

Далее, рассмотрим выражение

$$h(mn) = \sum_{c|mn} f(c)g(mn/c).$$

Если $(m, n) = 1$, то, по лемме 1, мы можем представить $c = ab$, где $a|m$, $b|m$ и $(a, b) = 1$. Поэтому получаем, используя мультипликативность f и g,

$$h(mn) = \sum_{c|mn} f(c)g(mn/c) = \sum_{a|m}\sum_{b|n} f(ab)g(\frac{m}{a}\frac{n}{b}) =$$
$$= \sum_{a|m}\sum_{b|n} f(a)f(b)g(m/a)g(n/b) =$$
$$= \sum_{a|m} f(a)g(m/a)\sum_{b|n} f(b)g(n/b) = h(m)h(n).$$

Что и требовалось доказать. ■

Предупреждение. Свертка двух вполне мультипликативных функций может не быть вполне мультипликативной.

Теорема 20. Пусть $f = g * u$. Функции f и g мультипликативны одновременно.

Доказательство. Утверждение следует из следствия теоремы 18, фактов, что функции u и μ мультипликативны, и из мультипликативности свертки двух мультипликативных функций (теорема 19). ■

Теорема 21. Если функции g и $f * g$ обе мультипликативны, то функция f мультипликативна.

Доказательство. Доказательство от противного. Предположим, что f – не мультипликативная функция. Тогда существуют два взаимно простых числа m и n, таких, что $f(mn) \neq f(m)f(n)$. Выберем mn самое маленькое с таким условием. Если $mn = 1$, то $f(1) \neq f(1)f(1)$ и $f(1) \neq 1$. Пусть $h = f * g$. Так как $1 = h(1) = f(1)g(1) = f(1) \neq 1$, то получаем противоречие. Следовательно, $mn \neq 1$.

Если $mn > 1$, то $f(ab) = f(a)f(b)$ для всех $1 \le ab < mn$ и $(a, b) = 1$. Тогда

$$h(mn) = f(mn)g(1) + \sum_{\substack{a|m,b|n \\ ab<mn}} f(ab)g(mn/(ab)) =$$

$$= f(mn) + \sum_{\substack{a|m,b|n \\ ab<mn}} f(a)f(b)g(m/a)g(n/b) =$$

$$= f(mn) - f(m)f(n) + h(m)h(n).$$

Другими словами, $h(mn) - h(m)h(n) = f(mn) - f(n)f(m)$. Так как $f(mn) \ne \ne f(m)f(n)$, то $h(mn) \ne h(m)h(n)$, что противоречит мутипликативности функции h. ■

Следствие. Если f – мультипликативная функция, то Дирихле обратная функция f^{-1} – также мультипликативная функция.

Доказательство. Достаточно заметить для доказательства, что функции f и $f * f^{-1} = \varepsilon$ мультипликативны. ■

Теорема 22. Множество мультипликативных арифметических функций образует абелеву подгруппу группы всех арифметических функций со свойством $f(1) \ne 0$ и с операцией свертки Дирихле и нейтральным элементом ε.

Доказательство сразу следует из теорем 17, 19 и следствия из теоремы 21. ■

Функция MoebiusMu[n] языка Wolfram вычисляет значения функции Мёбиуса для целого n. При этом выполняются равенства: MoebiusMu[n] = MoebiusMu[–n] и MoebiusMu[0] = = 0. В некоторых случаях выражения, содержащие MoebiusMu, могут быть упрощены с использованием FullSimplify. Следующий пример взят из справочной службы *Mathematica*.

```
FullSimplify[Abs[Sum[MoebiusMu[i],{i, 1, n}]] < 8 + n/9, n ∈ Integers
&& n ≥ 1]
True
```

DirichletConvolve[f[n], g[n], n, m] вычисляет свертку Дирихле арифметических функций $(f * g)(m)$. Если m – натуральное число, то значением свертки всегда будет конечная сумма, при других значениях m выдается сообщение об ошибке. Функции f и g могут иметь произвольные численные значения, даже комплексные.

Примеры 9 (вычисление свертки Дирихле).

1. $(f * g)(12)$

DirichletConvolve[*f*[*n*], *g*[*n*], *n*, 12]

$f[12]\,g[1] + f[6]\,g[2] + f[4]\,g[3] + f[3]\,g[4] + f[2]\,g[6] + f[1]\,g[12]$

2. $(a * b)(m) = ab\ \tau(m)$, в частности, $\tau(m) = (1 * 1)(m)$

DirichletConvolve[*a*, *b*, *n*, *m*]

$a\ b$ DivisorSigma[0, m]

3. $(1 * n)(m) = \sigma(m)$

DirichletConvolve[1, *n*, *n*, *m*]

DivisorSigma[1, m]

4. $(1 * n^k)(m) = \sigma_k(m)$

DirichletConvolve[1, *n*^{k}, *n*, *m*]

DivisorSigma[k, m]

5. $(n^3 * n^k)(m) = \sigma_{k-3}(m)$

DirichletConvolve[*n*^{3}, *n*^{k}, *n*, *m*]

m^3 DivisorSigma[$-3 + k$, m]

В качестве нейтрального элемента ε в группе мультипликативных функций в языке Wolfram используется функция δ_n (дельта Кронекера):

Simplify[{KroneckerDelta[0], KroneckerDelta[*n*]}, *n* ≠ 0]

{0, 1}

6. $(\mathrm{f} * \varepsilon)(m) = f(m)$

DirichletConvolve[*f*[*n*], KroneckerDelta[*n* – 1], *n*, *m*]

$f[m]$

7. $(\mu * 1)(m) = \varepsilon(m)$ – функция Мёбиуса и функция $u(n) = 1$ являются взаимно-обратными относительно свертки Дирихле

DirichletConvolve[MoebiusMu[*n*], 1, *n*, *m*]

KroneckerDelta[$1 - m$]

8. $f * (g + h) = (f * g) + (f * h)$ – дистрибутивность свертки Дирихле относительно сложения. Действительно,

DirichletConvolve[*f*[*n*], *g*[*n*] + *h*[*n*], *n*, *m*]

DirichletConvolve[$f[n]$, $g[n]$, n, m] + DirichletConvolve[$f[n]$, $h[n]$, n, m]

9. $n = \sum_{d|n} \varphi(d)$ (см. теорема 3)

```
DirichletConvolve[EulerPhi[n], 1, n, m]
```
m

10. $f_{k+1} = f_k * \varphi f_k$

```
Table[DirichletConvolve[n^k, n^k EulerPhi[n], n, m],{k, Range[0, 10]}]
```
$\{m, m^2, m^3, m^4, m^5, m^6, m^7, m^8, m^9, m^{10}, m^{11}\}$

В общем случае в языке Wolfram данная свертка не вычисляется, но тождество 10 справедливо – см. доказательство теоремы 26.

11. $(\mu * \sigma_k)(m)$

```
DirichletConvolve[MoebiusMu[n], DivisorsSigma[k, n], n, m]
```
m^k

12. $f_1 = \varphi * u$ и $\varphi = f_1 * \mu$ – частный случай формулы обращения Мёбиуса, где $f_1(n) = n$

```
DirichletConvolve[n, MoebiusMu[n], n, m]
```
EulerPhi[m]

Формула обращения Мёбиуса – частный случай формулы обращения Дирихле, но для того, чтобы с пользой использовать последнюю формулу, необходимо знать какие-то другие пары взаимно-обратных функций кроме $u(n)$ и $\mu(n)$. В теореме 16 явно указан алгоритм, вычисляющий значения Дирихле обратной функции. Пусть g – арифметическая функция и $g(1) \neq 0$. Напишем программу для вычисления значений $g^{-1}(n)$. Будем использовать рекурсию с запоминанием промежуточных значений.

```
reverseD[g_, 1] := reverseD[g, 1] = 1 / g[1]
reverseD[g_, n_] := reverseD[g, n] =
– (1 / g[1]) (Plus @@ Map[g[#] × reverseD[g, n / #]&,
Rest[Divisors[n]]])
```

Задача 9. Нахождение Дирихле обратной функции для функции f_1.

Найдем первые значения функции $f_1^{-1}(n)$ – Дирихле обратной для функции $f_1(n) = n$. Функцию f_1 будем представлять в виде анонимной функции #&.

```
reverseD[#&, #] & /@ Range[20]
```
{1, –2, –3, 0, –5, 6, –7, 0, 0, 10, –11, 0, – 13, 14, 15, 0, – 17, 0, – 19, 0}

Напечатаем значения функции вместе с аргументами:

```
{#, reverseD[#&, #]}& /@ Range[20] // Column
```

```
{1, 1}
{2, – 2}
{3, – 3}
{4, 0}
{5, – 5}
{6, 6}
{7, – 7}
{8, 0}
{9, 0}
{10, 10}
{11, – 11}
{12, 0}
{13, – 13}
{14, 14}
{15, 15}
{16, 0}
{17, – 17}
{18, 0}
{19, – 19}
{20, 0}
```

Нетрудно заметить, что в данном диапазоне аргументов $f_1^{-1}(n) = \mu(n) \times n$. Проверим это равенство для первой тысячи натуральных чисел:

```
AllTrue[(reverseD[#&, #] = = MoebiusMu[#] #)& /@ Range[1000], #&]
```

```
True
```

Докажем наше предположение с помощью Mathematica:

```
DirichletConvolve[n, n MoebiusMu[n], n, m]
```

$$m\delta_{1-m}$$

```
FullSimplify[%]
```

$$\begin{cases} 1 & m == 1 \\ 0 & \text{True} \end{cases}$$

Получили:

Теорема 23. $f_1^{-1}(n) = \mu(n) \times n.$

Задача 10. Нахождение Дирихле обратной функции для $f_k\ (n) = n^k$, где k – произвольное натуральное число.

Начнем с экспериментов: $k = 2$ и $k = 3$:

```
{#, reverseD[#^2& ,#]}& /@ Range[20]
```
{{1, 1}, {2, – 4}, {3, – 9}, {4, 0}, {5, – 25}, {6, 36}, {7, – 49},
{8, 0}, {9, 0}, {10, 100}, {11, – 121}, {12, 0}, {13, – 169}, {14, 196},
{15, 225}, {16, 0}, {17, – 289}, {18, 0}, {19, – 361}, {20, 0}

```
{#, reverseD[#^3&, #]}& /@ Range[20]
```
{{1, 1}, {2, –8}, {3, –27}, {4, 0}, {5, –125}, {6, 216},
{7, –343}, {8, 0}, {9, 0}, {10, 1000}, {11, –1331}, {12, 0},
{13, –2197}, {14, 2744}, {15, 3375}, {16, 0}, {17, –4913},
{18, 0}, {19, –6859}, {20, 0}}

Рассмотрение чисел n также из большего диапазона, как было проделано для $k = 1$, позволяет выдвинуть гипотезы, что

$$f_2^{-1}(n) = \mu(n) \times n^2 \text{ и } f_3^{-1}(n) = \mu(n) \times n^3.$$

Используя Mathematica, убедимся, что эти гипотезы верны.

```
DirichletConvolve[n^2, n^2 MoebiusMu[n], n, m]
```
$m^2 \delta_{1-m}$

```
FullSimplify[%]
```
$\begin{cases} 1 & m == 1 \\ 0 & \text{True} \end{cases}$

```
DirichletConvolve[n^3, n^3 MoebiusMu[n], n, m]
```
$m^3 \delta_{1-m}$

```
FullSimplify[%]
```
$\begin{cases} 1 & m == 1 \\ 0 & \text{True} \end{cases}$

Можно проверить и для других значений, скажем, для $k = 17$:

```
FullSimplify[DirichletConvolve[n^17, n^17 MoebiusMu[n], n, m]]
```
$\begin{cases} 1 & m == 1 \\ 0 & \text{True} \end{cases}$

По-видимому, и для любого натурального k имеем аналогичное равенство

$$f_k^{-1}(n) = \mu(n) \times n^k,$$

но с помощью Mathematica это не установить:

```
DirichletConvolve[#^k&[n], n^k MoebiusMu[n], n, m]]
DirichletConvolve[#^k&[n], n^k MoebiusMu[n], n, m]]
```

Но, оказывается, если функция является вполне мультипликативной, то найти для нее Дирихле обратную функцию просто.

Теорема 24. Пусть функция f мультипликативна. Тогда для того, чтобы f была вполне мультипликативна необходимо и достаточно, чтобы $f^{-1}(n) = \mu(n)f(n)$ для всех $n \geq 1$.

Доказательство. Необходимость. Пусть $g(n) = \mu(n)f(n)$. Так как функция f вполне мультипликативна, то имеем

$$(g * f)(n) = \sum_{d|n} \mu(d)f(d)f(n/d) =$$
$$= f(n)\sum_{d|n} \mu(d) = f(n)\varepsilon(n) = \varepsilon(n),$$

так как $f(1) = 1$ и $\varepsilon(n) = 0$ для $n > 1$. Следовательно, $g = f^{-1}$.

Достаточность. Пусть $f^{-1}(n) = \mu(n)f(n)$. Чтобы доказать, что f вполне мультипликативна, достаточно доказать, что $f(p^a) = (f(p))^a$ для всех простых p и целых $a \geq 1$ (теорема 2 (b)). Уравнение $f^{-1}(n) = \mu(n)f(n)$ влечет

$$\sum_{d|n} \mu(d)f(d)f(n/d) = 0$$

для всех $n > 1$. Следовательно, полагая $n = p^a$, мы имеем

$$\mu(1)f(1)f(p^a) + \mu(p)f(p)f(p^{a-1}) = 0,$$

что влечет $f(p^a) = f(p)f(p^{a-1})$, т.е f вполне мультипликативна. ■

Следствие 1. Для любого натурального k имеем равенство

$$f_k^{-1}(n) = \mu(n) \times n^k.$$

Для доказательства достаточно заметить, что функция $f_k(n) = n^k$ является вполне мультипликативной.

Следствие 2. Дирихле обратная функция для функции Эйлера

$$\varphi^{-1}(n) = \sum_{d|n} d\mu(d).$$

Доказательство. Так как $\varphi = \mu * f_1$, то имеем $\varphi^{-1} = \mu^{-1} * (f_1)^{-1}$. Но так как f_1 вполне мультипликативна, то $(f_1)^{-1} = \mu f_1$, и поэтому

$$\varphi^{-1} = \mu^{-1} * \mu f_1 = u * \mu f_1. \blacksquare$$

Следующая теорема показывает, что $\varphi^{-1}(n) = \prod_{p|n}(1-p)$.

Теорема 25. Если f – мультипликативная функция, то

$$\sum_{d|n} \mu(d) f(d) = \prod_{p|n}(1 - f(p)).$$

Доказательство. Пусть

$$g(n) = \sum_{d|n} \mu(d) f(d).$$

Тогда g – мультипликативная функция, так что для ее определения достаточно знать значения $g(p^a)$. Но имеем

$$g(p^a) = \sum_{d|p^a} \mu(d) f(d) = \mu(1) f(1) + \mu(p) f(p) = 1 - f(p).$$

Следовательно,

$$g(n) = \prod_{p|n} g(p^a) = \prod_{p|n}(1 - f(p)). \blacksquare$$

Теорема 26. Пусть $k \geq 0$ – целое число, тогда Дирихле обратная функция

$$(f_k \varphi)^{-1} = f_k * \mu f_{k+1}.$$

Доказательство. Докажем сначала тождество $f_{k+1} = f_k \varphi * f_k$ (см. пример 9(10)). Имеем, по определению,

$$(f_k(n)\varphi(n)) * f_k(n) = \sum_{d|n} d^k \varphi(d)(n/d)^k = n^k \sum_{d|n} \varphi(d) = n^{k+1}.$$

Последнее равенство следует из теоремы 3. Используя формулу обращения Дирихле (теорема 18), получаем $f_k \varphi = f_{k+1} * (f_k)^{-1}$. Следовательно,

$(f_k\ \varphi)^{-1} = (f_{k+1} * (f_k)^{-1})^{-1} = (f_{k+1})^{-1} * f_k =$ (следствие из теоремы 18) $= \mu f_{k+1} * f_k$. ■

Определим теперь Дирихле обратные функции для σ_a, где a – неотрицательное целое число.

Теорема 27. Для $n > 1$ имеем

$$\sigma_a^{-1}(n) = \sum_{d|n} d^a \mu(d)\mu(n/d).$$

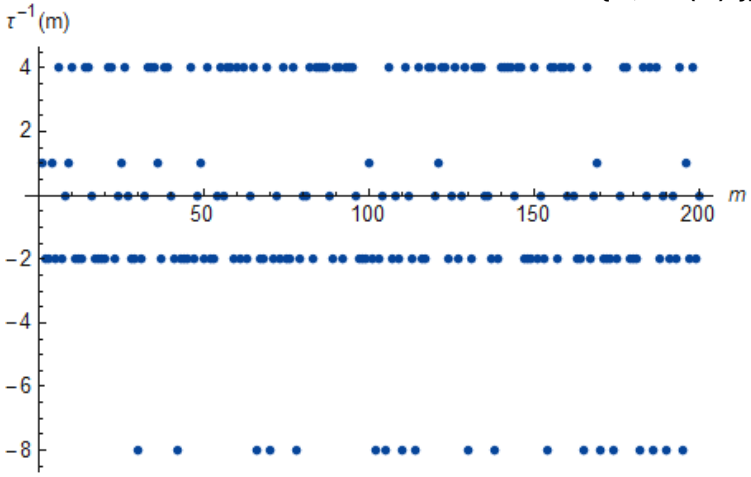
В частности, $\tau^{-1} = \mu * \mu$.

Доказательство. Так как $\sigma_a = f_a * u$ и f_a вполне мультипликативна, то имеем

$$\sigma_a^{-1} = (f_a * u)^{-1} = f_k^{-1} * u^{-1} = (\mu f_a) * \mu. \blacksquare$$

Получим визуальное впечатление от поведения функций τ^{-1} и σ^{-1}. Сначала рассмотрим функцию τ^{-1}.

```
reverseTau[m_] := Module[{n}, DirichletConvolve[MoebiusMu[n],
MoebiusMu[n], n, m]]
ListPlot[reverseTau /@ Range[200], PlotSyle →{PlotSize → Medium},
                                   AxesLabel → {m, "τ^-1(m)"}]
```

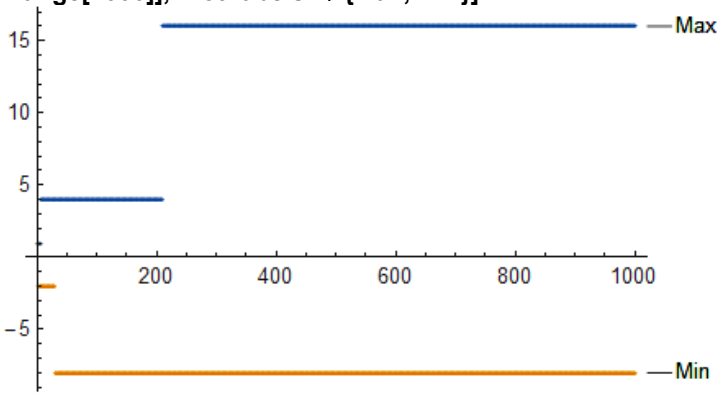


Мы видим, что значениями функции являются степени числа –2. Наибольшее и наименьшее значения $\tau^{-1}(m)$ медленно возрастают по модулю с ростом числа m.

```
{Max[#], Min[#]}& [reverseTau /@ Range[10000]]
```
{16, –32}

Более наглядно это видно на графике (в написанном ниже виде вычисление значения функции {Max[#], Min[#]}& происходит каждый раз заново с увеличением диапазона – это очень неэффективно):

```
ListPlot[Transpose[{Max[#], Min[#]}& [reverseTau /@ Range[#]]& /@
Range[1000]], PlotLabels → {Max, Min}]
```



Детальное изучение поведения τ^{-1} на большом диапазоне приводит к предположению, что значения функции $\tau^{-1}(m) = (\mu * \mu)(m)$ можно определить с помощью следующих правил:

1. Если $m \neq 1$ и m делится на куб целого числа, то $\tau^{-1}(m) = 0$.
2. Если m – точный квадрат, то $\tau^{-1}(m) = 1$.
3. Если в разложении m на простые множители степени сомножителей не превышают 2 и число различных сомножителей в первой степени равно k, то $\tau^{-1}(m) = (-2)^k$.

Функция τ^{-1} – мультипликативная, поэтому достаточно узнать значения функции на простых числах.

Теорема 28. Для простого числа p имеем:

(a) $\tau^{-1}(p) = -2$;

(b) $\tau^{-1}(p^2) = 1$;

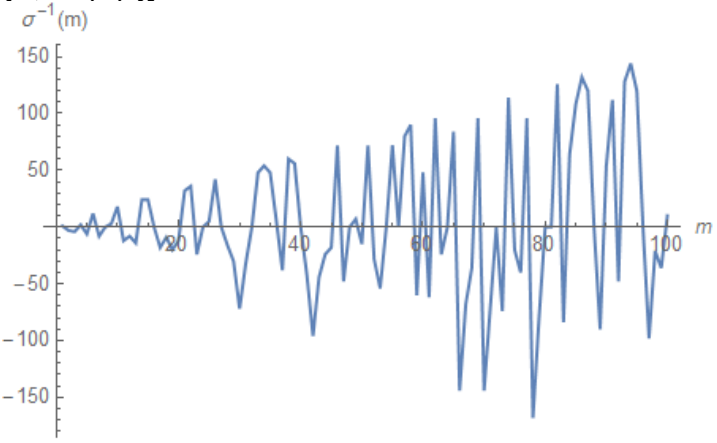
(c) $\tau^{-1}(p^3) = 0$.

Доказательство. a) $(\mu * \mu)(p) = \mu(1)\mu(p) + \mu(p)\mu(1) = -1 + -1 = -2$. b) $(\mu * \mu)(p^2) = \mu(1)\mu(p^2) + \mu(p)\mu(p) + \mu(p^2)\mu(1) = 0 + (-1)(-1) + 0 = 1$. c) $(\mu * \mu)(p^3) = \mu(1)\mu(p^3) + \mu(p)\mu(p^2) + + \mu(p^2)\mu(p) + \mu(p^3)\mu(1) = 0$. ■

Вышеприведенные правила 1–3 следуют из доказанной теоремы.

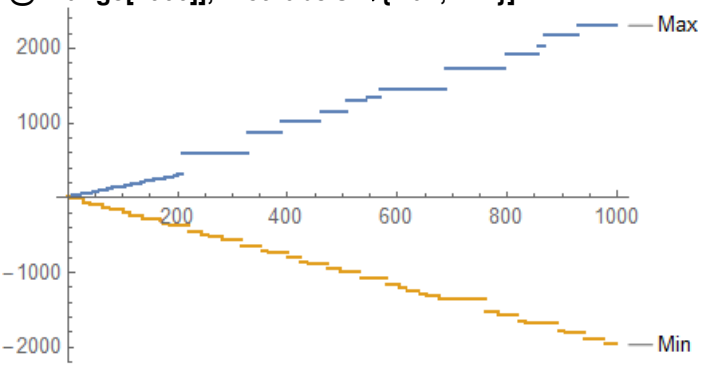
Теперь рассмотрим функцию σ^{-1}.

```
reverseSigma[m_] := Module[{n}, DirichletConvolve[n MoebiusMu[n],
MoebiusMu[n], n, m]]
ListPlot[reverseSigma /@ Range[100], Joined → True, AxesLabel →
{m,"σ^-1(m)"}]
```



Рост по модулю наибольшего и наименьшего значений $\sigma^{-1}(m)$ мы наблюдаем на следующем графике:

```
ListPlot[Transpose[{Max[#], Min[#]}& [reverseSigma /@ Range[#]]&
/@  Range[1000]], PlotLabels →{Max, Min}]
```



Функция σ^{-1} – мультипликативная, поэтому достаточно узнать значения функции на простых числах. При изучении результатов проведенных экспериментов получено следующее утверждение:

Теорема 29. Для простого числа p имеем
(a) $\sigma^{-1}(p) = -1 - p$;
(b) $\sigma^{-1}(p^2) = p$;
(c) $\sigma^{-1}(p^3) = 0$.

Доказательство. a) $(f_1\mu * \mu)(p) = 1\times\mu(1)\mu(p) + p\times\mu(p)\mu(1) = -1 + -p$. b) $(f_1\mu * \mu)(p^2) = 1\times\mu(1)\mu(p^2) + p\times\mu(p)\mu(p) + p^2\times\mu(p^2)\mu(1) = 0 + p(-1)(-1) + 0 = p$; c) $(f_1\mu * \mu)(p^3) = 1\times\mu(1)\mu(p^3) + p\times\mu(p)\mu(p^2) + p^2\times\mu(p^2)\mu(p) + p^3\times\mu(p^3)\mu(1) = 0$. ■

Глава 5

ЧИСЛА ФИБОНАЧЧИ

§ 1. Определения и некоторые свойства

Одной из самых замечательных последовательностей целых чисел является последовательность Фибоначчи[32]. Ее можно определить рекурсивно следующим образом:

Для $n \geq 0$ обозначим через F_n n-е число Фибоначчи и положим

$$F_0 = 0, \quad F_1 = 1,$$

$$F_n = F_{n-1} + F_{n-2} \text{ при } n \geq 2.$$

Имеется функция Fibonacci[n], вычисляющая числа Фибоначчи в языке Wolfram. Вот несколько первых чисел последовательности Фибоначчи:

```
Fibonacci /@ Range[0, 20]
```

{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765}

Числа Фибоначчи можно вычислять, используя выражение в «замкнутой форме»[33]. Рассмотрим квадратное уравнение $x^2 = x + 1$; оно имеет корни $\tau = (1+\sqrt{5})/2$ и $\sigma = -1/\tau$. Число τ, приблизительно равное 1.61803…, известно как «золотое сечение». Формула Бине[34] – удивительный результат о том, что определенная комбинация n-х степеней τ и σ равна F_n.

---

[32] Леонардо Пизанский, по прозвищу Фибоначчи (приблизительно, 1170–1250) – первый крупный математик средневековой Европы.

[33] Понятие «замкнутая форма» поясняется в [67, с. 24]: «выражение для величины $f(n)$ представлено в замкнутой форме, если его можно вычислить с помощью некоторого фиксированного числа «известных» стандартных операций, независимо от n. Например, выражения $2^n - 1$ и $n(n + 1)/2$ представлены в замкнутой форме, поскольку они включают в себя только сложение, умножение, деление и возведение в степень».

[34] Жак Бинé (1786–1856) – французский математик, механик и астроном.

Теорема 1 (формула Бине).

$$F_n = \frac{\tau^n - \sigma^n}{\sqrt{5}}. \qquad (1)$$

Доказательство. Используем математическую индукцию. Легко видеть, что формула справедлива для $n = 0$ и $n = 1$. Для доказательства индуктивного перехода предварительно заметим, что $1/\tau = (-1 + \sqrt{5})/2$. Поэтому достаточно показать, что правая сторона в (1) подчиняется тому же рекурсивному правилу, как и числа Фибоначчи:

$$\frac{\tau^n - \sigma^n}{\sqrt{5}} + \frac{\tau^{n+1} - \sigma^{n+1}}{\sqrt{5}} = \frac{\tau^{n+2} - \sigma^{n+2}}{\sqrt{5}}.$$

Но это следует немедленно из того факта, что

$$\tau^n + \tau^{n+1} - \tau^{n+2} = \tau^n (1 + \tau - \tau^2) = 0$$

и аналогичного соотношения для σ. ■

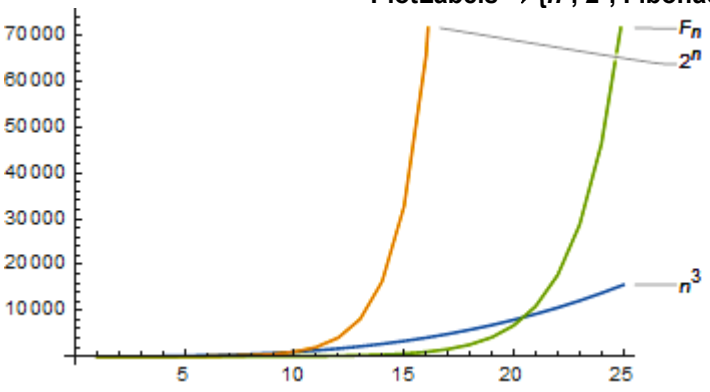
Так как $|\sigma| < 1$, то степени σ стремятся к 0. Это означает, что формула Бине имеет следующее

Следствие. F_n равно ближайшему целому числу к $\tau^n / \sqrt{5}$.

Доказательство. Так как $|\sigma| < 1$ и $\sqrt{5} > 2$, то $-\sigma^n / \sqrt{5}$ по абсолютной величине меньше чем 1/2. Этого достаточно для доказательства. ■

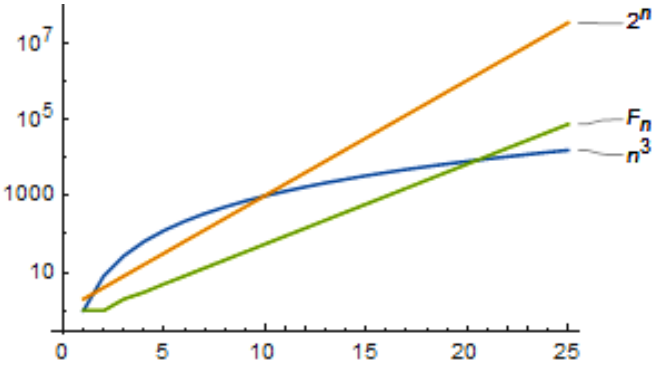
Сравним рост последовательности Фибоначчи и последовательностей n^3 и $2n$.

```
{data1, data2, data3} = Transpose[Table[{n^3, 2^n, Fibonacci[n]}, {n, 25}]];
ListPlot[{data1, data2, data3}, Joined → True,
                                PlotLabels → {n^3, 2^n, Fibonacci[n]}]
```



Можем использовать логарифмический масштаб на оси ординат.

```
ListLogPlot[{data1, data2, data3}, Joined → True,
                                   PlotLabels → {n^3, 2^n, Fibonacci[n]}]
```



Число Фибоначчи f_n равно примерно $2^{0{,}694n}$. Для быстрого вычисления чисел Фибоначчи можно использовать матричное тождество [68, с. 9]

$$\begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n.$$

Используя данное тождество, можно определить новую функцию на языке Wolfram для вычисления n-го числа Фибоначчи.

```
fib[n_] := Module[{k = n, b = {{1, 0}, {0, 1}}, c = {{1, 1}, {1, 0}}},
While[k ≠ 0, If[Mod[k, 2] == 0, k = k/2; c = c . c, k = k – 1; b = b . c]];
b[[2, 1]]]
```

Инфиксная матричная операция Dot, обозначаемая точкой, является произведением матриц.

```
fib /@ Range[10]
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55}
```

Литература о числах Фибоначчи многочисленна. В частности, известно очень много тождеств, связывающих числа Фибоначчи (см., например, [11]).

Приведем три тождества:

$$\sum_{r=0}^{5} F_{n+r} = 4F_{n+4};$$

$$\sum_{r=0}^{9} F_{n+r} = 11F_{n+6};$$

$$\sum_{r=0}^{n} F_r^2 = F_n F_{n+1}.$$

Если тождества найдены, то доказать их можно часто с помощью математической индукции. Данные тождества можно доказать на языке Wolfram, используя функцию FullSimplify.

```
FullSimplify[Sum[Fibonacci[n + r], {r, 0, 5}] == 4 Fibonacci[n + 4]]
```

True

```
FullSimplify[Sum[Fibonacci[n + r], {r, 0, 9}] == 11 Fibonacci[n + 6]]
```

True

```
FullSimplify[Sum[Fibonacci[n]^2, {r, 0, n}] == Fibonacci[n] Fibonacci[n + 1]]
```

True

Рассмотрим несколько задач с числами Фибоначчи

Задача 1. Чему равно $\gcd(F_n, F_{n+1})$?

Вычислим значения $\gcd(F_n, F_{n+1})$ для $n = 1, 2, \ldots, 10$.

```
GCD[Fibonacci[#], Fibonacci[# + 1]]& /@ Range[10]
```

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1}

Проверим для первой сотни ненулевых чисел Фибоначчи:

```
AllTrue[Map[# == 1&, GCD[Fibonacci[#],
                Fibonacci[# + 1]]& /@ Range[100]], #&]
```

True

Теорема 2. Для любого $n \geq 1$ выполнено $\gcd(F_n, F_{n+1}) = 1$.

Доказательство. Имеем $\gcd(F_1, F_2) = \gcd(1, 1) = 1$. Следовательно, если утверждение неверно, то существует наименьшее n, скажем $n = r$, для которого $\gcd(F_r, F_{r+1}) > 1$ и $\gcd(F_{r-1}, F_r) = 1$. Тогда существует целое $d > 1$, делящее F_r и F_{r+1}. Так как $F_{r-1} = F_{r+1} - F_r$, поэтому d делит также F_{r-1}. Но это противоречит $\gcd(F_{r-1}, F_r) = 1$. Следовательно, для $n \geq 1$ выполнено $\gcd(F_n, F_{n+1}) = 1$. ■

Задача 2. Чему равно $\gcd(F_n, F_{n+2})$?

Вычисляя значения $\gcd(F_n, F_{n+2})$ для $n = 1, 2, \ldots, 10$, получаем результат такой же, как в задаче 1.

```
GCD[Fibonacci[#], Fibonacci[# + 2]]& /@ Range[10]
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
```

Проверим для первой тысячи ненулевых чисел Фибоначчи.

```
Select[Range[1000], GCD[Fibonacci[#], Fibonacci[# + 2]] > 1&]
{}
```

Вычисления подтверждают равенство $\gcd(F_n, F_{n+2}) = 1$ для $0 < n \le 1000$. Используя рассуждения, подобные доказательству теоремы 2, устанавливаем истинность следующего утверждения.

Теорема 3. Для любого $n \ge 1$ выполнено $\gcd(F_n, F_{n+2}) = 1$.

Задача 3. Для каких $k \ge 1$ выполнено $\gcd(F_n, F_{n+k}) = 1$ для всех $n \ge 1$?

Вычисляя значения $\gcd(F_n, F_{n+3})$ для $n = 1, 2, \ldots, 20$, получаем

```
GCD[Fibonacci[#], Fibonacci[# + 3]]& /@ Range[20]
{1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1}
```

Как видим, для $k = 3$ равенство $\gcd(F_n, F_{n+3}) = 1$ не выполняется для всех n. Эксперименты для других $k > 2$ приводят к такому же результату. По-видимому равенство нарушается для всех $k > 2$. Как это выяснить? Продолжим эксперименты.

Задача 4. Если $k \ge 3$, то для каких n имеем $\gcd(F_n, F_{n+k}) > 1$?

Зафиксируем несколько первых значений k и для каждого k найдем значения $n \le 50$, для которых $\gcd(F_n, F_{n+k}) > 1$. Определим функцию $f[k]$, которая выдает список таких $n \le 50$ и вычислим ее для некоторых k.

```
f[k_] := Select[Range[50], GCD[Fibonacci[#], Fibonacci[# + k]] ≠ 1&]
f[3]
{3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48}

f[4]
{4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48}

f[5]
{5, 10, 15, 20, 25, 30, 35, 40, 45, 50}
```

Изучая результаты, получаем, что если n кратно k, то $n + k$ также кратно k и имеем $\gcd(F_n, F_{n+k}) > 1$. На самом деле справедлива

Теорема 4[65]. Если $n > 2$, то F_m делится на F_n тогда и только тогда, когда m делится на n.

Из этой теоремы следует, что если n делится на k, то $\gcd(F_n, F_{n+k})$ делится на F_k. Более того, справедлив более сильный результат:

Теорема 5[67, с. 326]. Если $n, m \geq 1$, то $F_{\gcd(m,n)} = \gcd(F_n, F_m)$.

Многочлен для чисел Фибоначчи

Напомним определения перечислимости множеств и диофантовости множеств [89, 73].

Множество A натуральных чисел называется *диофантовым* тогда и только тогда, когда оно является множеством всех положительных значений некоторого полинома с целыми коэффициентами при натуральных значениях переменных.

Множество натуральных чисел называется *перечислимым*, если оно перечисляется некоторым алгоритмом, то есть если существует алгоритм, который выдает в качестве результата в произвольном порядке все элементы этого множества и только их.

Теорема 6 [91]. Диофантовость множества равносильна его перечислимости.

В частности, любое множество положительных целых чисел, являющееся результатом работы некоторого алгоритма, может быть получено при нахождении положительных значений некоторого полинома.

Числа Фибоначчи порождаются положительными значениями многочлена пятой степени от двух переменных [89]:

$$p(a, b) = 2a^4b + a^3b^2 - 2a^2b^3 - a^5 - ab^4 + 2a.$$

Оценим полезность этого многочлена для генерации чисел Фибоначчи.

Фиксируем верхнюю границу *maxab* для значений переменных a и b. В качестве первого значения возьмем *maxab* = 1000. Переберем значения переменных a и b от 1 до *maxab* и получим 1 миллион значений многочлена $p(a, b)$.

```
p[a_, b_] := 2 a^4 b + a^3 b^2 - 2 a^2 b^3 - a^5 - a b^4 + 2 a
s1000 = Union[Flatten[Table[p[a, b], {a, 1, 1000}, {b, 1, 1000}]]];
Max[Abs[s]]
```

1080971238104521

```
Length[s1000]
999135

Select[s1000, # > 0&]
{1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987}
```

Таким образом, наибольшее число равно 1 080 971 238 104 521, а неповторяющихся чисел будет 999 135. Но если удалить все отрицательные значения многочлена, то останется всего 16 первых чисел Фибоначчи.

Увеличим теперь *maxab* до 2000. Теперь получаем 4 миллиона значений многочлена $p(a, b)$.

```
s2000 = Union[Flatten[Table[p[a, b], {a, 1, 2000}, {b, 1, 2000}]]];
Max[Abs[s2000]]
34591102466944803

Length[s2000]
3998054

Select[s2000, # > 0&]
{1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597}
```

Теперь наибольшее значение многочлена равно 34 591 102 466 944 803, а неповторяющихся чисел – 3 998 054. Но отметим, что 17-е число 1597 является единственным числом Фибоначчи, полученным, когда a и b были больше 1000.

Наконец, положим *maxab* = 3000. Получаем 9 миллионов значений многочлена $p(a, b)$.

```
s3000 = Union[Flatten[Table[p[a, b], {a, 1, 3000}, {b, 1, 3000}]]];
Max[Abs[s3000]]
262676167803027544

Length[s3000]
8996885
```

Наибольшее значение многочлена стало 262 676 167 803 027 544 среди 8 996 885 неповторяющихся чисел.

```
Select[s3000, # > 0&]
{1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584}
```

Но, несмотря на то, что диапазон изменений переменных a и b значительно вырос, получено только единственное новое число Фибоначчи

2584. Мы видим, что многочлен

$$p(a, b) = 2a^4b + a^3b^2 - 2a^2b^3 - a^5 - ab^4 + 2a$$

практически непригоден для генерации чисел Фибоначчи.

Подстановка Фибоначчи

Теорема 7[35]. Пусть на строке символов задана подстановка $\{x \to xy, y \to x\}$. Если начинать с единственного символа x и совершать указанную подстановку n раз, то результирующая строка содержит F_{n+1} раз символ x и F_n раз символ y.

Проверим истинность утверждения. Сначала будем представлять строки в виде списков символов. Используем функцию NestList для визуализации результатов последовательных подстановок.

```
NestList[Flatten[# /. {x → {x, y}, y → x}]&, {x}, 6] // Column
{x}
{x, y}
{x, y, x}
{x, y, x, x, y}
{x, y, x, x, y, x, y, x}
{x, y, x, x, y, x, y, x, x, y, x, x, y}
{x, y, x, x, y, x, y, x, x, y, x, x, y, x, y, x, x, y, x, y, x}
```

Функция Count применяется для подсчета символов x и y для каждого шага выполнения функции NestList.

```
{Count[#, x], Count[#, y]}& /@ NestList[Flatten[# /. {x → {x, y}, y →
x}]&, {x}, 10]]
{{1, 0}, {1, 1}, {2, 1}, {3, 2}, {5, 3}, {8, 5}, {13, 8}, {21, 13}, {34, 21},
{55, 34}, {89, 55}}
```

Сейчас будем использовать только строки. Вместо функции Replace (в примере со списками использовалось инфиксное обозначение «/.») применяем функцию StringReplace.

```
NestList[StringReplace[#, {"x" → "xy", "y" → "x"}]&, "x", 6]
{x, xy, xyx, xyxxy, xyxxyyx, xyxxyyxxyxxy, xyxxyyxxyyxxyyxxyyx}
```

Функция CountString применяется для подсчета символов x и y для каждого шага выполнения функции NestList.

[35] http://functions.wolfram.com/IntegerFunctions/Fibonacci/31/02/

```
{StringCount[#, "x"], StringCount[#, "y"]}& /@
  NestList[StringReplace[#, {"x" → "xy", "y" → "x"}]&, "x", 10]
```

{{1, 0}, {1, 1}, {2, 1}, {3, 2}, {5, 3}, {8, 5}, {13, 8}, {21, 13}, {34, 21}, {55, 34}, {89, 55}}

Более удобно использовать SubstitutionSystem – «систему переписывания по правилам» (в русскоязычной литературе используется термин «системы переписывания термов»). Функция SubstitutionSystem[*rule*, *unit*, *t*] создает список, представляющий эволюцию системы по правилам *rule* с начальным состоянием *unit* за *t* шагов. Эта функция работает со списками и строками и с некоторыми другими более сложными выражениями.

Вызовы функции

```
SubstitutionSystem[{"x" → "xy", "y" → "x"}, "x", 6]
```

и

```
SubstitutionSystem[{x} → {x, y}, y → {x}}, {x}, 6]
```

дают уже рассмотренные результаты для строк и списков соответственно.

§ 2. Фибоначчиевая система счисления

В книге [67, с. 327–333] рассмотрена фибоначчиевая система счисления, основанная на следующей теореме Цеккендорфа [57]. Для чисел Фибоначчи используется обозначение F_n.

Каждое натуральное число имеет единственное представление вида

$$n = F_{k_1} + F_{k_2} + \cdots + F_{k_r}, \tag{2}$$

где соседние номера чисел Фибоначчи удовлетворяют условию

$$k_i \geq k_{i+1} + 2.$$

Например, представление одного миллиарда оказывается таким:

$$10^9 = 701408733 + 267914296 + 24157817 + 5702887 + 514229 +$$
$$+ 196418 + 75025 + 28657 + 1597 + 233 + 89 + 13 + 5 + 1 = F_{44} + F_{42} +$$
$$+ F_{37} + F_{34} + F_{29} + F_{27} + F_{25} + F_{23} + F_{17} + F_{13} + + F_{11} + F_7 + F_5 + F_2.$$

Мы можем для представления натуральных чисел использовать последовательности нулей и единиц, записывая $n = (b_m b_{m-1} \ldots b_2)_F$ тогда и только тогда, когда $n = \sum_{k=2}^{m} b_k F_k$. Эта система счисления отчасти напо-

минает двоичное представление, за исключением того, что в ней никогда не встречаются две 1 подряд. Вот, например, числа от 1 до 12, представленные по Фибоначчи:

$$1 = (1)_F,\ 2 = (10)_F,\ 3 = (100)_F,\ 4 = (101)_F,\ 5 = (1000)_F,$$
$$6 = (1001)_F,\ 7 = (1010)_F,\ 8 = (10000)_F,\ 9 = (10001)_F,$$
$$10 = (10010)_F,\ 11 = (10100)_F,\ 12 = (10101)_F.$$

Наша цель – написать функцию, представляющую натуральные числа в фибоначчиевой системе счисления. Представление (2) будем находить с помощью жадного алгоритма: в качестве F_{k_1} выбирается наибольшее число Фибоначчи $\le n$, затем в качестве F_{k_2} выбирается наибольшее число $\le n - F_{k_1}$, и т. д.

Следствие из теоремы 1 говорит, что $F_k = \text{GoldenRatio}^k / \sqrt{5}$, округленное до ближайшего целого (GoldenRatio – числовая константа языка Wolfram – отношение золотого сечения равное $\frac{1}{2}\left(\sqrt{5}+1\right)$). Для нахождения ближайшего к n снизу числа Фибоначчи сначала, используя Reduce, решим уравнение

```
Reduce[{GoldenRatio^k/Sqrt[5] == n, n > 0}, {k}, Reals]
```

$$n > 0 \;\&\&\; k == \frac{\text{Log}\left[\sqrt{5}\,n\right]}{-\text{Log}[2] + \text{Log}\left[1+\sqrt{5}\right]}$$

Если k четно, то F_k немного меньше, чем $\text{GoldenRatio}^k / \sqrt{5}$; в противном случае оно немного больше. Поэтому ближайшее к n снизу число Фибоначчи можно вычислить с помощью функции:

```
near[n_] := If[Fibonacci[#] > n, Fibonacci[#-1], Fibonacci[#]] &
 [Round[Log[Sqrt[5] n]/(-Log[2] + Log[1 + Sqrt[5]])]]
```

Проверим:

```
Map[near, {70, 100}]
{55, 89}
```

Но поскольку, в первую очередь, нас интересует не само число, а номер ближайшего числа Фибоначчи, не превосходящего n, то определим функцию:

```
nearN[n_] := If[Fibonacci[#] > n, #-1, #] &
    [Round[Log[Sqrt[5] n] / (-Log[2] + Log[1 + Sqrt[5]])]]
```

Проверим:

```
Map[nearN, {70, 100}]
```
{10, 11}

Теперь мы собираемся создать список чисел Фибоначчи, которые являются слагаемыми в представлении (2) натурального числа n. Для этого удобно предварительно построить список пар вида

$$\{\{n,k_1\},\{n-F_{k_1},k_2\},...,\{n-F_{k_1}-F_{k_2}-\cdots-F_{k_{r-1}},k_r\},$$
$$\{n-F_{k_1}-F_{k_2}-\cdots-F_{k_{r-1}}-F_{k_r},\text{Nothing}\}\}.$$

Nothing представляет элемент в списке, который будет автоматически удален. Например:

```
{a, b, Nothing, c, d, Nothing}
```
{a, b, c, d}

Вспомогательная функция $g[r, k]$ из каждой пары этого списка выдает следующую пару, точнее

```
g[{r_, n_}] := Module[{d = r - Fibonacci[n]}, If[d > 0, {d, nearN[d]}, Nothing]]
```

В качестве примера возьмем $n = 100$. Имеем

```
near[100]
```
89
```
nearN[100]
```
11
```
g[{100, nearN[100]}]
```
{11, 6}
```
g[%]
```
{3, 4}
```
g[%]
```
Nothing

```
Plus@@Fibonacci[{11, 6, 4}]
100
```

Таким образом, $100 = F_{11} + F_6 + F_4 = 89 + 8 + 3$.

С помошью функции NestWhileList получаем нужный список:

```
NestWhileList[g, {#, nearN[#]}& [100], Not[SameQ[#, Nothing]]&]
{{100, 11}, {11, 6}, {3, 4}}
```

Поясним функцию SameQ. Функция SameQ[x, y] выдает True, если выражения x и y идентичны, в противном случае выдает False.

```
SameQ[x, y]
False
SameQ[x, x]
True
```

В отличие от функции SameQ, функция Equal (==) остается невычисленной, если символические аргументы не имеют значения.

```
x == y
x == y
```

Из полученного списка пар создадим список номеров чисел Фибоначчи, сумма которых дает число n. Для этого определим функцию:

```
h[n_] :=# [[2]]&/@NestWhileList[g, {#, nearN[#]}&[n], Not[SameQ[#,
Nothing]]&]
```

Как мы знаем, $10^9 = F_{44} + F_{42} + F_{37} + F_{34} + F_{29} + F_{27} + F_{25} + F_{23} + F_{17} + F_{13} + F_{11} + F_7 + F_5 + F_2$. Проверим функцию h:

```
h[10^9]
{44, 42, 37, 34, 29, 27, 25, 23, 17, 13, 11, 7, 5, 2}
```

Осталось получить список цифр числа n (это единицы и нули) в системе счисления Фибоначчи. Алгоритм прост:

1. Пусть r – значение функции $h[n]$. Тогда для числа n количество его фибоначчиевых цифр $len = h[[1]] - 1$.

2. Создадим список s, состоящий из нулей в количестве len штук.

3. В списке s в соответствии с номерами фибоначчиевых слагаемых нули заменим на единицы.

4. Преобразуем в списке s числа 1 и 0 на символы "1" и "0", соответственно.

5. Объединим полученные символы в одну строку.

Функция *fs*[*n*] выдает представление натурального числа *n* в фибоначчиевой системе счисления:

```
fs[n_] := Module[{r = h[n], len, s, i},
                 len = r[[1]] - 1; s = Table[0, {len}];
                 Do[If[MemberQ[r, i], s[[i - 1]] = 1, 0], {i, 2, len + 1}];
                 s = Reverse[s]; StringJoin[s /. {1 → "1", 0 → "0"}]
               ]
```

Получим представление чисел 100 и 10^9 в фибоначчиевой системе счисления.

```
fs[100]
```

1000010100

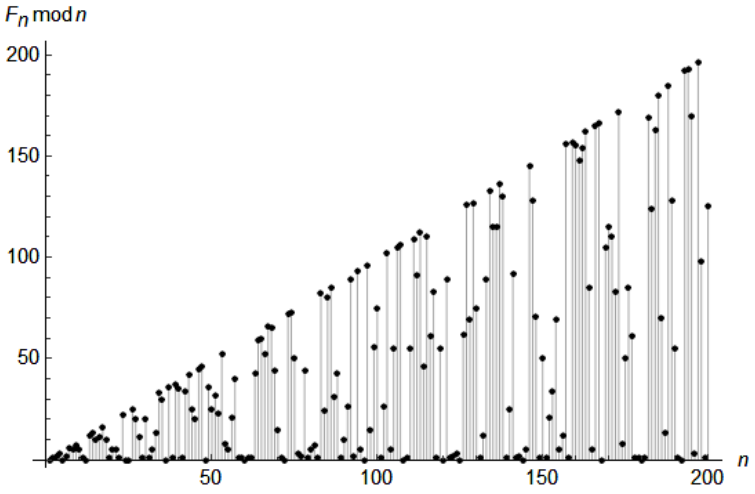
```
fs[10^9]
```

101000010010000101010100000100010100010100 1

§ 3. Последовательность Fibonacci(*n*) mod *n*

Последовательность F_n mod *n* (остаток от деления F_n на *n*) показывает замечательно сложное поведение. Рассмотрим график первых двести чисел этой последовательности.

```
ListPlot[Table[Mod[Fibonacci[n], n], {n, 200}],
         AxesLabel → {n, Mod[Subscript[F, n], n]}, Filling → Bottom]
```



Стивен Вольфрам [49] полагает, что возможно провести полный анализ поведения этой последовательности, например представив значение $F_n \bmod n$ в терминах стандартных теоретико-числовых функций от n. Он также считает, что возможное поведение последовательности описывается простой примитивной рекурсией.

Последовательность $F_n \bmod n$ представлена в on-line энциклопедии Слоана последовательностей целых чисел [24], но там отсутствует анализ ее поведения. В данном параграфе исследуются подпоследовательности вида $F_{q\times p} \bmod q\times p$, где q – фиксированное натуральное число, а p пробегает простые числа. Вычисляются начальные отрезки подпоследовательностей, высказываются гипотезы, проводятся эксперименты для проверки. Доказаны достаточные условия на q, при которых значения последовательности $F_{q\times p} \bmod q\times p$ лежат только на двух прямых. В частности, первые 13 значений q суть 1, 2, 5, 10, 12, 24, 25, 36, 48, 50, 60, 72, 96. Полученные результаты ранее опубликованы в [80].

Когда в последовательности $F_n \bmod n$ встречаются нули?

Получим числа $n < 200$ и обладающие свойством $F_n \bmod n = 0$:

```
Select[Range[200], Mod[Fibonacci[#], #]==0&]
```
{1, 5, 12, 24, 25, 36, 48, 60, 72, 96, 108, 120, 125, 144, 168, 180, 192}

Перечислим известные факты:

1. Числа $F(5^k) \bmod 5^k$ равны нулю для натурального k [11].

2. Числа $F(4\times3^k) \bmod 4\times3^k$ равны нулю для любого положительного целого k [12].

Но не только для $n = 5^k$ или $n = 4\times3^k$ числа $F_n \bmod n$ равны 0. Примеры смотрите ниже. Известен факт[36]: если $n > 1$ и $F_n \bmod n = 0$, то n делится на 5 или на 12.

Были рассмотрены первые 500 тысяч чисел Фибоначчи. Для их вычисления использовалось матричное тождество

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n .$$

При возведении в степень матрицы приведение по модулю n осуществлялось для каждого произведения. Полученный алгоритм имеет вычислительную сложность $O(\ln n)$. Приведем соответствующую программу вычисляющую $F_n \bmod m$:

[36] Последовательность A023172 (http://oeis.org/A023172) в [24].

```
fibMod[n_, m_] :=
  Module[{k = n, b = {{1, 0}, {0, 1}}, c = {{1, 1}, {1, 0}}},
    While[k ≠ 0, If[Mod[k, 2] == 0, k = k/2; c = Mod[c.c, m], k = k – 1;
      b = Mod[b.c, m]]];
    b[[2,1]]]
```

Отметим, что F_{500000} содержит 104 494 десятичных цифры:

```
IntegerLength[Fibonacci[500000]]
```
104494

Пусть R обозначает множество $\{F_n \mid n \le 500\,000, F_n \bmod n = 0\}$. Оказывается, это множество состоит из 2139 чисел

```
R = Parallelize[Select[Range[500000], fibMod[#, #] == 0&]];
Length[R]
```
2139

Какие числа не делятся на 12?

```
Select[R, Mod[#, 12] > 0&]
```
{1, 5, 25, 125, 625, 3125, 15625, 75025, 78125, 375125, 390625}

```
FactorInteger[%]
```
{{{1, 1}}, {{5, 1}}, {{5, 2}}, {{5, 3}}, {{5, 4}}, {{5, 5}}, {{5, 6}}, {{5, 2}, {3001, 1}}, {{5, 7}}, {{5, 3}, {3001, 1}}, {{5, 8}}}

Чисел вида $F(5^k)$ только девять, для n = 1, 5, 25, 125, 625, 3125, 15625, 78125, 390625. Два числа из R есть $F(5^2\times3001)$ и $F(5^3\times3001)$. Все остальные числа в R имеют вид F_{12k}.

Чисел вида $F(4\times3^k)$ только десять, для n = 12, 36, 108, 324, 972, 2916, 8748, 26244, 78732, 236196. Три числа F_{24}, F_{36} и F_{60} из R имеют вид F_{12p}, где p – простое число. Остальные числа из R имеют вид F_{12k}, где k – составное число.

```
R1 = Select[R, Mod[#, 12] == 0&];
Length[R1]
```
2128

```
Select[R1, IntegerQ[Log[3, #/4]]&]
```
{12, 36, 108, 324, 972, 2916, 8748, 26244, 78732, 236196}

```
Select[R1, PrimeQ[#/12]&]
```
{24, 36, 60}

Но обратное не верно. Среди первых 500 000 чисел Фибоначчи имеется 35182 числа, которые не принадлежат R, хотя номера их имеют вид

$12k$ и k – составное число.

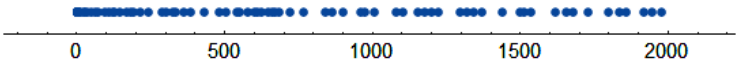
```
Length[Select[12 Select[Range[⌊500000/12⌋], Not[PrimeQ[#]]&],
 fibMod[#, #] > 0&]]
```
35182

Первые три таких числа $F(12\times21)$, $F(12\times22)$ и $F(12\times26)$.

```
Select[12 Select[Range[30], Not[PrimeQ[#]]&], fibMod[#, #] > 0&]
```
{252, 264, 312}

Покажем распределение нулей $F_n \bmod n = 0$ при $n \le 2000$.

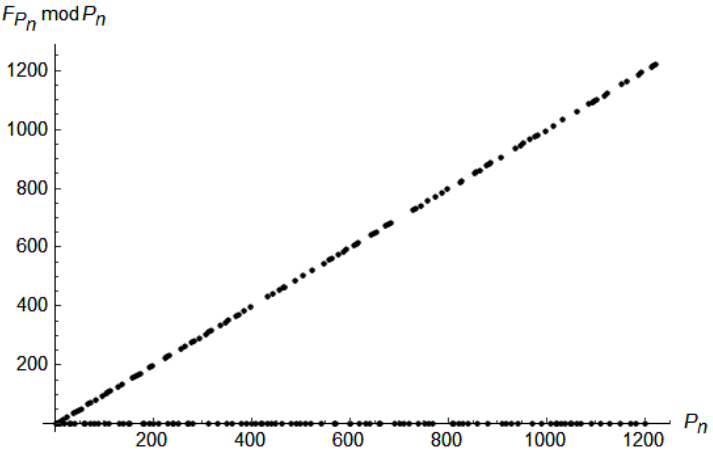
```
NumberLinePlot[Select[Table[{n, Mod[Fibonacci[n], n]}, {n, 1, 2000}],
 #[[2]] == 0&]]
```



Подпоследовательность $F_p \bmod p$, p – простое

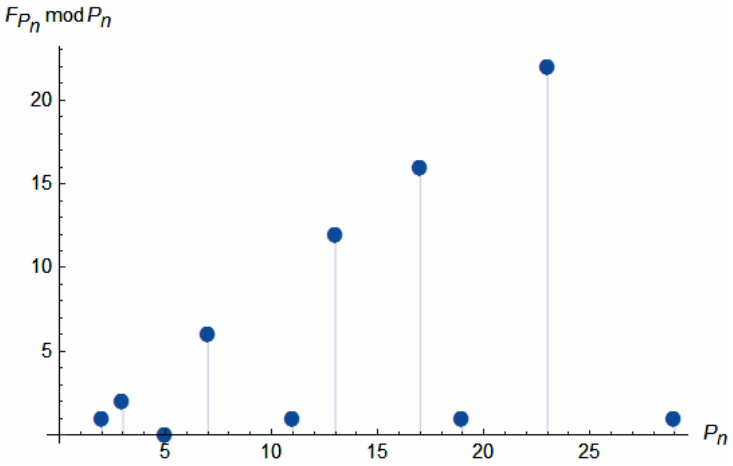
Рассмотрим числа Фибоначчи с номерами, которые являются простыми числами. Отложим на оси абсцисс первые 200 простых чисел P_n ($n = 1, 2, \ldots, 200$), а на оси ординат соответствующие значения $F(P_n) \bmod P_n$. Изобразим график полученных точек с координатами $(P_n, F(P_n) \bmod P_n)$.

```
ListPlot[Table[{Prime[n], Mod[Fibonacci[Prime[n]], Prime[n]]}, {n, 1, 200}],
 AxesLabel → {Subscript[P, n], Mod[Subscript[F, Subscript[P, n]],
 Subscript[P, n]]}]
```



Изучение графика приводит к предположению, что $F_p \bmod p$ может быть равно только 1 или $p - 1$. Более детальное изображение приводит к предположению, которое доказываем как теорему 6.

```
ListPlot[Table[{Prime[n], Mod[Fibonacci[Prime[n]], Prime[n]]}, {n, 1, 10}],
AxesLabel → {Subscript[P, n], Mod[Subscript[F, Subscript[P, n]],
Subscript[P, n]]}]
```



Теорема 6.

a. Если простое p имеет вид $5t \pm 1$, то $F_p \bmod p = 1$.

b. Если простое p имеет вид $5t \pm 2$, то $F_p \bmod p = p - 1$.

Для доказательства теоремы нам потребуется лемма.

Лемма 1 [65, с. 53–54].

a. Если простое p имеет вид $5t \pm 1$, то p делит $5^{(p-1)/2} - 1$.

b. Если простое p имеет вид $5t \pm 2$, то p делит $5^{(p-1)/2} + 1$.

***Доказательство* теоремы 6**. Если $p = 2$, то $F_p = F_2 = 1 \equiv -1 \pmod 2$. Поэтому в дальнейшем предполагаем, что p нечетно.

По формуле Бине мы имеем

$$F_p = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^p - \left(\frac{1-\sqrt{5}}{2}\right)^p\right) = \frac{1}{\sqrt{5}}\frac{1}{2^p}\sum_{k=0}^{p}\left(C_p^k\left((\sqrt{5})^k - (-1)^k(\sqrt{5})^k\right)\right) =$$

$$= \frac{1}{\sqrt{5}}\frac{1}{2^p}\sum_{\substack{k=1,\\ k \text{ нечетно}}}^{p}\left(2C_p^k\left((\sqrt{5})^k\right) = \frac{1}{2^{p-1}}\sum_{\substack{k=1,\\ k \text{ нечетно}}}^{p}\left(C_p^k\left((\sqrt{5})^{k-1}\right) = \frac{1}{2^{p-1}}\sum_{\substack{k=1,\\ k \text{ нечетно}}}^{p} C_p^k 5^{(k-1)/2} =\right.\right.$$

$$= \frac{1}{2^{p-1}}(C_p^1 + C_p^3 5 + C_p^5 5^2 + \ldots + C_p^{p-2} 5^{(p-3)/2} + C_p^p 5^{(p-1)/2}).$$

Получаем

$$2^{p-1} F_p = C_p^1 + C_p^3 5 + C_p^5 5^2 + \ldots + C_p^{p-2} 5^{(p-3)/2} + C_p^p 5^{(p-1)/2}.$$

Так как

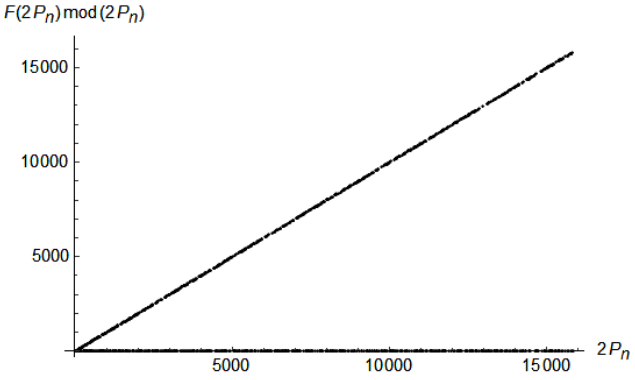
$$C_p^k = \frac{p!}{k!(p-k)!}$$

и, если $0 < k < p$, то p в числителе C_p^k не сокращается. Поэтому все биномиальные $C_p^p = 1$. Отсюда имеем $2^{p-1} F_p \equiv 5^{(p-1)/2} (\mathrm{mod}\ p)$. По малой теореме Ферма $2^{p-1} \equiv 1 \pmod{p}$. Поэтому, используя лемму 1, получаем утверждение теоремы. ■

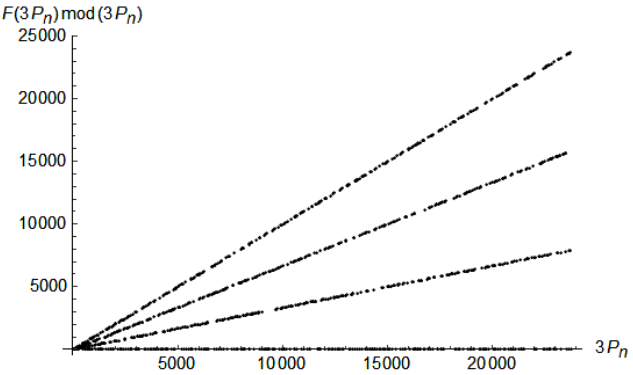
Подпоследовательности $F_{q\times p}$ mod $q\times p$, p – простое

Будем изучать подпоследовательности $F_{q\times p}$ mod $q\times p$, где q – фиксированное натуральное число, а p пробегает простые числа. Меняя q, мы имеем разное поведение подпоследовательностей. На следующих графиках, создаваемых функцией $g[q]$, мы видим типичные ситуации для некоторых небольших q. На оси *абцисс* размещаем числа $q\times P_n$, а на оси *ординат* – соответствующие значения $F(q\times P_n)$ mod P_n.

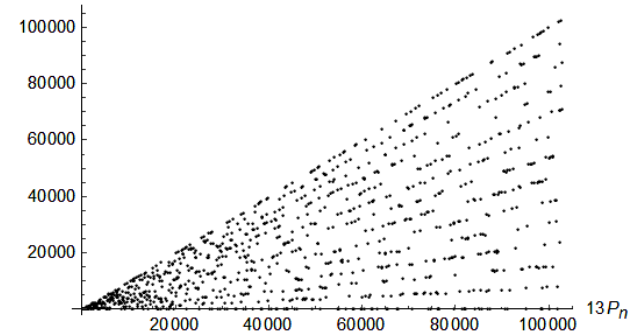
```
g[q_] :=
ListPlot[Table[{q Prime[n], Mod[Fibonacci[q Prime[n]], q Prime[n]]},
{n, 4, 1000}],
AxesLabel → {q Subscript[P, n], Mod[F[q Subscript[P, n]], q
Subscript[P, n]]]
g[2]
```



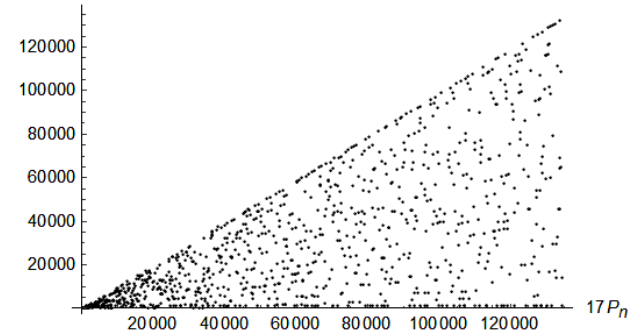
g[3]


$F(3P_n) \bmod (3P_n)$
25000
20000
15000
10000
5000
5000
10000
15000
20000
$3P_n$

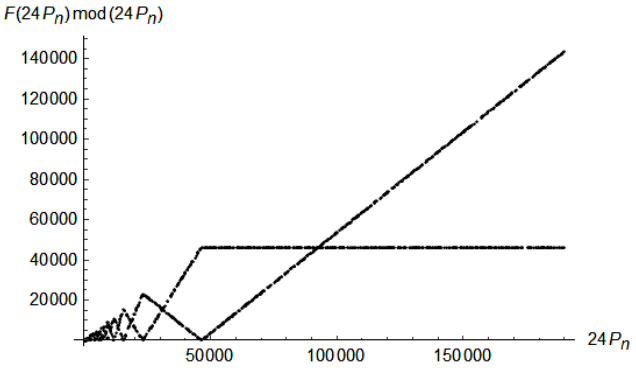
g[13]


$F(13P_n) \bmod (13P_n)$
100000
80000
60000
40000
20000
20000
40000
60000
80000
100000
$13P_n$

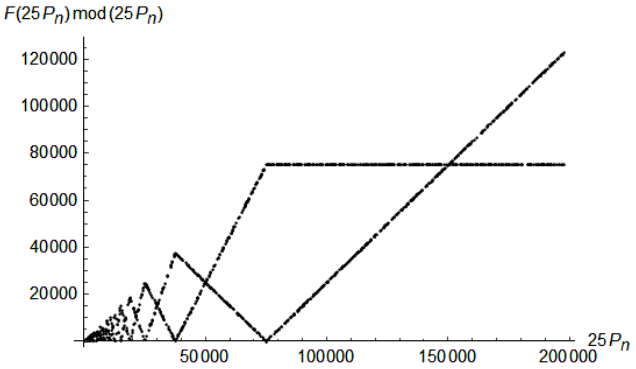
g[17]


$F(17P_n) \bmod (17P_n)$
120000
100000
80000
60000
40000
20000
20000
40000
60000
80000
100000
120000
$17P_n$

g24]



g[25]



На графиках для $q = 2$, 24 и 25 значения ($q \times p$, $F_{q\times p} \bmod q\times p$) находятся на двух прямых, начиная с некоторого $q\times p_0$. Такие случаи изучены. Для $n = 3p$ и $n = 4p$ высказаны только гипотезы без доказательств.

Ситуации, в которых поведение подпоследовательности $F_{q\times p} \bmod q\times p$ описывается двумя прямыми, первоначально изучались по отдельности для разных значений q. Затем полученные результаты были обобщены. Мы же сейчас начнем с доказательства общих результатов, а потом приведем следствия.

Теорема 7. Пусть q – такое натуральное число, что выполнено условие

$$\forall p > \max(5, F_q) \Rightarrow F_{q\times p} \equiv \pm F_q \pmod{q}. \qquad (3)$$

(Сравнение в (3) выполнено одновременно как для положительных, так и для отрицательных значений F_q).

Тогда

a. Если простое p имеет вид $5t \pm 1$, то $F_{q\times p} \bmod q\times p = F_q$.

b. Если простое p имеет вид $5t \pm 2$, то $F_{q\times p} \bmod q\times p = q\times p - F_q$.

Для доказательства теоремы 7 нам потребуется две леммы.

Лемма 2 (теорема Десмонда). Пусть p – простое число. Тогда для любого натурального n имеем

$$F_{p\times n} \equiv F_n F_p \pmod{p}.$$

Оригинал изложен в статье [5]. Доступное доказательство см. в [11].

Лемма 3. Если сравнение $a \equiv b$ имеет место по нескольким разным модулям, то оно имеет место и по модулю, равному наименьшему общему кратному этих модулей.

Доказательство. Если $a \equiv b \pmod{m}$ и $a \equiv b \pmod{n}$, то $a - b$ делится на m и n, значит, $a - b$ делится на наименьшее общее кратное m и n. ■

Доказательство **теоремы 7**. Теорема Десмонда дает $F_{p\times q} \equiv F_q F_p \pmod{p}$. Поэтому по теореме 6

*a*1. Если простое p имеет вид $5t \pm 1$, то $F_{p\times q} \equiv F_q \pmod{p}$.

*b*1. Если простое p имеет вид $5t \pm 2$, то $F_{p\times q} \equiv -F_q \pmod{p}$.

Если $p > \max(5, F_q)$, то по условию (3) имеем

*a*2. Если простое p имеет вид $5t \pm 1$, то $F_{p\times q} \equiv F_q \pmod{q}$.

*b*2. Если простое p имеет вид $5t \pm 2$, то $F_{q\times p} \equiv -F_q \pmod{q}$.

Так как p – простое и $p > q$, то наименьшее общее кратное p и q равно qp. Поэтому по лемме 3 имеем

*a*3. Если простое p имеет вид $5t \pm 1$, то $F_{q\times p} \equiv F_q \pmod{q\times p}$.

*b*3. Если простое p имеет вид $5t \pm 2$, то $F_{q\times p} \equiv -F_q \pmod{q\times p}$.

Следовательно, для любого $p > \max(5, F_q)$ выполнено утверждение теоремы. ■

Чтобы получить следствие из теоремы, нам необходима следующая лемма.

Лемма 4 [65; 67, с. 326]. Если $n > 2$, то F_m делится на F_n тогда и только тогда, когда m делится на n.

Следствие 1. Пусть $F_q \bmod q = 0$. Тогда для каждого $p > \max(5, F_q)$ имеем

a. Если простое p имеет вид $5t \pm 1$, то $F_{q\times p} \bmod q\times p = F_q$

b. Если простое p имеет вид $5t \pm 2$, то $F_{q\times p} \bmod q\times p = q\times p - F_q$

Доказательство. Так как q делит $\pm F_q$ и, по лемме 4, F_q делит $F_{q\times p}$, то $F_{q\times p} \equiv \pm F_q \pmod{q}$. Получили условие (3). ■

Как мы знаем, среди первых 500 000 значений q условие следствия 1 выполнено для 2139 значений q. Приведем несколько первых пар таких

значений (q, F_q); вторая компонента пары ограничивает снизу начальное значение p, начиная с которого поведение последовательности $F_{q\times p} \bmod q\times p$ описывается двумя прямыми.

```
{#, Fibonacci[#]}& /@ Select[Range[200], Divisible[Fibonacci[#], #]&]
// TableForm
```

```
1       1
5       5
12      144
24      46 368
25      75 025
36      14 930 352
48      4 807 526 976
60      1 548 008 755 920
72      498 454 011 879 264
96      51 680 708 854 858 323 072
108     16 641 027 750 620 563 662 096
120     5 358 359 254 990 966 640 871 840
125     59 425 114 757 512 643 212 875 125
144     555 565 404 224 292 694 404 015 791 808
168     57 602 132 235 424 755 886 206 198 685 365 216
180     18 547 707 689 471 986 212 190 138 521 399 707 760
192     5 972 304 273 877 744 135 569 338 397 692 020 533 504
```

Следствие 2. Пусть $q = 2t$, причем t не делится на 3 и t делитель F_q. Тогда для каждого $p > \max(5, F_q)$ имеем

a. Если простое p имеет вид $5t \pm 1$, то $F_{q\times p} \bmod q\times p = F_q$.

b. Если простое p имеет вид $5t \pm 2$, то $F_{q\times p} \bmod q\times p = q\times p - F_q$.

Доказательство. Покажем, что F_q нечетно. Действительно, если $2 = F_3$ делит F_{2t}, то, по лемме 4, имеем 3 – делитель t, что невозможно. Поэтому $F_q \equiv \pm t \pmod q$. Так как p – простое и $p > q$, то pq не делится на 3. По лемме 4, $F_{q\times p}$ нечетно, но $F_{q\times p}$ делится на F_q. Отсюда следует, что $F_{q\times p}$ делится на t. Поэтому получаем $F_{q\times p} \equiv \pm t \pmod q$. Так как $F_q \equiv \pm t \pmod q$ и $F_{q\times p} \equiv \pm t \pmod q$, то $F_{q\times p} \equiv \pm F_q \pmod q$. Получили условие (3). ■

Первые шесть значений q, для которых выполнено условие следствия 2, есть 2, 10, 50, 110, 250 и 550:

```
2 Select[Range[300], Divisible[Fibonacci[2#], #] && (Mod[#, 3] > 0)&]
```

```
{2, 10, 50, 110, 250, 550}
```

Покажем и соответствующие значения F_q:

```
{#, Fibonacci[#]}& /@  Most[%] // TableForm
2       1
10      55
50      12 586 269 025
110     43 566 776 258 854 844 738 105
250     7 896 325 826 131 730 509 282 738 943 634 332 893 686 268 675 876 375
```

Не показываем F_{550}, так как это число содержит 115 десятичных цифр:

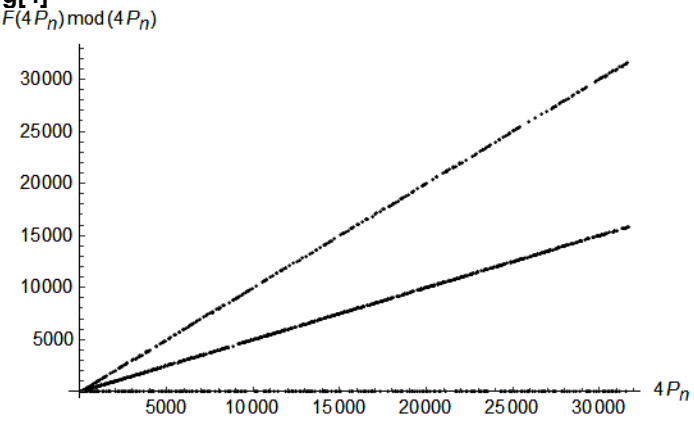
```
IntegerLength[Fibonacci[550]]
115
```

Предположения без доказательств

Теорема 9 вместе со следствиями дает достаточные условия того, когда поведение подпоследовательностей $F_{q\times p} \bmod q\times p$ описывается двумя прямыми. В более сложных ситуациях удалось сформулировать только гипотезы, и то только в двух простейших случаях.

Рассмотрим поведение последовательности $F_{4p} \bmod 4p$.

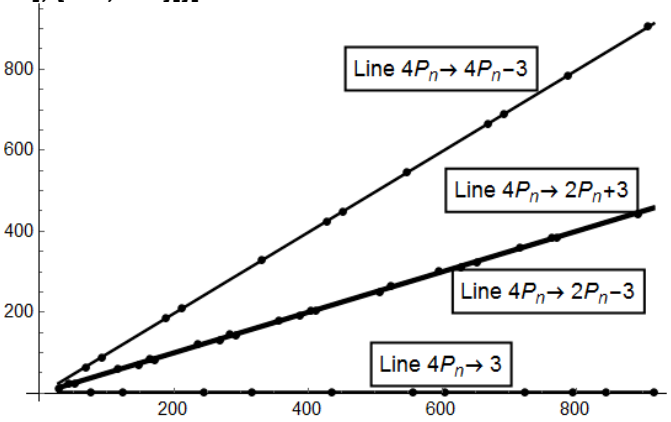
```
g[4]
```



Кажется, что значения лежат на трех прямых, но увеличивая масштаб, обнаруживаем, что имеется четыре прямые.

```
Show[ListPlot[Table[{4Prime[n], 4Prime[n] – 3}, {n, 4, 50}],
Joined → True],
ListPlot[Table[{4Prime[n], 2Prime[n] + 3}, {n, 4, 50}], Joined → True],
```

```
ListPlot[Table[{4Prime[n], 2Prime[n] – 3}, {n, 4, 50}], Joined → True],
ListPlot[Table[{4Prime[n], 3}, {n, 4, 50}], Joined → True],
ListPlot[Table[{4Prime[n], Mod[Fibonacci[4Prime[n]], 4 Prime[n]]},
{n, 4, 50}]],
Epilog → { Inset[Framed[Style["Line 4Subscript[P, n] → 3", 12],
{600, 70}],
Inset[Framed[Style["Line 4Subscript[P, n] → 2Subscript[P, n] +
3",12], {750, 500}],
Inset[Framed[Style["Line 4Subscript[P, n] → 2Subscript[P, n] – 3",
12], {760, 250}],
Inset[Framed[Style["Line 4Subscript[P, n] → 4Subscript[P, n] – 3",
12], {600, 800}]}]
```



С помощью Mathematica были высказаны гипотезы (проверены для первой тысячи простых чисел).

Возможно, для всех простых чисел справедливы следующие утверждения:

Гипотеза 1.

Если $p \equiv 1$ или $p \equiv 19$ по модулю 15, то $F_{4p} \bmod 4p = 3$.

Если $p \equiv 7$ или $p \equiv 13$ по модулю 15, то $F_{4p} \bmod 4p = 2p - 3$.

Если $p \equiv 2$ или $p \equiv 8$ по модулю 15, то $F_{4p} \bmod 4p = 4p - 3$.

Если $p \equiv 11$ или $p \equiv 14$ по модулю 15, то $F_{4p} \bmod 4p = 2p + 3$.

Гипотеза 2.

Если $p \equiv 1$ или $p \equiv 19$ по модулю 30, то $F_{4p} \bmod 4p = 3$.

Если $p \equiv 7$ или $p \equiv 13$ по модулю 30, то $F_{4p} \bmod 4p = 2p - 3$.

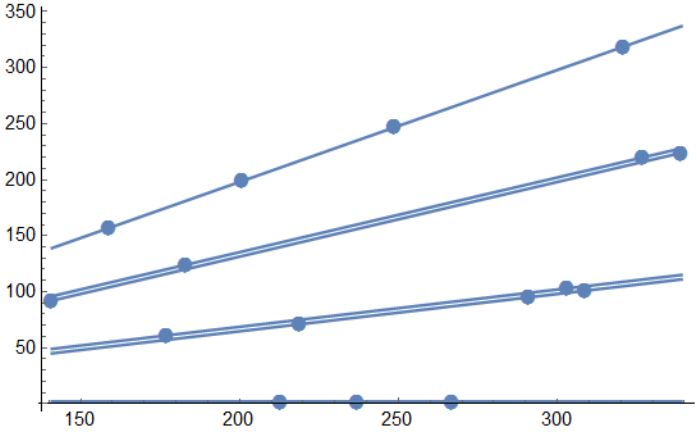
Если $p \equiv 17$ или $p \equiv 23$ по модулю 30, то $F_{4p} \bmod 4p = 4p - 3$.

Если $p \equiv 11$ или $p \equiv 29$ по модулю 30, то $F_{4p} \bmod 4p = 2p + 3$.

Поиск подходящего модуля m осуществлялся среди тех m, у которых значение функции Эйлера $\varphi(m)$ равно 4 или 8. Для $m \le 10\,000$ только два значения 15 и 30 оказались подходящими.

Изучалось также поведение последовательности $F_{3p} \bmod 3p$. Если рассмотреть распределение значений последовательности в диапазоне изменения простых чисел от $p_{15} = 47$ до $p_{30} = 113$, то наблюдается следующая картина.

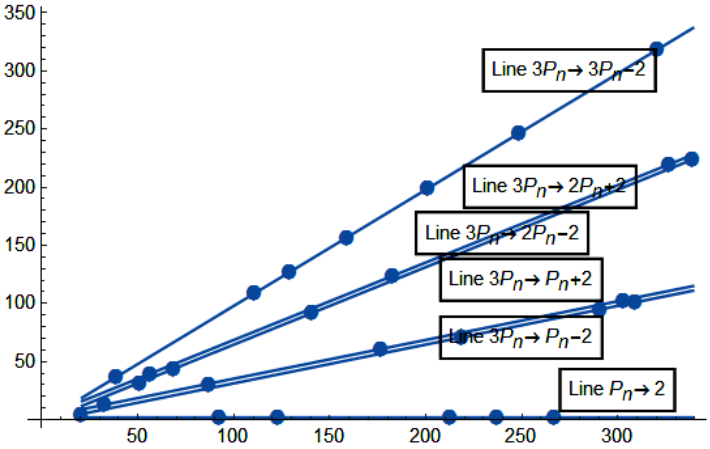
```
Show[ListPlot[Table[{3Prime[n], 3Prime[n] – 2}, {n, 15, 30},
Joined → True],
ListPlot[Table[{3Prime[n], 2Prime[n] + 2}, {n, 15, 30}], Joined → True],
ListPlot[Table[{3Prime[n], 2Prime[n] – 2}, {n, 15, 30}], Joined → True],
ListPlot[Table[{3Prime[n], Prime[n] + 2}, {n, 15, 30}], Joined → True],
ListPlot[Table[{3Prime[n], Prime[n] – 2}, {n, 15, 30}], Joined → True],
ListPlot[Table[{3Prime[n], 2}, {n, 15, 30}], Joined → True],
ListPlot[Table[{3Prime[n], Mod[Fibonacci[3Prime[n]], 3Prime[n]]},
{n, 15, 30}],
PlotMarkers → {Automatic, Medium}]]
```



То, что точки $(3p,\ F_{3p} \bmod 3p)$ последовательности $F_{3p} \bmod 3p$ располагаются на шести прямых, проверено для первой тысячи простых чисел.

```
Show[ListPlot[Table[{3Prime[n], 3Prime[n] – 2}, {n, 4, 30}],
Joined → True],
ListPlot[Table[{3Prime[n], 2Prime[n] + 2}, {n, 4, 30}], Joined → True],
ListPlot[Table[{3Prime[n], 2Prime[n] – 2}, {n, 4, 30}], Joined → True],
ListPlot[Table[{3Prime[n], Prime[n] + 2}, {n, 4, 30}], Joined → True],
```

```
ListPlot[Table[{3Prime[n], Prime[n] – 2}, {n, 4, 30}], Joined → True],
ListPlot[Table[{3Prime[n], 2}, {n, 4, 30}], Joined → True],
ListPlot[Table[{3Prime[n], Mod[Fibonacci[3Prime[n]], 3Prime[n]]},
{n, 4, 30}], PlotMarkers → {Automatic, Medium}],
Epilog → {
Inset[Framed[Style["Line 3Pn → 2",10]],  {300, 25}],
Inset[Framed[Style["Line 3Pn  → 2Pn  + 2", 10]], {265, 200}],
Inset[Framed[Style["Line 3Pn → 2Pn – 2", 10]], {240, 160}],
Inset[Framed[Style["Line 3Pn  → Pn – 2", 10]], {250, 70}],
Inset[Framed[Style["Line 3Pn → Pn + 2", 10]], {250, 120}],
Inset[Framed[Style["Line 3Pn  →  3Pn  – 2", 10]], {275, 300}]}]
```



Попытки сделать и в этом случае похожие предположения, как и в случае $q = 4$ оказались безуспешны. Поскольку прямых шесть, то подходящий модуль m отыскивался среди тех m, для которых функция Эйлера $\varphi(m)$ была бы кратна 6, точнее, $\varphi(m)$ должно быть равно 6, 12, 18 или 24. Были просмотрены все $m \le 10000$, но подходящий модуль m не обнаружен.

Глава 6

ПРИМИТИВНЫЕ КОРНИ

§ 1. Порядок числа по модулю

Возьмем натуральное число x и рассмотрим его степени x, x^2, x^3, … . Так как для них имеется лишь конечное число значений по модулю m, то найдется два таких целых числа k и n, что $x^k \equiv x^n \pmod m$, где $k > n$. Если x и m взаимно просты, то обе части сравнения можно сократить на x^n. Получаем $x^{k-n} \equiv 1 \pmod m$.

Определение. Наименьшее число h, такое, что $x^h \equiv 1 \pmod m$, называется **порядком числа *x* по модулю *m***. Порядок числа x по модулю m традиционно обозначается $\mathrm{ord}_m(x)$. Часто $\mathrm{ord}_m(x)$ называют также **мультипликативным порядком**.

Рассмотрим примеры. Порядок 9 по модулю 17 равен 8:

```
Mod[9^Range[20], 17]
{9, 13, 15, 16, 8, 4, 2,1, 9, 13, 15, 16, 8, 4, 2, 1, 9, 13, 15, 16}
```

Порядок 9 по модулю 10 равен 2:

```
Mod[9^Range[8], 10]
{9, 1, 9, 1, 9, 1, 9, 1}
```

Порядок числа x не существует, если $gcd(x, m) > 1$:

```
Mod[9^Range[8], 6]
{3, 3, 3, 3, 3, 3, 3, 3}

Mod[24^Range[15], 21]
{3, 9, 6, 18, 12, 15, 3, 9, 6, 18, 12, 15, 3, 9, 6, 18, 12, 15, 3, 9, 6}
```

По теореме Эйлера, если x и m взаимно просты, то $x^{\varphi(m)} \equiv 1 \pmod m$, где $\varphi(m)$ –функция Эйлера, измеряющая количество чисел в ряду 1, 2, 3, …, n, взаимно простых с m. Поэтому порядок $\mathrm{ord}_m(x) \le \varphi(m)$.

Мы можем детально рассмотреть ситуации с порядками чисел, фиксируя модуль и перебирая основания и степени. Для этого полезно написать функцию, создающую необходимые таблицы.

```
g[m_] :=
  Grid[{{"x^n"} ~ Join ~ Range[m]} ~ Join ~
    Table[{x} ~ Join ~ Mod[x^Range[m], m],
      {x, Select[Range[m], GCD[#, m]==1&]}],
    Frame → {{True}, {True}}]
```

Поясним программу. Создается список подсписков чисел, который с помощью функции Grid превращается в прямоугольную таблицу с оглавлением (самый левый столбец и самая верхняя строк); остальных столбцов – m и строк – $\varphi(m)$. Встроенная функция Join в инфиксной записи производит конкатенацию (соединение) списков в один список. Подробности смотрите в справочной системе языка Wolfram.

Рассмотрим $m = 13$, тогда $\varphi(13) = 12$. В таблице столбец с надписью «*x*^*n*» содержит различные x, а строка-заголовок содержит последовательные показатели степени.

```
g[13]
```

x^n	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	3	6	12	11	9	5	10	7	1	2
3	3	9	1	3	9	1	3	9	1	3	9	1	3
4	4	3	12	9	10	1	4	3	12	9	10	1	4
5	5	12	8	1	5	12	8	1	5	12	8	1	5
6	6	10	8	9	2	12	7	3	5	4	11	1	6
7	7	10	5	9	11	12	6	3	8	4	2	1	7
8	8	12	5	1	8	12	5	1	8	12	5	1	8
9	9	3	1	9	3	1	9	3	1	9	3	1	9
10	10	9	12	3	4	1	10	9	12	3	4	1	10
11	11	4	5	3	7	12	2	9	8	10	6	1	11
12	12	1	12	1	12	1	12	1	12	1	12	1	12

Как мы видим, порядки чисел $\mathrm{ord}_{13}(2) = \mathrm{ord}_{13}(6) = \mathrm{ord}_{13}(7) = \mathrm{ord}_{13}(11) = 12$; $\mathrm{ord}_{13}(4) = \mathrm{ord}_{13}(10) = 6$; $\mathrm{ord}_{13}(5) = \mathrm{ord}_{13}(8) = 4$; $\mathrm{ord}_{13}(3) = \mathrm{ord}_{13}(9) = 3$, $\mathrm{ord}_{13}(12) = 2$ и $\mathrm{ord}_{13}(1) = 1$.

Возьмем $m = 12$, тогда $\varphi(12) = 4$. Порядок существует только для четырех чисел, меньших 12: $\mathrm{ord}_{12}(5) = \mathrm{ord}_{12}(7) = \mathrm{ord}_{12}(11) = 2$ и $\mathrm{ord}_{12}(1) = 1$.

g[12]

x^n	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	1	1	1	1	1
5	5	1	5	1	5	1	5	1	5	1	5	1
7	7	1	7	1	7	1	7	1	7	1	7	1
11	11	1	11	1	11	1	11	1	11	1	11	1

Теорема 1. Пусть $\gcd(a, m) = 1$. Тогда $a^x \equiv 1 \pmod m$ достаточно и необходимо для того, чтобы $\mathrm{ord}_m(a)$ был делителем x.

Доказательство. Не указывая модуля, предполагаем, что все порядки и сравнения далее имеют место для модуля m. Необходимость: $a^x = a^{\mathrm{ord}(a)\cdot k} = (a^{\mathrm{ord}(a)})^k \equiv 1$. Достаточность: пусть $a^x \equiv 1$ и, от противного, запишем x как $q\cdot\mathrm{ord}(a) + r$, где $0 \le r < \mathrm{ord}(a)$. Тогда $1 \equiv a^x = a^{q\cdot\mathrm{ord}(a)\,+\,r} = a^{q\cdot\mathrm{ord}(a)}a^r \equiv a^r$, что противоречит минимальности $\mathrm{ord}(a)$. Следовательно, $r = 0$. ■

Следствие 1. Если $\gcd(a, m) = 1$, то $\mathrm{ord}_m(a)$ делит $\varphi(m)$. В частности, если p – простое число, не делящее a, то $\mathrm{ord}_p(a)$ делит $p - 1$.

Из доказанного сразу получаем алгоритм нахождения мультипликативного порядка $\mathrm{ord}_m(a)$. Перебираем в порядке возрастания все делители $\varphi(m)$ и первый делитель d, для которого $a^d \equiv 1 \pmod m$, есть $\mathrm{ord}_m(a)$.

В языке Wolfram $\mathrm{ord}_m(x)$ вычисляется функцией MultiplicativeOrder[x, m].

MultiplicativeOrder[9, 13]
3

Функция MultiplicativeOrder[x, m] остается невычисленной, если $\mathrm{ord}_m(x)$ не существует.

MultiplicativeOrder[9, 12]
MultiplicativeOrder[9, 12]

Можно использовать традиционное математическое обозначение мультипликативного порядка:

MultiplicativeOrder[*n*, *m*]//TraditionalForm
$\mathrm{ord}_m(n)$

ord$_9$(5)
6

Следствие 2. Если $\gcd(a, m) = 1$, то $a^i \equiv a^j \pmod m$ тогда и только тогда, когда $i \equiv j \pmod{\mathrm{ord}_m(a)}$.

Доказательство. Предположим $a^i \equiv a^j \pmod m$ и $i \geq j$. Так как $\gcd(a, m) = 1$, то $\gcd(a^j, m) = 1$. Поэтому существует $(a^j)^{-1}$ по модулю m. Имеем $a^i \cdot (a^j)^{-1} \equiv a^j \cdot (a^j)^{-1} \equiv 1$ по модулю m, т.е. $a^i \cdot a^{-j} \equiv a^{i-j} \equiv 1 \pmod m$. По теореме, $\mathrm{ord}_m(a) \mid i - j$, т.е. $i \equiv j \pmod{\mathrm{ord}_m(a)}$.

Обратно, пусть $i \equiv j \pmod{\mathrm{ord}_m(a)}$, где $i \geq j$. Тогда, если обозначить через $t = \mathrm{ord}_m(a)$, то для некоторого целого k выполнено $i = j + kt$. Поэтому

$$a^i = a^{j+kt} = a^j \cdot (a^t)^k \equiv a^j \cdot 1^k \equiv a^j \pmod m. \blacksquare$$

Следующая теорема говорит, как связаны мультипликативные порядки различных степеней числа a по одному и тому же модулю m.

Теорема 2. Если $\mathrm{ord}_m(a) = t$, то для любого u имеем $\mathrm{ord}_m(a^u) = t / \gcd(t, u)$.

Доказательство. Не указывая модуля, предполагаем, что все порядки и сравнения далее имеют место для модуля m. Пусть $s = \mathrm{ord}(a^u)$, $d = \gcd(t, u)$, $t_1 = t/d$ и $u_1 = u/d$. Тогда t_1 и u_1 – взаимно простые. Имеем

$$(a^u)^{t_1} = a^{u\,t/d} = (a^t)^{u_1} \equiv 1,$$

т.е. s делит t_1. Наша цель – равенство $s = t_1$, поэтому нужно доказать, что t_1 делит s. Имеем $a^{us} = (a^u)^s \equiv 1$, поэтому t делит us, т.е $t_1 d$ делит $u_1 ds$. Так что t_1 делит $u_1 s$. Поскольку $\gcd(t_1, u_1) = 1$, то t_1 делит s. ■

Следствие. Пусть $\mathrm{ord}_m(a) = t$ и u – целое положительное число. Тогда $\mathrm{ord}_m(a^u) = t$ в том и только том случае, когда $\gcd(t, u) = 1$.

В языке Wolfram можно вычислить так называемый **обобщенный мультипликативный порядок** числа x по модулю m. Вызов функции MultiplicativeOrder[x, m, {r_1, r_2, …}] выдает наименьшее число h, такое, что $x^h \equiv r_i \pmod m$ для какого-то r_i из списка {r_1, r_2, …}. Таким образом, $\mathrm{ord}_m(x)$ также вычисляется при вызове MultiplicativeOrder[x, m, {1}].

В качестве примера рассмотрим степени 10 по модулю 17:

```
Mod[10^Range[16], 17]
{10, 15, 14, 4, 6, 9, 5, 16, 7, 2, 3, 13, 11, 8, 12, 1}
```

Имеем

```
MultiplicativeOrder[10, 17, {5, 7, 1, 13}]
7
MultiplicativeOrder[10, 17, {1, 2, 3, 4}]
4
```

Используя обобщенный мультипликативный порядок, можно решать сравнения с неизвестным показателем; например, требуется найти x, для которого $117^x \equiv 15 \pmod{73}$.

```
MultiplicativeOrder[117, 73, {15}]
65
```

Проверка:

```
Mod[117^65, 73]
15
```

§ 2. Примитивные корни – образующие в U(m)

Как уже отмечалось, $\mathrm{ord}_m(x) \le \varphi(m)$. Естественно поставить вопрос, когда мультипликативный порядок по модулю m равен $\varphi(m)$? В этой связи имеется следующее определение.

Определение. Число a называется **примитивным корнем** для m (называют также **первообразным корнем**) если $\mathrm{ord}_m(a) = \varphi(m)$.

Например, как ранее было уже замечено, $\mathrm{ord}_{13}(2) = \mathrm{ord}_{13}(6) = \mathrm{ord}_{13}(7) = \mathrm{ord}_{13}(11) = 12 = \varphi(13)$, поэтому 2, 6, 7 и 11 являются примитивными корнями для 13, а остальные вычеты по модулю 13 не являются примитивными корнями.

Если $m = 12$, то только четыре числа имеют порядки $\mathrm{ord}_{12}(5) = \mathrm{ord}_{12}(7) = \mathrm{ord}_{12}(11) = 2$ и $\mathrm{ord}_{12}(1) = 1$, а так как $\varphi(m) = 4$, то примитивный корень для 12 не существует.

Пусть $\mathbf{U}(m)$ – множество натуральных чисел $\{a \mid 1 \le a \le m$ и $\gcd(a, m) = 1\}$. Как мы знаем, любой элемент $\mathbf{U}(m)$, рассматриваемый как соответствующий класс вычетов по модулю m, является обратимым (делителем единицы) в кольце $\mathbf{Z}_m$. Нетрудно убедиться, что $\mathbf{U}(m)$ является группой относительно операции умножения классов вычетов в $\mathbf{Z}_m$.

Следующая теорема подчеркивают центральную роль примитивных корней: степени с показателями 1, 2, …, $\varphi(m)$ любого примитивного корня для m образуют мультипликативную группу $\mathbf{U}(m)$ обратимых элементов в $\mathbf{Z}_m$.

Теорема 3. Если g есть примитивный корень для m, то каждое целое число, взаимно простое с m, сравнимо по модулю m с g^i для какого-то показателя $i = 1, 2, \ldots, \varphi(m)$. Другими словами, множество $\{g^i \mid i = 1, 2, \ldots, \varphi(m)\}$ есть приведенная система вычетов по модулю m.

Доказательство. Числа g, g^2, g^3, …, $g^{\varphi(m)}$ все различны по модулю m, так как, если $x^i \equiv x^j \pmod m$, где $i < j$, то $x^{j-i} \equiv 1 \pmod m$, что противоречит тому, что g – примитивный корень. Все степени g взаимно просты с m и их количество равно $\varphi(m)$, поэтому g, g^2, g^3, …, $g^{\varphi(m)}$ исчерпывают все элементы $\mathbf{U}(m)$. ■

Например, 2 и 11 являются примитивными корнями для 13, поэтому их степени образуют группу $\mathbf{U}(13)$:

```
Mod[2^Range[12], 13]
{2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7, 1}
Mod[11^Range[12], 13]
{11, 4, 5, 3, 7, 12, 2, 9, 8, 10, 6, 1}
```

Возьмем m =14, тогда имеем

```
EulerPhi[14]
6
Select[Range[13],  MultiplicativeOrder[#, 14] = = 6 &]
{3, 5}
Mod[3^Range[6], 14]
{3, 9, 13, 11, 5, 1}
Mod[5^Range[6], 14]
{5, 11, 13, 9, 3, 1}
```

Таким образом, мы видим, что любой примитивный корень для m является генератором (образующей) для циклической группы $\mathbf{U}(m)$. Если же примитивный корень для m не существует, то группа не является циклической. В качестве примера возьмем m = 12:

```
EulerPhi[12]
4
Select[Range[12], GCD[#, 12]==1&]
{1, 5, 7, 11}
```

Ни один из элементов $\mathbf{U}(12)$ не генерирует всю группу:

```
Mod[#^Range[4], 12]& /@ {1, 5, 7, 11}//Column
{1, 1, 1, 1}
{5, 1, 5, 1}
{7, 1, 7, 1}
{11, 1, 11, 1}
```

Переформулировка следствия из теоремы 2 говорит, в каком случае степени примитивного корня являются также примитивными корнями.

Предложение. Пусть g – примитивный корень по модулю m. Тогда g^u есть примитивный корень по модулю m тогда и только тогда, когда $(\varphi(m), u) = 1$.

Теорема 4. Если m имеет примитивный корень, то количество примитивных корней для m равно $\varphi(\varphi(m))$.

Доказательство. Если g – примитивный корень, то множество $\{g^u \mid u = 1, 2, \ldots, \varphi(m)\}$ есть приведенная система вычетов по модулю m. Остается только проверить, сколько чисел из этого множества являются примитивными корнями. В силу предыдущего предложения, $\mathrm{ord}_m(g^u) = \varphi(m)$ каждый раз, когда $(\varphi(m), u) = 1$, а количество таких чисел u равно $\varphi(\varphi(m))$. ■

Например, если $m = 13$, то $\varphi(\varphi(13)) = 4$ и примитивными корнями для 13 являются 2, 6, 7 и 11.

Примитивный корень числа m (но не обязательно наименьший для составного числа m) может быть вычислен в языке Wolfram с использованием функции PrimitiveRoot[m]. Вызов PrimitiveRoot[m] выдает генератор группы $\mathbf{U}(m)$ делителей единицы кольца $\mathbf{Z}_m$.

```
PrimitiveRoot[37]
2
```

Проверим, что это 2 – действительно генератор:

```
Sort[Mod[2^#, 37]& /@ Range[EulerPhi[37]]]
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36}
```

А если m – составное?

```
EulerPhi[10]
4
PrimitiveRoot[10]
7
Mod[7^#, 10]& /@ Range[4]
{7, 9, 3, 1}
```

Но наименьший примитивный корень для 10 есть 3:

```
Mod[3^#,10]&/@Range[4]
{3, 9, 7, 1}
```

Вызов PrimitiveRoot[m, k] выдает наименьший примитивный корень m, больший или равный k. Вызов PrimitiveRoot[m, 1] выдает наименьший примитивный корень m.

```
PrimitiveRoot[10, 1]
3

PrimitiveRoot[37, 3]
5
```

Для простых m функции PrimitiveRoot[m, 1] и PrimitiveRoot[m] выдают один и тот же примитивный корень.

```
AllTrue[(PrimitiveRoot[#] == PrimitiveRoot[#, 1]&) /@ Table[Prime[n],
{n, 1000}], #&]
True
```

Функция PrimitiveRootList[m] вычисляет список всех примитивных корней числа m, расположенных в порядке возрастания:

```
PrimitiveRootList[37]
{2, 5, 13, 15, 17, 18, 19, 20, 22, 24, 32, 35}

PrimitiveRootList[10]
{3, 7}
```

Как мы знаем, количество примитивных корней для m равно $\varphi(\varphi(m))$:

```
EulerPhi[EulerPhi[37]]
12

EulerPhi[EulerPhi[10]]
2
```

Следующие теоремы, принадлежащие Гауссу, говорят о том, в каких случаях натуральные числа имеют примитивные корни.

Теорема 5. Любое простое число имеет примитивный корень.

(Простое доказательство см. в [71, с. 60, 61]).

Теорема 6. Все случаи, когда существуют примитивные корни по модулю m, превосходящие 1, суть $m = 2, 4, p^n, 2p^n$, p – простое нечетное число и $n > 0$.

Доказательство этой теоремы см., например, в [64, с. 148–152].

Теорема 7. Функция $\mathrm{ord}_m(x)$ мультипликативна, т.е. для любых a и b, для которых $\mathrm{ord}_m(a)$ и $\mathrm{ord}_m(b)$ существуют, из взаимной простоты a и b следует

$$\mathrm{ord}_m(a) \cdot \mathrm{ord}_m(b) = \mathrm{ord}_m(a \cdot b).$$

Доказательство. Пусть $\mathrm{ord}_m(a) = l$ и $\mathrm{ord}_m(b) = k$. Тогда

$$(ab)^{lk} \equiv (a^l)^k (b^k)^l \equiv 1^k 1^l \equiv 1 (\mathrm{mod}\, m).$$

В силу теоремы 1, мы доказали, что порядок ab является делителем lk. Докажем, что $\mathrm{ord}_m(ab)$ не является собственным делителем lk. Рассмотрим от противного. Предположим, что $\mathrm{ord}_m(ab) = l_1 k_1$, где l_1 – делитель l, а k_1 – делитель k. Тогда

$$a^{l_1 k_1} b^{l_1 k_1} \equiv 1 (\mathrm{mod}\, m).$$

Возведем обе части этого сравнения в степень l_2, где $l_1 l_2 = l$. Так как $a^l \equiv 1 \pmod m$, то мы получаем $b^{lk_1} \equiv 1 (\mathrm{mod}\, m)$. Из этого следует, что lk_1 кратно $\mathrm{ord}_m(b) = k$. Но l взаимно просто с k, значит k_1 кратно k; будучи также делителем k, k_1должно быть равно k. Аналогично $l_1 = l$ и, таким образом, $\mathrm{ord}_m(a \cdot b) = \mathrm{ord}_m(a) \cdot \mathrm{ord}_m(b)$. ■

Задача 1. Пусть a' – обратный элемент a по модулю m. Как связаны $\mathrm{ord}_m(a)$ и $\mathrm{ord}_m(a')$?

Рассмотрим сначала примеры. Возьмем простой модуль $m = 11$ и сравним $\mathrm{ord}_m(a)$ и $\mathrm{ord}_m(a')$ для для всех a из диапазона $1 \le a \le 10$.

```
m = 11;
{MultiplicativeOrder[#, m],
MultiplicativeOrder[ModularInverse[#, m], m]}& /@
    Range[10]
```

{{1, 1}, {10, 10}, {5, 5}, {5, 5}, {5, 5}, {10, 10}, {10, 10}, {10, 10}, {5, 5}, {2, 2}}

Теперь возьмем составной модуль $m = 12$ и найдем аналогичные мультипликативные порядки для тех чисел a из диапазона $1 \le a \le 10$, для которых существует обратный элемент по модулю 12.

```
m = 12;
{MultiplicativeOrder[#, m],
MultiplicativeOrder[ModularInverse[#, m], m]}& /@
   Select[Range[10], GCD[#, m] == 1&]
```

{{1, 1}, {2,2}, {2,2}}

Естественно предположить, что $\mathrm{ord}_m(a) = \mathrm{ord}_m(a')$ для любых m и a. Докажем это. Равенство $\mathrm{ord}_m(a) = \mathrm{ord}_m(a')$ следует из того, что $a^t \equiv 1 \pmod m$ тогда и только тогда, когда $(a')^t \equiv 1 \pmod m$. Действительно, предположим $a^t \equiv 1 \pmod m$. Тогда $(a')^t \equiv (a')^t\, a^t\, a^t \equiv ((a'\, a)^t\, a^t \equiv \equiv 1^t \cdot 1 \equiv 1 \pmod m$. Обратное показывается аналогично.

Задача 2. Пусть $(a, m) = 1$. Для каких значений m выполнено $\mathrm{ord}_m(a) = m - 1$? Проделаем вычислительные эксперименты, выскажем гипотезу и докажем.

Решение. Определим функцию $f[m]$, которая выдает список чисел a, для которых выполнено $\mathrm{ord}_m(a) = m - 1$:

```
f[m_] := Select[Range[m – 1], GCD[#, m] == 1&&
MultiplicativeOrder[#, m] == m – 1&]
```

Выясним, для каких значений m соответствующие списки чисел a не пусты:

```
{#, f[#]}& /@ Select[Range[30], f[#] ≠ {}&] // Column
{2, {1}}
{3, {2}}
{5, {2, 3}}
{7, {3, 5}}
{11, {2, 6, 7, 8}}
{13, {2, 6, 7, 11}}
{17, {3, 5, 6, 7, 10, 11, 12, 14}}
{19, {2, 3, 10, 13, 14, 15}}
{23, {5, 7, 10, 11, 14, 15, 17, 19, 20, 21}}
{29, {2, 3, 8, 10, 11, 14, 15, 18, 19, 21, 26, 27}}
```

Гипотеза очевидна – m должно быть простым числом.

Доказательство. От противного. Пусть m – составное число. Тогда $\varphi(m) < m - 1$. Так как $\mathrm{ord}_m(a) \mid \varphi(m)$, то $\mathrm{ord}_m(a) < m - 1$. Полученное противоречие завершает доказательство.

Задача 3. Пусть $m = a^n - 1$, где $a > 1$ и n – натуральные числа. Изучим, какие значения принимает $\mathrm{ord}_m(a)$.

Вычислим $\mathrm{ord}_m(a)$ для случайных пар небольших чисел (a, n). Сначала создадим список s из 10 псевдослучайных пар (a, n).

```
s2 = Table[RandomInteger[{2, 10}, 2], 10]
{{4, 9}, {10, 5}, {4, 3}, {4, 8}, {4, 7}, {2, 9}, {9, 10}, {9, 9}, {5, 3}, {5, 5}}
```

Теперь из пар (a, n) сделаем тройки (a, n, m).

```
s3 = {#[[1]], #[[2]], #[[1]]^#[[2]] – 1}& /@ s2
{{4, 9, 262143}, {10, 5, 99999}, {4, 3, 63}, {4, 8, 65535}, {4, 7, 16383},
{2, 9, 511}, {9, 10, 3486784400}, {9, 9, 387420488}, {5, 3, 124}, {5, 5,
3124}}
```

И, наконец, получим список четверок $(a, n, m, \mathrm{ord}_m(a))$:

```
s4 = {#[[1]], #[[2]], #[[3]], MultiplicativeOrder[#[[1]], #[[3]]]}& /@ s3
{{4, 9, 262143, 9}, {10, 5, 99999, 5}, {4, 3, 63, 3}, {4, 8, 65535, 8},
{4, 7, 16383, 7}, {2, 9, 511, 9}, {9, 10, 3486784400, 10}, {9, 9,
387420488, 9}, {5, 3, 124, 3}, {5, 5, 3124, 5}}
```

Мы видим в нашем примере, что $\mathrm{ord}_m(a) = n$. Докажем, что это справедливо в общем случае. Заметим, что $a^t < m = a^n - 1$ для любого $1 \le t < n$. Следовательно, a^t не сравнимо с 1 по модулю m при $t < n$. Так как $a^n \equiv 1 \pmod m$, то $\mathrm{ord}_m(a) = n$. И поскольку $\mathrm{ord}_m(a)$ делит $\varphi(m)$, то также имеем $n \mid \varphi(m)$.

Задача 4. Пусть g есть примитивный корень по модулю m и g' – обратный элемент g по модулю m. Является ли g' также примитивным корнем m?

Проделаем вычислительные эксперименты и выскажем гипотезу. Используя теоремы 5 и 6, легко получить тест, который для натурального числа $m > 1$ определяет, существует ли примитивный корень по модулю m:

```
primitiveRootExistQ[m_] := Module[{fs = FactorInteger[m]},
        Which[m == 1 || m == 2 || m == 4, True,
                Length[fs] == 1 && fs[[1, 1]] > 2, True,
                Length[fs] == 2 && fs[[1]] == {2, 1}, True,
                True, False]]
```

Далее напишем программу, которая для данного натурального числа m и примитивных корней по модулю m отвечает на поставленный вопрос. Предварительно, определим модули m, не превосходящие 20, для которых существуют примитивные корни.

```
ms = Select[Range[20], primitiveRootExistQ]
{2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 17, 18, 19}
```

Возьмем для примера $m = 17$. Вычислим список всех примитивных корней по модулю m:

```
rs = PrimitiveRootList[17]
{3, 5, 6, 7, 10, 11, 12, 14}
```

Для примитивного корня $g = 10$ вычисляем обратный элемент:

```
Solve[10 x == 1, {x}, Modulus → 17][[1, 1, 2]]
12
```

Индексация [[1, 1, 2]] необходима, так как функция Solve выдает сложное выражение:

```
Solve[10 x == 1, {x}, Modulus → 17]
{{x →12}}
```

Как мы видим, обратный элемент для 10 – число 12 – также является примитивным корнем. Будет ли это справедливо для всех примитивных корней по модулю 17? Значение следующего выражения говорит, что для модуля 17 обратные элементы для каждого примивного корня сами являются примитивными корнями (в данном случае функция MemberQ проверяет, входит ли найденный обратный элемент в список *rs*):

```
{#, MemberQ[rs, Solve[# x == 1, {x}, Modulus → 17][[1,1,2]]]}& /@ rs
{{3, True}, {5, True}, {6, True}, {7, True}, {10, True}, {11, True}, {12,
True}, {14, True}}
```

Теперь определим функцию для проверки ситуации для произвольного модуля m, имеющего примитивные корни:

```
f[m_] := Module[{rs = PrimitiveRootList[m]},
{#, MemberQ[rs, Solve[# x == 1, {x}, Modulus → m][[1, 1, 2]]]}& /@ rs]
```

Вызов функции *f* для других значений 5 и 13 подсказывает гипотезу, что обратный элемент для примитивного корня является примитивным корнем для любого модуля.

```
f [5]
{{2, True}, {3, True}}

f[13]
{{2, True}, {6, True}, {7, True}, {11, True}}
```

Проверим гипотезу для всех подходящих модулей $m \le 1000$. Для этого переопределим функцию f так, чтобы ее значение было булевской величиной:

```
f[m_] := Module[{rs = PrimitiveRootList[m]},
AllTrue[MemberQ[rs, Solve[# x == 1, {x},
        Modulus → m][[1, 1, 2]]]& /@ rs, #&]]
f /@ {5, 14, 17}
{True, True, True}
```

И, наконец, следующее вычисление показывает, что гипотеза справедлива для всех модулей, не превосходящих 1000.

```
AllTrue[f /@ Select[Range[2, 1000], primitiveRootExistQ], #&]
True
```

Для доказательства при произвольном m воспользуемся утверждением задачи 1. Пусть g есть примитивный корень по модулю m и g' – обратный элемент g по модулю m. Имеем из решения задачи 1, что $\mathrm{ord}_m(g) = \mathrm{ord}_m(g')$. Так как $\mathrm{ord}_m(g) = \varphi(m)$, поэтому $\mathrm{ord}_m(g') = \varphi(m)$, следовательно, g' – примитивный корень по модулю m.

§ 3. Периодические десятичные дроби

Рациональное число есть число вида r/s, где r и s – целые. Все рациональные числа в десятичной системе счисления представляются периодической десятичной дробью (в частном случае период состоит из цифры 0). И наоборот, любая периодическая десятичная дробь представляет рациональное число. Мы рассмотрим связь между десятичными периодическими дробями и примитивными корнями.

Теорема 8. Каждое рациональное число имеет десятичное представление, которое состоит из какой-то начальной части (**предпериода**) и следуемой за ней периодической части.

Доказательство. Приведем только «эскиз» доказательства. Сделать этот «эскиз» полным доказательством будет хорошим упражнением. Если r/s – рациональное число, то представим, что мы выполняем деление в столбик, как это мы делали в школе. На каждом шаге у нас был остаток, меньший s. Поэтому после конечного числа шагов остаток должен повторяться. Это означает, что цифры десятичного представления отношения начнут повторяться с некоторой позиции.

Доказательство в обратную сторону. Предположим, что

$$x = m.abc\ldots defg\ldots hefg\ldots hefg\ldots h\ldots,$$

где m – целое число, предпериод $abc\ldots d$ имеет длину u и периодическая часть $efg\ldots h$ имеет длину q. Используя формулу для суммы геометрической прогрессии, получаем

$$x = m + \frac{abc\ldots d}{10^u} + \frac{\frac{\overline{efg\ldots h}}{10^{u+q}}}{1-\frac{1}{10^q}} = m + \frac{1}{10^u}\left(abc\ldots d + \frac{\overline{efg\ldots h}}{10^q - 1}\right).$$

Правая часть равенства есть сумма рациональных чисел, поэтому x – рациональное число. ■

Язык Wolfram имеет две функции для работы с десятичными представлениями рациональных чисел. Функция RealDigits для рационального числа x с периодом в десятичном представлении выдает список цифр в виде $\{\{a_1, a_2, \ldots, a_u\},\{b_1, b_2,\ldots, b_q\}, n\}$, где $a_1, a_2, \ldots, a_u$ – цифры предпериода и $b_1, b_2,\ldots, b_q$ – цифры периодической части. Если целая часть у числа x имеется, то n равно количеству цифр в целой части. Если целая часть отсутствует, то n не больше нуля и информирует о количестве цифр 0, с которых начинается дробная часть x в десятичном представлении. Функция FromDigits является обратной для RealDigits.

Небольшой код позволит показать сразу несколько ситуаций:

```
N[{123456/13, 123/13, 12/13, 1/13, 1/17}, 20] // Column
9496.6153846153846154
9.4615384615384615385
0.92307692307692307692
0.076923076923076923077
0.058823529411764705882
```

```
xs = {#, RealDigits[#]}& /@ {123456/13, 123/13, 12/13, 1/13, 1/17};
xs//Grid
```

$123456/13$	$\{\{9,4,9,6,\{6,1,5,3,8,4\}\},4\}$
$123/13$	$\{\{9,\{4,6,1,5,3,8\}\},1\}$
$12/13$	$\{\{\{9,2,3,0,7,6\}\},0\}$
$1/13$	$\{\{\{7,6,9,2,3,0\}\},-1\}$
$1/17$	$\{\{\{5,8,8,2,3,5,2,9,4,1,1,7,6,4,7,0\}\},-1\}$

Мы замечаем, что если знаменатель 13, то длина периода равна 6, а для знаменателя 17 длина периода максимально возможная – 16 (по доказательству теоремы 8). Это приводит к вопросу, в каких случаях рациональное число r/s в десятичном представлении имеет длину периодической части равной $s - 1$?

Теорема 9 [64, с. 206; 98, с. 168–184]. Рациональное число r/s имеет длину десятичного периода равную $s - 1$ тогда и только тогда, когда s – простое число и 10 является примитивным корнем для s.

Проверим справедливость утверждения теоремы для дробей вида $1/s$, где $s \le 250$ (программа выдает список знаменателей).

```
Select[Range[2, 250], Length[RealDigits[1/#][[1, –1]]] = = # – 1&]
{7, 17, 19, 23, 29, 47, 59, 61, 97, 109, 113, 131, 149, 167, 179, 181, 193,
223, 229, 233}
```

И для сравнения найдем числа $s \le 250$, которые имеют 10 в качестве примитивного корня:

```
Select[Range[2, 250], MultiplicativeOrder[10, #] == # – 1&]
{7, 17, 19, 23, 29, 47, 59, 61, 97, 109, 113, 131, 149, 167, 179, 181, 193,
223, 229, 233}
```

Следующая теорема дает полный ответ о длинах предпериода и периода для рациональных дробей в десятичном представлении.

Теорема 10. Пусть $x = r/s$, где r и s – положительные целые и $r < s$. Запишем s в виде $2^a 5^b y$, где $\gcd(y, 10) = 1$. Тогда длина периодической части десятичного представления x есть $\mathrm{ord}_y(10)$ и длина предпериода равна наибольшему из чисел a и b.

Подобное утверждение справедливо и для других оснований системы счисления, отличных от 10 [22, теорема 12.4].

В справке языка Wolfram по поводу функции MultiplicativeOrder приведено определение функции для нахождения длины периода, основанное на применении FixedPoint:

```
digitCycleLength[r_Rational, b_Integer?Positive]:=
MultiplicativeOrder[b, FixedPoint[Quotient[#, GCD[#, b]]&,
Denominator[r]]]
```

Находим периоды дроби 1/7 в системах счисления по основанию 10 и 4 (RealDigits[x, b] дает список цифр x по основанию b):

```
digitCycleLength[1/7, 10]
6
```

N[1/7, 20]
0.14285714285714285714

***digitCycleLength*[1/7, 4]**
3

Вызов функции RealDigits[*x*, *b*] дает список цифр *x* по основанию *b*, а вызов с тремя аргументами RealDigits[*x*, *b*, *len*] дает список цифр длиной *len*.

RealDigits[1/7, 4]
{{{2, 1, 0}}, –1}

RealDigits[1/7, 4, 10]
{{2, 1, 0, 2, 1, 0, 2, 1, 0, 2},-1}

Глава 7

ПРОИЗВОДЯЩИЕ ФУНКЦИИ

Производящие функции обеспечивают действенными средствами работы с последовательностями чисел. Используются методы преобразования бесконечных рядов, которые «порождают» эти последовательности.

§ 1. Формальные степенные ряды

Теория формальных рядов в полной общности описана в [63, с. 64–81]. Рассмотрим частный случай – множество формальных степенных рядов.

Назовем **формальным степенным рядом** (от одной переменной) формальное выражение вида

$$\sum_{k=0}^{\infty} a_k x^k = a_0 + a_1 x + a_2 x^2 + \dots,$$

где коэффициенты a_k принадлежат числовому кольцу или полю R (например, $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$).

Переменная x является формальной, и нас не будет интересовать сходимость такого ряда. Вполне возможно, что, только подставив $x = 0$, в результате получим некоторое число. Например, так обстоит дело для ряда

$$\sum_{k=0}^{\infty} k! x^k = 1 + 1x + 2x^2 + 6x^3 + 24x^4 + 120x^5 + \dots.$$

Множество формальных степенных рядов, определяемое числовым кольцом или полем R, будем обозначать как

$$R[[x]] = \{\sum_{k=0}^{\infty} a_k x^k \mid a_k \in R\}.$$

Формальные степенные ряды являются обобщением многочленов от одной переменной, и поэтому в степенном ряду может быть и конечное число членов, т.е. только конечное число коэффициентов $a_k \neq 0$. И так же как для многочленов, можно ввести на множестве $R[[x]]$ алгебраическую структуру коммутативного кольца с единицей.

Определим операции сложения и умножения формальных степенных рядов посредством равенств

$$\sum_{k=0}^{\infty} a_k x^k + \sum_{k=0}^{\infty} b_k x^k = \sum_{k=0}^{\infty} (a_k + b_k) x^k,$$

$$(\sum_{k=0}^{\infty} a_k x^k)(\sum_{k=0}^{\infty} b_k x^k) = \sum_{k=0}^{\infty} c_k x^k, \text{ где } c_k = \sum_{i=0}^{k} a_i b_{k-i} x^i.$$

Таким образом, чтобы найти коэффициент c_n при x^n в произведении двух рядов, нам не надо рассматривать эти ряды полностью, достаточно взять первые $n + 1$ членов в каждом из них.

Пример 1. Вычисляем на языке *Wolfram*:

```
Expand[(1 - x) Sum[x^k, {k, 0, 10}]]
```

$1 - x^{11}$

```
Expand[(Sum[x^k, {k, 0, 5}])(Sum[x^k, {k, 0, 5}])]
```

$1 + 2x + 3x^2 + 4x^3 + 5x^4 + 6x^5 + 5x^6 + 4x^7 + 3x^8 + 2x^9 + x^{10}$

Можно провести вычисления и с бесконечными рядами:

```
(1 - x) Sum[x^k, {k, 0, Infinity}]
```

1

```
Simplify[(Sum[x^k, {k, 0, Infinity}])(Sum[x^k, {k, 0, Infinity}]) == Sum[(k + 1) x^k, {k, 0, Infinity}]]
```

True

Эти вычисления подсказывают, что

$$(1-x)\sum_{k=0}^{\infty} x^k = 1,$$

$$(\sum_{k=0}^{\infty} x^k)(\sum_{k=0}^{\infty} x^k) = \sum_{k=0}^{\infty} (k+1)x^k.$$

Как обычно, элемент a некоторого кольца называется обратимым, если в этом кольце найдется элемент b, такой что $ab = 1$ (в этом случае пишем $b = 1/a$ или $b = a^{-1}$). Во множестве $R[[x]]$ роль нуля выполняет ряд, все коэффициенты которого равны нулю, а роль единицы – ряд, в котором первый коэффициент равен 1, а остальные равны 0.

Теорема 1. Множество $R[[x]]$ с операциями сложения и умножения и элементами 0 и 1 является коммутативным кольцом, в котором элемент $\sum_{k=0}^{\infty} a_k x^k$ обратим тогда и только тогда, когда $a_0 \neq 0$.

Доказательство. Из определения операций сложения и умножения формальных степенных рядов все свойства коммутативного кольца с единицей следуют очевидным образом, за исключением ассоциативности умножения и утверждения о необходимом и достаточном условии обратимости формального степенного ряда.

Докажем ассоциативность. Пусть даны ряды

$$u = a_0 + a_1 x + a_2 x^2 + \dots,$$
$$v = b_0 + b_1 x + b_2 x^2 + \dots,$$
$$w = c_0 + c_1 x + c_2 x^2 + \dots.$$

Коэффициенты ряда $(uv)w$ вычисляются по формуле

$$d_n = \sum_{k=0}^{n} (\sum_{i=0}^{k} a_i b_{k-i}) c_{n-k},$$

а ряда $u(vw)$ – по формуле

$$e_n = \sum_{k=0}^{n} a_k (\sum_{i=0}^{n-k} b_j c_{n-k-i}).$$

И окончательно получаем

$$d_n = \sum_{\substack{i,j,k \geq 0 \\ i+j+k=n}} a_i b_j c_k = e_n.$$

Теперь выясним, когда существует обратный элемент для ряда $a_0 + a_1x + a_2x^2 + \ldots$. Пусть $(a_0 + a_1x + a_2x^2 + \ldots)(u_0 + u_1x + u_2x^2 + \ldots) = 1$, тогда $a_0 u_0 = 1$. Следовательно, $u_0 \neq 0$. И получаем

$$\begin{cases} a_0u_0 = 1, \\ a_0u_1 + a_1u_0 = 0, \\ a_0u_2 + a_1u_1 + a_2u_0 = 0, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_0u_k + a_1u_{k-1} + \ldots + a_ku_0 = 0, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \end{cases}$$

Следовательно, $u_0 = 1/a_0, u_1 = -a_1u_0/a_0$, и так далее. Более точно, если определены значения u_0, u_1, …, u_{k-1}, то u_k находится посредством формулы

$$u_k = -\frac{a_1u_{k-1} + a_2u_{k-2} + \ldots + a_ku_0}{a_0}.$$

Таким образом, последовательно находятся все коэффициенты искомого ряда. ■

Следовательно, операция деления возможна для некоторых формальных степенных рядов. Приведем программу для вычисления обратного элемента (делителя единицы) в кольце $R[[x]]$.

Пусть *as* – список первых n коэффициентов формального степенного ряда A. Функция *unitDivisor*[*as*, *k*], где $k \le n$, выдает список первых k коэффициентов обратного формального степенного ряда $B = 1/A$:

```
unitDivisor[as_, 1] := {1/as[[1]]}
unitDivisor[as_, k_] /; Length[as] >= k > 1 := Module[{bs =
unitDivisor[as, k – 1]},
        Append[bs, –Dot[Take[as, {2, k}], Reverse[bs] / as[[1]]]]]
```

Пример 2. Найдем обратные ряды для рядов $\sum_{k=0}^{\infty} x^k$ и $1 - x$.

```
unitDivisor[{1, 1, 1, 1, 1, 1, 1, 1, 1, 1}, 10]
{1, –1, 0, 0, 0, 0, 0, 0, 0, 0}

unitDivisor[{1, –1, 0, 0, 0, 0, 0, 0, 0, 0}, 10]
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
```

Как мы уже видели в примере 1, эти два ряда взаимно-обратные.

Пример 3. Формальные степенные ряды $1 - x - x^2$ и $\sum_{k=0}^{\infty} F_k x^k$ взаимно-обратные (F_k – k-е число Фибоначчи, $F_0 = F_1 = 1$, и для $k > 1$ имеем $F_k = F_{k-1} + F_{k-2}$).

Находим первые 10 коэффициентов ряда $1/(1 - x - x^2)$:

```
unitDivisor[{1, –1, –1, 0, 0, 0, 0, 0, 0, 0}, 10]
{1, 1, 2, 3, 5, 8, 13, 21, 34, 55}
```

Получили первые 10 чисел Фибоначчи (в данном случае нумерация начинается с 0). Проверим, что если будем вычислять первые 500 коэффициентов формального степенного ряда $1/(1 - x - x^2)$, то также получим соответствующие числа Фибоначчи (встроенная функция Fibonacci нумерует числа Фибоначчи, начиная с 1).

```
unitDivisor[Join[{1, –1, –1}, Table[0, 497]], 500] ==
Fibonacci[Range[500]]
True
```

Нетрудно проверить, что *unitDivisor* дает нужный результат и для нахождения $1/\sum_{k=0}^{\infty} F_k x^k$:

```
unitDivisor[Fibonacci[Range[500]], 500] == Join[{1, –1, –1},
Table[0, 497]]
True
```

Докажем равенство $(1 - x - x^2)\sum_{k=1}^{\infty} F_k x^k = 1$, не используя Mathematica. Имеем

$$(1 - x - x^2)\sum_{k=0}^{\infty} F_k x^k = (1 - x - x^2)(F_0 + F_1 x + F_2 x^2 + \ldots) =$$
$$= F_0 + (F_1 - F_0)x + (F_2 - F_1 + F_0)x^2 + \ldots .$$

В силу начальных условий и рекуррентного соотношения для чисел Фибоначчи полученный ряд равен 1.

Кроме кольцевых операций (сложение и умножение) над формальными степенными рядами определяются и другие операции.

Формальные степенные ряды часто обозначаются одной буквой, где аргумент указывается в скобках:

$$f(x) = \sum_{k=0}^{\infty} a_k x^k.$$

Хотя вышенаписанное выглядит как функциональная запись, но это не означает, что $f(x)$ является «обычной» функцией. Не следует думать, что существует «значение $f(x_0)$ для аргумента x_0», так как переменная x является формальной и сумма ряда смысла не имеет.

Кроме кольцевых операций (сложение и умножение) над формальными степенными рядами определяются и другие операции.

Дифференцирование и интегрирование. Пусть дан формальный степенной ряд $f(x) = \sum_{k=0}^{\infty} a_k x^k$. **Производной** этого ряда называется формальный степенной ряд

$$f'(x) = \sum_{k=1}^{\infty} k a_k x^{k-1}.$$

Интегралом называется формальный степенной ряд

$$\int f(x) = \sum_{k=0}^{\infty} a_k \frac{x^{k+1}}{k+1}.$$

Очевидно, что для функций, представимых в виде степенных рядов, формула производной соответствует обычной. Формула для интеграла соответствует значению интеграла с переменным верхним пределом

$$\int f(x) = \int_0^x f(t)dt.$$

Композиция формальных степенных рядов. Эта операция является обобщением подстановки многочлена в другой многочлен вместо переменной. Возьмем два степенных ряда

$$f(x) = \sum_{k=0}^{\infty} a_k x^k \text{ и } g(x) = \sum_{k=0}^{\infty} b_k x^k,$$

где $g(0) = b_0 = 0$. Композицией формальных рядов $g(x)$ и $f(x)$ (другими словами «подстановкой ряда $g(x)$ в ряд $f(x)$») называется формальный

степенной ряд

$$f(g(x)) = \sum_{k=0}^{\infty} a_k (g(x))^k = \sum_{k=0}^{\infty} c_k x^k,$$

где $c_0 = a_0$ и для $n > 0$

$$c_n = \sum_{k \in \mathbb{N}, |i|=n} a_k b_{i_1} b_{i_2} \ldots b_{i_k}.$$

В последней сумме суммирование идет по $i \in \mathbb{N}^k$ и $|i| = i_1 + i_2 + \ldots + i_k$. Например, первые четыре слагаемые в $f(g(x))$ суть

$$a_0 + a_1 b_1 x + (a_1 b_2 + a_2 b_1^2) x^2 + (a_1 b_3 + 2 a_2 b_1 b_2 + a_3 b_1^3) x^3.$$

Обратите внимание, что подстановка ряда с $b_0 \neq 0$ неопределенна, ибо тогда пришлось бы суммировать бесконечный числовой ряд.

§ 2. Производящие функции в математике и в языке Wolfram

Определение производящей функции. Пусть a_0, a_1, a_2, … – произвольная бесконечная последовательность чисел (они могут быть натуральными, целыми, рациональными, вещественными или комплексными). Производящей функцией для этой последовательности будем называть формальный степенной ряд

$$\sum_{k=0}^{\infty} a_k x^k.$$

Производящая функция определяется и для конечной последовательности a_0, a_1, a_2, …, a_n. В этом случае в формальном степенном ряду все коэффициенты, начиная с a_{n+1}, полагают равным нулю.

В примере 3 мы получили равенство

$$\frac{1}{1 - x - x^2} = \sum_{k=0}^{\infty} F_k x^k.$$

Каков его смысл? Например, бессмысленно подставлять в него $x = 1$. Дело в том, что выражения

$$\frac{1}{1 - x - x^2} \quad \text{и} \quad \sum_{k=0}^{\infty} F_k x^k$$

равны как элементы кольца формальных степенных рядов. И в соответствии с предыдущим определением производящая функция последовательности чисел Фибоначчи 1, 1, 2, 3, 5, 8, 13, 21, … представляется как в виде формального степенного ряда, так и в виде функции $1/(1 - x - x^2)$. Про последнее выражение говорят, что оно является производящей функцией, записанной в «замкнутой» форме (содержит только элементарные функции и не содержит бесконечные суммы).

Теория производящих функций дает аппарат для создания производящих функций в замкнутой форме для различных последовательностей. И возможно, поэтому производящие функции находят разнообразные применения, в первую очередь в комбинаторике и теории чисел.

В книге [67, с. 353–417] описана теория производящих функций и множество примеров применения. По-видимому, все, что алгоритмизуемо в теории производящих функций, в частности разнообразные приемы преобразований производящих функций из указанной книги, реализовано в языке Wolfram. В дальнейшем это продемонстрируем в системе Mathematica на некоторых задачах.

Ранее было дано определение последовательности Фибоначчи и для этой последовательности была найдена производящая функция. Но числа последовательности Фибоначчи в языке Wolfram нумеруются другим образом:

```
Fibonacci[Range[0, 10]]
{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55}
```

В соответствии с этим определением требуется заново найти производящую функцию в замкнутой форме для последовательности Фибоначчи. Воспользуемся полученным ранее равенством:

$$\frac{1}{1-x-x^2} = 1x^0 + 1x^1 + 2x^2 + 3x^3 + 5x^4 + \ldots.$$

Умножая обе части на x, получаем

$$\frac{x}{1-x-x^2} = 0x^0 + 1x^1 + 1x^2 + 2x^3 + 3x^4 + 5x^5 + \ldots.$$

Таким образом, последовательность {Fibonacci[n] | n = 0, 1, 2, …} имеет производящую функцию $x/(1 - x - x^2)$ и в дальнейшем будет использоваться именно такая производящая функция для последовательности Фибоначчи.

Рассмотрим основные функции языка Wolfram, имеющие отношение к производящим функциям.

В языке Wolfram имеется возможность для последовательности (в общем случае бесконечной) получить производящую функцию в замкнутом виде. Основная функция для этой цели GeneratingFunction[*expr*, *n*, *x*], где выражение *expr* представляет *n*-й член последовательности. Найдем производящую функцию для последовательности Фибоначчи:

```
GeneratingFunction[Fibonacci[n], n, x]
```

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

Прежде чем рассмотреть другие примеры, необходимо познакомиться с функцией Series, которая в некотором смысле является «обратной» по своему действию функции GeneratingFunction.

Функция Series[f, {x, x_0, n}] языка Wolfram создает конечный степенной ряд

$$\sum_{k=0}^{n} f^{(k)}(x_0)\frac{(x-x_0)^k}{k!},$$

где $f^{(k)}(x_0)$ – значение k-й производной функции f в точке x_0. Когда f является производящей функцией последовательности, то Series обычно вызывается в виде Series[f, {x, 0, n}]. Например, для производящей функции последовательности Фибоначчи получаем

```
Series[-x/(-1+x+x^2), {x, 0, 10}]
```

$$x+x^2+2x^3+3x^4+5x^5+8x^6+13x^7+21x^8+34x^9+55x^{10}+O[x]^{11}$$

Результатом является не «обыкновенное» выражение, а объект SeriesData, об этом говорит последнее «слагаемое» O[x]. С помощью функции Normal получаем обыкновенный конечный степенной ряд

```
Normal[%]
```

$$x+x^2+2x^3+3x^4+5x^5+8x^6+13x^7+21x^8+34x^9+55x^{10}$$

Функция SeriesCoefficient[*series*, *n*] извлекает коэффициент при степени *n* из степенного ряда, созданного функцией Series.

```
f10 = Series[-x/(-1+x+x^2), {x, 0, 10}];
SeriesCoefficient[f10, 10]
```
55

Список всех коэффициентов до порядка n включительно или коэффициенты из нужного диапазона можно получить следующим образом:

```
SeriesCoefficient[f10, #]& /@ Range[0, 10]
```
{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55}

А теперь рассмотрим ряд других примеров.

```
GeneratingFunction[n, n, x]
```
$$\frac{x}{(-1+x)^2}$$

```
SeriesCoefficient[Series[%, {x, 0, 7}], #]& /@ Range[0, 7]
```
{0, 1, 2, 3, 4, 5, 6, 7}

```
GeneratingFunction [1/(n + 1), n, x]
```
$$-\frac{\text{Log}[1-x]}{x}$$

```
SeriesCoefficient[Series[%, {x, 0, 7}], #]& /@ Range[0, 7]
```
$$\left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}\right\}$$

Если в последнем вычислении GeneratingFunction в качестве n-го члена взять $1/n$, то производящая функция не может быть вычислена и исходное выражение возвращается невычисленным, так как для $n = 0$ получаем бесконечность.

```
GeneratingFunction[1/n, n, x]
```
GeneratingFunction: Infinite expression 1/0 encountered
GeneratingFunction[1/n, n, x]

```
GeneratingFunction[1/n!, n, x]
```
e^x

```
SeriesCoefficient[Series[%, {x, 0, 7}], #]& /@ Range[0, 7]
```
$$\left\{1, 1, \frac{1}{2}, \frac{1}{6}, \frac{1}{24}, \frac{1}{120}, \frac{1}{720}, \frac{1}{5040}\right\}$$

GeneratingFunction[*c*^*n*, *n*, *x*]

$$\frac{1}{1-cx}$$

SeriesCoefficient[Series[%, {*x*, 0, 7}], #]& /@ Range[0, 7]

$$\left\{1, c, c^2, c^3, c^4, c^5, c^6, c^7\right\}$$

GeneratingFunction[(*n* + 1)^(1/2), *n*, *x*]

$$\frac{\text{PolyLog}\left[-\frac{1}{2}, x\right]}{x}$$

В этих случаях в результате выдается значение полилогарифма – функции *Li*. Опреление функции [41]: $Li_n(z) = \sum_{k=1}^{\infty} z^k / k^n$. В языке *Wolfram* имеется соответствующая функция – PolyLog[*n*, *x*].

SeriesCoefficient[Series[%, {*x*, 0, 7}], #]& /@ Range[0, 7]

$$\left\{1, \sqrt{2}, \sqrt{3}, 2, \sqrt{5}, \sqrt{6}, \sqrt{7}, 2\sqrt{2}\right\}$$

GeneratingFunction[(*n* + 1)^(–1/2), *n*, *x*]

$$\frac{\text{PolyLog}\left[\frac{1}{2}, x\right]}{x}$$

SeriesCoefficient[Series[%, {*x*, 0, 7}], #]& /@ Range[0, 7]

$$\left\{1, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{3}}, \frac{1}{2}, \frac{1}{\sqrt{5}}, \frac{1}{\sqrt{6}}, \frac{1}{\sqrt{7}}, \frac{1}{2\sqrt{2}}\right\}$$

SeriesCoefficient[Series[%, {*x*, 0, 14}], #]& /@ Range[0, 14]

{0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0}

Для многих основных последовательностей соответствующие производящие функции уже определены в языке *Wolfram*. Но большее значение имеют алгоритмы, реализованные в языке, решающие задачи следующего вида:

Пусть даны последовательности и их производящие функции. Каким образом преобразования исходных последовательностей связаны с преобразованиями соответствующих производящих функций? Самый мощный на сегодняшний день метод работы с бесконечными последовательностями чисел – это преобразования производящих функций, которые «порождают» эти последовательности.

В следующих примерах $f[n]$ и $g[n]$ задают вид членов последовательностей $f = \{f_0, f_1, f_2, \ldots\}$ и $g = \{g_0, g_1, g_2, \ldots\}$ соответственно. Формируя из этих последовательностей новую последовательность, рассмотрим, как преобразуются производящие функции.

Линейное преобразование – общий член новой последовательности есть $\alpha f_n + \beta g_n$:

```
GeneratingFunction[αf[n] + βg[n], n, x]
```
α GeneratingFunction[$f[n]$, n, x] + β GeneratingFunction[$g[n]$, n, x]

Сдвиг исходной позиции вправо на *m* позиций (в начало последовательности добавляется m нулей):

```
GeneratingFunction[x^m f[n], n, x, Assumptions → m ∈ Integers && m ≥ 0]
```
x^m GeneratingFunction[$f[n]$, n, x]

Задание опции Assumptions $\rightarrow m \in$ Integers && $m \geq 0$ дает ограничение на m.

Проверим это для последовательности Фибоначчи при сдвиге на 4 позиции вправо:

```
x^4 GeneratingFunction[Fibonacci[n], n, x]
```
$$-\frac{x^5}{-1+x+x^2}$$

```
SeriesCoefficient[Series[%, {x, 0, 10}], #]& /@ Range[0, 10]
```
{0, 0, 0, 0, 0, 1, 1, 2, 3, 5, 8}

Как сдвинуть исходную позицию влево на *m* позиций (начало новой последовательности начинается с элемента $f[m]$)? К сожалению, Mathematica не может это показать в самом общем виде. Требуется явно задать или вид $f[n]$, или m. Покажем сначала, как меняется производящая функция, если в исходной последовательности удаляется m членов (m меняется в диапазоне от 1 до 5):

```
GeneratingFunction[f[n + #], n, x]& /@ Range[5] // TableForm
```

$$-\frac{f[0]}{x}+\frac{\text{GeneratingFunction}[f[n],n,x]}{x}$$

$$-\frac{f[0]}{x^2}-\frac{f[1]}{x}+\frac{\text{GeneratingFunction}[f[n],n,x]}{x^2}$$

$$-\frac{f[0]}{x^3}-\frac{f[1]}{x^2}-\frac{f[2]}{x}+\frac{\text{GeneratingFunction}[f[n],n,x]}{x^3}$$

$$-\frac{f[0]}{x^4}-\frac{f[1]}{x^3}-\frac{f[2]}{x^2}-\frac{f[3]}{x}+\frac{\text{GeneratingFunction}[f[n],n,x]}{x^4}$$

$$-\frac{f[0]}{x^5}-\frac{f[1]}{x^4}-\frac{f[2]}{x^3}-\frac{f[3]}{x^2}-\frac{f[4]}{x}+\frac{\text{GeneratingFunction}[f[n],n,x]}{x^5}$$

Пусть $f[n]$ есть Fibonacci$[n]$, тогда сдвигу влево на пять позиций последовательности Фибоначчи соответствует производящая функция, получаемая после упрощения:

```
-f[0]/x^5 - f[1]/x^4 - f[2]/x^3 - f[3]/x^2 - f[4]/x + GeneratingFunction[f[n], n, x]/x^5 /.
 f[n] → Fibonacci[n]
```

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

```
Together[%]
```

$$\frac{-5-3x}{-1+x+x^2}$$

Теперь m – произвольное целое неотрицательное, но исходная последовательность есть последовательность Фибоначчи:

```
GeneratingFunction[Fibonacci[n + m], n, x, Assumptions → m ∈
 Integers && m ≥ 0]
```

$$\frac{\left(-1-\sqrt{5}\right)^{-m}\left(2^{1-m}\left(1+\sqrt{5}\right)\left(4^m-\left(-6-2\sqrt{5}\right)^m\right)-2\left(2\left(-3-\sqrt{5}\right)^m+2^m\left(3+\sqrt{5}\right)\right)x\right)}{\sqrt{5}\left(1+\sqrt{5}+2x\right)\left(-2+x+\sqrt{5}x\right)}$$

Упростим для $m = 5$:

```
FullSimplify[%] /. m → 5
```

$$\frac{-\left(-3-\sqrt{5}\right)^5\left(1+\sqrt{5}+2x\right)+32\left(1+\sqrt{5}-\left(3+\sqrt{5}\right)x\right)}{\left(-1-\sqrt{5}\right)^5\left(5+\sqrt{5}\right)\left(-1+x+x^2\right)}$$

```
Apart[%]
```

$$\frac{-5-3x}{-1+x+x^2}$$

С помощью функции Series из производящей функции получим первые пять слагаемых ряда:

```
Series[%, {x, 0, 5}]
```

$$5+8x+13x^2+21x^3+34x^4+55x^5+O[x]^6$$

Преобразование последовательности f_n в $c^n f_n$ соответствует умножению аргумента производящей функции на c (f_n и $c^n f_n$ – общие члены последовательностей, c – константа):

```
Series[GeneratingFunction[f[n], n, c x], {x, 0, 5}]
```

GeneratingFunction$[f[n], n, 0] + c\,f[n]\,x + c^2 f[n]\,x^2 + c^3 f[n]\,x^3 + c^4 f[n]\,x^4 + O[x]^5$

Два примера:

```
GeneratingFunction[Fibonacci[n], n, c x]
```

$$-\frac{cx}{-1+cx+c^2x^2}$$

```
SeriesCoefficient[Series[%, {x, 0, 7}], #]& /@ Range[0, 7]
```

$$\left\{0, c, c^2, 2c^3, 3c^4, 5c^5, 8c^6, 13c^7\right\}$$

```
GeneratingFunction[1, n, (–1) x]
```

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

```
SeriesCoefficient[Series[%, {x, 0, 7}], #]& /@ Range[0, 7]
```

{1, –1, 1, –1, 1, –1, 1, –1}

Преобразование последовательности f_n в $(n + 1) f_{n+1}$ соответствует дифференцированию производящей функции (f_n и $(n + 1) f_{n+1}$ – общие члены последовательностей):

```
D[GeneratingFunction[f[n], n, x], x]
```
$$\text{GeneratingFunction}[(1+n) f[1+n], n, x]$$
```
GeneratingFunction[c, n, x]
```
$$\frac{c}{1-x}$$
```
SeriesCoefficient[Series[D[%, x], {x, 0, 7}], #]& /@ Range[0, 7]
```
$$\{c, 2c, 3c, 4c, 5c, 6c, 7c, 8c\}$$

Если после дифференцирование выполним сдвиг на одну позицию вправо, то получим производящую функцию последовательности с общим членом $n f_n$:

```
x D[GeneratingFunction[f[n], n, x], x]
```
$$x\ \text{GeneratingFunction}[(1+n) f[1+n], n, x]$$
```
x D[GeneratingFunction[c, n, x], x]
```
$$\frac{cx}{(1-x)^2}$$
```
SeriesCoefficient[Series[%, {x, 0, 7}], #]& /@ Range[0, 7]
```
$$\{0, c, 2c, 3c, 4c, 5c, 6c, 7c\}$$

Повторное дифференцирование позволяет умножить f_n на любой желаемый многочлен от n.

Деление элементов последовательности на n можно представить в виде формального степенного ряда только с ненулевым знаменателем для каждого члена. В качестве примеров рассмотрим производящие функции для трех последовательностей.

Получим производящую функцию для последовательности с общим членом c^n/n:

```
x GeneratingFunction[c^(n + 1)/(n + 1), n, x]
```
$$-\text{Log}[1 - c\,x]$$
```
SeriesCoefficient[Series[%, {x, 0, 7}], #]& /@ Range[0, 7]
```
$$\left\{0, c, \frac{c^2}{2}, \frac{c^3}{3}, \frac{c^4}{4}, \frac{c^5}{5}, \frac{c^6}{6}, \frac{c^7}{7}\right\}$$

Теперь производящая функция для последовательности с общим членом *Fibonacci*(*n*)/*n*:

```
x GeneratingFunction[Fibonacci[n + 1]/(n + 1), n, x]
```

$$\frac{\text{Log}\left[\frac{1+\sqrt{5}+2x}{1+\sqrt{5}}\right]-\text{Log}\left[\frac{1}{2}\left(2-x-\sqrt{5}x\right)\right]}{\sqrt{5}}$$

```
FullSimplify[SeriesCoefficient[Series[%, {x, 0, 7}], #]& /@ Range[0, 7]]
```

$$\left\{0,1,\frac{1}{2},\frac{2}{3},\frac{3}{4},1,\frac{4}{3},\frac{13}{7}\right\}$$

Вычислим производящую функцию для последовательности с общим членом $(-1)^n/n$:

```
x GeneratingFunction[(–1)^(n + 1)/(n + 1), n, x]
```

$$-\text{Log}[1+x]$$

```
FullSimplify[SeriesCoefficient[Series[%, {x, 0, 7}], #]& /@ Range[0, 7]]
```

$$\left\{0,-1,\frac{1}{2},-\frac{1}{3},\frac{1}{4},-\frac{1}{5},\frac{1}{6},-\frac{1}{7}\right\}$$

Свертка последовательностей. Сверткой последовательностей $f = \{f_0, f_1, f_2, \ldots\}$ и $g = \{g_0, g_1, g_2, \ldots\}$ называется последовательность $h = \{h_0, h_1, h_2, \ldots\}$, члены которой вычисляются по формуле

$$h_n = \sum_{k=0}^{n} f_k g_{n-k}.$$

Легко увидеть, что произведением двух формальных степенных рядов $f_0 + f_1 x + f_2 x^2 + \ldots$ и $g_0 + g_1 x + g_2 x^2 + \ldots$ является ряд $h_0 + h_1 x + h_2 x^2 + \ldots$. Таким образазом, получаем, что произведение производящих функций для последовательностей f и g является производящей функцией для свертки h последовательностей f и g.

Рассмотрим свертку последовательностей $\{f_0, f_1, f_2, \ldots\}$ и $g = \{1, 1, 1, 1, \ldots\}$, получаем $h_n = \sum_{k=0}^{n} f_k$, т.е. умножение производящей функции на $1/(1 - x)$ дает производящую функцию для последовательности частичных сумм исходной последовательности (так как производящая функция последовательности g есть $1/(1 - x)$).

Пример 4. Найдем последовательность частичных сумм для последовательности с общим членом $f_k = k^3\,3^k$.

```
FullSimplify[1/(1 - x) GeneratingFunction[n^3 3^n, n, x]]
```

$$-\frac{3x(1+3x(4+3x))}{(1-3x)^4(-1+x)}$$

Определяем общий член полученной последовательности:

```
SeriesCoefficient[%, {x, 0, n}]
```

$$\begin{cases}\frac{3}{8}\left(11-11\times 3^n+4\times 3^{1+n}n-2\times 3^{1+n}n^2+4\times 3^n n^3\right) & n\geq 0\\ 0 & \text{True}\end{cases}$$

Таким образом, установили

$$\sum_{k=0}^{n} k^3 3^k = \frac{3}{8}\left(11-11\times 3^n+4\times 3^{1+n}n-2\times 3^{1+n}n^2+4\times 3^n n^3\right).$$

Как найти производящую функцию для последовательности с общим членом $f(n + a)$, если известна производящая функция для последовательности с общим членом $f(n)$? Это возможно, если использовать опцию Assumptions:

```
GeneratingFunction[f[n + a], n, x, Assumptions → a ∈ Integers && a > 0]
```

$$x^{-a}\,\text{GeneratingFunction}\left[f[n],n,x\right]-\sum_{n=0}^{-1+a} x^{-a+n} f[n]$$

Пример 5. Найдем производящие функции для последовательностей $\{n \bmod m\}_{n=0}^{\infty}$ и $\left\{\left\lfloor \frac{n}{m}\right\rfloor\right\}_{n=0}^{\infty}$, где $m > 1$ – целое число и $n \bmod m$ – остаток от деления n на m, а $\left\lfloor \frac{n}{m}\right\rfloor$ – неполное частное от деления n на m. Для последовательности $\{n \bmod m\}_{n=0}^{\infty}$ используем опцию Assumptions:

```
GeneratingFunction[Mod[n, m], n, x,
                   Assumptions → Element[m, Integers] && m > 1]
```

$$-\frac{\begin{cases} x & m=2\\ \frac{x-mx^m-x^{1+m}+mx^{1+m}}{(-1+x)^2} & \text{True}\end{cases}}{-1+x^m}$$

Полученное выражение является записью на языке Wolfram кусочно-заданной функции. Определим эту функцию в равносильном, но более привычном виде:

```
r[x_, m_] := Module[{ y = 1 – x^m}, If[m == 2, x/y,
(x – mx^m – x^(1+m) + mx^(1+m))/((– 1 + x)^2 y)]]
```

Последовательность $\{n \bmod m\}_{n=0}^{\infty}$ является периодической с периодом 0, 1, …, $m - 1$. Рассмотрим случай $m = 5$. Найдем производящую функцию и члены степенного ряда до x^{17}:

```
r[x, 5]
```

$$\frac{x - 5x^5 + 4x^6}{(-1+x)^2\left(1-x^5\right)}$$

```
Series[r[x, 5], {x, 0, 17}]
```

$$x+2x^2+3x^3+4x^4+x^6+2x^7+3x^8+4x^9+x^{11}+2x^{12}+3x^{13}+4x^{14}+x^{16}+O[x]^{17}$$

Для нахождения производящей функции для неполного частного воспользуемся равенством

$$\lfloor n/m \rfloor = (n - n \bmod m)/m.$$

Поэтому производящую функцию находим как

```
GeneratingFunction[(n – Mod[n, m])/m, n, x, Assumptions →
Element[m, Integers] && m > 1]
```

$$\frac{\dfrac{x}{(-1+x)^2} + \dfrac{\begin{cases} x & m = 2 \\ \dfrac{x - mx^m - x^{1+m} + mx^{1+m}}{(-1+x)^2} & \text{True} \end{cases}}{-1+x^m}}{m}$$

Полученную функцию запишем в более удобном виде:

```
q[x_, m_] := x/(m(-1+x)^2) - r[x, m]/m
```

Найдем для $m = 5$ производящую функцию и члены степенного ряда до x^{16}.

```
Series[q[x, 5], {x, 0, 15}]
```

$$x^5 + x^6 + x^7 + x^8 + x^9 + 2x^{10} + 2x^{11} + 2x^{12} + 2x^{13} + 2x^{14} + 3x^{15} + O[x]^{16}$$

§ 3. Преобразования Дирихле

Важная альтернатива обычным производящим функциям получается, если вместо ядер z^n используются ядра $1/n^z$; они рассчитаны на последовательности $\{g_1, g_2, g_3, \ldots\}$, начинающиеся с $n = 1$, а не с $n = 0$:

$$\tilde{G}(z) = \sum_{n\geq 1} \frac{g_n}{n^z}.$$

Такие функции называются **производящими функциями Дирихле** (ПФД). Например, ПФД для последовательности $\{1, 1, 1, \ldots\}$ равна

$$\sum_{n\geq 1} \frac{1}{n^z} = \zeta(z).$$

Это известная дзета-функция Римана.

Производящие функции Дирихле особенно полезны, когда последовательность $\{g_1, g_2, g_3, \ldots\}$ является мультипликативной функцией. Каким образом производящие функций Дирихле связаны с мультипликативными функциями? Пусть $\tilde{F}(z)$ и $\tilde{G}(z)$ – производящие функции Дирихле для последовательностей $\{f(1), f(2), \ldots\}$ и $\{g(1), g(2), \ldots\}$ соответственно. Тогда произведение этих производящих функций является также ПФД:

$$\tilde{F}(z)\tilde{G}(z) = \sum_{k,m\geq 1} \frac{f_k}{k^z}\frac{g_m}{m^z} = \sum_{n\geq 1} \frac{1}{n^z}\Big(\sum_{\substack{k,m\geq 1\\ k\cdot m=n}} f_k g_m\Big) = \tilde{H}(z).$$

Если рассматривать функции f и g как соответствующие последовательности $\{f(1), f(2), \ldots\}$ и $\{g(1), g(2), \ldots\}$, то суммы вида

$$\sum_{\substack{k,m\geq 1\\ k\cdot m=n}} f_k g_m$$

встретились нам в главе при определении свертки Дирихле $f * g$. Следовательно, произведение производящих функций $\tilde{F}(z)\tilde{G}(z)$ есть производящая функция $\tilde{H}(z)$ для свертки Дирихле $h = f * g$.

Кроме того, так как значение любой мультипликативной функции $f(n)$ определяются значениями для n, являющихся степенями простых чисел, то можно разложить ПФД в произведение по простым числам [67, с. 407]:

$$\tilde{F}(z) = \prod_{p \text{ простое}} (1 + \frac{f(p)}{p^z} + \frac{f(p^2)}{p^{2z}} + \frac{f(p^3)}{p^{3z}} + \ldots).$$

Если, например, $f(n) = 1$ – постоянная функция, то получаем представление дзета-функции Римана в виде произведения

$$\zeta(z) = \prod_{p \text{ простое}} (\frac{1}{1 - p^{-z}}).$$

Для функции Мебиуса $\mu(p) = -1$ и $\mu(p^k) = 0$, для $k > 1$, следовательно, ее ПФД равна

$$\tilde{M}(z) = \prod_{p \text{ простое}} (1 - p^{-z}).$$

Производящую функцию Дирихле для функции f часто называют **преобразованием Дирихле** функции f. В языке Wolfram преобразования Дирихле вычисляются с помощью функции DirichletTransform. В частности, имеем

```
DirichletTransform[1, n, s]
Zeta[s]
```

Примеры 6 (преобразования Дирихле)

```
DirichletTransform[MoebiusMu[n], n, s]
1 / Zeta[s]

DirichletTransform[DivisorSum[n, f], n, s]
DirichletTransform[f[n], n, s] Zeta[s]

DirichletTransform[DivisorSum[n, EulerPhi], n, s]
Zeta[-1 + s]

DirichletTransform[EulerPhi[n], n, s]
Zeta[-1 + s] / Zeta[s]

DirichletTransform[m^a, m, s]
Zeta[-a + s]

DirichletTransform[DivisorSigma[#, n], n, s]& /@ Range[0, 3]
{Zeta[s]^2, Zeta[-1 + s] Zeta[s], Zeta[-2 + s] Zeta[s], Zeta[-3 + s] Zeta[s]}
```

Можно предположить, что в общем случае

```
DirichletTransform[DivisorSigma[k, n], n, s]
```

равно

Zeta[$-k+s$] Zeta[s]

но Mathematica исходное выражение не вычисляет:

```
DirichletTransform[DivisorSigma[k, n], n, s]
DirichletTransform[DivisorSigma[k, n], n, s]
```

Примеры 7 (преобразования Дирихле)

```
DirichletTransform[DivisorSum[n, #^–2&], n, s]
Zeta[s] Zeta[2 + s]
DirichletTransform[DivisorSum[n, f], n, s]
DirichletTransform[f[n], n, s] Zeta[s]
```

Преобразование Дирихле линейно:

```
DirichletTransform[a f[n] + b g[n], n, s]
a DirichletTransform[f[n], n, s] + b DirichletTransform[g[n], n, s]
DirichletTransform[EulerPhi[n] n^a, n, s]
Zeta[–1 – a + s] / Zeta[–a + s]
DirichletTransform[Mod[n, 3], n, s]
3^–s (Zeta[s, 1/3] + 2 Zeta[s, 2/3])
```

С помощью производящих функций Дирихле мы получаем возможность вычислять свертку Дирихле для мультипликативных функций. Если мы предполагаем, что $f * g = h$, то для доказательства тождества вычисляем ПФД для функций f, g, h и убеждаемся, что произведение ПФД функций f и g равно ПФД функции h.

Сначала известные примеры.

Пример 8. $\varphi(n) * \sigma(n) = n\,\tau(n)$.

Для доказательства тождества вычисляем производящие функции для функций $\varphi(n)$, $\sigma(n)$ и $n\,\tau(n)$ и убеждаемся, что произведение двух первых ПФД равно третьей ПФД.

```
DirichletTransform[EulerPhi[n], n, s]
Zeta[–1 + s] / Zeta[s]
DirichletTransform[DivisorSigma[1, n], n, s]
Zeta[–1 + s] Zeta[s]
DirichletTransform[EulerPhi[n], n, s]
DirichletTransform[DivisorSigma[1, n], n, s]
Zeta[–1 + s]^2
```

В этом случае тождество можно было проверить, просто вычисляя $\varphi(n) * \sigma(n)$:

```
DirichletConvolve[EulerPhi[n], DivisorSigma[1, n], n, m]
m DivisorSigma[0, m]
```

Пример 9. $n^a\ \varphi(n) * n^a = n^{a+1}$.

Это тождество доказано при доказательстве теоремы 26 из главы 4. Его можно также доказать, вычисляя ПФД, как в примере 8. Но явное вычисление свертки ничего не дает:

```
DirichletConvolve[n^a EulerPhi[n], n^a, n, m]
DirichletConvolve[n^a EulerPhi[n], n^a, n, m]
```

Пример 10. $n^a\ \mu(n) * n^{a+1} = n^a\ \varphi(n)$.

Тождество $\mu(n) * n = \varphi(n)$ можно доказать, прямо вычисляя свертку $\mu(n) * n$. Но при доказательстве более общего тождества $n^a\ \mu(n) * n^{a+1} = n^a\ \varphi(n)$ необходимы вычисления трех ПФД; прямое вычисление свертки $n^a\ \mu(n) * n^{a+1}$ ничего не дает. Следующие вычисления ПФД показывают справедливость тождества.

```
DirichletTransform[ n^(a+1), n, s]
Zeta[–1 – a + s]

DirichletTransform[n^^a MoebiusMu[n], n, s]
1/Zeta[–a + s]

DirichletTransform[n^a EulerPhi[n], n, s]
Zeta[–1 – a + s] / Zeta[–a + s]
```

Пример 11. $\varphi(n) * \sigma_{k+1}(n) = n\sigma_k(n)$, $k \geq 0$ – целое.

Это тождество обобщает тождество из примера 8. Но доказать это тождество так же, как тождество из примера 10, не получается. Не вычисляется преобразование Дирихле для функции $\sigma_{k+1}(n)$:

```
DirichletTransform[DivisorSigma[k + 1, n], n, s]
DirichletTransform[DivisorSigma[k + 1, n], n, s]
```

Прямое вычислении свертки $\varphi(n) * \sigma_{k+1}(n)$ также не проходит:

```
DirichletConvolve[EulerPhi[n], DivisorSigma[k + 1, n], n, m]
DirichletConvolve[EulerPhi[n], DivisorSigma[k + 1, n], n, m]
```

Но свертка Дирихле для частных значений k вычисляется:

```
DirichletConvolve[EulerPhi[n], DivisorSigma[3, n], n, m]
```
m DivisorSigma[2, m]

```
DirichletConvolve[EulerPhi[n], DivisorSigma[7, n], n, m]
```
m DivisorSigma[6, m]

Тождество $\varphi(n) * \sigma_{k+1}(n) = n\sigma_k(n)$ выполняется для всех неотрицательных целых $k \le 5000$:

```
AllTrue[Table[DirichletConvolve[EulerPhi[n], DivisorSigma[k + 1, n],
n, m] == m DivisorSigma[k, m], {k, 0, 5000}], #&]
```
True

По-видимому, тождество справедливо для всех целых $k \ge 0$. Но это остается недоказанным.

Глава 8

ВОКРУГ ТЕОРЕМЫ ЭЙЛЕРА О СУММАХ ДЕЛИТЕЛЕЙ

В этой главе покажем аппарат производящих функций в действии. Рассмотрим две задачи, которые решал Эйлер. В одной задаче подсчитывается число разбиений для произвольного натурального числа, в решении другой задачи появляется рекуррентная закономерность, связывающая суммы делителей натуральных чисел. Эйлер не имел определений формального степенного ряда и производящей функции, тем не менее, используя индукцию, на интуитивном уровне получил то, что сейчас можно получить с помощью производящих функций и вычислений на языке Wolfram.

§ 1. Последовательности Эйлера

Разбиения

Разбиением натурального числа n называется всякая невозрастающая последовательность натуральных чисел $\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_m$, для которых

$$\sum_{i=1}^{m} \lambda_i = n \,.$$

Для чисел n, $0 < n < 6$, перечислим все разбиения:

$n = 1$ 1

$n = 2$ 2 = 1+1

$n = 3$ 3 = 2+1 = 1+1+1

$n = 4$ 4 = 3+1 = 2+2 = 2+1+1 = 1+1+1+1

$n = 5$ 5 = 4+1 = 3+2 = 3+1+1 = 2+2+1 = 2+1+1+1 = 1+1+1+1+1

Пусть p_n обозначает число разбиений для неотрицательного целого числа n (условимся считать, что $p_0 = 1$), тогда последовательность $p_0, p_1, p_2, \ldots$ начинается с чисел 1, 1, 2, 3, 5, 7, 11, 15, …

Достаточно просто найти производящую функцию для последовательности p_n (доказательство см. в [88, с. 88]).

Теорема 1 (Эйлер). Производящая функция для числа разбиения числа n имеет вид

$$P(x)=\frac{1}{\prod_{n=0}^{\infty}\left(1-x^{n+1}\right)}.$$

В языке *Wolfram* имеется функция PartitionsP[40] для вычисления значений последовательности p_n:

```
PartitionsP/@Range[0,10]
{1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42}
```

Mathematica знает, чему равна производящая функция для последовательности p_n, и выдает ее в «замкнутом» виде:

```
GeneratingFunction[PartitionsP[n], n, x]
        1
-------------------
QPochhammer[x, x]
```

Функция QPochhammer[a, q] дает значение «q-символа Похгаммера» $(a;q)_\infty$, который определяется как

$$(a;q)_\infty=\prod_{k=0}^{\infty}\left(1-aq^k\right).$$

Этот символ был введен вскоре после обычных гипергеометрических функций; q-функции долгое время изучались как теоретические обобщения гипергеометрических и других функций. Язык *Wolfram* впервые дает возможность полностью вычислить числовые q-функции и также совершить символьные преобразования с ними, что позволяет использовать q-функции в замкнутом виде для сумм, произведений, решений рекуррентных уравнений и т. д. [42].

Можно получить q-символ Похгаммера при упрощении формального произведения

```
Simplify[∏_{n=0}^{∞} (1 - x^(n+1))]
QPochhammer[x, x]
```

Можно явно получить конечный степенной ряд, коэффициенты которого являются начальными элементами последовательности $p_0, p_1, p_2, \ldots$:

```
Series[1/QPochhammer[x, x], {x, 0, 9}]
```

$$1 + x + 2x^2 + 3x^3 + 5x^4 + 7x^5 + 11x^6 + 15x^7 + 22x^8 + 30x^9 + O[x]^{10}$$

Последовательность с производящей функцией QPochhammer[*x*, *x*]

Изучим сейчас последовательность $a_0, a_1, a_2, \ldots$, для которой производящей функцией является QPochhammer[x, x]. Посмотрим на первые члены этой последовательности:

```
SeriesCoefficient[QPochhammer[x, x], {x, 0, #}]& /@ Range[0, 51]
```

{1, –1, –1, 0, 0, 1, 0, 1, 0, 0, 0, 0, –1, 0, 0, –1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, –1, 0, 0, 0, 0, –1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1}

Впервые эту последовательность получил Эйлер, который вручную находил коэффициенты формального степенного ряда, получаемого в результате бесконечного произведения [95, с. 117, 118]:

$$(1 - x)(1 - x^2)(1 - x^3)\,(1 - x^4)(1 - x^5)\ldots.$$

Сейчас это можно сделать быстрее:

```
Series[Product[(1 - x^n), {n, ∞}], {x, 0, 51}]
```

$$1 - x - x^2 + x^5 + x^7 - x^{12} - x^{15} + x^{22} + x^{26} - x^{35} - x^{40} + x^{51} + O[x]^{52}$$

Рассмотрение последнего ряда привело Эйлера к гипотезе, которая сейчас известна как

Теорема 2 (пентагональная теорема Эйлера).

$$\prod_{n=1}^{\infty}\left(1 - x^n\right) = 1 + \sum_{m=1}^{\infty}(-1)^m\left(x^{m(3m-1)/2} + x^{m(3m+1)/2}\right).$$

Доказательство смотрите, например, в [88, с. 91, 92; 103, с. 26].

Найдем формулу для последовательности $a_0, a_1, a_2, \ldots$ в замкнутом виде. Если $n = m(3m \pm 1)/2$, то выразим m через n. Получаем

$$m = \frac{1}{6}\left(\sqrt{1 + 24n} \pm 1\right).$$

Таким образом, если $t = \sqrt{1+24n}$ – целое число, делящееся на 6, то $a_n = \pm 1$, иначе $a_n = 0$. Заметим, что t должно быть нечетным.

Определим функцию от произвольного натурального t

$$\alpha(t) = \begin{cases} (-1)^{(t\pm 1)/6}, & \text{если } t \equiv \mp 1 (\mathrm{mod}\, 6); \\ 0, & \text{иначе.} \end{cases}$$

Если $t = \sqrt{1+24n}$ – целое число, делящееся на 6, то $a_n = \alpha(t)$.

Так как элементы последовательности a_0, a_1, a_2, … являются только ± 1 или 0, то естественно предположить, что они суть значения некоторого символа Якоби для t. Для нечетных t в качестве простейших кандидатов можно взять $\left(\frac{t}{3}\right)$ или $\left(\frac{3}{t}\right)$. Проверка показывает, что $\left(\frac{t}{3}\right) \neq \alpha(t)$ в общем случае. Проверим предположение $\left(\frac{3}{t}\right) = \alpha(t)$ для первой тысячи нечетных натуральных чисел:

```
AllTrue[(JacobiSymbol[3, #] ==
    Piecewise[{{(–1)^((# + 1)/6), Mod[#, 6] == 5},
            {(–1)^((# – 1)/6), Mod[#, 6] == 1}}, 0])& /@
            Select[Range[1000], OddQ],
        #&]
True
```

Лемма. Для нечетных натуральных t выполнено равенство $\alpha(t) = \left(\frac{3}{t}\right)$.

Доказательство. Если t не кратно 3, то по квадратичному закону взаимности и по критерию Эйлера квадратичного вычета по простому модулю имеем

$$\left(\frac{3}{t}\right) = \left(\frac{t}{3}\right)(-1)^{\frac{(3-1)(t-1)}{4}} = \left(\frac{t}{3}\right)(-1)^{(t-1)/2} \equiv t^{(3-1)/2}(-1)^{(t-1)/2} \equiv t(-1)^{(t-1)/2} (\mathrm{mod}\, 3).$$

Далее разберем два случая для t.

1. $t = 6k + 5$ (k – целое). В этом случае $(t-1)/2 = 3k + 2$ и поэтому

$$\left(\frac{3}{t}\right) \equiv (6k+5)(-1)^{3k+2} \equiv (-1)^{3k+3} \equiv (-1)^{k+1} (\mathrm{mod}\, 3).$$

Для $t = 6k + 5$ значение $\alpha(t)$ равно $(-1)^{(t+1)/6} = (-1)^{k+1}$.

2. $t = 6k + 1$. В этом случае $(t-1)/2 = 3k$ и поэтому

$$\left(\frac{3}{m}\right) \equiv (6k+1)(-1)^{3k} \equiv (-1)^k (\mathrm{mod}\, 3).$$

Для $t = 6k + 1$ значение $\alpha(t)$ равно $(-1)^{(t-1)/6} = (-1)^k$.

3. Осталось рассмотреть ситуацию, когда m кратно 3. Но в этом случае $\left(\frac{3}{m}\right) = 0 = \alpha(m)$ по определению. ■

Теорема 3 (формула для последовательности a_0, a_1, a_2, …в замкнутом виде). Для целого $n > 0$ положим $t = \sqrt{1+24n}$. Тогда элементы последовательности a_0, a_1, a_2, … можно вычислить по правилу

$$a_n = \begin{cases} \left(\dfrac{12}{t}\right), \text{ если } t \text{ – целое число;} \\ 0, \text{ иначе.} \end{cases} \qquad (1)$$

Доказательство. Из леммы следует, что если t – нечетное целое число, то $a_n = \left(\frac{3}{t}\right)$.

Желательно было бы иметь такое x, чтобы $a_n = \left(\frac{x}{t}\right)$ для всех натуральных t. Для этого достаточно взять $x = 12$, тогда

$$\left(\frac{12}{t}\right) = \left(\frac{2}{t}\right)^2 \left(\frac{3}{t}\right)$$

и (1) справедливо, так как для нечетных t символ Якоби $\left(\frac{2}{t}\right) = \pm 1$, а для четных t выполнено $\left(\frac{2}{t}\right) = 0$. ■

В соответствии с теоремой a_n можно вычислять на языке Wolfram с помощью следующей функции:

```
a[n_] := With[{t = Sqrt[24n + 1]}, If[IntegerQ[t], JacobiSymbol[12, t], 0]]
```

Последовательность 1, 2, 5, 7, 12, 15, 22, 26, 35, 40, 51, …

Эйлер также рассматривал последовательность $\{b_n \mid n = 0, 1, 2, \ldots\}$:

$$1, 2, 5, 7, 12, 15, 22, 26, 35, 40, 51, \ldots \text{ [37]}, \qquad (2)$$

состоящую из положительных показателей степеней ненулевых членов ряда

$$1 - x - x^2 + x^5 + x^7 - x^{12} - x^{15} + x^{22} + x^{26} - x^{35} - x^{40} + x^{51} + \ldots .$$

Он экспериментально обнаружил закономерность в построении последовательности (2). Далее цитируем Эйлера по книге [95, с. 114, 115].

---

[37] Эта последовательность с добавленным числом 0 в качестве первого члена называется последовательностью *обобщенных пятиугольных чисел*.

«Закон чисел 1, 2, 5, 7, 12, 15, … < … > станет ясен, если мы возьмем их разности:

Числа 1, 2, 5, 7, 12, 15, 22, 26, 35, 40, 51, 57, 70, 77, 92, 100, …

Разности ***1***, 3, ***2***, 5, ***3***, 7, ***4***, 9, ***5***, 11, ***6***, 13, ***7***, 15, ***8***, …

В самом деле, мы имеем здесь поочередно все целые числа 1, 2, 3, 4, 5, 6, … и нечетные числа, и поэтому мы можем продолжать последовательность этих чисел сколь угодно далеко».

Слова Эйлера подсказывают, что члены последовательности (2) удовлетворяют следующему правилу:

$$b_n = \begin{cases} 1, \text{если } n = 0; \\ b_{n-1} + n + 1, \text{если } n \text{ четно}; \\ b_{n-1} + (n+1)/2, \text{если } n \text{ нечетно}. \end{cases} \qquad (3)$$

На языке Wolfram рекурсивная программа с запоминанием уже вычисленных промежуточных значений выглядит следующим образом:

```
b[0] = 1;
b[n_?EvenQ] := b[n] = b[n – 1] + n + 1
b[n_?OddQ] := b[n] = b[n – 1] + (n + 1)/2
```

Проверим

```
b /@ Range[0, 15]
{1, 2, 5, 7, 12, 15, 22, 26, 35, 40, 51, 57, 70, 77, 92, 100}
```

Найдем производящую функцию для последовательности b_n. Простое применение функции GeneratingFunction к $b[n]$ выдает результат невычисленным. Поэтому будем применять способ решения рекуррентных соотношений в четыре шага, изложенный в [67, с. 364–376].

Шаг 1 требует записать рекуррентное соотношение (3) в виде «одного уравнения» для последовательности b_n, т.е. нужна формула, не содержащая конструкций с перечислением случаев. Искомое уравнение имеет вид

$$b_n = b_{n-1}\,[n > 0] + (n+1)/2\ [n \text{ нечетно}] + \\ + (n+1)[n > 0 \text{ и } n \text{ четно}] + [n = 0]. \qquad (4)$$

Запись с квадратными скобками широко используется в [67] в соответствием с правилом:

Если S – некоторое утверждение, которое может быть истинно или ложно, квадратно-скобочное обозначение $[S]$ означает 1, если S истинно, и 0 в противном случае.

Поэтому в правой части (4) присутствует только 1, когда $n = 0$, а при $n > 0$ справа всегда присутствует b_{n-1} и еще одно слагаемое $(n + 1)/2$ или $(n + 1)$ в зависимости от четности числа n.

На ш а г е 2 умножаем обе части уравнения на x^n и просуммируем по всем n. В левой части получается сумма $\sum_{n=0}^{\infty} b_n x^n$, которая равна производящей функции $B(x)$, а правую часть следует преобразовать с тем, чтобы она превратилась в какое-то другое выражение, включающее $B(x)$.

Слагаемое b_{n-1} $[n > 0]$ преобразуется в $\sum_{n=0}^{\infty} b_{n+1} x^{n+1} = xB(x)$, слагаемое $[n = 0]$ становится 1. Слагаемое $(n + 1)/2$ [n нечетно] преобразуется в $\sum_{n=0}^{\infty}((n+1)/2)x^n$ [n нечетно]. Из ряда $\sum_{n=0}^{\infty}((n+1)/2)x^n$ получаем с помощью Mathematica производящую функцию $C1$ в замкнутой форме:

```
C1 = GeneratingFunction[(n + 1)/2, n, x]
```

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

Но в ряде $\sum_{n=0}^{\infty}((n+1)/2)x^n$ должны присутствовать члены только с нечетными n. Есть общий метод извлечения членов с нечетными номерами $\{0, g_1, 0, g_3, 0, g_5, 0, \ldots\}$ из любой последовательности $g_0, g_1, g_2, g_3, \ldots$. Имеем

$$\sum_{n=0}^{\infty} g_{2n+1} x^{2n+1} = \frac{G(x) - G(-x)}{2},$$

где $G(x)$ – производящая функция последовательности $g_0, g_1, g_2, g_3, \ldots$. В соответствии с этим преобразуем второе слагаемое в (4):

```
Simplify[(C1 – (C1 /. x → –x)) / 2]
```

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

Третье слагаемое в (4) $(n + 1)[n > 0$ и n четно] преобразуем сначала в выражение

$$\sum_{n=0}^{\infty}(n+1)x^n \ [n \text{ четно}].$$

Из ряда $\sum_{n=0}^{\infty}(n+1)x^n$ получаем с помощью Mathematica производящую функцию $C2$ в замкнутой форме:

```
C2 = GeneratingFunction[n + 1, n, x]
```

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

Также имеется общий метод извлечения членов с четными номерами $\{g_0, 0, g_2, 0, g_4, 0, \ldots\}$ из любой последовательности $g_0, g_1, g_2, g_3, \ldots$. Имеем

$$\sum_{n=0}^{\infty} g_{2n}x^{2n} = \frac{G(x)+G(-x)}{2}.$$

При извлечении из ряда членов с четными n, заметим, что поскольку член $1\times x^0$ не нужен, то нужно отнять 1:

```
Simplify[(C2 + (C2/. x → – x))/2] – 1
```

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

Проверим, что действительно сейчас получили члены с ненулевыми четными степенями x:

```
Series[%, {x, 0, 10}]
```

$$3x^2+5x^4+7x^6+9x^8+11x^{10}+O[x]^{11}$$

Последнее слагаемое $[n = 0]$ в (4) дает просто 1. Суммируя все полученные выражения в правой части (4), получаем уравнение для производящей функции B.

$$B = xB+\frac{x}{\left(-1+x^2\right)^2}+\frac{1+x^2}{\left(-1+x^2\right)^2}.$$

Шаг 3. Решая уравнение, получаем для производящей функции $B(x)$ выражение в замкнутом виде:

```
Solve[B == x B + x/(-1 + x^2)^2 + (1 + x^2)/(-1 + x^2)^2, {B}]
```

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

***B* /. %[[1]]**

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

Следовательно, выражение

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

является производящей функцией в замкнутом виде для последовательности Эйлера

1, 2, 5, 7, 12, 15, 22, 26, 35, 40, 51, 57, 70, 77, 92, 100, … .

На последнем четвертом шаге, имея производящую функцию, в языке Wolfram легко получить выражение b_n в замкнутом виде:

SeriesCoefficient[- (1+ x + x^2)/((-1+ x)^3 (1+ x)^2), {x, 0, n}]

$$\begin{cases}\frac{1}{16}(13+3(-1)^n+2(9+(-1)^n)n+6n^2) & n\geq 0\\ 0 & True\end{cases}$$

Определим b_n в замкнутом виде

b[n_] := (13 + 3(–1)^n + 2(9 + ((–1)^n)n + n^2) / 16 **(5)**

Вычислим 20 первых членов последовательности

***b* /@ Range[0, 19]**

{1, 2, 5, 7, 12, 15, 22, 26, 35, 40, 51, 57, 70, 77, 92, 100, 117, 126, 145, 155}

В языке Wolfram имеется функция FindSequenceFunction[{x_1, x_2, …, x_k}, n], которая пытается найти простую функцию, которая дает последовательность x_1, x_2, x_3, … при задании последовательных целых аргументов n. Необходимо в качестве первого аргумента задать конечную последовательность, но членов должно быть достаточно, для того, чтобы начало последовательности было «характерно». Если с каким-то началом последовательности FindSequenceFunction выдает в качестве

результата какую-то функцию, то надо попробовать увеличить количество членов и убедиться, что функция не меняется.

Для нашей последовательности 1, 2, 5, 7, 12, 15, 22, 26, 35, 40, 51, … вызов

```
FindSequenceFunction[{1, 2, 5, 7, 12, 15, 22}, n]
FindSequenceFunction[{1, 2, 5, 7, 12, 15, 22}, n]
```

ничего не дает. И только после 13 членов последовательности функция определяется и не меняется при увеличении количества членов.

```
FindSequenceFunction[{1, 2, 5, 7, 12, 15, 22, 26, 35, 40, 51, 57, 70}, n]
```

$$\frac{1}{16}\left(1+(-1)^{1+n}+6n-2(-1)^n n+6n^2\right) \qquad (6)$$

Сравним формулу (6) с формулой (5). Только надо не забыть, что функция FindSequenceFunction нумерует члены последовательности, начиная с 1, а в формуле (5) нумерация членов идет с 0.

```
Simplify[1/16 (1 + (-1)^(1 + n) + 6 n - 2 (-1)^n n + 6 n^2) /. n -> n + 1 ==
  1/16 (13 + 3 (-1)^n + 2 (9 + (-1)^n) n + 6 n^2)]
True
```

Учитывая различную нумерацию членов, получаем эквивалентность формул (5) и (6).

§ 2. Теоремы Эйлера

Гипотеза Эйлера о законе чисел, относящемся к суммам их делителей

Эйлер путем индуктивных рассуждений обнаружил удивительную рекуррентную закономерность, связывающую суммы делителей натуральных чисел. Начальное представление об этой связи дает следующая формула ($\sigma(n)$ обозначает сумму положительных делителей натурального числа n):

$$\begin{aligned}\sigma(n) = {} & \sigma(n-1)+\sigma(n-2)-\sigma(n-5)-\sigma(n-7)+\sigma(n-12)+\sigma(n-15)- \\ & -\sigma(n-22)-\sigma(n-26)+\sigma(n-35)+\sigma(n-40)-\sigma(n-51)- \\ & -\sigma(n-57)+\sigma(n-70)+\sigma(n-77)-\sigma(n-92)-\sigma(n-100)+\ldots \end{aligned} \qquad (7)$$

Как точно выглядит это выражение и как Эйлер пришел к своей догадке, все это хорошо изложено в [95, с. 111–127]. Там же имеется и

библиографическая ссылка на оригинал. Важно следующее: в этом выражении встречаются последовательности (1) и (2). Члены последовательности (2) вычитаются из n в аргументах функции σ. Коэффициенты, стоящие перед σ в правой части выражения (7) совпадают с ненулевыми членами последовательности (1), умноженными на –1.

Получим гипотезу Эйлера, используя систему Mathematica. Возьмем конечный ряд из 51 члена формального степенного ряда для последовательности (1). Производящая функция для последовательности (1) есть QPochhammer[x, x].

```
t1 = Series[QPochhammer[x, x], {x, 0, 50}]
```

$$1 - x - x^2 + x^5 + x^7 - x^{12} - x^{15} + x^{22} + x^{26} - x^{35} - x^{40} + O[x]^{51}$$

Продифференцируем этот ряд по x и результат умножим на $-x$:

```
t2 = –x D[Series[QPochhammer[x, x], {x, 0, 50}], x]
```

$$x + 2x^2 - 5x^5 - 7x^7 + 12x^{12} + 15x^{15} - 22x^{22} - 26x^{26} + 35x^{35} + 40x^{40} + O[x]^{51}$$

Как видим, коэффициенты ряда $t2$ отличаются от соответствующих коэффициентов ряда $t1$ для члена x^n множителем $-n$. Это сохраняется и для рассмотренных больших n. Определим новую последовательность $c_n = -n\,a_n$ для всех $n = 0, 1, 2, \ldots$

Теперь найдем частное двух рядов $t3 = t2/t1$, заметим, что мы учитываем в примере только степени, не превосходящие 50.

```
t3 = t2/t1
```

$$x + 3x^2 + 4x^3 + 7x^4 + 6x^5 + 12x^6 + 8x^7 + 15x^8 + 13x^9 + 18x^{10} + 12x^{11} + 28x^{12} + 14x^{13} + 24x^{14} + 24x^{15} + 31x^{16} + 18x^{17} + 39x^{18} + 20x^{19} + 42x^{20} + 32x^{21} + 36x^{22} + 24x^{23} + 60x^{24} + 31x^{25} + 42x^{26} + 40x^{27} + 56x^{28} + 30x^{29} + 72x^{30} + 32x^{31} + 63x^{32} + 48x^{33} + 54x^{34} + 48x^{35} + 91x^{36} + 38x^{37} + 60x^{38} + 56x^{39} + 90x^{40} + 42x^{41} + 96x^{42} + 44x^{43} + 84x^{44} + 78x^{45} + 72x^{46} + 48x^{47} + 124x^{48} + 57x^{49} + 93x^{50} + O[x]^{51}$$

Создадим список коэффициентов ряда:

```
list = SeriesCoefficient[t3, #]& /@ Range[50]
```

{1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28, 14, 24, 24, 31, 18, 39, 20, 42, 32, 36, 24, 60, 31, 42, 40, 56, 30, 72, 32, 63, 48, 54, 48, 91, 38, 60, 56, 90, 42, 96, 44, 84, 78, 72, 48, 124, 57, 93}

Проверим, что эти числа являются суммами делителей первых 50 натуральных чисел (DivisorSigma[1, x] – встроенная функция в языке Wolfram для вычисления $\sigma(x)$).

```
AllTrue[(list[[#]] == DivisorSigma[1, #])& /@ Range[50], #&]
True
```

Дополнительные проверки для конечных рядов $t1$ и $t2$ с большим количеством членов также показали, что $t3 = t2/t1$ есть конечный степенной ряд для соответствующей конечной последовательности $\sigma(1)$, $\sigma(2)$, $\sigma(3)$, …, $\sigma(m)$. Можно предположить, что частное рядов $t3 = t2/t1$, при расширении $t1$ и $t2$ до бесконечных, является формальным степенным рядом для последовательности $\sigma(1)$, $\sigma(2)$, $\sigma(3)$, … .

Если это так, то произведение формальных степенных рядов $t3$ и $t1$ дает формальный степенной ряд для последовательности c_0, c_1, c_2, …, которая является сверткой последовательностей a_0, a_1, a_2, …, и 0, $\sigma(1)$, $\sigma(2)$, $\sigma(3)$, … . Следовательно, для $n = 1, 2, 3, \ldots$ имеем формулу

$$c_n = \sum_{k=0}^{n} a_k \sigma(n-k).$$

Отсюда получаем гипотезу Эйлера

$$\sigma(n) = n a_n - \sum_{k=1}^{n-1} a_k \sigma(n-k). \qquad (8)$$

Раньше была написана функция языка Wolfram для вычисления a_n,

```
a[n_ ] := With[{m = Sqrt[24n + 1]},  – If[IntegerQ[m],
JacobiSymbol[12, m], 0]]
```

Напишем программу для вычисления $-\sum_{k=1}^{n-1} a_k \sigma(n-k) + n c_n$ в символьном виде

```
s0[n_] := -Sum[(a[k] σ[n - k]), {k, 1, n - 1}] + n a[n]
```

Посмотрим, как выглядит правая часть (8) для $n = 97$ и $n = 100$:

```
s0[97]
-σ[5] + σ[20] + σ[27] - σ[40] - σ[46] + σ[57] + σ[62] - σ[71] - σ[75] +
 + σ[82] + σ[85] - σ[90] - σ[92] + σ[95] + σ[96]
```

```
s0[100]
```
$$-100-\sigma[8]+\sigma[23]+\sigma[30]-\sigma[43]-\sigma[49]+\sigma[60]+\sigma[65]-\sigma[74]-$$
$$-\sigma[78]+\sigma[85]+\sigma[88]-\sigma[93]-\sigma[95]+\sigma[98]+\sigma[99]$$

Теперь определим функцию для численного вычисления правой части в (8) и проверим выполнения равенства для $n = 97$ и $n = 100$.

```
s[n_] := n a[n] + Sum[a[k] DivisorSigma[1, n - k], {k, 1, n - 1}]
{DivisorSigma[1, 97], s[97]}
{98, 98}
{DivisorSigma[1, 100], s[100]}
{217, 217}
```

И, наконец, проверим выполнения равенства (8) для всех $n \le 1000$.

```
AllTrue[(s[#] == DivisorSigma[1, #])& /@ Range[1000], #&]
True
```

Эйлер не смог доказать найденную им закономерность, она была доказана значительно позже.

Рекуррентная формула о числе разбиений

Из теоремы 1 следует, что

$$(\sum_{n=0}^{\infty} a_n x^n)(\sum_{n=0}^{\infty} p(n)x^n) = 1,$$

где $p(n)$ – количество разбиений числа n (§1) и a_0, a_1, a_2, a_3, … – последовательность, определенная в §1. Таким образом, произведение формальных степенных рядов дает единичный ряд, следовательно, свертка последовательности $p(0)$, $p(1)$, $p(2)$, … и последовательности a_0, a_1, a_2, a_3, … дает последовательность 1, 0, 0, 0, … . Поэтому для $n > 0$, получаем

$$\sum_{m=0}^{n} a_m p_{n-m} = 0.$$

Следовательно,

$$p_n = -\sum_{m=1}^{n} a_m p_{n-m}. \qquad (9)$$

Напишем программу для вычисления правой части (9) в символьном виде:

$$w0[n_] := -\left(\sum_{m=1}^{n-1} a[m]\,p[n-m] + a[n]\right)$$

Посмотрим, как выглядит правая часть (9) для $n = 50$ и $n = 51$:

w0[50]

$$p[10]+p[15]-p[24]-p[28]+p[35]+p[38]-p[43]-p[45]+p[48]+p[49]$$

w0[51]

$$-1+p[11]+p[16]-p[25]-p[29]+p[36]+p[39]-p[44]-p[46]+p[49]+p[50]$$

Теперь определим функцию для численного вычисления правой части в (9) и проверим выполнения равенства для $n = 50$ и $n = 1000$.

$$w[n_] := -\left(\sum_{m=1}^{n-1} a[m]\,\text{PartitionsP}[n-m] + a[n]\right)$$

{PartitionsP[50], w[50]}

{204226, 204226}

{PartitionsP[1000], w[1000]}

{24061467864032622473692149727991,
24061467864032622473692149727991}

И, наконец, проверим выполнения равенства (9) для всех $n \le 1000$.

AllTrue[(w[#] == PartitionsP[#])& /@ Range[1000], #&]

True

Формула (9) представляет собой алгоритм для вычисления $p(n)$.

Полученные результаты этой главы ранее опубликованы в [74].

Глава 9

РАСПОЗНАВАНИЕ ПРОСТЫХ И СОСТАВНЫХ ЧИСЕЛ

§ 1. Псевдопростые числа Ферма

Контропозиция малой теоремы Ферма (глава 7, теорема 10) дает тест на разложимость натурального числа.

Теорема 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.

Случай $a = n - 1$ исключается, так как $(n-1)^{n-1} \equiv 1 \pmod{n}$ для любого нечетного n и поэтому новой информации о числе n мы не получим.

Пример 1. Определим $r(n) = (10^n - 1)/9$. Десятичными цифрами чисел $r(n)$ являются только единицы; их n штук. Если число n составное, то число $r(n)$ – тоже составное (если n делится на m, то $r(n)$ делится на $r(m)$). А является ли $r(n)$ простым числом для простого n? Проверим для простых n меньших 1000.

```
PrimePi[1000]
168

Select[Prime[Range[168]], PrimeQ[(10^# – 1)/9]&]
{2, 19, 23, 317}
```

Нашлось всего четыре простых числа $r(2)$, $r(19)$, $r(23)$ и $r(317)$. Для остальных простых чисел p меньших 1000 числа $r(p)$ являются составными. Для больших простых p проверить с помощью функции PrimeQ, что $r(p)$ является простым, невозможно. Но можно использовать тест на разложимость. Число 3571 является простым. Проверим на разложимость число $r(3571)$.

```
PrimeQ[3571]
True

(PowerMod[2, # – 1, #] ≠ 1)&[(10^3571 – 1)/9]
True
```

Число $(10^{3571}–1)/9$ является составным.

Составное число n называется **псевдопростым числом Ферма** по основанию a, $1 < a < n – 1$, если выполнено сравнение

$$a^n \equiv a \pmod{n}. \qquad (1)$$

Найдем в пределах первой тысячи псевдопростые числа Ферма по основаниям 2 и 3:

```
pseudoprimeQ[n_, a_] := Not[PrimeQ[n]] && PowerMod[a, n, n] == a
Select[Range[3,1000,2], pseudoprimeQ[#, 2]&]
{341, 561, 645}

Select[Range[3,1000,2], pseudoprimeQ[#, 3]&]
{91, 121, 561, 671, 703, 949}
```

Теорема 2 [86, с. 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.

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 2 утверждает, что при фиксированном выборе основания a вероятные простые числа по основанию a скорее всего являются простыми.

Функция PowerMod[a, # – 1, #] = = 1&] реализует тест на вероятное простое число по основанию a.

Как сказано выше, вероятные простые числа, являющиеся составными, достаточно редки. Однако это не мешает существованию бесконечного множества таких чисел.

Предварительно докажем лемму.

Лемма. Пусть d и b – натуральные числа, тогда если d делит b, то $2^d - 1$ делит $2^b - 1$.

Доказательство. Пусть $b = d\,k$, тогда

$$2^b - 1 = 2^{d\,k} - 1 = (2^d)^k - 1 = (2^d - 1 + 1)^k - 1 \equiv 1^k - 1 = 0 \pmod{2^d - 1}. \blacksquare$$

Теорема 3 [20, с. 106]. Существует бесконечное множество псевдопростых чисел Ферма по основанию 2.

Доказательство. Мы построим бесконечный список псевдопростых чисел по основанию 2, показывая, что если n – псевдопростое число, то таким же является $m = 2^n - 1$. Лемма говорит, что если n – составное число, то таким же является и m. Нам остается показать, что $2^{m-1} \equiv 1 \pmod{m}$. Так как n – псевдопростое число по основанию 2, то $2^n = 2 \cdot 2^{n-1} \equiv 2 \pmod{n}$, следовательно, n делит $2^n - 2$. Положим, чтобы воспользоваться леммой, $d = n$, $b = 2^n - 2$. Тогда получаем по лемме, $2^n - 1$ делит

$$2^{(2^n-2)} - 1 = 2^{(2^n-1)-1} - 1 = 2^{m-1} - 1,$$

т.е. $2^{m-1} \equiv 1 \pmod{m}$. Что и требовалось доказать. ■

Доказательство теоремы подсказывает, как мы можем, начиная с 341, получить бесконечную последовательность псевдопростых чисел по основанию 2. Для компактного изображения чисел из указанной последовательности можно использовать функцию HoldForm языка Wolfram. Эта функция при вызове печатает свой аргумент в невычисленном виде. Приведем пример:

```
HoldForm[{100!, 2x + 3x, 100/50 + 2}]
```

$$\left\{100!, 2x + 3x, \frac{100}{50} + 2\right\}$$

Определим вспомогательную функцию

```
f[x_] := HoldForm[2^x – 1]
```

и напечатаем первые семь псевдопростых чисел, начиная с 341:

```
NestList[f, 341, 6]
```

$$\{341, 2^{341} - 1,\ 2^{2^{341}-1} - 1,\ 2^{2^{2^{341}-1}-1} - 1,\ 2^{2^{2^{2^{341}-1}-1}-1} - 1,$$
$$2^{2^{2^{2^{2^{341}-1}-1}-1}-1} - 1,\ 2^{2^{2^{2^{2^{2^{341}-1}-1}-1}-1}-1} - 1\}$$

Можно думать, что существуют такие основания a, для которых отсутствуют псевдопростые числа или их конечное число. Но это не так.

Теорема 4 [86, с. 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 теста (2) мы бы получили детерминированный тест распознавания простых чисел.

Рассмотрим дополнительно псевдопростые числа по основаниям 5 и 7, меньшие 1000:

```
Select[Range[1000], pseudoprimeQ[#, 5]&]
{15, 65, 217, 435, 561, 781}
Select[Range[1000], pseudoprimeQ[#, 7]&]
{21, 25, 105, 33, 231, 301, 325, 525, 561, 703, 817}
```

Отметим, что функция *pseudoprimeQ*[*n*, *a*] выдает True для некоторых *n* кратных *a*. Если удалить *n* кратные *a*, то получаем меньше составных чисел:

```
Select[Range[3, 1000, 2], pseudoprimeQ[#, 5] && Not[Divisible[#, 5]]&]
{217, 561, 781}
Select[Range[3, 1000, 2], pseudoprimeQ[#, 7] && Not[Divisible[#, 7]]&]
{25, 325, 561, 703, 817}
```

А если ли числа, которые сразу по четырем основаниям 2, 3, 5 и 7 являются псевдопростыми?

```
pseudoprimeManyQ[n_, s_] := AllTrue[s, pseudoprimeQ[n, #]&]
Select[Range[1000], pseudoprimeManyQ[#, {2, 3, 5, 7}]&]
{561}
```

Как мы видим, число 561 является псевдопростым числом Ферма по основаниям 2, 3, 5 и 7. Докажем, что на самом деле число $561 = 3 \cdot 11 \cdot 17$ является псевдопростым по всем основаниям a, для которых $\gcd(a, 561) = 1$. Чтобы показать, что $a^{560} \equiv 1 \pmod{561}$, достаточно (по теореме 2 (9) из главы 3) убедиться, что это сравнение имеет место $a^{560} \equiv 1$ по модулям 3, 11 и 17. Но $a^{560} \equiv a^{10 \cdot 56} \equiv 1^{56} \equiv 1 \pmod{11}$ по малой теореме Ферма. Подобным образом, $a^{560} \equiv a^{16 \cdot 35} \equiv 1^{35} \equiv 1 \pmod{17}$ и $a^{560} \equiv a^{2 \cdot 280} \equiv 1^{280} \equiv 1 \pmod{3}$. Сравнение $a^{560} \equiv 1 \pmod{561}$ приводит к сравнению $a^{561} \equiv a \pmod{561}$.

Нетрудно видеть, что $n = 561$ есть произведение нечетных простых чисел p_i и каждое $p_i - 1$ делит $n - 1$.

Кажется невероятным, что такие числа, как 561, существуют, но и в самом деле это так. Впервые они были найдены Р. Кармайклом в 1910 г., с тех пор такие числа носят имя этого математика.

Составные числа n, такие, что для любого целого a, $1 < a < n$, $\gcd(a, n) = 1$, выполнено сравнение (1), называются **числами Кармайкла**.

Числа Кармайкла легко распознать по разложению на простые множители.

Теорема 5 (критерий Корсельта [86, с. 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 [31], что, начиная с достаточно большого n, выполнено двойное неравенство

$$n^{2/7} < C(n) < n\exp(-\frac{\ln n \ln\ln\ln n}{\ln\ln n}). \quad (3)$$

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 тест будет более эффективным, чем тест (2).

Теорема 6[38] (вариант малой теоремы Ферма). Пусть n – нечетное простое число и $n - 1 = 2^s t$, где число t нечетно. Если число a не делится на n, то

$$\begin{cases} \text{или } a^t \equiv 1 (\mathrm{mod}\, n), \\ \text{или } a^{2^i t} \equiv -1 (\mathrm{mod}\, n) \text{ для некоторого } i, \text{ где } 0 \le i \le s-1. \end{cases} \quad (4)$$

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

$$a^{2^k t} - 1 = (a^{2^{k-1} t} - 1)(a^{2^{k-1} t} + 1).$$

[38] Теорема приведена в [86, с. 158] без доказательства.

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.

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 1 (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 mod n;
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 1 оставляет всего 1282 числа, не распознавая их как составные.

Более того, для алгоритма 1 не существуют числа, аналогичные кармайкловским числам. Это следует из результата, независимо полученного в работах [14, 18].

Теорема 7 (доступное доказательство см. в [86, с. 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.

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 7 утверждает, что если n – нечетное составное число, то как минимум 3/4 всех целых чисел отрезка $[1, n - 1]$ являются свидетелями для n. Поскольку программа *highlyProbablePrimeQ*[n, a] работает очень быстро, 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 1 со случайным выбором основания a.

Алгоритм 2 (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 1 решается, является ли n сильно вероятным простым числом по основанию a;
2. (* Объявление результата *)
If n – сильно вероятное простое число по основанию a
    then return(a, НЕТ)
    else return(a, ДА);

Согласно теореме 7, если $n > 9$ – нечетное составное число, то вероятность того, что алгоритму 2 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 боль-

шим, мы можем сделать эту вероятность пренебрежимо малой. Алгоритм 2 – 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 6 у простых чисел свидетелей нет.

Пусть нам неизвестно, является данное большое нечетное число n простым или составным. Предположим, что, скажем, 20 итераций алгоритма 2 не позволили найти для n 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 2 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 2 сделать меньше $n - 2$? Следующий результат – самое лучшее, что известно.

Теорема 8 [86, с. 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 $\chi(s)$ – так называемый характер Дирихле. Знаменитая гипотеза Римана и ее расширенный вариант еще не доказаны, но имеется много свидетельств в их справедливости (см. [70]).

Приведем детерминистический тест Миллера на простоту. Он основан на теореме 8 и при выполнении расширенной гипотезы Римана устанавливает простоту числа n за полиномиальное время относительно количества цифр в n.

Алгоритм 3 (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 число n или простое, или расширенная гипотеза Римана не верна.

$a = 2$;
While $a \le \lfloor 2 \ln^2 n \rfloor$ do
  begin
  При помощи алгоритма 1 проверяем, является ли число n
  сильно вероятным простым числом по основанию a;
  If n не является 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 [86, с. 161] можно применять для получения случайных чисел, которые, скорее всего, являются простыми.

Алгоритм 4 (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 2.

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 2 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}}]
501720 654 095 427 233 202 505 694 811 793

§ 4. Вероятные тесты на простоту, связанные с числами Фибоначчи

В этом параграфе рассматриваются псевдопростые числа Ферма по основанию 2 и псевдопростые числа Фибоначчи. Проверяется, существуют ли нечетные составные числа вида $n \equiv \pm 2 \pmod 5$, которые были бы одновременно псевдопростыми числами Ферма и Фибоначчи. Кроме обычного теста Фибоначчи предлагаются еще два теста, связанные с числами Фибоначчи. Таким образом, всего проверялось три теста на простоту, и вычисления показали, что до 10^{10} данные тесты работают правильно. Найдены формулы для аппроксимации распределения псевдопростых чисел Фибоначчи. Полученные результаты опубликованы в [79].

Так как псевдопростых чисел Ферма по любому основанию бесконечно много, поэтому соотношение (1) не может служить тестом на простоту.

Последовательность чисел Фибоначчи F_n (нумерация начинается с 1) обладает следующим свойством [86, с. 166]:

Теорема 1(9). Если n – простое число, то

$$F_{n-\varepsilon_n} \equiv 0 (\mathrm{mod}\, n), \tag{5}$$

где ε_n есть символ Лежандра $(\frac{n}{5})$. Имеем $(\frac{n}{5}) = 1$ при $n \equiv \pm 1 (\mathrm{mod}\ 5)$, $(\frac{n}{5}) = -1$ при $n \equiv \pm 2 \pmod 5$ и $(\frac{n}{5}) = 0$ при $n \equiv 0 \pmod 5$.

Составные числа, для которых выполнено (5) называются псевдопростыми числами Фибоначчи. Отношение (5) также не может служить тестом на простоту, так как псевдопростых чисел Фибоначчи бесконечно много. Более того, псевдопростых чисел Фибоначчи вида $n \equiv \pm 1 \pmod 5$ бесконечно много [20, с. 127, 128].

Неизвестно, существует ли нечетное составное число n вида $n \equiv \pm 2 \pmod 5$, которое одновременно являлось бы псевдопростым числом Ферма по основанию 2 и псевдопростым числом Фибоначчи. В [86, с. 194] даже предлагается денежный приз за нахождение такого числа или опровержение его существования.

Кроме (5) существуют еще и другие, связанные с числами Фибоначчи, необходимые условия для простых чисел.

Теорема 10. Если n – простое число, то

$$F_n \equiv (\tfrac{n}{5}) (\mathrm{mod}\, n). \tag{6}$$

Теорема 11. Если n – простое число большее 5, то

$$F_{2n} \equiv (\tfrac{n}{5}) (\mathrm{mod}\, 2n). \tag{7}$$

Эти теоремы есть частные случаи теоремы 12, сформулированной в конце параграфа.

Каждая из трех пар свойств (1) и (5), (1) и (6), (1) и (7) является потенциальным тестом на простоту для нечетных чисел вида $n \equiv \pm 2 \pmod 5$.

Если рассматривать нечетные числа вида $n \equiv \pm 1 (\mathrm{mod}\ 5)$ то легко обнаружить контрпримеры. В диапазоне до 10^6 среди нечетных составных чисел вида $n \equiv \pm 1 (\mathrm{mod}\ 5)$ встречаются:

1) 15 чисел (наименьшее среди них 6601), имеющих свойства (1) и (5);

2) 7 чисел (наименьшее число 219 781), имеющих свойства (1) и (6);

3) 14 чисел (наименьшее число 6601), имеющих свойства (1) и (7) .

Отношение (1) проверялось для основания $a = 2$.

Причем в данном диапазоне множество всех чисел со свойствами (1) и (5) включает множество всех чисел со свойствами (1) и (6) и множество всех чисел со свойствами (1) и (7).

Был произведен поиск нечетных составных чисел n вида $n \equiv \pm 2 \pmod 5$, которые были бы контрпримером того, что каждая из трех пар свойств (1) и (5), (1) и (6), (1) и (7) является потенциальным тестом на простоту. Отношение (1) проверялось для $a = 2$.

Обозначим через M множество всех меньших 10^{10} нечетных составных чисел n вида $n \equiv \pm 2 \pmod 5$. Необходимо вычислять для каждого $n \in M$ значения $2^n \bmod n$, $F_{n+1} \bmod n$, $F_n \bmod n$ и $F_{2n} \bmod 2n$ (здесь mod обозначает операцию нахождения остатка от деления).

Хорошо известны быстрые алгоритмы вычисления $2^n \bmod n$. Простейший из них – рекурсивный алгоритм с приведением по модулю при каждом умножении имеет вычислительную сложность $O(\ln n)$; при промежуточных вычислениях числа не превосходят n^2.

Для вычисления чисел $F_{n+1} \bmod n$, $F_n \bmod n$ и $F_{2n} \bmod 2n$ использовалась функция *fibMod*[*n_*, *m_*] , определенная в § 3 главы 5.

Алгоритм поиска контрпримеров достаточно прост. Опишем его для случая условий (1) и (5) и диапазона натуральных чисел [min, max].

1. Создаем список чисел $A_1 = [\min, \max] \cap M$.

2. Из списка A_1 выбираем псевдопростые числа Фибоначчи, удовлетворяющие условию (5). Получаем список чисел A_5.

3. Из списка A_5 выбираем псевдопростые числа Ферма по основанию 2, удовлетворяющие условию (1). Получаем A_6 – список чисел-контпримеров.

Если переставить местами шаги 2 и 3, то алгоритм работает быстрее, так как нахождение псевдопростых чисел Ферма требует меньше времени, чем псевдопростых Фибоначчи. Но получение полного списка псевдопростых чисел Фибоначчи A_6 понадобилось для изучения распределения этих чисел.

Этот алгоритм естественен для функционального программирования, так как единственная сложная структура – списки и операции над списками – фильтрация, т.е. выделение элементов списка, удовлетворяющих некоторому условию. Поскольку списки весьма длинные, то естественно фильтрацию списка проводить параллельно. С помощью функции PrimeQ создается список A_1.

Например, следующий код создает список $[1, 10^{10}] \cap M$.

```
Parallelize[Select[Range[1, 10^10, 2], Not[PrimeQ[#]] && (Mod[#, 5] ==
2 || Mod[#, 5] == 3)&]];
```

В результате проверки ни одного контрпримера не найдено. Тем самым, возможно, каждая из трех пар свойств (1) и (5), (1) и (6), (1) и (7) является тестом на простоту для нечетных чисел вида $n \equiv \pm 2 \pmod 5$.

Будем называть составные нечетные числа вида $n \equiv \pm 2 \pmod 5$ псевдопростыми числами Фибоначчи, указывая дополнительно, какое из условий (5), (6) или (7) выполняется. Обозначим соответствующие множества псевдопростых чисел Фибоначчи меньших 10^{10} через S_5, S_6 и S_7. Индексы в обозначениях этих множеств соответствуют условиям (5), (6) и (7). Нас интересуют вопросы о распределении и свойствах псевдопростых чисел Фибоначчи.

Приведем количественные оценки. Прежде всего, заметим, что среди чисел меньших 10^{10} имеется приблизительно $1.5 \cdot 10^8$ составных нечетных чисел вида $n \equiv \pm 2 \pmod 5$ и среди них – 14887 псевдопростых чисел Ферма по основанию 2.

Множество S_5 содержит 3498 чисел, а во множестве S_6 чисел более чем в четыре с половиной раза меньше – их 756. Причем только 13 чисел принадлежит разности $S_6 \setminus S_5$. Количество чисел во множество S_7 больше чем чисел в S_5. Проверка условий (5), (6) и (7) имеет одинаковую вычислительную сложность $O(\ln n)$, но мультипликативный мно-

житель в оценке сложности для условия (7) больше, чем для условий (5) и (6). Из всего этого можно заключить, что тест на простоту с условиями (1) и (6) предпочтительней, чем два других рассматриваемых теста.

При поиске контрпримеров была побочно выдвинута гипотеза, которая была подтверждена для чисел, меньших 10^{10}.

Каждое псевдопростое числа Фибоначчи, удовлетворяющее условию (6) *или* (7)*, свободно от квадратов.*

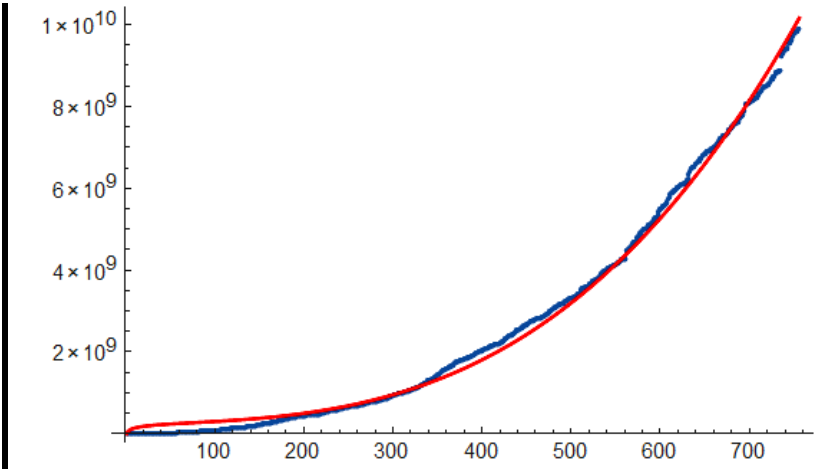
Для изучения распределения псевдопростых чисел Фибоначчи, удовлетворяющих условию (6), введены две целочисленных последовательности:

$psp(n)$ = n-е псевдопростое число Фибоначчи
(нумерация начинается с 1);

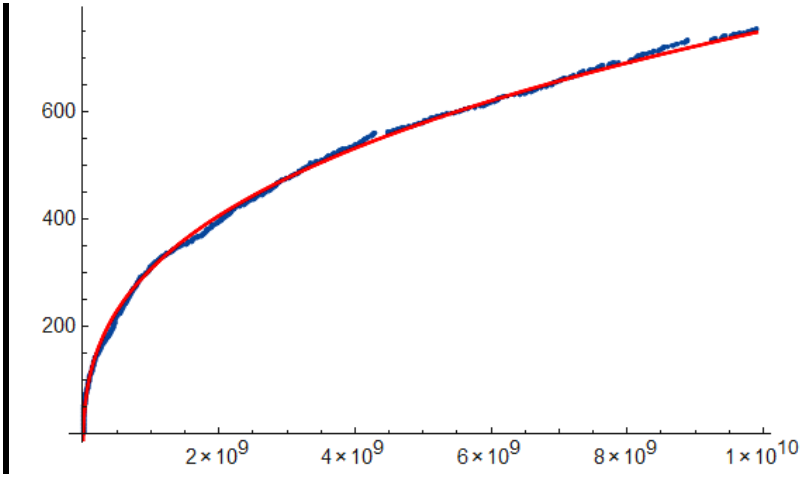
$psppi(n)$ = количество псевдопростых чисел Фибоначчи,
не превосходящих n.

Достаточно просто найти математические функции для аппроксимации этих последовательностей. Как это делается в общем случае, смотрите пример в § 4. «Самоописательная последовательность» Голомба из главы 2. Для последовательности $psp(n)$ аппроксимирующая функция искалась среди функций $c_1 n^3 + c_2 \ln n$, а для последовательности $psppi(n)$ – среди функций $c_1 n^{1/3} + c_2 \ln n$.

На интервале от 1 до 10^{10} найдены аппроксимации $psp(n) \approx 22{,}6n^3 + 6{,}0 \cdot 10^7 \ln n$ (см. следующий график)



и $psppi(n) \approx 0{,}4n^{1/3} - 4{,}1\ln n$ (график ниже).



Кроме свойств (6) и (7) получены и другие необходимые свойства простых чисел. Теоремы 10 и 11 являются частными случаями теоремы 12 для $q = 1$ и $q = 2$.

Теорема 12. Пусть Q – множество натуральных чисел q, удовлетворяющих любому из условий:

1. $F_q \equiv 0 \pmod q$.
2. Пусть $q = 2t$, причем t не делится на 3 и t – делитель F_q.

Тогда для каждого простого $p > \max(5, f_q)$ имеем

$$F_{qp} \equiv F_q \cdot (\tfrac{p}{5}) \pmod{qp}. \qquad (8)$$

Первые числа q, принадлежащие множеству Q, суть 1, 2, 5, 10, 12, 24, 25, 36, 48, 50, 60, 72, 96, 108, 110, 120, 125, 144, 168, 180, 192, 216, 240.

Данное утверждение есть переформулировка следствий из теоремы 7, доказанных в §3 главы 5.

Можно было бы проверить, не являются ли тестами на простоту условия (1) и (8) для некоторых $q > 2$ из вышеприведенного списка. Вычислительная сложность таких тестов не будет превышать сложность теста с условиями (1) и (6), но мультипликативный множитель в оценке сложности для условия (8) больше, чем для условия (6).

Глава 10

СИЛЬНО СОСТАВНЫЕ ЧИСЛА

§ 1. Вычисление сильно составных чисел

Определение. Положительное целое n называется **сильно составным числом**, если для всех целых $m < n$ выполнено $\tau(m) < \tau(n)$.

Понятие сильно составного числа ввел Сриниваса Рамануджан [19]. Легко проверить, что первые шесть сильно составных чисел есть 1, 2, 4, 6, 12, 24. Принято считать, что 1 и 2 являются сильно составными, хотя они даже не составные. Современные знания о сильно составных числах описаны в [37].

Условимся использовать сокращение «*hc*-число» вместо «сильно составное число».

Предложение 1. Пусть h – *hc*-число и для натурального m имеем $\tau(h) < \tau(m)$, тогда существует такое *hc*-число k, что $h < k \le m$.

Доказательство. Из сильносоставности h следует неравенство $\tau(n) < \tau(h) < \tau(m)$ для любого $n < h$. Пусть k – наименьшее целое, для которого выполнено $h < k$ и $\tau(k) \ge \tau(m)$. Если такого числа k нет между h и m, то положим $k = m$. Так что существование такого числа k гарантировано, причем, по определению k, имеем $\tau(n) < \tau(m)$ для всех $n = h + 1$, $h + 2$, …, $k - 1$. Тогда $\tau(k) > \tau(n)$ для любого $n < k$ и, следовательно, k – *hc*-число. ■

Следствие. Сильно составных чисел бесконечно много.

Доказательство. Так как, например, $\tau(2^{\alpha}) = \alpha + 1$ может быть произвольно большим, то число m, у которого $\tau(m) > \tau(h)$, всегда существует и, следовательно, для каждого *hc*-числа существует большее *hc*-число. ■

Предложение 2. Если h есть наибольшее *hc*-число, меньшее или равное n, то максимальное количество делителей у чисел, меньших или равных n, есть $\tau(h)$.

Доказательство. Если $m \in [h, n]$, то $\tau(m) \le \tau(h)$; ибо наименьшее число $m > h$, обладающее свойством $\tau(m) > \tau(h)$, было бы *hc*-числом, что противоречит определению h. ■

Пусть h_n обозначает n-е по порядку возрастания *hc*-число, т.е. $h_1 = 1$, $h_2 = 2$, $h_3 = 4$, $h_4 = 6$ и т.д. Можно указать верхнюю границу для *hc*-числа в зависимости от его порядкового номера.

Предложение 3. Пусть n – натуральное число. Тогда, $h_n \le 2^n$.

Доказательство. Покажем сначала, что если $m > 0$ – натуральное число, то существует такое *hc*-число k, что $m < k \le 2m$. Пусть a будет наибольшее *hc*-число, меньшее или равное m. Заметим, что $2a \le 2m$ и $\tau(a) < \tau(2a)$. В силу предложения 1 существует такое *hc*-число k, что $a < k \le 2a$. Если $k \le m$, то это противоречит выбору a. Поэтому $m < k \le 2m$.

Отсюда следует, что если будем брать степени двойки в качестве m: $2^0, 2^1, 2^2, \ldots$ то существует также *hc*-число h, такое, что $2^n < h \le 2^{n+1}$, где числа n начинаются с 0. Поэтому имеется, по крайней мере, n *hc*-чисел, не превосходящих 2^n. Поэтому для $n > 0$ имеем $h_n \le 2^n$. ■

Эта оценка конечно очень завышенная. Следующая простая программа на языке Wolfram выдает все 56 *hc*-чисел, не превосходящих 10^8.

```
s = {}; a = 0;
Do[b = DivisorSigma[0, n]; If[b > a, a = b; AppendTo[s, n]], {n, 10^8}];
s
```

```
{1, 2, 4, 6, 12, 24, 36, 48, 60, 120, 180, 240, 360, 720, 840, 1260, 1680,
2520, 5040, 7560, 10080, 15120, 20160, 25200, 27720, 45360, 50400,
55440, 83160, 110880, 166320, 221760, 277200, 332640, 498960,
554400, 665280, 720720, 1081080, 1441440, 2162160, 2882880,
3603600, 4324320, 6486480, 7207200, 8648640, 10810800, 14414400,
17297280, 21621600, 32432400, 36756720, 43243200, 61261200,
73513440}
```

Программа, реализуя алгоритм «грубой силы», перебирает последовательно натуральные числа, вычисляя каждый раз сумму делителей, и поэтому работает медленно. Для создания более быстрого алгоритма рассмотрим некоторые свойства *hc*-чисел.

Пусть p_i, $i > 1$, есть строго возрастающая последовательность простых чисел. Факторизации числа

$$n = p_{i_1}^{\alpha_1} \cdots p_{i_r}^{\alpha_r}, \quad 1 \le i_1 < \cdots < i_r,$$

на простые множители сопоставим число

$$\boldsymbol{n} = p_1^{\beta_1} \cdots p_r^{\beta_r}, \quad \beta_1 \ge \beta_2 \ge \cdots \ge \beta_r, \tag{1}$$

где показатели степеней β_i получены из α_i переупорядочиванием в убывающем порядке. Например, для $n = 3^5 \cdot 7^2 \cdot 13^3 \cdot 17^2 \cdot 23^9$ получаем число $\boldsymbol{n} = 3^9 \cdot 7^5 \cdot 13^3 \cdot 17^2 \cdot 23^2$.

Теорема 1 (Рамануджан). Имеем $\tau(\boldsymbol{n}) = \tau(n)$ и $\boldsymbol{n} \le n$.

Доказательство. Первое равенство следует из формулы для количества делителей $\tau(n) = (\alpha_1 + 1) \cdots (\alpha_r + 1)$ и из (1). Чтобы получить неравенство, заметим, что так как $(p^u q^v)/(p^v q^u) = (p/q)^{u-v}$, то из $u \ge v$ и $p \le q$ получаем $p^u q^v \le p^v q^u$. Если мы переставим α_i, чтобы получить β_i, мы уменьшим n:

$$p_{i_1}^{\beta_1} \cdots p_{i_r}^{\beta_r} \le p_{i_1}^{\alpha_1} \cdots p_{i_r}^{\alpha_r} = n.$$

Завершаем доказательство очевидным неравенством

$$\boldsymbol{n} \le p_{i_1}^{\alpha_1} \cdots p_{i_r}^{\alpha_r}. \ \blacksquare$$

Следствие. Любое *hc*-число n имеет вид

$$n = 2^{\alpha_2} 3^{\alpha_3} \cdots p^{\alpha_p}, \quad \alpha_2 \ge \alpha_3 \ge \cdots \ge \alpha_p \ge 1. \tag{2}$$

С. Рамануджан оценил значения α_i в факторизации (2).

Теорема 2 (Рамануджан). Пусть $h = 2^{\alpha_2} 3^{\alpha_3} \cdots p^{\alpha_p}$ есть *hc*-число и P – простое число, которое следует после p. Тогда для каждого $q \le p$ имеем

$$\left\lfloor \frac{\log p}{\log q} \right\rfloor \le \alpha_q \le 2 \left\lfloor \frac{\log P}{\log q} \right\rfloor. \tag{3}$$

Доказательство. Ограничим сначала α_q снизу. Предполагаем, что $q < p$; учитывая, что

$$q^{\frac{\log p}{\log q}} = p,$$

получаем $q^x < p < q^{x+1}$, где $x = \lfloor \log p / \log q \rfloor$. Рассмотрим число

$$k = \frac{h}{p} q^x < h.$$

Имеем $\tau(k) < \tau(h)$, так как h – сильно составное число и $k < h$. Сокращая общие сомножители в обеих сторонах неравенства, получаем

$(\alpha_q + x + 1)\alpha_p < (\alpha_q + 1)(\alpha_p + 1)$. Решаем неравенство (используя функцию Reduce языка Wolfram):

```
Reduce[{α_q > 1, α_p > 1, (α_q + x + 1) α_p < (α_q + 1)(α_p + 1)}, x]
```

$$\alpha_q > 1 \,\&\&\, \alpha_p > 1 \,\&\&\, x < \frac{1+\alpha_q}{\alpha_p}$$

Так как x – целое, то $x \le \alpha_q$. В случае $q = p$ имеем $x = 1$ и поэтому $x \le \alpha_q$.

Оценим теперь α_q сверху. Положим $y = \lfloor \log P / \log q \rfloor$, что влечет $q^y < P < q^{y+1}$. Если $\alpha_q > y + 1$, то мы можем рассмотреть число

$$k = \frac{h}{q^{y+1}} P < h.$$

Как и прежде, имеем $\tau(k) < \tau(h)$, так как h – hc-число и $k < h$. Сокращая общие сомножители в обеих сторонах неравенства, получаем $2(\alpha_q - y) < \alpha_q + 1$. Отсюда следует, что $\alpha_q < 2y + 1$. ■

Утверждение теоремы 2 имеется в [19], где оно является следствием более сильного результата, принадлежащего С. Рамануджану. Приведенное здесь доказательство является более простым прямым доказательством теоремы.

Замечание. С. Рамануджан уточнил [37], что для наибольшего простого делителя p в разложении (2) показатель α_p всегда равен 1, если $p > 3$.

Если p – наибольший простой делитель в разложении (2) hc-числа h, то теорема 2 позволяет оценить диапазоны значений показателей α_i. С учетом уточнения Рамануджана функция $d[p]$ выдает границы диапазонов возможных значений α_i.

```
d[2] = {{1, 2}};
d[3] = {{1, 2, 3, 4}, {1, 2}};
d[p_] := Append[Most[Table[Range[⌊Log[p] / Log[Prime[k]]⌋,
                2 ⌊Log[NextPrime[p]] / Log[Prime[k]]⌋], {k,
PrimePi[p]}]], {1}]
```

Например, для $p = 7$ получаем

```
d[7]
{{2, 3, 4, 5, 6}, {1, 2, 3, 4}, {1, 2}, {1}}
```

То есть, $2 \le \alpha_2 \le 6$, $1 \le \alpha_3 \le 4$, $1 \le \alpha_5 \le 2$, $\alpha_7 = 1$.

Для описания алгоритмов вычисления *hc*-чисел введем необходимую терминологию и обозначения.

Число n называется **потенциально сильно составным числом**, если для n выполнены условия (2) и (3). Множество всех потенциально сильно составных чисел будем обозначать Λ.

Если $n \in \Lambda$, то функция $maxpd(n)$ выдает в качестве значения наибольший простой делитель n.

Если p – простое число, то обозначим

$$min(p) = \text{минимум } \{n \mid n \in \Lambda \text{ и } maxpd(n) = p\},$$

$$max(p) = \text{максимум } \{n \mid n \in \Lambda \text{ и } maxpd(n) = p\}.$$

Если p – простое число, то определим множество $\Lambda_p = \{n \mid n \in \Lambda$ и $maxpd(n) = p\}$. Очевидно, Λ_p – подмножество замкнутого интервала $[min(p), max(p)]$.

Множество чисел Λ_p вычисляется с помощью вложенного цикла. Если $p > 3$, то вложенность на 1 меньше порядкового номера простого числа p.

Пример 1. Например, для $p = 7$ имеем код на языке Wolfram (значение переменной *s*7 после выполнения есть список элементов из Λ_7)

```
s7 = {}; Do[Do[Do[AppendTo[s7, Times @@ {2^i2, 3^i3, 5^i5, 7}],
   {i5, Min[i3, 2], Min[i3, 1], –1}], {i3, Min[i2, 4],
    Min[i2, 1], –1}], {i2, 6, 2, –1}]; s7
{907200, 181440, 302400, 60480, 100800, 20160, 6720, 453600, 90720,
151200, 30240, 50400, 10080, 3360, 226800, 45360, 75600, 15120,
25200, 5040, 1680, 37800, 7560, 12600, 2520, 840, 6300, 1260, 420}
```

Из определения *hc*-чисел следует, что их вычислять можно только по очереди в порядке возрастания. Для этого достаточно просматривать в возрастающем порядке потенциально сильно составные числа. Рассматривая первые 56 *hc*-чисел, вычисленные с помощью «грубой силы», обнаруживаем, что если h_1 и h_2 – какие-то два подряд идущих *hc*-числа, то может быть выполнено любое из соотношений $maxpd(h_1) = maxpd(h_2)$, $maxpd(h_1) < maxpd(h_2)$ или $maxpd(h_1) > maxpd(h_2)$. Это свойство естественно сохраняется, если h_1 и h_2 – два подряд идущих потенциально сильно составных числа.

Предлагаемые ниже алгоритмы вычисления *hc*-чисел представляют все множество Λ в виде объединения непересекающихся диапазонов потенциально сильно составных чисел. Эти диапазоны могут быть открытыми, замкнутыми или полуоткрытыми интервалами. Таким обра-

зом, получение всех *hc*-чисел сводится к вычислению *hc*-чисел в последовательных интервалах.

Для определенности возьмем некоторое простое число q и рассмотрим, как разбивается множество $[2, min(q))\cap\Lambda$ на интервалы. Для этого сначала определим числа, которые будут концами этих интервалов. Пусть последовательность B состоит из упорядоченных по возрастанию элементов множества

$$\{b \mid \exists\ (\text{простое } p < q)\ ((b = min(p)) \vee (b = max(p))) \ \&$$

$$\&\ (b < min(q))\} \cup \{min(q)\}. \qquad (4)$$

Заметим, что для каждого b значение $maxpd(b)$ равно некоторому простому числу, не превосходящему q.

Для вычисления элементов последовательности B необходимо предварительно вычислить элементы всех множеств Λ_p, где $p \le q$. А далее числа b определяем из (4). Список чисел множества Λ_p определяем для каждого p с помощью программы, подобной программе в примере 1.

Пример 2. Рассмотрим случай Λ_{19}. Сначала вычислим границы для показателей степеней простых сомножителей в факторизации чисел из Λ_{19}.

```
d[19]
{{4, 5, 6, 7, 8}, {2, 3, 4}, {1, 2}, {1, 2}, {1, 2}, {1, 2}, {1, 2}, {1}}
```

Программа для вычисления чисел из Λ_{19}:

```
s19 = {}; Do[Do[Do[Do[Do[Do[Do[Do[AppendTo[s19, Times@@{2^i2,
3^i3, 5^i5, 7^i7, 11^i11, 13^i13, 17^i17, 19}], {i17, Min[i13, 2], Min[i13,
1], –1}], {i13, Min[i11, 2], Min[i11, 1], –1}], {i11, Min[i7, 2], Min[i7, 1], –
1}], {i7, Min[i5, 2], Min[i5, 1], –1}], {i5, Min[i3, 2], Min[i3, 1], –1}], {i3,
Min[i2, 4], Min[i2, 2], –1}], {i2, 8, 4, –1}]; s19
```

Выполнив эту программу, найдем $min(19) = 232792560$, $max(19) = 2852230315334400$.

Пример 3. Последовательность B для разбиения множества $[2, min(29)) \cap \Lambda$ на интервалы. Вычисляем $min(p)$ и $max(p)$ для всех простых $p \le 29$ (как в примере 2). Упорядочивая по возрастанию полученные числа, получаем последовательность B:

$$2 = min(2),\ 4 = max(2),\ 6 = min(3),\ 60 = min(5),\ 144 = max(3),$$
$$420 = min(7),\ 720 = max(5),\ 27720 = min(11),\ 360360 = min(13),$$
$$907200 = max(7),\ 12252240 = min(17),\ 69854400 = max(11),$$
$$232792560 = min(19),\ 5354228880 = min(23),$$
$$39956716800 = max(13),\ 2329089562800 = min(29).$$

Теперь нужно для каждого элемента $b \in B$ определить, в какой интервал S_i входит b в качестве граничного элемента. Каждый интервал S_i будет характеризоваться своим множеством $D_i = \{maxpd(n) \mid n \in S_i\}$. Множества D_i и S_i одновременно определяются рекурсивно по следующим правилам:

1. Начало. S_1 есть замкнутый интервал [2, 4] и $D_1 = \{2\}$. Очевидно, S_2 есть полузамкнутый интервал [6, 60) и $D_2 = \{3\}$.

Пусть D_i и S_i определены. Рекурсивный переход к D_{i+1} и S_{i+1} осуществляются разбором случаев.

2. Интервал S_i открыт справа. Это происходит только тогда, когда для S_i правый граничный элемент $b = min(p)$ для некоторого p. В этом случае b входит в качестве первого элемента в S_{i+1} и интервал S_{i+1} является замкнутым слева. Полагаем $D_{i+1} = D_i \cup \{p\}$.

3. Интервал S_i замкнут справа. Это происходит только тогда, когда для S_i правый граничный элемент $b = max(p)$ для некоторого p. В этом случае интервал S_{i+1} открыт слева и полагаем $D_{i+1} = D_i \setminus \{p\}$.

4. Правая граница интервала S_{i+1} зависит только от значения правого b для S_{i+1}. Если $b = max(p)$ для некоторого p, то интервал S_{i+1} замкнут справа и данное b принадлежит S_{i+1}. Если $b = min(p)$ для некоторого p, то интервал S_{i+1} открыт справа и данное b не принадлежит S_{i+1}.

Пример 4. Пусть $q = 29$. В соответствии с примером 3 получаем следующее разбиение $[2, min(q)) \cap \Lambda$ на интервалы:

$$S_1 = [2, 4] \text{ и } D_1 = \{2\},\ S_2 = [6, 60) \text{ и } D_2 = \{3\},$$
$$S_3 = [60, 144] \text{ и } D_3 = \{3, 5\},\ S_4 = (144, 420) \text{ и } D_4 = \{5\},$$
$$S_5 = [420, 720] \text{ и } D_5 = \{5, 7\},\ S_6 = (720, 27720) \text{ и } D_6 = \{7\},$$
$$S_7 = [27720, 360360) \text{ и } D_7 = \{7, 11\},\ S_8 = [360360, 907200]$$
$$\text{и } D_8 = \{7, 11, 13\},\ S_9 = (907200, 12252240) \text{ и } D_9 = \{11, 13\},$$
$$S_{10} = [12252240, 69854400] \text{ и } D_{10} = \{11, 13, 17\},$$
$$S_{11} = (69854400, 232792560) \text{ и } D_{11} = \{13, 17\},$$
$$S_{12} = [232792560, 5354228880) \text{ и } D_{12} = \{13, 17, 19\},$$
$$S_{13} = [5354228880, 39956716800] \text{ и } D_{13} = \{13, 17, 19, 23\},$$
$$S_{14} = (39956716800, 2329089562800) \text{ и } D_{14} = \{17, 19, 23\}.$$

Теперь все готово для описания алгоритма вычисления *hc*-чисел.

Алгоритм 1. Вычисление *hc*-чисел из интервала разбиения множества$[2, min(q)) \cap \Lambda]$. Входные данные: 1) некоторый интервал S с концами $b1$ и $b2$ (конец интервала может входить или не входить в интервал); 2) множество простых чисел $D = \{maxpd(n) \mid n \in S\}$; 3) наибольшее *hc*-

число h, меньшее всех чисел из интервала S. Выход: Γ – множество *hc*-чисел из интервала S.

1. Для каждого $p \in D$ выполняем следующее:

1*a*. Находим множество M всех последовательностей натуральных чисел $\alpha_2, \alpha_3, \alpha_5, ..., \alpha_p$, удовлетворяющих условиям (2) и (3).

1*b*. Для каждой последовательности $\alpha_2, \alpha_3, \alpha_5, ..., \alpha_p$ из M вычисляем число

$$n = 2^{\alpha_2} 3^{\alpha_3} \cdots p^{\alpha_p}.$$

Множество полученных таким образом чисел n есть множество Λ_p.

1*c*. Учитывая границы интервала S, получаем множество потенциально сильно составных чисел $S \cap \Lambda_p$.

2. Объединяем все множества $S \cap \Lambda_p$, полученные на шаге 1:

$$\bigcup_{p \in D} (S \cap \Lambda_p) = S \cap \Lambda.$$

3. Упорядочиваем по возрастанию множество $S \cap \Lambda$. Получили возрастающую последовательность всех потенциально сильно составных чисел из интервала S. Обозначим эту последовательность через T.

4. Из последовательности T извлекаем только *hc*-числа. Создаем последовательность Γ по возрастанию *hc*-чисел. Первоначально, $\Gamma = \{h\}$. Последовательно проходим числа из T и каждый раз, когда встречаем число n со свойством $\tau(n) > \tau(m)$ (m есть последнее число в Γ), добавляем n в конец Γ. Завершая шаг 4, удаляем первый элемент h из Γ.

Программа для алгоритма 1 не приводится, поскольку вычисления по данному алгоритму на языке Wolfram на сервере с двумя процессорами Intel Xeon E5-2620 v2 оказались медленными и требующими наличия большой оперативной памяти у компьютера.

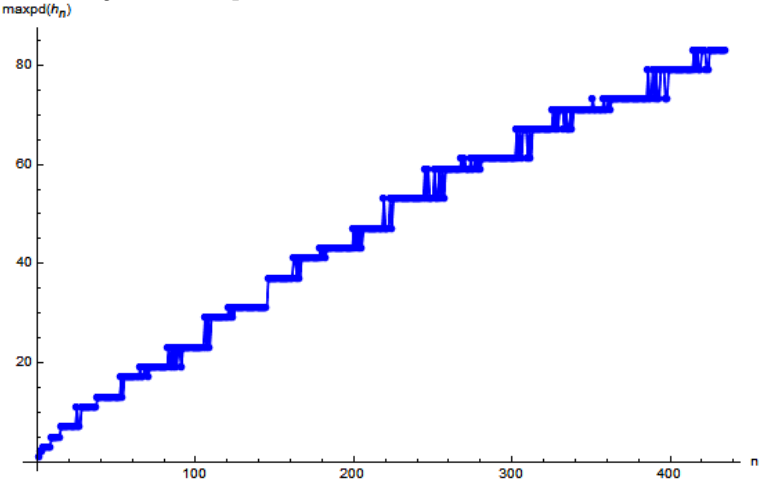
Но тем не менее были найдены первые 435 *hc*-чисел. Покажем последнее число:

$$h_{435} = 692923455891269609720491232091948991680000.$$

Имеем $maxpd(h_{435}) = 83$.

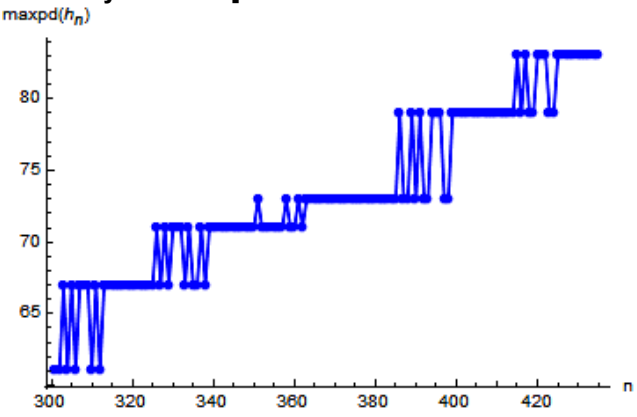
Для создания нового эффективного алгоритма было исследовано, какие значения принимает наибольший простой делитель для найденных *hc*-чисел. Следующий код выдает график зависимости $maxpd(h_n)$ от номера n, который меняется от 1 до 435. Список *hc*435 состоит из первых 435 *hc*-чисел, перечисленных по возрастанию.

```
ListPlot[FactorInteger[#][[-1, 1]]& /@ hc435, Joined → True,
  AxesLabel → {"n", "maxpd(hn])"}, PlotTheme → "Monochrome",
  PlotStyle → Blue]
```



Чтобы более детально рассмотреть график указанной зависимости для h_n, выберем диапазон от $n = 301$ до $n = 435$. Предварительно создаем список пар вида $\{n, h_n\}$:

```
pairs = MapIndexed[{First[#2] + 300, #1}&,Take[hc435, {301, 435}]];
ListPlot[{#[[1]], FactorInteger[#[[2]]][[-1, 1]]}& /@ pairs, Joined → True,
  AxesLabel → {"n", "maxpd(hn])"}, PlotTheme → "Monochrome",
  PlotStyle → Blue]
```



Изучение графиков привело к гипотезе, но прежде чем её сформулировать, введем обозначение

$$H(p) = \{n \mid n - \text{сильно составное число и } maxpd(n) = p\}.$$

Гипотеза. Пусть p_0, p_1 и p_2 – три подряд идущих простых числа. Обозначим через n_1, n_2 – наименьшие числа множеств $H(p_1)$ и $H(p_2)$ соответственно. Тогда, если n – сильно составное число $n_1 \le n < n_2$, то $maxpd(n)$ равно p_0 или p_1.

В соответствии с этим предположением *hc*-числа можно вычислять с помощью следующего алгоритма.

Алгоритм 2. Вход: p_0, p_1 и p_2 – три подряд идущих простых числа и n_1 – наименьшие число множества $H(p_1)$. Выход: n_2 и все *hc*-числа n в интервале $n_1 \le n \le n_2$.

1. Находим $\Lambda(p_0)$, $\Lambda(p_1)$ и $\Lambda(p_2)$. (Вместо обозначения Λ_x используется $\Lambda(x)$).

2. Получаем список s – упорядоченные по возрастанию потенциально сильно составные числа из объединения

$$\{n \mid n \in \Lambda(p_0) \text{ и } n > n_1\} \cup \Lambda(p_1) \cup \Lambda(p_2).$$

3. Находим все *hc*-числа из списка s, не превосходящие n_2.

В примере 2 описывается, как находятся множества в шаге 1. Если $s0$, $s1$, $s2$ – списки чисел, входящих во множества $\Lambda(p_0)$, $\Lambda(p_1)$ и $\Lambda(p_2)$ соответственно, то список s в шаге 2 получаем как

```
s = Sort[Join[Select[s0, # > n1&], s1, s2]];
```

Шаг 3 реализуется функцией

```
hcRange[s_, n1_, p2_] :=
  Module[{r = {}, a = DivisorSigma[0, n1], go = True, b, i = 1,
      l = Length[s]},
    While[go && i <= l, b = DivisorSigma[0, s[[i]]]; If[b > a, a = b;
        AppendTo[r, s[[i]]];
        If[FactorInteger[s[[i]]][[-1,1]] == p2, go = False]]; i++
      ]; r
    ]
```

Аргумент $p2$ у функции *hcRange* используется для нахождения n_2: если сумма делителей очередного числа n из списка s стала равна $p2$, то полагаем n_2 равным n и поиск *hc*-чисел заканчивается. Вычисление значений функции DivisorSigma осуществляется быстро, так как потенциально сильно составные числа легко факторизуются.

Применив алгоритм 2 для входа p_0, p_1 и p_2 и *hc*-числа n_1 и получив результат, можно запустить этот алгоритм для входа p_1, p_2 и p_3 и *hc*-числа n_2, где p_3 – следующее простое число большее p_2. Алгоритм 2 не нуждается в параллельных вычислениях и не требует для выполнения много оперативной памяти. Фактически соответствующая программа на языке Wolfram выполнялась на персональном компьютере.

Было вычислено 1003 сильно составных числа. Число h_{1003} равно

5946324504744078862063876146973507710583877301704448717931780201244084179200 0.

Это число имеет 77 цифр, его факторизация есть

$$2^9\times3^5\times5^3\times7^3\times11^2\times13^2\times17^2\times19\times23\times29\times31\times37\times41\times43\times47\times$$
$$\times53\times59\times61\times67\times71\times73\times79\times83\times89\times97\times101\times103\times107\times109\times$$
$$\times113\times127\times\times131\times137\times139\times149\times151\times157\times163\times167$$

и количество делителей равно 111325552312320.

Рассмотрим распределение сильно составных чисел среди потенциально сильно составных. Если взять 500 000 в качестве верхней границы для рассматриваемых чисел, то имеется 35 сильно составных чисел, меньших 500 000. Вычисляя $min(p)$ для первых простых чисел (сделать это легко с помощью функции $d[p]$), находим что $min(p) < 500\ 000$ для всех $p < 13$. Найдем объединение всех Λ_p для простых $p < 13$. И выберем из объединения все числа меньшие 500 000. Получим 58 потенциально сильно составных чисел. Пусть $s1$ – список всех *hc*-чисел меньших 500 000, а $s2$ – список всех потенциально сильно составных чисел меньших 500 000.

Определим список пар вида {номер, соответствующее по порядку число в $s2$}:

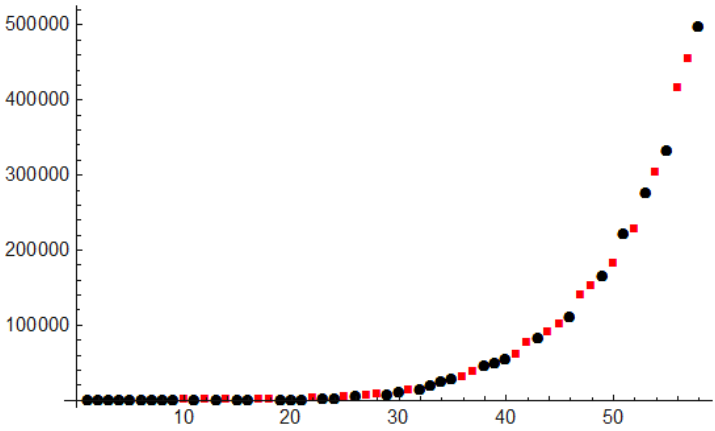
```
pairs2 = MapIndexed[{First[#2], #1}&, s2];
```

Список $s1$ содержит только пары для *hc*-чисел из $s2$:

```
pairs1=Select[pairs2, MemberQ[s1, #[[2]]]&]
{{1, 1}, {2, 2}, {3, 4}, {4, 6}, {5, 12}, {6, 24}, {7, 36}, {8, 48}, {9, 60},
{11, 120}, {13, 180}, {15, 240}, {16, 360}, {19, 720}, {20, 840}, {21,
1260}, {23, 1680}, {24, 2520}, {26, 5040}, {29, 7560}, {30, 10080},
{32, 15120}, {33, 20160}, {34, 25200}, {35, 27720}, {38, 45360}, {39,
50400}, {40, 55440}, {43, 83160}, {46, 110880}, {49, 166320}, {51,
221760}, {53, 277200}, {55, 332640}, {58, 498960}}
```

Изобразим на графике:

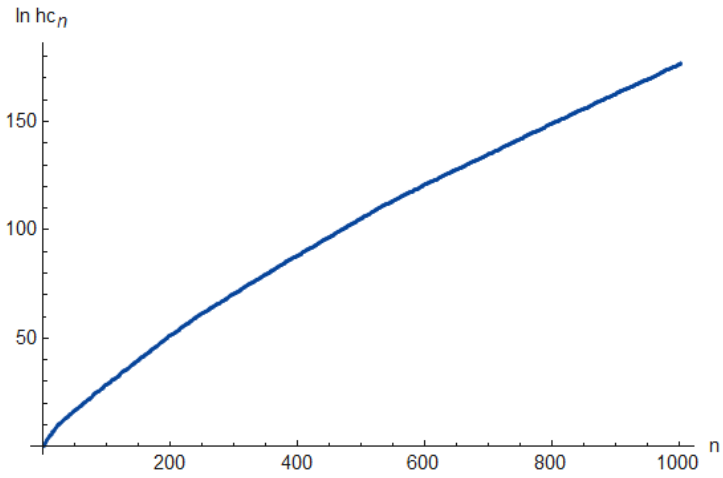
```
ListPlot[{pairs1, Complement[pairs2, pairs1]}, PlotStyle → {Black, Red},
  PlotMarkers → {Automatic}]
```



Кружками обозначены сильно составные числа, а квадратиками – остальные потенциально сильно составные числа.

Так как *hc*-числа растут очень быстро, то более наглядным является график зависимости ln h_n от номера n (*hc*1003 – список первых 1003 *hc*-чисел).

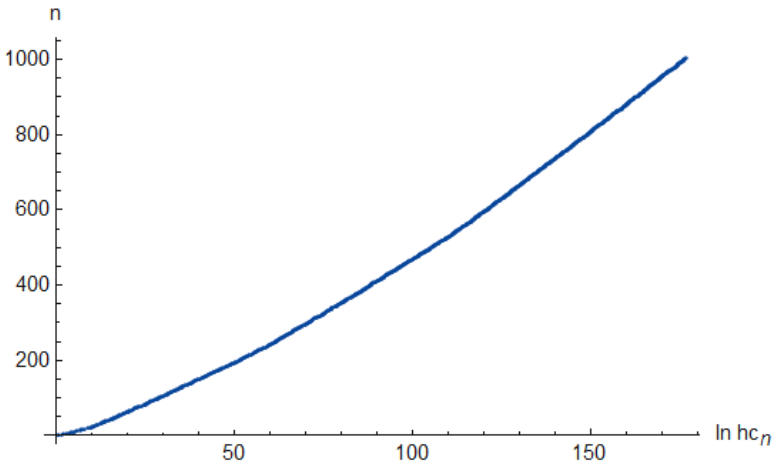
```
pairsLog = MapIndexed[{First[#2], #1}&, Log[hc1003]];
ListPlot[pairsLog, AxesLabel → {"n", "ln(hc(n))"}]
```



Определим значение функции $\gamma(x)$ для натурального x как количество hc-чисел, не превосходящих x. Будем считать, что значения функции $\gamma(x)$ известны для всех $x \le h_{1003}$, но хотелось бы иметь приближенную формулу, аппроксимирующую $\gamma(x)$.

Будем на оси абсцисс откладывать значения $\ln h_n$, а на оси ординат n, для $n \le 1003$, тогда точки с координатами $\{\ln h_n, n\}$ принадлежат графику функции $\ln x \to \gamma(x)$. Построим этот график. Для этого каждую пару в списке *pairsLog* надо реверсировать, т.е. пару $\{n, \ln h_n\}$ заменить парой $\{\ln h_n, n\}$.

```
pairsLog1 = MapIndexed[{#1, First[#2]}&, Log[hc1003]];
ListPlot[pairsLog1, AxesLabel → {"ln hc_n)", "n"}]
```



Теперь ищем простую функцию, график которой визуально совпадает с полученной кривой. В результате экспериментов найдена функция, имеющая вид $a\,x + b\,x^{3/2}$. Коэффициенты a и b определяются с помощью функции Fit языка Wolfram.

```
Fit[pairsLog1, {x^(3/2), x}, x]
```

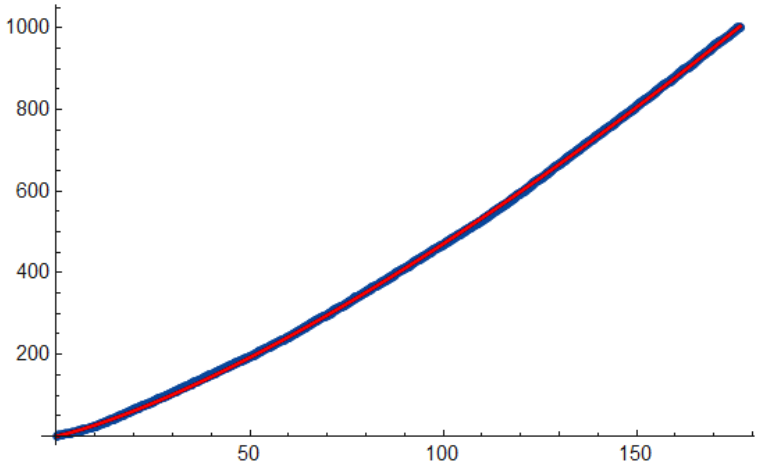
$1.77457\,x + 0.293921\,x^{3/2}$

Определим последнее выражение как функцию $f(x)$ и вычислим ее график g.

```
f[x_ ] := 1.77457 x + 0.293921x^(3/2)
g = Plot[f[x], {x, 0, 176.779}, PlotStyle → Red]
```

Число 176.779 есть $\ln h_{1003}$. Изобразим вместе графики функций $\ln x \to \gamma(x)$ (толстая кривая) и $\ln x \to 1.77457 \ln x + 0.293921(\ln x)^{3/2}$ (тонкая кривая). На оси абсцисс располагаются значения $\ln x$.

```
Show[ListPlot[pairsLog1, PlotStyle → PointSize[Medium]], g]
```



Визуально эти графики совпадают, но полученное равенство

$$\gamma(x) \approx 1.77457 \ln x + 0.293921(\ln x)^{3/2} \qquad (5)$$

весьма приближенное. Тем не менее можно оценивать значение hc_n для $n > 1003$.

Пример 5. Пусть $n = 2000$. Решаем уравнение $2000 = 1.77457 \ln x + 0.293921(\ln x)^{3/2}$.

```
Solve[2000 == 1.77457 Log[x] + 0.293921(Log[x])^(3/2), x]
```

Solve: Solve was unable to solve the system with inexact coefficients. The answer was obtained by solving a corresponding exact system and numericizing the result.

$\{\{x \to 3.34675 \times 10^{127}\}\}$

Получили, что 2000-е сильно составное число приближенно имеет 128 цифр.

Результаты этого параграфа опубликованы в [75].

§ 2. Итерация функции, определяющей число делителей

Рассмотрим два вопроса, касающихся поведения функции $\tau(n)$ – число делителей натурального числа n.

Вопрос 1. Заметим, что $\tau(p^{k-1}) = k$, когда p – простое и $k > 1$. Следовательно, уравнение $\tau(n) = k$ имеет бесконечное число решений. Поэтому любое натуральное число является значением функции τ для соответствующего аргумента.

Будем искать для данного k наименьшее n, для которого $\tau(n) = k$. Начнем с вычислений, где перебираются в цикле последовательные натуральные числа n до тех пор, пока не встретится n, такое, что $\tau(n) = k$. Определим список *ts*, состоящий из пар $\{t, k\}$, где t = минимум $\{n \mid n \le 10^5$ и $\tau(n) = k\}$.

```
s = {}; ts = {};
Do[k = DivisorSigma[0, t]; If[FreeQ[s, k], AppendTo[s, k];
AppendTo[ts, {t, k}]], {t, 10^5}];
ts
```

{{1, 1}, {2, 2}, {4, 3}, {6, 4}, {12, 6}, {16, 5}, {24, 8}, {36, 9}, {48, 10}, {60, 12}, {64, 7}, {120, 16}, {144, 15}, {180, 18}, {192, 14}, {240, 20}, {360, 24}, {576, 21}, {720, 30}, {840, 32}, {900, 27}, {960, 28}, {1024, 11}, {1260, 36}, {1296, 25}, {1680, 40}, {2520, 48}, {2880, 42}, {3072, 22}, {3600, 45}, {4096, 13}, {5040, 60}, {5184, 35}, {6300, 54}, {6480, 50}, {6720, 56}, {7560, 64}, {9216, 33}, {10080, 72}, {12288, 26}, {14400, 63}, {15120, 80}, {15360, 44}, {20160, 84}, {25200, 90}, {25920, 70}, {27720, 96}, {32400, 75}, {36864, 39}, {44100, 81}, {45360, 100}, {46080, 66}, {46656, 49}, {50400, 108}, {55440, 120}, {60480, 112}, {61440, 52}, {65536, 17}, {82944, 55}, {83160, 128}}

Список *s* накапливает во время выполнения цикла Do неповторяющиеся значения $\tau(t)$. Логическая функция FreeQ[*s*, *k*] выдает истину, если *k* не принадлежит *s*. В этом случае в списки *s* и *ts* добавляются новые элементы. Какие значения *t* попали в список *s* и сколько их?

```
Sort[s]
```

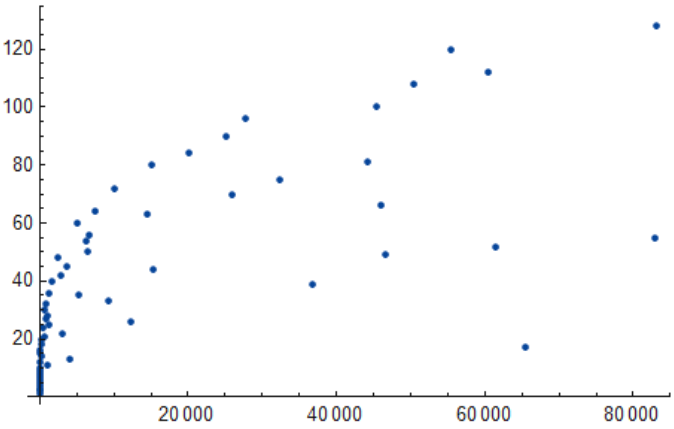
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 24, 25, 26, 27, 28, 30, 32, 33, 35, 36, 39, 40, 42, 44, 45, 48, 49, 50, 52, 54, 55, 56, 60, 63, 64, 66, 70, 72, 75, 80, 81, 84, 90, 96, 100, 108, 112, 120, 128}

```
Length[s]
```

60

Некоторое представление о *ts* дает диаграмма разброса данных.

ListPlot[*ts*]



Полный перебор, который демонстрируется в описанной программе, очень неэффективен. Мы знаем, что для простого k наименьшее n со свойством $\tau(n) = k$ есть 2^{k-1}. Поэтому в списке s отсутствуют все простые числа $k \geq 19$, так как $2^{18} = 262144 > 10^5$.

Как для составного k найти число n, удовлетворяющее условиям $\tau(n) = k$ и $n < 2^{k-1}$? В этом случае будем использовать мультипликативные разбиения k.

Мультиликативным разбиением (также используется термин **неупорядоченная факторизация**) [46] натурального k называется факторизация k на произведение сомножителей больших 1, причем порядок сомножителей игнорируется. В случае простого k имеем единственное разбиение, сводящееся к одному сомножителю k.

Пример 6. Покажем мультипликативные разбиения нескольких первых чисел k:

$$1 = 1;$$
$$2 = 2;$$
$$3 = 3;$$
$$4 = 2 \times 2 = 4;$$
$$5 = 5;$$
$$6 = 2 \times 3 = 6;$$
$$7 = 7;$$
$$8 = 2 \times 2 \times 2 = 2 \times 4 = 8.$$

Пусть $k = (a_1 + 1)(a_2 + 1) \cdots (a_r + 1)$, где все $a_i > 0$, – какое-то мультипликативное разбиение. Тогда для любых r простых чисел $p_{i_1}, p_{i_2}, \ldots, p_{i_r}$ имеем $\tau(\prod_{t=1}^{r} p_{i_t}^{a_t}) = k$. Среди всех чисел $\prod_{t=1}^{r} p_{i_t}^{a_t}$, пользуясь теоремой 1, легко найти наименьшее. Следующий пример показывает, как это сделать.

Пример 7. Пусть $k = 8$ и рассмотрим мультипликативное разбиение $8 = 2 \times 2 \times 2$. Соответствующие числа n, для которых $\tau(n) = k$, имеют вид $n = pqs$, где p, q, s – произвольные попарно различные простые числа. Из теоремы 1 следует, что для разбиения $8 = 2 \times 2 \times 2$ число $\boldsymbol{n} = 2 \times 3 \times 5 = 30$ есть наименьшее число n, для которого $\tau(n) = 8$.

Но имеются еще два варианта мультипликативных разбиения числа 8: $8 = 2 \times 4$ и собственно 8. Для $8 = 2 \times 4$ соответствующие числа n, для которых $\tau(n) = 8$**,** имеют вид $n = p^3 q$, где p, q – произвольные разичные простые числа, и в этом случае $\boldsymbol{n} = 2^3 \times 3 = 24$.

Для разбиения из единичного сомножителя 8 соответствующие числа n, для которых $\tau(n) = 8$**,** имеют вид $n = p^7$, где p – произвольное простое число, и в этом случае $\boldsymbol{n} = 2^7 = 128$.

Из трех чисел 30, 24 и 128 выбираем наименьшее число. Таким образом, получили число 24, которое есть наименьшее число n, для которого $\tau(n) = 8$.

Как видим, для данного k нахождение наименьшего n со свойством $\tau(n) = k$ можно выполнить с помощью следующего алгоритма.

Алгоритм 1. Вход: натуральное число k. Выход: наименьшее натуральное число n, для которого $\tau(n) = k$.

1. Находим все мультипликативные разбиения числа k.

2. Каждому разбиению $k = (a_1 + 1)(a_2 + 1) \cdots (a_r + 1)$, все $a_i > 0$, сопоставляем число $\boldsymbol{n}$, о котором говорится в формулировке теоремы 1.

3. Среди всех чисел $\boldsymbol{n}$, полученных на шаге 2, находим наименьшее число. Оно и будет результатом алгоритма.

Наиболее трудным шагом является первый. В настоящее время для его выполнения неизвестно никакого способа, кроме полного перебора во множестве всех делителей числа k в поиске подходящих сомножителей. Приведем программу на языке Wolfram, генерирующую все мультипликативные разбиения числа k [9].

```
unorderedFactorizations[m_, 1] := {{}}
unorderedFactorizations[1, k_] := {{}}
```

```
unorderedFactorizations[m_, k_?PrimeQ] := If[m < k, {}, {{k}}]
unorderedFactorizations[m_, k_] := unorderedFactorizations[m, k] =
  Module[{d}, Flatten[Function[d, Append[#, d]& /@
              unorderedFactorizations[d, k/d]] /@
              Select[Divisors[k], 1< # ≤ m&], 1]]
unorderedFactorizations[1] := {{1}}
unorderedFactorizations[k_] := unorderedFactorizations[k, k]
```

Рекурсивная функция *unorderedFactorizations*[*m*, *k*] с запоминанием промежуточных значений рекурсивных вызовов находит все мультипликативные разбиения числа *k*, где сомножители *m* не больше *k*. Для каждого делителя *d* числа *k* функция *unorderedFactorizations*[*m*, *k*] вызывает себя рекурсивно с аргументами *d* и *n*/*d*. Функция *unorderedFactorizations*[*k*] выдает все мулитипликативные разбиения числа *k*.

Найдем все разбиения чисел от 8 до 16:

```
{#, unorderedFactorizations[#]}& /@ Range[8, 16] // Column
```

```
{8, {{2, 2, 2}, {2, 4}, {8}}}
{9, {{3, 3}, {9}}}
{10, {{2, 5}, {10}}}
{11, {{11}}}
{12, {{2, 2, 3}, {3, 4}, {2, 6}, {12}}}
{13, {{13}}}
{14, {{2, 7}, {14}}}
{15, {{3, 5}, {15}}}
{16, {{2, 2, 2, 2}, {2, 2, 4}, {4, 4}, {2, 8}, {16}}}
```

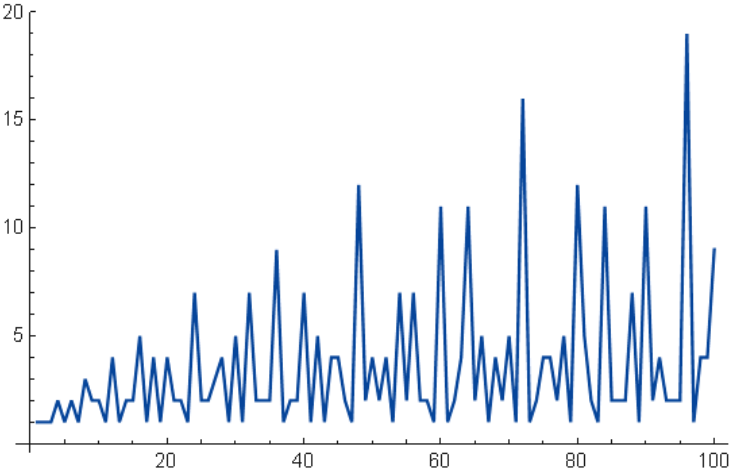
Чтобы узнать число мультипликативных разбиений для *k* досточно выполнить суперпозицию функций *unorderedFactorizations* и Length. Найдем число разбиений для всех чисел *k* от 1 до 100.

```
Length[unorderedFactorizations[#]]& /@ Range[100]
```

```
{1, 1, 1, 2, 1, 2, 1, 3, 2, 2, 1, 4, 1, 2, 2, 5, 1, 4, 1, 4, 2, 2, 1, 7, 2, 2, 3, 4, 1,
5, 1, 7, 2, 2, 2, 9, 1, 2, 2, 7, 1, 5, 1, 4, 4, 2, 1, 12, 2, 4, 2, 4, 1, 7, 2, 7, 2, 2,
1, 11, 1, 2, 4, 11, 2, 5, 1, 4, 2, 5, 1, 16, 1, 2, 4, 4, 2, 5, 1, 12, 5, 2, 1, 11, 2,
2, 2, 7, 1, 11, 2, 4, 2, 2, 2, 19, 1, 4, 4, 9}
```

Построим график для данной последовательности:

```
ListLinePlot[#, PlotRange → All]
```



Перейдем к реализации шага 2. Числа a_1, a_2, …, a_r из разбиения числа $k = (a_1 + 1)(a_2 + 1) \cdots (a_r + 1)$ должны стать показателями первых простых чисел в разложении $\boldsymbol{n}$, причем они должны быть переупорядочены в порядки невозрастания. Создание всех разбиений и соответствующие переупорядочения выполняет суперпозиция функций:

```
Reverse /@ unorderedFactorizations[k]
```

Число 12 имеет 4 разбиения {{2, 2, 3}, {3, 4}, {2, 6}, {12}}. Покажем, что получается после вычисления:

```
Reverse /@ unorderedFactorizations[12]
{{3, 2, 2}, {4, 3}, {6, 2}, {12}}
```

Теперь каждое разбиение из полученного списка должно превратиться в число вида $\boldsymbol{n}$ (см. формулировку теоремы 1). Должен получиться список $\{2^2\times3\times5, 2^3\times3^2, 2^5\times3, 2^{11}\}$. Нужные действия совершает функция

```
toTimes[s_] :=Times @@Flatten[MapIndexed[{Prime[First[#2]]^#1}&,
                s – 1]]
toTimes[{{3, 2, 2}, {4, 3}, {6, 2}, {12}}]
{60, 72, 96, 2048}
```

Для выполнения шага 3 достаточно найти минимальное число из полученного списка. И получаем функцию, которая находит наименьшее число с данным количеством делителей.

```
f[k_] := Min[toTimes /@ (Reverse/@ unorderedFactorizations[k])]
f[12]
60

f /@ Range[50]
{1, 2, 4, 6, 16, 12, 64, 24, 36, 48, 1024, 60, 4096, 192, 144, 120, 65536,
180, 262144, 240, 576, 3072, 4194304, 360, 1296, 12288, 900, 960,
268435456, 720, 1073741824, 840, 9216, 196608, 5184, 1260,
68719476736, 786432, 36864, 1680, 1099511627776, 2880,
4398046511104, 15360, 3600, 12582912, 70368744177664, 2520,
46656, 6480}
```

Функция f использует много памяти и работает медленно.

```
Timing[f /@ Range[50000];]
{71.4173, Null}
```

Для больших значений k вычисления $f[k]$ практически невозможны.

Предлагается более быстрый эвристический алгоритм.

Алгоритм 2. Вход: натуральное число k. Выход: «наименьшее» натуральное число n, для которого $\tau(n) = k$.

1. Находим мультипликативное разбиение числа k, состоящее только из простых сомножителей.

2. Полученному единственному разбиению $k = (a_1 + 1)(a_2 + 1) \cdots (a_r + 1)$, все $a_i > 0$, сопоставляем число $\boldsymbol{n}$, о котором говорится в формулировке теоремы 1. Число $\boldsymbol{n}$ и будет результатом алгоритма.

Шаг 1 сводится, например, для $k = 120$ к вычислению следующего выражения:

```
Flatten[Table[#[[1]], #[[2]]]& /@ Reverse[FactorInteger[120]]]
{5, 3, 2, 2, 2}
```

А вся программа для алгоритма 2 задается функцией

```
z[k_] := toTimes[Flatten[Table[#[[1]],
#[[2]]]&/@Reverse[FactorInteger[k]]]]
```

Как и положено эвристическому алгоритму, функция $z[k]$ не всегда дает верное решение. Проверим функцию z для ранее полученного списка ts.

```
AllTrue[z[#[[2]]] == #[[1]]& /@ ts, #&]
False
```

Следовательно, для каких-то чисел k функция $z[k]$ не находит наименьшего числа n с условием $\tau(n) = k$. Определим, для каких чисел k это происходит.

```
errors = Select[{#[[2]], #[[1]], z[#[[2]]]}& /@ ts, #[[2]] ≠ #[[3]]&]
```

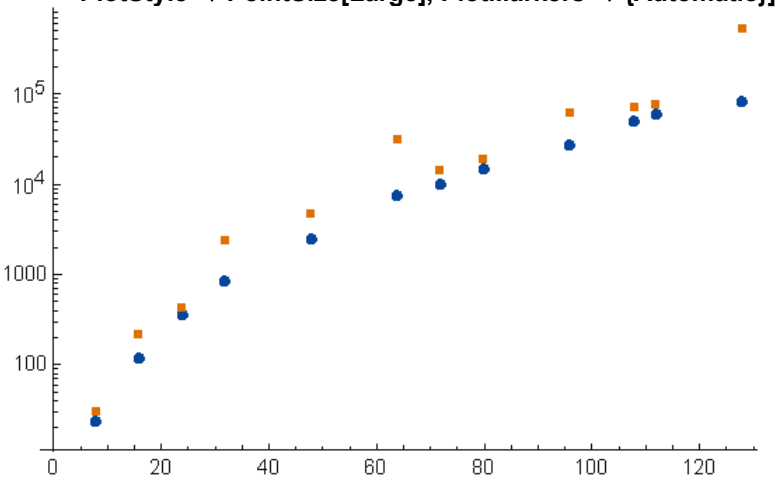
{{8, 24, 30}, {16, 120, 210}, {24, 360, 420}, {32, 840, 2310}, {48, 2520, 4620}, {64, 7560, 30030}, {72, 10080, 13860}, {80, 15120, 18480}, {96, 27720, 60060}, {108, 50400, 69300}, {112, 60480, 73920}, {128, 83160, 510510}}

```
Length[errors]
```

12

Нашлось 12 значений k (первые компоненты троек) из 60 чисел списка s. Вторые компоненты – это наименьшие числа n, такие, что $\tau(n) = k$; третьи компоненты – значения $z[k]$. На графике с логарифмическим масштабом по оси ординат списка *ers* представлены одновременно наименьшие значения n для k и значения $z[k]$.

```
ListLogPlot[{{#[[1]], #[[2]]}& /@ errors, {#[[1]], z[#[[1]]]}& /@ errors},
  PlotStyle → PointSize[Large], PlotMarkers → {Automatic}]
```



Значения функции z изображены квадратиками.

Рассмотрим одно множество натуральных чисел k, для которых алгоритм 2 возможно значительно чаще выдает наименьшее n с условием $\tau(n) = k$.

Будут ли значения функций *f* и *z* совпадать для сильно составных чисел? Возьмем первые 100 сильно составных чисел, полученных в предыдущем параграфе.

```
hc100 = {1, 2, 4, 6, 12, 24, 36, 48, 60, 120, 180, 240, 360, 720, 840,
1260, 1680, 2520, 5040, 7560, 10080, 15120, 20160, 25200, 27720,
45360, 50400, 55440, 83160, 110880, 166320, 221760, 277200, 332640,
498960, 554400, 665280, 720720, 1081080, 1441440, 2162160,
2882880, 3603600, 4324320, 6486480, 7207200, 8648640, 10810800,
14414400, 17297280, 21621600, 32432400, 36756720, 43243200,
61261200, 73513440, 110270160, 122522400, 147026880, 183783600,
245044800, 294053760, 367567200, 551350800, 698377680,
735134400, 1102701600, 1396755360, 2095133040, 2205403200,
2327925600, 2793510720, 3491888400, 4655851200, 5587021440,
6983776800, 10475665200, 13967553600, 20951330400, 27935107200,
41902660800, 48886437600, 64250746560, 73329656400,
80313433200, 97772875200, 128501493120, 146659312800,
160626866400, 240940299600, 293318625600, 321253732800,
481880599200, 642507465600, 963761198400, 1124388064800,
1606268664000, 1686582097200, 1927522396800, 2248776129600};
```

Числа из второй половины этого списка являются уже большими аргументами, чтобы функция *f* смогла вычислить значения на используемом компьютере автором. Поэтому выберем только первые 66 сильно составных чисел.

```
Select[{#, f[#], z[#]}& /@ Take[hc100, 66], #[[2]] ≠ #[[3]]&]
```
{{24, 360, 420},{48, 2520, 4620}}

Можно высказать **предположние *H*1**: *для сильно составных чисел k, за исключением k = 24 и k = 48, функция z[k] выдает наименьшее число n, для которого* $\tau(n) = k$.

Заметим, что функция $z[k]$ выдает наименьшее число *n,* для которого $\tau(n) = k$, не только для сильно составных *k*.

```
Complement[#[[2]]& /@ Select[{#[[2]], #[[1]], z[#[[2]]]}& /@ ts, #[[2]]
== #[[3]]&], hc100]
```
{16, 64, 144, 192, 576, 900, 960, 1024, 1296, 2880, 3072, 3600, 4096, 5184, 6300, 6480, 6720, 9216, 12288, 14400, 15360, 25920, 32400, 36864, 44100, 46080, 46656, 61440, 65536, 82944}

Этот программный код делает следующее. Берется список, элементы которого имеют вид $\{n, k\}$, где n – наименьшее число с условием $\tau(n) = k$. В этом списке оставляем только те элементы $\{n, k\}$, для которых $z[k]$ равно n. Далее из полученного списка пар создаем только список вторых компонентов – значений k. И, окончательно, удаляя из списка элементы $hc100$, получаем список чисел k, не являющихся сильно составными, но для которых $z[k]$ выдает наименьшее число n с условием $\tau(n) = k$.

```
Length[%]
30
```

Вопрос 2. Пусть $n > 1$ – натуральное число. Определим последовательность целых чисел $n_1, n_2, n_3, \ldots$ по правилу $n_1 = \tau(n)$ и $n_{k+1} = \tau(n_k)$ для $k = 1, 2, 3, \ldots$. Определим функцию $h(n)$ – «**высота *n***», значение которой для целого $n > 1$ равно наименьшему r со свойством $n_r = 2$. Другими словами, «высота n» равна числу итераций функции τ для достижения $n_r = 2$. Каково поведение функции $h(n)$?

Теорема 3. Функция «высота n» определена для всех $n > 1$.

Доказательство. Если $n = 2$, то $\tau(2) = n_1 = 2$ и $r = 1$. Докажем, что $\tau(n) < n$, если $n > 2$. Пусть p – нечетное простое число, тогда $p^a \geq (1 + 2)^a = 1 + 2a + \ldots + 2^a > a + 1$ для $a \geq 1$. Также имеем $2^a = (1+1)^a = 1 + a + C_a^2 + \ldots + 1 > a + 1$ для $a \geq 2$. Поэтому $\tau(p^a) = (a + 1) < p^a$, если $p^a \neq 2$. Далее, $\tau(n) = \prod \tau(p_i^{a_i}) < \prod p_i^{a_i} = n$, если какое-нибудь $p_i^{a_i} \neq 2$. Следовательно, $n > n_1 > n_2 > \ldots > n_r = 2$. Заметим, что $\tau(n_k) = 2$ для всех $k > r$. ■

Первое впечатление о поведении последовательностей $n_1, n_2, n_3, \ldots, n_r$ для различных начальных n получим, используя функцию NestWhileList языка Wolfram:

```
NestWhileList[DivisorSigma[0, #]&, #, # ≠ 2&]& /@ Range[2, 16] //
Column
{2}
{3, 2}
{4, 3, 2}
{5, 2}
```

```
{6, 4, 3, 2}
{7, 2}
{8, 4, 3, 2}
{9, 3, 2}
{10, 4, 3, 2}
{11, 2}
{12, 6, 4, 3, 2}
{13, 2}
{14, 4, 3, 2}
{15, 4, 3, 2}
{16, 5, 2}
```

Определим на языке Wolfram функцию g, вычисляющую значения $h(n)$:

```
g[2] = 0;
g[n_] := Length[NestWhileList[DivisorSigma[0, #]&, n, # ≠ 2&]] – 1
```

Первые 15 значений функции $h(n)$:

```
g /@ Range[2, 16]
{0, 1, 2, 1, 3, 1, 3, 2, 3, 1, 4, 1, 3, 3, 2}
```

Но для вычисления высоты n в большом диапазоне натуральных чисел n, применение функции g не эффективно, Например, вызов g /@ Range[2, 16] вычисляет $\tau(4)$ семь раз. Полезно использовать рекурсию с запоминанием уже вычисленных значений. Определим функцию h на языке Wolfram следующим образом:

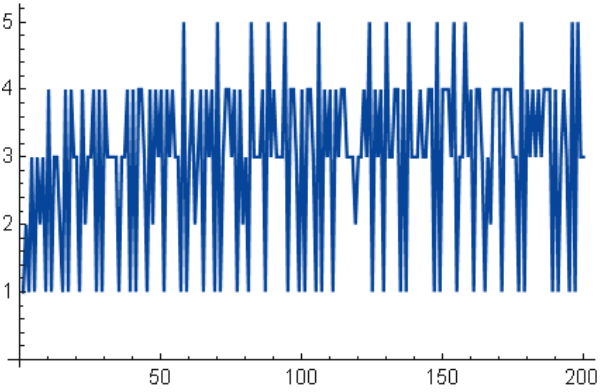
```
h[2] = 0;
h[n_] := h[n] = Module[{a = DivisorSigma[0, n]},If[a == 2, 1, h[a] + 1]]
h/@ Range[2, 16]
{0, 1, 2, 1, 3, 1, 3, 2, 3, 1, 4, 1, 3, 3, 2}
```

Вычислим значения функции h для натуральных чисел не превосходящих 10^6.

```
sr = h /@ Range[3, 1000000];
```

Изобразим на графике значения $h(n)$ для $3 \le n \le 200$:

```
ListPlot[Take[sr, 200], Joined → True]
```



Как видим, нижняя граница значений h равна 1, поскольку для простых чисел p значение $\tau(p) = 1$.

Верхняя граница значений $h(n)$ медленно растет. Попробуем оценить скорость роста $\max\{h(n) \mid 3 \le n \le m\}$. Ранее мы уже вычислили все значения $h(n)$ для всех $n \le 10^6$. Вычислим те значения $m \le 10^6$, для которых $h(m) > \max\{ h(n) \mid 3 \le n < m\}$.

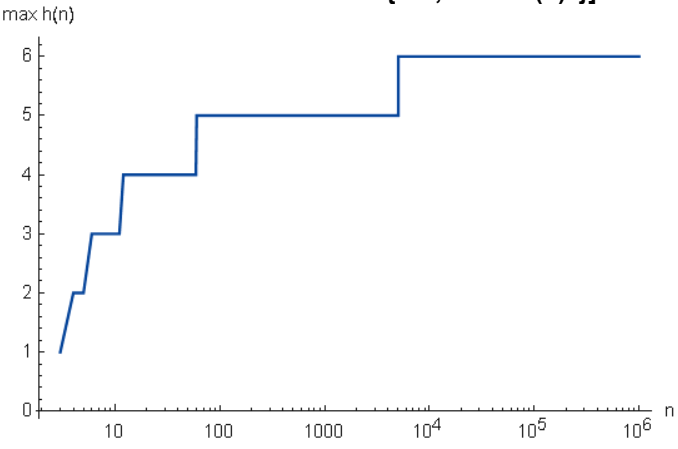
```
n = 3; s = {{3, 1}};
While[n ≤ 1000000, b = h[n]; a = s[[–1, 2]];
      If[b > a, AppendTo[s, {n, b}]]; n++];
s
```
{{3, 1}, {4, 2}, {6, 3}, {12, 4}, {60, 5}, {5040, 6}}

Будем называть число m **пороговым** (для функции h), если значение $h(m)$ увеличивается на единицу по сравнению с $\max\{h(n) \mid 3 \le n < m\}$. Таким образом, числа $b_1 = 3$, $b_2 = 4$, $b_3 = 6$, $b_4 = 12$, $b_5 = 60$, $b_6 = 5040$ являются пороговыми. Причем длины диапазонов, где максимумы остаются неизменными, резко увеличиваются. Заметим, что в диапазоне $5040 \le n \le 10^6$ наибольшее значение $h(n)$ равно 6. Для того чтобы изобразить это более наглядно, список s, состоящий из пар $\{m, h(m)\}$, дополним парами $\{m - 1, h(m - 1)\}$.

```
n = 3; s = {{3, 1}};
While[n ≤ 1000000, b = h[n]; a = s[[–1, 2]]; If[ b > a,
        AppendTo[s, {n – 1, a}]; AppendTo[s,{n, b}]]; n++];
      AppendTo[s, {1000000, s[[–1, 2]]}]; s
```
{{3, 1}, {3, 1}, {4, 2}, {5, 2}, {6, 3}, {11, 3}, {12, 4}, {59, 4}, {60, 5}, {5039, 5}, {5040, 6}, {1000000, 6}}

Для того чтобы графически изобразить список *s* полностью, не теряя наглядности, сделаем масштаб по оси *n* логарифмическим:

```
ListLogLinearPlot[s, Joined → True, PlotRange → All,
                  AxesLabel → {"n", "max h(n)"}]
```



Как мы видим, для натуральных чисел *n*, не превосходящих миллиона, максимальное значение $h(n) = 6$. А как найти следующие пороговые числа, в частности b_7 – наименьшее число с «высотой» равной 7?

Введенные функции *f* и *z* по определению обладают свойствами: для любого $k > 2$ выполнено $h(f(k)) = h(k) + 1$ и $h(z(k)) = h(k) + 1$. Оказывается также, что следующая итерация выдает первые 6 пороговых чисел.

```
NestList[f, 3, 5]
{3, 4, 6, 12, 60, 5040}
```

Кажется очевидным, что для пороговых чисел должно выполняться свойство

$$b_{i+1} = f(b_i). \tag{6}$$

Равенство (6) следовало бы из монотонности функции *f*. Но это не так, *f* не является монотонной функцией даже при аргументах одинаковой высоты. Одним из контрпримером является пара чисел 132 и 140 высотой равной 5. Хотя $132 < 140$, но $f(132) = 322560 > f(140) = 181440$. Равенство (6) не доказано и совершенно не понятно, как это доказывать.

Определим две бесконечные последовательности натуральных чисел bf_i и bz_i: начальные значения последовательностей есть $bf_1 = bz_1 = 3$, и для $i > 1$ положим $bf_i = f(bf_{i-1})$ и $bz_i = z(bz_{i-1})$. Известно, что $bf_i = bz_i$ вы-

полнено для $1 \le i \le 6$ и $bf_6 = 5040$. Значения bf_7 и bz_7 также совпадают:

```
{f[5040], z[5040]}
{293318625600, 293318625600}
```

Выполнено свойство

$$b_i \le bf_i \le bz_i \text{ для всех } i, \tag{7}$$

причем известно, что равенства выполнены для $1 \le i \le 6$.

Но это не доказывает и не опровергает, что $b_7 = 293318625600$, так как вычисление b_7 требует много времени.

Можно выдвинуть **предположение *H*2**:

$$b_i = bf_i = bz_i \text{ для всех } i, \tag{8}$$

причем известно, что равенства справедливы для $1 \le i \le 6$.

Хотя следующие пороговые числа вычислить не можем, но мы можем получить ограничения для них сверху, вычисляя bz_i.

Имеем $bz_8 = z(293318625600)$.

```
bz8 = z[293318625600]
67005916820458516837147643892742111293383729764099090415466
7968000000000000
```

К сожалению, $f(293318625600)$ невозможно вычислить, но поскольку 293318625600 является сильно составным числом (принадлежит списку *hc*100), то, если надеяться на справедливость предположения *H*1, получаем $bf_8 = bz_8$.

А является ли bz_8 сильно составным числом? Известно два необходимых условия для сильно составных чисел. Первое условие – это соотношение (1) из теоремы 1 с уточнением С. Рамануджана о том, что для наибольшего простого делителя p в разложении (1) показатель α_p всегда равен 1, если $p > 3$. По определению все значения функции z удовлетворяют соотношению (1). Степень наибольшего простого делителя bz_8 (это число 61) равна 1:

```
FactorInteger[bz8][[–1]]
{61, 1}
```

О втором необходимом условие говорится в теореме 2. Если p – наибольший простой делитель в разложении (1) сильно составного числа n, то в §1 оцениваются диапазоны значений показателей α_i. Функция $d[p]$ выдает границы диапазонов возможных значений α_i. Например, для $p = 7$ получаем

```
d[7]
{{2, 3, 4, 5, 6}, {1, 2, 3, 4}, {1, 2}, {1}}
```

Следующая булева функция проверяет второе необходимое условие для сильносоставности.

```
ramQ[1] = True;
ramQ[n_] := Module[{ps, as, l, gs},
      {ps, as} = Transpose[FactorInteger[n]]; l = Length[as];
      gs = {#[[1]], #[[-1]]}& /@ d[ps[[- 1]]];
      AllTrue [(#[[1]] ≤ #[[2]] ≤ #[[3]])& /@ Table[{gs[[i, 1]], as[[i]],
            gs[[i, 2]]}, {i, l}], #&]]
ramQ[bz8]
False
```

Следовательно, bz_8 – не сильно составное число, поэтому нельзя воспользоваться предположением $H1$ и это несколько уменьшает шансы того, что $bz_9 = bf_9$.

Найдем bz_9 и bz_{10}.

```
bz9 = z(bz8);
```

Целиком число bz_9 показать смысла нет, но посмотрим на несколько первых и последних цифр и вычислим общее количество цифр в десятичной записи числа.

```
Short[bz9]
22150113494786440950264049795892<<1364>>00000000000000000000000000000000

IntegerLength[bz9]
1429
```

И в завершении вычислим bz_{10}.

```
bz10 = z(bz9);
Short[bz10]
50865206389039571266591772197304<<189619>>00000000000000000000000000000000

IntegerLength[bz10]
189683
```

Таким образом, получили численную верхнюю оценку $b_i \le bz_i$ для всех $i \le 10$.

Основные результаты этого параграфа были доложены на конференции [76].

СПИСОК ЛИТЕРАТУРЫ

На высокую башню можно подняться
лишь по винтовой лестнице.

Фрэнсис Бэкон

1. *Троицкий* вариант. – № 13N (839). – 30 сентября 2008.
2. *Bailey D.* Mathematics by Experiment: Plausible Reasoning in the 21st Century. – Wellesley, MA: AK Peters, 2003.
3. *Bailey D., Borwein J., Calkin N., Girgensohn R., Luke D. R., Moll V.* Experimental Mathematics in Action. – Natick, MA: A.K. Peters, 2006.
4. *Borwein J., Bailey D., Girgensohn R.* Experimentation in Mathematics. – Wellesley, MA: AK Peters, 2003. – 358 p.
5. *Desmond J.E.* Problem B-182 // The Fibonacci Quarterly. – 1970. – V. 20. – No. 1 (Feb.). – P. 96.
6. *Guy R.K.* Unsolved Problems in Number Theory. 3nd ed. – New York: Springer-Verlag, 2004. – 438 p.
7. *Hastings C., Mischo K., Morrison M.* Hands-on Start to Wolfram Mathematica and Programming with the Wolfram Language, Second Edition. – Wolfram Media, Inc., 2016. – 484 p.
8. http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fractions/egyptian.html#section5.1
9. *Knopfmacher A., Mays M.* Ordered and Unordered Factorizations of Integers // The Mathematica Journal. – Wolfram Media, Inc., 2006. – V. 10. – No. 1.
10. *Koshy T.* Elementary Number Theory with Application. 2nd ed. – Academic Press, 2007. – 774 p.
11. *Koshy T.* Fibonacci and Lucas Numbers with Applications. V. 1. 2nd ed. – Jons Wiley & Sons, Inc., 2017.
12. *Kramer J., and Hoggatt V.E., Jr.* Special Cases of Fibonacci Periodicity // The Fibonacci Quarterly. – 1972. – V. 10. – No. 5 (Nov.). – P. 519–22.
13. *Lemmermeyer F.* Reciprocity Laws I. – Berlin: Springer-Verlag, 2000.
14. *Monier L.* Evaluation and comparison of two efficient probabilistic primality testing algorithms // Theoret. Comput. Sci. – 1980. – V. 12. – P. 97–108.
15. *Niven I., Zuckerman H.S., Montgomery H.L.* An introduction of the theory of numbers. 5th ed. – New York: Wiley, 1991. – 529 p.
16. *Ochem P., Rao M.* Odd Perfect Numbers Are Greater than 10^{15000}// Math. Comput. – 2012. – V. 81. – P. 1869–1877.
17. *Proofs* of the Quadratic Reciprocity Law. – URL: http://www.rzuser.uni-heidelberg.de/~hb3/fchrono.html
18. *Rabin M.* Probabilistic algorithm for testing primality // J. Number Theory. – 1980. – V. 12. – P. 128–138.

19. *Ramanujan S*. Highly Composite Numbers // Proc. London Math. Soc. XIV. – 1915. P. 347–409. http://ramanujan.sirinudi.org/Volumes/published/ram15.pdf
20. *Ribenboim P*. The new book of prime number records. 3rd ed. – Springer-Verlag, 1996. – 542 p.
21. *Rivest R.L., Shamir A., Adleman L.M.* A method for obtaining digital signatures and public-key cryptosystems // Communications of the ACM. – 1978. – V. 21. – P. 120–126.
22. *Rosen Kennet. H*. Elementary number theory and its applications. 6th ed. – Addison-Wesley, 2011. – 756 p.
23. *Sierpinski W*. Elementary theory of numbers. – Elsevier, 1988. – 513 p.
24. *Sloan's* On-Line Encyclopedia of Integer Sequences (OEIS, http://oeis.org/).
25. *Stopple J.* A Primer of Analytic Number Theory: From Pythagoras to Riemann. – Cambridge, UK; New York: Cambridge University Press, 2003.
26. *The Haskell* Programming Language. – URL: http://www.haskell.org
27. *The Wolfram Language*: Fast Introduction for Programmers. – URL: http://www.wolfram.com/language/fast-introduction-for-programmers/en/
28. *Vardi I.* Computational Recreations in Mathematica. – Redwood City, CA: Addison-Wesley, 1991. – 286 p.
29. *Wagon S.* Mathematica in Action: Problem Solving Through Visualization and Computation. – Springer Science & Business Media, 2010. – 580 p.
30. *Weisstein Eric W*. Wolfram MathWorld: Totient Valence Function. – URL: http://mathworld.wolfram.com/TotientValenceFunction.html
31. *Weisstein Eric W*. *Wolfram* MathWorld: CarmichaelNumber. – URL: http://mathworld.wolfram.com/CarmichaelNumber.html
32. *Weisstein Eric W*. Wolfram MathWorld: Carmichaels Totient Function Conjecture. – URL: http://mathworld.wolfram.com/CarmichaelsTotientFunctionConjecture.html
33. *Weisstein Eric W*. Wolfram MathWorld: Collatz Problem. – URL: http://mathworld.wolfram.com/CollatzProblem.html
34. *Weisstein Eric W*. Wolfram MathWorld: Egyptian Fraction. – URL: http://mathworld.wolfram.com/EgyptianFraction.html
35. *Weisstein Eric W*. Wolfram MathWorld: Euclid Number. – URL: http://mathworld.wolfram.com/EuclidNumber.html
36. *Weisstein Eric W*. Wolfram MathWorld: Feigenbaum Constant. – URL: http://mathworld.wolfram.com/FeigenbaumConstant.html
37. *Weisstein Eric W*. Wolfram MathWorld: Highly Composite Number. – URL: http://mathworld.wolfram.com/HighlyCompositeNumber.html
38. *Weisstein Eric W*. Wolfram MathWorld: Lorenz Attractor. – URL: http://mathworld.wolfram.com/LorenzAttractor.html
39. *Weisstein Eric W*. Wolfram MathWorld: Mersenne Prime. – URL: http://mathworld.wolfram.com/MersennePrime.html
40. *Weisstein Eric W*. Wolfram MathWorld: PartitionFunctionP. – URL: http://mathworld.wolfram.com/PartitionFunctionP.html
41. *Weisstein Eric W*. Wolfram MathWorld: Polylogarithm. – URL: http://mathworld.wolfram.com/Polylogarithm.html

42. *Weisstein Eric W.* Wolfram MathWorld: q-PochhammerSymbol. – URL: http://mathworld.wolfram.com/q-PochhammerSymbol.html
43. *Weisstein Eric W.* Wolfram MathWorld: Riemann hypothesis. – URL: http://mathworld.wolfram.com/RiemannHypothesis.html
44. *Weisstein Eric W.* Wolfram MathWorld: Totient Summatory Function. – URL: http://mathworld.wolfram.com/TotientSummatoryFunction.html
45. *Weisstein Eric W.* Wolfram MathWorld: Ulam Sequence. – URL: http://mathworld.wolfram.com/UlamSequence.html
46. *Weisstein Eric W.* Wolfram MathWorld: Unordered Factorization. – URL: http://mathworld.wolfram.com/UnorderedFactorization.html
47. *Wellin P.* Essentials of Programming in Mathematica. – Cambridge University Press, 2016. – 432 p.
48. *Wolfram* Mathematica. – URL: http://www.wolfram.com/mathematica
49. *Wolfram S.* A New Kind of Science. – Wolfram Media, 2002. – 1197 p. – URL: http://www.wolframscience.com/
50. *Wolfram S.* An Elementary Introduction to the Wolfram Language, Second Edition. – Wolfram Media, Inc., 2017. – 340 p. – URL: http://www.wolfram.com/language/elementary-introduction/2nd-ed/
51. *Wolfram S.* Computational Knowledge and the Future of Pure Mathematics. – URL: http://blog.wolfram.com/2014/08/12/computational-knowledge-and-the-future-of-pure-mathematics/ Пер. на рус. яз.: https://habr.com/company/wolfram/ blog/236199/
52. *Wolfram S.* Mathematical Notation: Past and Future. – URL: http://www.stephen wolfram.com/publications/mathematical-notation-past-future/
53. *Wolfram S.* The Background and Vision of *Mathematica.* – URL: http://blog. stephenwolfram.com/2011/10/the-background-and-vision-of-mathematica/
54. *Wolfram S.* Two Hours of Experimental Mathematics. – URL: http://blog. stephen wolfram.com/2017/03/two-hours-of-experimental-mathematics/#more-12672
55. *Wolfram S.* What Should We Call the Language of *Mathematica*? – URL: http://blog.stephenwolfram.com/2013/02/what-should-we-call-the-language-of-mathematica/#more-4997
56. *Wolfram S.* Who Was Ramanujan? – URL: http://blog.stephenwolfram.com/2016/04/who-was-ramanujan/
57. *Zeckendorf E.* Représentation des nombres naturels par une somme de nombres de Fibonacci ou de nombres de Lucas // Bullitin de la Société Royale des Sciences de Liège. – 1972. – V. 41. – P. 179–182.
58. *Айерлэнд К., Роузен М.* Классическое введение в современную теорию чисел. – М.: Мир, 1987. – 416 с.
59. *Арсак Ж.* Программирование игр и головоломок. – М.: Наука, 1990. – 224 с.
60. *Барендрегт Х.* Ламбда-исчисление. Его синтаксис и семантика. – М.: Мир, 1985. – 606 с.
61. *Бернстайн П.* Против богов: укрощение риска. – М.: ЗАО «Олимп-Бизнес», 2000. – 400 с.
62. *Бурбаки Н.* Теория множеств. Начала математики: пер. с франц. – М.: Мир, 1965. – 456 с.

63. *Бурбаки Н.* Алгебра. Многочлены и поля. Упорядоченные группы. – М.: Мир, 1965. – 300 с.
64. *Бухштаб А.А.* Теория чисел. – М.: Просвещение, 1966. – 384 с.
65. *Воробьев Н.Н.* Числа Фибоначчи. – М.: Наука, 1978. – 144 с.
66. *Гарднер М.* Математические головоломки и развлечения. – 2-е изд. – М.: Мир, 1999. – 447 с.
67. *Грэхем Р., Кнут Д., Паташник О.* Конкретная математика. Основание информатики. – М.: Мир; БИНОМ. Лаборатория знаний, 2006. – 703 с.
68. *Дасгупта С., Пападимитриу Х., Вазирани У.* Алгоритмы. – М.: МЦНМО, 2014. – 320 с.
69. *Деза Е.И.* Специальные числа натурального ряда: учебное пособие. М.: Книжный дом «ЛИБРОКОМ», 2011. – 240 с.
70. *Дербишир Д.* Простая одержимость. Бернхард Риман и величайшая нерешенная проблема в математике. – М.: Астрель: CORPUS, 2010. – 463 с.
71. *Дэвенпорт Г.* Высшая арифметика: Введение в теорию чисел. 2-е изд. – М.: Книжный дом «ЛИБРОКОМ», 2010. – 176 с.
72. *Зюзьков В.М.* Введение в математическую логику: учебное пособие. – 2-е изд., испр. – СПб.: Лань, 2018. – 268 с.
73. *Зюзьков В.М.* Введение в математическую логику: учебное пособие. – Томск: Издательский Дом Томского государственного университета, 2017. – 256 с. http://vital.lib.tsu.ru/vital/access/manager/Repository/vtls:000581547
74. *Зюзьков В.М.* Вокруг теоремы Эйлера о суммах делителей // Вестник Томского государственного университета. Математика и механика. – 2018. – № 51. – С. 19–32.
75. *Зюзьков В.М.* Вычисление сильно составных чисел // Девятая Сибирская конференция по параллельным и высокопроизводительным вычислениям: сборник статей / под ред. А.В. Старченко. – Томск: Издательский Дом Томского государственного университета, 2017. – С. 35–41.
76. *Зюзьков В.М.* Итерации функции Эйлера и функции, определяющей число делителей // Всероссийская конференция по математике и механике, 2–4 октября 2018 г.: тез. докл. – Томск: Изд-во Том. ун-та, 2018. – С. 40.
77. *Зюзьков В.М.* Ленивое функциональное программирование: учеб. пособие. – 2-е изд., перераб. и доп. – Томск: Изд-во Том. ун-та, 2008. – 294 с.
78. *Зюзьков В.М.* Начала компьютерной алгебры: учебное пособие. – Томск: Издательский дом ТГУ, 2015. – 128 с. – URL: http://vital.lib.tsu.ru/vital/access/manager/Repository/vtls:000509029
79. *Зюзьков В.М.* Новые вероятностные тесты на простоту // Восьмая Сибирская конференция по параллельным и высокопроизводительным вычислениям. – Томск: Изд-во Том. ун-та, 2015. С. 60–66.
80. *Зюзьков В.М.* Последовательность Fibonacci(n) mod n // Вестник Томского государственного университета. Математика и механика. – 2013. – № 4 (24). – С. 15–23.
81. *Интервью* с профессором Рональдом Грэхемом // Квант. – 1988. – № 8.
82. *Клини С.К.* Введение в метаматематику. – М.: ЛИБРОКОМ, 2009. – 582 с.

83. *Кострикин А.И.* Введение в алгебру. Часть I: Основы алгебры. – М.: Физико-математическая литература, 2000. – 272 с.
84. *Кранц С.* Изменчивая природа математического доказательства. Доказать нельзя поверить. – М.: Лаборатория знаний, 2016. – 320 с.
85. *Крылов А.Н.* Мои воспоминания. 7-е изд. – Л.: Судостроение, 1979. – 480 с.
86. *Крэндалл Р., Померанс К.* Простые числа: Криптографические и вычислительные аспекты. – М.: УРСС: Книжный дом «ЛИБРОКОМ», 2011. – 664 с.
87. *Лакатос И.* Доказательства и опровержения: Как доказываются теоремы. – 2-е изд. – М.: Издательство ЛКИ, 2010. – 152 с.
88. *Ландо С.К.* Лекции о производящих функциях. – 3-е изд., испр. – М.: МЦНМО, 2007. – 144 с.
89. *Манин Ю.И.* Вычислимое и невычислимое. – М.: Сов. радио, 1980. – 128 с.
90. *Математический* энциклопедический словарь. М.: Советская энциклопедия. – 1988. – 847 с.
91. *Матиясевич Ю.В.* Диофантовы множества // УМН. – 1972. – Т. 22. – Вып. 5. – С. 185–222.
92. *Пайтген Х.-О., Рихтер П. Х.* Красота фракталов. Образы комплексных динамических систем. – М.: Мир, 1993. – 176 с.
93. *Петер Р.* Рекурсивные функции. М.: ИЛ, 1954. – 264 с.
94. *Пиковер К.* Великая математика. От Пифагора до 57-мерных объектов. 250 основных вех в истории математики. – М.: БИНОМ. Лаборатория знаний, 2015. – 539 с.
95. *Пойа Д.* Математика и правдоподобные рассуждения. – М.: Наука, 1975. – 464 с.
96. *Пойа Д.* Математическое открытие: Решение задач: основные понятия, изучение и преподавание: пер. с англ. – 3-е изд. – М.: КомКнига, 2010. – 448 с.
97. *Прасолов В.В.* Доказательство квадратичного закона взаимности по Золотареву // Математическое просвещение. – 2000. – Т. 4. – С. 140–144.
98. *Радемахер Г., Теплиц О.* Числа и фигуры. – Ижевск: Ижевская республиканская типография, 2000. – 264 с.
99. *Сосинский А.Б.* Умер ли Бурбаки? // Математическое просвещение. Третья серия. – 1998. – Вып. 2.
100. *Харди Г.*Двенадцать лекций о Рамануджане. – М.: Институт компьютерных исследований, 2002. – 336 с.
101. *Цагир Д.* Первые 50 миллионов простых чисел // Живые числа: сб. ст. 1981 г.: пер. с нем. – М.: Мир, 1985. – 128 с.
102. *Черемушкин А.В.* Лекции по арифметическим алгоритмам в криптографии. – М.: МЦНМО, 2002. – 104 с.
103. *Эндрюс Г.* Теория разбиений. М.: Наука, 1982. – 256 с.

ПРЕДМЕТНЫЙ И ПЕРСОНАЛЬНЫЙ УКАЗАТЕЛИ

Bailey D., 12
Borwein J., 12
Mathematica, 15
q-символ Похгаммера, 282
алгоритм
    RSA, 168
    возведения в степень, 116
    вычислительные трудности алгоритма Евклида, 100
    генерация простого числа, 305
    для вычисление gcd и инвертирования, 117
    Евклида, 98
    извлечения квадратного корня по модулю p, 164
    тест Миллера–Рабина, 303
аликвотная дробь, 64
аппроксимация поведения последовательности, 82
Белл Эрик Темпл, 12
Бурбаки Н., 23
Вольфрам Стивен, 14, 15, 22, 229
выражение в «замкнутой форме», 216
Гарднер Мартин, 109
Гаусс Карл, 7, 9, 10, 11, 12, 153
Гельфанд Михаил, 7
гипотеза
    Римана, 304
    тест Миллера на простоту, 305
    Эйлера о суммах делителей, 290
группа мультипликативных арифметических функций, 205
Грэхем Рональд, 84
Дон Загир, 7
евклидовы числа, 58
задача о господине *S* и господине *P*, 75
Золотарёв Егор, 153
Киплинг Редьярд, 171
Коллатц Лотар, 67
Коллатца последовательность, 68
Кранц Стивен, 9, 343
Лакатос Имре, 9, 343

Лежандр Анри, 148
Лоренц Эдвард, 10
Мандельброт Бенуа, 10
Манин Юрий, 7
Маршак Самуил, 171
множество
    диофантово, 221
    перечислимое, 221
обобщенный мультипликативный порядок, 245
пифагорова тройка, 61
Пойа Дьёрдь, 6, 9
порядок числа по модулю, 242
последовательность полиномиальных остатков, 101
последовательность Фибоначчи, 216
примитивный корень, 246
производящая функция Дирихле, 276
производящая функция для последовательности, 264
прообраз функции Эйлера, 132
разбиение мультипликативное, 326
разбиение натурального числа, 281
Рамануджан Сриниваса, 7, 10, 15, 311, 313, 314
рекуррентная формула о числе разбиений, 293
решение диофантовых уравнений, 107
Риман Бернхард, 9, 342
свёртка Дирихле, 201, 278
свёртка последовательностей, 273
свидетель числа, 303
символ Лежандра, 148
символ Якоби, 156
сравнение
    класс вычетов, 111
    наименьшие неотрицательные вычеты, 112
    полная система вычетов, 112
    приведенная система вычетов, 135
теорема
    Вильсона – Лагранжа, 115
    Десмонда, 236
    диофантовость равносильна перечислимости, 221
    китайская об остатках, 121
    критерий Гаусса, 151
    критерий Корсельта, 299
    критерий Эйлера, 148
    о количестве примитивных корней по модулю, 248
    о обратимости ряда, 260
    о существовании примитивных корней по модулю, 249

о функции Эйлера, 135
основные свойства сравнений, 114
Ферма малая, 136
формула обращения Дирихле, 203
формула обращения Мёбиуса, 198
Эйлера пентагональная, 283
Тест Люка – Лемера, 192
Улам Станислав, 93
Улама числа, 93
Фейгенбаум Митчелл, 10
фибоначчиевая система счисления, 224
функция
аддитивная, 175
Аккермана, 90
арифметическая, 175
вполне аддитивная, 175
вполне мультипликативная, 175
Дирихле обратная, 203
Мёбиуса, 196
мультипликативная, 175
полилогарифм, 268
сумма делителей, 179
сумматорная, 177
число делителей, 180
Эйлера, 127
число
вероятное простое, 296
Кармайкла, 299
Мерсенна, 192
Мерсенна простое, 192
пороговое, 335
потенциально сильно составное, 315
псевдопростое Ферма, 296
псевдопростое Фибоначчи, 306
сильно вероятное простое, 301
сильно составное, 311
совершенное, 189
Фибоначчи, 216
число разбиений, 281
Эйлер Леонардо, 7, 9, 10, 12, 135, 153, 281, 282, 283, 285, 290, 293
экспериментальная математика, 6, 9
язык Wolfram, 13, 20, 21, 23
анонимная функция, 44
ввод с помощью палитры, 24
ввод с помощью шаблонов, 23

ввод свободный на английском языке, 24
ввод строковый, 23
выражение, 25
    атомарное, 36
    головная часть, 26
    определения функции, 37
    переменная, 36
    полное представление с помощью FullForm, 26
    преобразование в различные формы, 32
    список, 37
    функция, 25
отличия от математической нотации, 21
парадигмы программирования, 35
    программирование с правилами преобразований, 46
    процедурное, 36
    функциональное, 40
параллельные вычисления, 52
принципы языка, 20
рекурсивные функции, запоминающие промужуточные значения, 39
упрощение
    автоматическое, 28
    заданное, 30
    с допущениями, 33
форма представления InputForm, 24
форма представления StandartForm, 25
форма представления TraditionalForm, 25

Научное издание

Зюзьков Валентин Михайлович

ЭКСПЕРИМЕНТЫ В ТЕОРИИ ЧИСЕЛ

Редактор *Т.С. Портнова*
Дизайн, верстка *Д.В. Фортеса*

ООО «Издательство научно-технической литературы»
634050, Томск, пл. Новособорная, 1, тел. (3822) 533-335

Изд. лиц. ИД № 04000 от 12.02.2001. Подписано к печати 30.01.2019.
Формат 60 × 84 $^{1}/_{16}$. Бумага офсетная. Печать офсетная. Гарнитура «Таймс».
Усл. п. л. 20,23. Уч.-изд. л. 22,65. Заказ № 6.

Отпечатано на оборудовании
Издательского Дома Томского государственного университета,
634050, г. Томск, пр. Ленина, 36, тел. (3822) 531-528, 529-849.
http://publish.tsu.ru E-mail: rio.tsu@mail.ru