



П. НОДЕН, К. КИТТЕ

АЛГЕБРАИЧЕСКАЯ АЛГОРИТМИКА

С упражнениями и решениями

АЛГЕБРАИЧЕСКАЯ АЛГОРИТМИКА

LOGIQUE MATHÉMATIQUES
I N F O R M A T I Q U E

ALGORITMIQUE ALGÈBRE

Avec exercices corrigés

Patrice NAUDIN

Maitre de conférences à l'université de Bordeaux I

Claude QUITTE

Maitre de conférences à l'université de Poitiers

Preface de Francis SERGERAERT

MASSON Paris Milan Barcelone Bonn 1992

Патрис НОДЕН, Клод КИТТЕ

АЛГЕБРАИЧЕСКАЯ АЛГОРИТМИКА

С упражнениями и решениями

Предисловие Франсиса Сержераера

Перевод с французского
В.А. Соколова

под редакцией
Л.С. Казарина

Рекомендовано Научно-методическим советом по прикладной математике
Учебно-методического объединения университетов для использования в учеб-
ном процессе для студентов вузов по специальности «Прикладная математи-
ка» и «Прикладная математика и информатика»



МОСКВА «МИР» 1999

УДК 512+519.6
ББК 22.14+22.19
Н72

Ноден П., Китте К.

Н72 Алгебраическая алгоритмика (с упражнениями и решениями):
Пер. с франц. — М.: Мир, 1999. — 720 с., ил.
ISBN 5-03-003318-1

Книга известных французских математиков — это по существу энциклопедия алгоритмов алгебры и теории чисел от Евклида и до наших дней. В ней прослеживается общая идея — представить основные алгебраические структуры и концепции в виде объектов, поддающихся машинной обработке. Главными для авторов являются два вопроса: что значит вычислить математический объект и как его вычислить наиболее эффективно.

Изложение отличается методическими достоинствами: тщательный отбор материала, многочисленные замечания теоретического и исторического характера, большое число упражнений с решениями в конце каждой главы.

Для математиков-прикладников, для всех изучающих и применяющих компьютерную алгебру и информатику как учебное и справочное пособие.

ББК 22.14+22.19



Издание осуществлено при поддержке Российского фонда
фундаментальных исследований по проекту 97-01-14001

Издание осуществлено в рамках программы помощи
издательскому делу «Пушкин» при поддержке
Министерства Иностранных Дел Франции
и Посольства Франции в России

Редакция литературы по математическим наукам

ISBN 5-03-003318-1 (русск.)
ISBN 2-225-82703-6 (франц.)

© Masson, Paris, 1992
© перевод на русский язык,
«Мир», 1999

Предисловие переводчика и редактора перевода.

Со времени появления монографии Г. Биркгофа и Т. Барти «*Современная прикладная алгебра*» (28 лет тому назад) сочетание прилагательных «*прикладной*» и «*абстрактный*» в применении к алгебре перестало шокировать или изумлять читателей. Прилагательное «*современный*», разумеется, должно исчезнуть по мере разрастания теории и завоевания все новых территорий (переведенная недавно книга Р. Лидла и П. Пильца тому свидетельство). Более интересным является сопоставление компьютерной математики и алгебры, появившееся сравнительно недавно, а уж алгебраическая алгоритмика и вовсе должна вызвать ассоциацию с «*маслом масляным*», ибо корни обоих понятий (алгебра и алгоритм), как известно, одни и те же. Тем не менее, несмотря на близкое родство, алгоритмика (читай, информатика) и алгебра длительное время развивались как бы обособленно, хотя слово «*метод*» в учебниках по высшей алгебре вытеснялось постепенно словом «*алгоритм*», а в информатику проникли модули и тензорные произведения. Вопросы сложности вычислений, явившиеся краеугольным камнем для информатики, позволили поставить весьма интересные чисто алгебраические задачи. Стало ясно, что в алгебре можно вычислять, даже если исходные вопросы касаются таких субстанций, как модуль, кольцо, группа. А раз можно вычислять, то нужно вычислять как можно быстрее, ибо уже студенту-первокурснику понятно, что «грубая сила» машины не сможет выручить при решении некоторых совсем простых, на первый взгляд, задач.

Многие алгоритмы алгебры, с которыми нам приходилось сталкиваться в жизни, становились известными из математического фольклора, другие изобретались самостоятельно (ибо посмотреть было негде). Потом появился изумительный Д. Кнут (Искусство программирования), пугающий своей фундаментальностью, куда все-таки приходилось заглядывать, невзирая на известные трудности. А что делать студенту? Книга «*Алгебраическая алгоритмика*» явилась нам неожиданно во всем своем блеске благодаря знакомству с ее авторами, работавшими в Пуатье. Начав работать, мы оба поняли, что этой-то книги нам не хватало и как преподавателям, и как математикам-профессионалам.

С одной стороны, — это новый взгляд на вещи, казавшиеся устоявшимися. Например, что может быть проще алгоритма Евклида или возведения в степень? С другой, некоторая систематизация, азбука информатики (как мы мечтали 30 лет тому назад о каталоге полиномиальных алгоритмов!). Наконец, это философия программирования, изложенная на небольшом пространстве и без присущего сугубым профессионалам высокомерия.

В то же время книга читается (мы отмечаем это с некоторым ужасом) как художественная, беллетристическая литература. Чего стоит одно дерево Штерна — Брок из задачи 35 в главе II!

Основной материал книги сопровождается огромным количеством упражнений (их около 300, но некоторые из упражнений содержат в себе от 3 до 5 задач), примеров, иллюстраций. Студента несомненно порадует то обстоятельство, что большинство упражнений приводится с решением (не сразу, а через несколько страниц). Пересечение с другими известными нам книгами по алгоритмике невелико.

Книга П. Нодена и К. Китте восполняет пробел, существовавший в литературе на русском языке (и по математике, и по информатике), и может быть использована для чтения специальных и основных курсов как для студентов-математиков, так и для информатиков. Кроме того, она, будучи использована в качестве дополнительного источника, окажется несомненно полезной при чтении курсов «Алгебра и теория чисел» и «Алгебра и геометрия».

Следует отметить, что терминология, применяемая французскими математиками, не вполне совпадает с принятой в нашей литературе. Так, натуральный ряд начинается с нуля, а не с 1, как обычно принято у нас. Отсюда появление обозначения N^* . Приведение матрицы к ступенчатому виду осуществляется по столбцам, а не по строкам, причем имени Гаусса авторы в этой связи не упоминают. Есть и другие мелкие детали, которые читатели, несомненно, заметят. Что ж, это заставляет читателя быть внимательным!

Приятно отметить, что в подготовке к изданию перевода книги активное участие приняли наши ученики и студенты С.В. Монахов, С.М. Медведев (подготовившие оригинал-макет русского перевода), И.А. Сагиров, Л.В. Гудкова, А.В. Хлямов, А.Е. Скрипкин. Мы им выражаем нашу глубокую благодарность. Мы также признательны заведующему лабораторией теоретико-групповых методов профессору А.Л. Онищику за предоставление технических средств для верстки этой книги. Мы благодарны издательству MASSON, давшему согласие на передачу права на издание перевода книги издательству «Мир», и Посольству Франции в Москве включение русского издания книги в программу «Пушкин», а также Российскому

фонду фундаментальных исследований, поддержавшему проект перевода книги издательским грантом.

В заключение мы хотим прямо сказать, что наш замысел вряд ли осуществился, если бы не помощь и активное сотрудничество наших французских друзей и коллег из университета Пуатье — авторов книги Патриса Нодена и Клода Китте. Они дали согласие на безвозмездное издание перевода их книги, любезно предоставили в наше распоряжение файлы с ее электронным вариантом и постоянно, на протяжении всего времени работы над переводом, поддерживали с нами связь, консультировали нас, полностью пересмотрели текст французского издания и внесли в него много исправлений и улучшений, которые учтены в русском издании.

Только благодаря этому сотрудничеству наш совместный проект успешно завершен.

Надеемся, что перевод этой примечательной книги будет интересен и полезен нашим читателям.

*Л.С. Казарин, В.А. Соколов,
Ярославль, 10 августа 1998 г.*

Предисловие к русскому изданию

Проект перевода Львом Казариным и Валерием Соколовым нашей книги «Алгебраическая алгоритмика», возникший более двух лет тому назад, подошел к завершению. Мы особенно рады этому событию, если учесть, что пройденный путь, вероятно, не был гладким. Основные очертания проект приобрел в результате нашего путешествия в Россию, которое мы предприняли весной 1996 года по приглашению Валерия Соколова от имени Ярославского государственного университета. Во время нашего пребывания в Ярославле Лев и Валерий оказали нам, как гостям университета, самый радушный прием. Лев сопровождал нас во всех поездках, которые Ярославский университет организовал для нас, чтобы мы открыли для себя маленький кусочек России — это то, что выходит за рамки профессиональной деятельности, но является, тем не менее, приятным (и необходимым) элементом любой поездки за границу. Благодаря Льву мы открыли ряд исторических уголков, входящих в «Золотое кольцо»; он опекал нас на первых шагах в нашей повседневной жизни иностранцев в России, что порой было очень непросто: вспомним автобус, который отвозил нас на вокзал в пятницу в 6 часов утра, когда все киоски были еще закрыты! «Автобус... А как насчет билетов?» А в Москве Аркадий Онищик, Сергей и Ирина Струнковы доказали нам на непостижимом для представителей Запада уровне, что российское гостеприимство вовсе не легенда. Обо всех этих встречах и людях мы сохраним самые волнующие воспоминания.

Прервем здесь дифирамбы, хотя это вступление в действительности лишь в небольшой степени передает наше душевное состояние. Нелегкое, все-таки, это дело — сочинять предисловие к переводу книги, которую сами написали! Поговорим, однако, об идеях, которые мы хотели изложить в этой книге.

В течение трех лет работы над ней (1989-91 гг.) нашей целью — амбициозной и одновременно наивной — было представить с одной конкретной точки зрения некоторые разделы алгебры, преподаваемые на старших курсах университета. В то время нам хотелось (и сейчас нам этого хочется более, чем когда-либо) изменений в изложении математики: не отказываясь от абстрактных понятий, использовать их так, чтобы гармонично сочетать теорию и практику.

Любой студент-математик знаком с основными понятиями, которые встречаются в арифметике: целые числа, сравнимость чисел по модулю n , многочлены и т.д. Но, чаще всего, эти объекты, по самой своей сути предназначенные для практики, определяются очень формально и абстрактно. Так ли уж необходимо вводить фактор-группу некоторой группы по отличной от нее подгруппе для изучения целых чисел, сравнимых по модулю p ? Что усвоят наши студенты при таком подходе? Так ли уж неразумно опираться, когда это возможно, на практику? Ведь освоив вычисления по модулю целого числа (или по модулю многочлена), уже гораздо позже можно рассмотреть общий случай вычислений по модулю идеала, «более естественно» приводящий к понятию фактор-кольца. В самом деле, нет недостатка в конкретных проблемах, где может с успехом использоваться вычислительная практика: тест на простоту чисел Лукаса — Лемера, критерий Пепина, RSA-метод в криптографии, генератор случайных чисел и т.д. Мы рассматриваем все эти темы в нашей книге и убеждены, что они лучше подходят для обучения, чем «теорема о факторизации (переход к фактор-множеству) некоторого отображения».

Что же изменилось с 1992 года, времени выхода нашей книги? В области математической алгоритмики на рынке появилось некоторое количество монографий, но это в основном узко специализированные книги научного характера, оказывающие весьма слабое влияние на преподавание математики. Что же касается информатики как таковой, то единственным плодом ее непрерывного бурного развития в данной области является увеличение числа систем формальных вычислений — часто настолько же мощных, насколько и последовательных, неприспособленных для научной работы из-за недостаточной строгости. Нынче можно осуществлять сложные вычисления в теории Галуа, в алгебраической геометрии, в теории чисел, в теории групп и т.д., но все это, как правило, недоступно для начинающего студента-математика. И против всех ожиданий, области, в которых системы формальных вычислений уже хорошо обкатаны — это, как ни странно, разделы курса алгебры — всегда излагаются в том же виде, в каком они были десятилетия назад. Использование этих прикладных программ осталось вне преподавания математики и касается, в основном, лишь научных исследований.

Сообществу математиков, кажется, грозит опасность раскола на два чуждых друг другу мира: «теоретиков» и «алгоритмистов». Эта ситуация, на наш взгляд, была бы достойна сожаления. В то же время, во Франции — особенно в преподавании математики — нередко шли на отказ от догматических воззрений в пользу нового хорошего подхо-

да к объяснению того или иного понятия. Преподавание (равно, как и научное исследование) может лишь обогатиться благодаря многообразию способов изложения материала: достаточно хотя бы раз в жизни прочитать студентам курс лекций по алгоритмике, чтобы убедиться в этом.

П. Ноден, К. Китте,
Пуатье, 2 февраля 1999 г.

Предисловие

Патрис Ноден и Клод Китте попросили меня написать предисловие к их книге, и я начну с благодарности им: быть вовлеченным в работу в приятной компании по случаю выхода в свет такого прекрасного произведения — это счастье.

История математики очень увлекательна. Для сюжета, который нас занимает сегодня, существенны три персонажа. Евклид пишет свои «Начала» за три столетия до нашей эры, и с этой даты большинство историков ведут отсчет появления той математики, которую мы знаем сегодня, имеющей характер некой игры, заключающейся в том, чтобы делать интересные выводы из множества хорошо подобранных аксиом. Природа этих аксиом и соблюдаемые правила игры с самого начала не были достаточно ясны, и математика XIX века познала такое развитие, что вскоре основы здания были едва способны выдерживать всю конструкцию. Итогом этого было то, что называют кризисом оснований математики.

Но тут появился Гильберт; он понял и талантливо объяснил, что математика может быть формализована. Гильберт описал работу математиков, как чисто формальную игру, манипулирующую множеством аксиом (комбинаторных объектов) с помощью доказательств (чисто комбинаторной техники) и приводящую к теоремам (также комбинаторным объектам). На самом деле так говорить — неправильно: Гильберт мог таким образом описать результаты, полученные его предшественниками, но никоим образом — с помощью какой процедуры они их открыли. Это главное различие для того сюжета, которым мы интересуемся. Гильберт знал это и вскоре поставил перед сообществом хорошие вопросы на эту тему: может ли аксиоматическая система быть полной, или, другими словами, всякое ли утверждение, имеющее смысл в этой системе, истинно или ложно? Кроме того, существует ли алгоритм, позволяющий определять, истинно или ложно то или иное утверждение (или же невыводимо в случае неполноты)?

После Евклида и Гильберта третьим является Гёдель. Он доказал, что всякая аксиоматическая достаточно «интересная» теория обязательно будет неполной (или противоречивой). Этот явно негативный результат странным образом является одним из наиболее позитивных.

Непосредственно вдохновленные доказательством Гёделя логики Чёрч и Тьюринг вскоре доказали (1936), независимо друг от друга и разными способами, что второй вопрос Гильберта также имеет отрицательный ответ: не может существовать алгоритм, даже чисто теоретический, грозящий математикам безработицей. Этот результат все еще отрицательный, но — терпение, и мы увидим, что на самом деле Чёрч и Тьюринг стоят у истоков фантастической — и позитивной — революции в математике. Ибо им пришлось очень тщательно обдумать, что же такое алгоритм. Первый из них изобрел для уточнения этого понятия λ -исчисление, второй — свою знаменитую теоретическую машину. И это стало рождением целой новой ветви в математике, называемой алгоритмикой, и книга, которую я предваряю предисловием, частично относится к этой области. В этом ее новизна.

Благодаря фон Нейману идеи Тьюринга вскоре привели к концепции и реализации того, что теперь называется компьютерами. Вначале в них видели лишь инструмент, предназначенный для того, чтобы использовать результаты труда математиков-теоретиков в различных приложениях. Этот взгляд, все еще распространенный, когда информатик ниже по течению принимает эстафету от математика, является сегодня совершенно неверным. Больше уже не считают, что математические теории, представление о которых существенно изменилось, находятся в верховьях по отношению к положению алгоритмического инструмента, и ставить эту ветвь в подчинение другим так же наивно, как пытаться выяснить в алгебраической топологии, что главнее: топология по отношению к алгебре или наоборот.

Математики все еще часто и довольно нелепо считают за честь получить результаты, не использующие новые алгоритмические средства. Это обычное явление, сопровождающее значительное и быстрое развитие в какой бы то ни было сфере жизни: подобное развитие порождает кризис, разрешающийся более или менее удачно. Когда возникла алгебраическая топология, многие строптивые топологи старательно отделяли топологические доказательства, не использующие алгебраическую топологию, считая их более хорошими, от других, незаконно рожденных. Но в конце концов алгебраическая топология утвердилась. В рамках алгебраической топологии в один прекрасный день появились спектральные последовательности', в это время доказательства, использующие такую технику, еще рассматривались как «грязные», и их, по-возможности, избегали. Мало-помалу, спектральные последовательности стали обычным делом. Совсем недавно новые исследования заново изучили вопрос, связанный со спектральными последовательностями, для того, чтобы сделать из них алгоритмический инструмент;

я знал редактора одного прекрасного математического журнала, сердито требовавшего представить аргумент, «убеждающий», что новые результаты, полученные таким способом, были бы недоступны при использовании «других средств» (sic) и не иначе, без чего представленная статья на эту тему была бы, очевидно, отвергнута. S.O.S. — расизм!

Книга Патриса Нодена и Клода Китте представляет классическую алгебру — ту, которая преподается на уровне второго цикла университета, но в свете новой алгоритмической идеологии. Для рассматриваемой темы алгебра подходит лучше всего: алгоритмика по самой своей природе комбинаторна, а среди всех основных математических дисциплин алгебра является, несомненно, «самой комбинаторной». Преподавать студентам алгебру таким способом — прекрасная идея с разных точек зрения. Очень полезно, что значительно усиливается конкретный характер теории, результатов, упражнений, отдельных тем; это облегчает понимание, требует хорошей точности, но и придает характер игры занятию, которое нередко воспринимается скучным и отталкивающим.

Но есть еще более важное обстоятельство. Богатство, присущее математике, чаще всего требует, как, впрочем, и в любой сфере деятельности, использования сразу нескольких средств очень разной природы. Самые большие успехи в математике почти всегда могут быть описаны именно так. В другой области, теоретической физике, показательна работа Луи де Бройля, объясняющая, что правильно понять свойства материи можно, лишь рассматривая ее одновременно с волновой и корпускулярной точек зрения. Эйнштейн поступил аналогично в отношении массы и энергии, Гильберт — с геометрией и функциональным анализом и т.д. Так и с момента возникновения алгоритмики всякая попытка пересмотреть тот или иной раздел математики с точки зрения алгоритмики может быть встречена с интересом и надеждой.

Так устанавливаются взаимные связи, природа которых богата и разнообразна. В некоторых случаях алгоритмика оказывается на службе у обычной математики, являясь тогда инструментом прикладной математики. Часто использование средств алгоритмики порождает новые области математических исследований. Проблемы сложности вычислений, составляющие значительную часть данного труда, — это как раз тот самый случай. Замечу, кстати, что именно к этой области принадлежит самая важная открытая проблема современной математики — сравнение сложностей P и NP ; если, основываясь на сравнительном подобию, распространить математику на все науки, то проблемы Римана, Пуанкаре, «обратная теорема Галуа» окажутся где-то недалеко от химии, тогда как проблема $P \neq NP$ не выйдет за пределы математики.

Иной раз алгоритмические проблемы порождают новый сюжет, который может быть затем полностью отделен от источника своего происхождения и превратиться в раздел «чистой» математики. Хотя исторически это и не вполне корректно, но таким образом можно представлять студентам гармонический анализ. Одна из глав этой книги посвящена дискретному преобразованию Фурье. Даже если бы Фурье не существовал и даже если бы никто с тех пор не мог его заменить, все равно алгоритмика умножения многочленов обязывает обнаружить так называемый анализ Фурье, его интерес, его богатство и его эффективность. А потом уже нетрудно, переходя к пределу, получить ряды Фурье и преобразование Фурье, исходя из дискретного преобразования Фурье; наиболее просто это сделать, используя нестандартный анализ — но берегись инквизиторов: сочетая две ереси в одном курсе, рискуешь сломать себе шею, если не быть осторожным или попросту скромным!

Другой интересный случай, представляющий, несомненно, очень богатый сюжет: для некоторой ранее существовавшей математической теории ее пересмотр может привести к таким точкам зрения, которые требуют иных методов, имеющих свой собственный интерес, не уменьшая, однако, тем самым, интерес к исходным теоретическим методам. Хорошим примером такого сорта является матричное исчисление. Классическая теория определителей, на базе знакопеременных полилинейных форм, непосредственно не используется при вычислении определителей. На самом деле ситуация даже еще лучше. Наши самые далекие предшественники, несомненно, были знакомы с сущностью так называемого метода Гаусса решения систем линейных уравнений. Однажды появились теоретики и вообразили, что лучше это делать с помощью определителей. Потом алгоритмисты снова поправили стрелки часов: для машины метод Гаусса намного лучше, чем метод Крамера, что, однако, вовсе не лишает интереса к этому методу, который с успехом используется для теоретического анализа первого. Этот и другие связанные с ним вопросы изыщно изложены в данной книге; содержащаяся в ней обширная библиография и комментарии позволят прилежным читателям быстро добраться до современных проблем, о богатстве которых они вначале и не подозревали.

Информатики сразу распознают одного из духовных наставников авторов: Дона Кнута. Отметим лишь составление макета книги, полностью реализованное самими авторами (без Postscript'a!) с помощью TEX'a. Мне не известна официальная градация в сообществе TEX пертов, но, несомненно, Патрис Ноден и Клод Китте в его первом эшелоне.

Вошло в поговорку безграничное богатство анализа всех аспектов проблемы в книгах Кнута; то же самое мы обнаруживаем здесь. Для читателя, располагающего достаточным временем, эта книга — неисчерпаемый кладёзь возможностей рассматривать отдельную тему с множества точек зрения. Когда, говоря о программе, авторы решаются перейти к следующей теме, они не забывают заранее указать полезную библиографию; часто это даже библиография в квадрате, содержащая цитируемые тексты, которые, в свою очередь, также содержат обширную библиографию — изящный пример использования подпрограмм. Как и у Кнута, в этой книге имеется обширный набор упражнений разного уровня — от простых примеров, предназначенных проверить понимание того или иного понятия, до самых сложных задач, приближающихся к темам научных исследований. И, как и у Кнута, решения упражнений также включены в книгу!

Для изучения алгоритмов требовалась техническая поддержка в виде языковых средств информатики. Авторы выбрали язык Ада — безусловно, один из лучших на сегодня языков. И пусть потенциальные пользователи не волнуются по поводу выбора языка, все еще не слишком распространенного во французских университетах: перевод Ада-программ в их любимый язык является несложным техническим упражнением; при этом строгость документации программ просто поразительна и может служить примером, так что на этот счет нет никаких опасений. Такой перевод, требующий ясно отличать алгоритмическую часть от реализации, очень поучителен. Кроме того, авторы зачастую предварительно дают описание алгоритмов в чистом виде в соответствии с аксиоматикой Хоара.

Это превосходная книга, которую я ставлю в моей библиотеке на полку со «священными» книгами, где находятся уже два «евангелия» — алгоритмика по Вирту и алгоритмика по Кнуту. Теперь я обладаю третьим «евангелием»; кто же напишет четвертое? Но ведь эта книга потребовала коллективной работы двух авторов — еще одно примечательное свойство, так что может быть моя коллекция «евангелий» уже полна? Нет: это противоречило бы Гёделю, но это уже совсем другая история.

Франсис Сержераер,
Мейлан, 27 сентября 1991 г.

От авторов

Вот и наступил момент, когда, закончив книгу, автор начинает свое введение и ищет подходящие формулировки, которые убедили бы читателя, что все дальнейшее было спланировано, обосновано... с самого начала. Мы не будем отступать от этого правила. Однако, скажем откровенно, наша точка зрения изменялась многократно с того момента, когда мы принялись за этот труд.

Вопрос «Возможно ли вычислять математические объекты?» является лейтмотивом этой работы; как можно убедиться, эта книга не дает окончательного ответа, но, надеемся, показывает, что этот вопрос не лишен интереса. В особенности в той области, которую затрагиваем здесь, а именно, в алгебре, мы убеждены, эффективное вычисление объектов — вручную или с помощью компьютера — проливает свет на используемые понятия. Но, начиная с таких элементарных понятий, как кольцо целых чисел по модулю p , и до более абстрактных структур, как модули в основных кольцах, что же все-таки можно вычислять и как? В начале работы мы были совсем уже готовы уступить намерению рассматривать алгебру без каких бы то ни было формальных средств, а только в чисто вычислительном аспекте; но изучение преобразования Фурье — и, в особенности, работы Ш. Винограда — убедительно подтолкнули нас к тому, чтобы избежать этого соблазна. В результате проект, сознательно ограниченный вначале, стал более полновесным и содержательным; мы даже позволили себе сочинить полную главу, посвященную дискретному преобразованию Фурье, с включением в нее билинейных форм и тензорного произведения; добавили раздел, излагающий классический подход к факторизуемости колец многочленов...

Однако мы не отказались от главной идеи, которая составляет основу этой книги: можно вполне конкретно представить понятия элементарной алгебры, т.е., опираясь на учебные примеры и приводя, по мере возможности, конструктивные доказательства.

Возьмем для примера теорему Цермело, согласно которой пространство можно преобразовать во вполне упорядоченное множество; приверженца Кантора будут очарованы строгостью доказательства — действительной и кажущейся; прагматики им возразят: Вы утверждаете, что можете преобразовать пространство во вполне упорядоченное множество; ну что ж, преобразуйте его! — Это слишком долго. — Тогда покажите нам по крайней мере, кого-то, кто нашел бы достаточно времени и терпения и кто смог бы выполнить это преобразование. — Нет, мы это не можем, потому что число необходимых для этого операций бесконечно, оно даже превосходит \aleph_0 . — А можете ли вы показать, как можно было бы выразить конечным числом слов то правило, которое позволило бы упорядочить пространство? — Нет — и прагматики делают вывод, что эта теорема или лишена смысла, или неверна, или же, по меньшей мере, недоказуема.

Анри Пуанкаре, *Последние мысли* (1913 [161])

Это идет вразрез с общепринятой практикой, которая заключается в преподавании математики на абстрактном уровне, без опоры на эксперимент, как в других научных дисциплинах. Скольких хлопот избегают при этом! Гораздо легче, например, доказать существование для любого простого числа p примитивного по модулю p многочлена, нежели построить процедуру, порождающую за 5 шагов в периоде длины 32 примитивный по модулю 2 многочлен $X^5 + X^2 + 1$.

Эта книга рискует удивить читателя; в ней иногда объясняются очень простые вещи, а иногда быстро проходят через более сложные понятия. Часто используются понятия группы, модуля, кольца, тела и т.п., но ни одно из этих понятий не определяется; существуют прекрасные книги по алгебре, содержащие эти определения, и нам хотелось не перефразировать их, а скорее попытаться взглянуть на эти структуры с точки зрения эффективности. Всякий раз, когда нам это представлялось возможным, мы стремились заменить традиционную манеру изложения темы и показать многосторонние связи между различными понятиями (например, различные типы колец в теории делимости, сходство между конечными абелевыми группами и редукцией эндоморфизмов и т.д.). Выбор рассматриваемых тем не бесспорен. Мы мало внимания уделяли многочленам, которые, однако, являются фундаментальными объектами в алгебре; в то же время сравнительно много внимания посвящено дихотомическому алгоритму возведения в степень и алгоритму Евклида. Цель, которую мы перед собой ставили, — изучить небольшое количество методов, относящихся к области, называемой ныне алгебраической алгоритмикой, стараясь связать их между собой и с классическими понятиями алгебры. Этот вывод привел нас к сознательному отказу от рассмотрения таких сюжетов как, например, теория Галуа или теория групп, которые очевидным образом допус-

кают алгоритмический подход. Таким образом, мы стремились предоставить необходимые средства для углубленного изучения этой дисциплины, оседлавшей математику и информатику. В действительности, существо описанных в данном курсе методов сконцентрировано в четырех основных алгоритмах: дихотомический алгоритм возведения в степень, алгоритм Евклида, китайская теорема об остатках и быстрое преобразование Фурье; они образуют основу для любой системы формальных вычислений. Тем не менее, в упражнениях содержатся введения в различные темы: от пошаговой трассировки до символа Якоби, включая р-адические числа, непрерывные дроби, многоразрядную арифметику...

Введение в предмет составляют основы информатики, необходимые для того, чтобы недвусмысленно рассуждать об алгоритмах, об их обоснованиях, о программах. Объекты информатики, определяемые здесь, очень разнообразны: от начал программирования очень быстро переходим к таким понятиям, как модулярность, шифрование информации, порождаемость..., затрагивая весьма тонкую технику программирования на языке Ада. И вот великое слово произнесено! На самом деле большинство реализаций алгоритмов, которые мы даем, написано на языке Ада. Многие удивлялись, что не был выбран Лисп (и мы готовы спорить), Паскаль или Си (в данном случае спор с нами бесполезен). Ада обладает той строгостью, которая, как нам кажется, вполне гармонирует с математической строгостью; мы также широко пользовались богатством систем контроля в языке Ада как для программирования, так и для описания алгоритмов.

Программисту на языке Лисп известно значение всего, но он никогда не знает, сколько это стоит.

Алан Дж. Перлис, *Программистские эпиграммы* (1978)

Пусть сторонников языка Лисп не смущает эта цитата. Мы далеки от того, чтобы быть невосприимчивыми к этому языку. И если мы по достоинству ценим определенные аспекты языка Ада, то мы всегда сожалели о хронической нехватке некоторых базовых средств в этом языке (арифметика высокой точности — лишь один тому пример).

Выбор материала по информатике для включения в эту книгу был труден: она адресована в первую очередь не информатикам, а математикам, чьи познания в информатике не всегда достаточно основательны. Тем не менее, в этой книге мы не предполагаем, что читателю

совершенно чужд этот предмет; однако, возможно, он будет удивлен не обычным подходом в математике. В информатике не редкость, когда формулировка проблемы бывает такой же длинной, как и ее решение.

В книге приведены несколько полных программ; они предназначены не для «производственных целей», а для построения поучительных примеров. И хотя мы не включили все программы, реализующие описанные методы, — книга бы просто лопнула... так же, как и наш издатель — тем не менее, все описанные методы реализованы или на языке Ада, когда имевшиеся в нашем распоряжении средства это позволяли, или в среде системы формализованных вычислений Scratch pad. Эта система была выбрана не за свои мнимые достоинства или за как будто бы богатое программное обеспечение, а исключительно потому, что она была нам доступна и позволяла умножать числа с 100 знаками. И если слабость и неадекватные реакции ее компилятора довольно быстро обескуражили одного из авторов, то другой еще продолжает борьбу.

Мы должны особо упомянуть наш основной источник вдохновения — монументальный труд Дональда Кнута «Искусство программирования». Часто нам случалось самостоятельно или в процессе чтения открыть некий новый алгоритм; так вот, мы можем утверждать, что более, чем в 70% случаев, оказывалось, что этот алгоритм уже был описан в его книге и чаще всего гораздо лучше, чем там, где мы его нашли. Мы не можем упоминать Кнута на каждой странице, где явно присутствует его вклад, ибо его имя появлялось бы сотни раз. Мы это делаем здесь.

Помимо этой серии книг, еще не оконченной в настоящий момент, Кнут также реализовал — чтобы иметь возможность точного написания своих книг — компилятор текстов поистине феноменальной мощности. Эта книга смогла быть написана ее авторами-любителями благодаря Кнуту и многочисленным бессонным ночам в попытках понять внутренние механизмы Т_EX'a. Без Т_EX'a мы никогда бы не смогли реализовать нашу книгу в данном виде. И вот теперь мы можем, на манер Кнута, сказать: *If this book has any merit, the credit should go to Knuth... On the other hand, if this book has deficiencies, the blame should be directed to us ...*¹

Даниель Лазар и Франсис Сержераер первыми связали математику и информатику в процессе преподавания в Пуатье. Мы им обязаны определенной концепцией преподавания этих дисциплин и считаем своим долгом поблагодарить их здесь.

Эта книга не является творением двух отдельных личностей; мы пользовались советами наших многочисленных коллег. Мы особенно благодарны Пьеру Берна, Жан-Мари Гурсо, Жан-Луи Паско, Мустафе Раису, Ги Рено и Жаку Валетту; мы обязаны не только идеями упражнений, доказательств, но также и другой точкой зрения на ряд методов, излагаемых в

¹Если эта книга имеет достоинства, это заслуга Кнута... С другой стороны, если в этой книге есть недостатки, то упрек должен быть адресован нам...

этой книге. Мы благодарим также сотрудников факультета математики и информатики университета Пуатье. В течение всех тех лет, когда шла работа над книгой, мы ощущали их поддержку и симпатию. Успешное продвижение проекта было значительно облегчено предоставлением места работы в лаборатории первому автору и творческого отпуска — второму.

Мы не забываем студентов специального курса «Информатика для математиков», которые на протяжении многих лет вносили различные поправки и предлагали идеи.

Мы не можем на этом закончить, чтобы не подмигнуть Жаку Боровчику, без которого не осмелились бы написать эту книгу, а также Жан-Пьеру Раду, виновному за выбор языка Ада, Ливью Соломону, дававшему нам советы по разным случаям, и Бернару Гарро, который неотлучно был с нами в течение всей этой работы.

Что же касается якобы очень взаимообогащающей работы вдвоем, то это не всегда было очевидным, и проблемы не всегда были столь же просты, как эта: « — Почему ты вставляешь цитаты по-английски? » Некоторые читатели не поймут. — Я предпочитаю оригинальную версию, которая содержит в себе порой непере译имый юмор. — Ну, а как насчет цитат на немецком? — Ну это совсем другое дело. Я не знаю немецкого».

Эта книга посвящается всем нашим друзьям, которые нас поддерживали и мирились с нашим переменчивым настроением в течение этих трех сумасшедших лет.

Глава I

Алгоритмика и программирование на языке Ада

В этой главе — вероятно самой «информатической» в нашей книге — мы введем некоторые алгоритмические понятия и опишем несколько получисленных алгоритмов, необходимых для реализации алгебраических методов.

Точное определение алгоритмических границ, в которых будут изложены методы вычисления, станет предметом особого внимания. Цель нашего труда не ознакомить с основами алгоритмики — предполагается, что читатель не новичок в данной области, — а определить без укоризненным способом семантику построений, используемых в алгоритмах¹.

После изложения общих направлений мы приступим к исследованию первого фундаментального алгоритма этой книги: дихотомического алгоритма возведения в степень. Для этого будут использоваться два метода построения алгоритмов. Один очень близок к математическим рассуждениям, другой — чисто алгоритмический. При этом у нас появится возможность открыть первый рекурсивный алгоритм, представленный в данной работе.

Начав с первого примера алгоритма, мы перейдем к изучению наглядных понятий программирования на языке Ада. И здесь тоже речь пойдет не об изложении полного курса языка Ада, а только о пред-

¹Что совсем не умаляет ни корректность (верность) программ, которые можно будет потом получить, ни соответствие компилятора его спецификациям. Эти условия, однако, необходимы для истинности математической теоремы, доказанной с помощью компьютера.

ставлении нескольких ключевых понятий, необходимых для понимания приведенных программ.

Вообще говоря, все понятия будут представлены (по мере надобности) на примерах, во избежание их формального описания (по последнему вопросу лучше обратиться к справочному изданию: Reference manual for the Ada programming language¹).

И наконец, завершит эту главу разработка совершенно неэффективного метода вычисления определителей целочисленных матриц. Обсуждаемый метод, основанный на математическом определении детерминанта, позволяет наилучшим образом использовать все возможности современных компьютеров и получить идеальное приближение к бесконечности на машине! Кроме того, обсуждение ситуации позволит нам несколько пополнить представление о программировании на языке Ада.

1. Введение в алгоритмику

Прежде чем приступить к описанию любого алгоритма, нужно говорить общепринятые соглашения, используемые в работе. В информатике чаще, чем в любой другой дисциплине, встречается взаимное непонимание между пользователем и разработчиком новых программ. Поэтому так важно сформулировать аксиоматику, позволяющую свободно рассуждать об алгоритмах.

Множество аксиом, которое мы собираемся сейчас описать, известно как *аксиоматика Хоара* [86], в честь исследователя, который первым формализовал семантику структур и действий последовательной алгоритмики.

1.1. Терминология и обозначения

Алгоритм — это последовательность более или менее элементарных действий, которая позволяет пошагово решить поставленную задачу за конечное время. Можно формально по индукции определить, что такое алгоритм, нижеследующим образом. Цель этого введения в алгоритмику — изучение и представление всех терминов, фигурирующих в предыдущем определении (стиль определения, который в информатике называется **грамматикой**).

Можно описать алгоритм в терминах состояний системы: состояние системы до применения алгоритма, состояние той же системы после

¹См. книгу [186]. — Прим. перев.

применения алгоритма. Эти два состояния образуют то, что называется спецификацией задачи. Таким же образом можно, описывая поэтапно

последовательные состояния (между действиями) рассматриваемой системы, проводить обоснование алгоритма. Система доказательства по Флойду[71] — это способ формализации данного процесса. Но эта аксиоматика не просто метод доказательства: в случае ее обобщения она может претендовать на роль хорошей методологии программирования.

- **Алгоритм — это действие**
- **Действие — это:**
 - **элементарное действие**
 - **последовательность действий**
 - **альтернатива**
 - **итерация**
- **Элементарное действие — это:**
 - **пустое действие**
 - **присваивание — ...**

Состояния данной системы выражаются в виде утверждений, т.е. логических формул. Рабочие гипотезы алгоритма называются предусловиями или данны ми алгоритма. Заключение называются постусловиями или результатам и. Вот, например, способ формального описания алгоритма:

[предусловия] Алгоритм [постусловия]

И это то, что можно назвать **спецификацией** алгоритма. Используя данную форму выражений, можно описать аксиомы Хоара в следующем виде:

▷ *Посылки* : предварительные условия к применению аксиомы. Эти условия выражаются, в основном, в терминах преобразований утверждений посредством некоторых действий.

▷ *Заключение* : результат построения, которым описывают аксиому.

1.2. Присваивание

Это элементарное действие, наиболее опасное и наиболее трудное для понимания в последовательных алгоритмах. Его семантика чрезвычайно проста: переменной (иногда очень сложной) придают значение какого-либо выражения. Действие присваивания представляется в алгоритмах, как правило, двумя способами: обычно с помощью стрелки, направленной влево, а иногда последовательностью знаков «:=», как, например, в языках программирования.

(1) Аксиома (присваивания).

▷ *Посылки*: x является переменной, e — выражением

▷ *Заключение*: $[] x \leftarrow e [x = \text{значению } e]$

Интерпретация этой аксиомы следующая: нет никакого предусловия, кроме природы его операндов, для использования присваивания. Каково бы ни было состояние системы, сведенной к переменной x , после выполнения присваивания x будет иметь то же самое значение, что и e .

Если задействовано несколько переменных, то иногда форму постуловия присваивания модифицируют. Возьмем, например, систему, состоящую из двух переменных x и y , предположим к тому же, что y имеет значение e . Выполним предыдущее присваивание. В предусловие можно включить переменную y , хотя она не входит в присваивание. В результате имеем: $[y = e]$. Довольно часто вместо того, чтобы писать постусловие в виде $[y = e \text{ и } x = e]$, запись сокращают до $[x = y]$, что, разумеется, приводит к исчезновению некоторой начальной информации.

Все это может показаться тривиальным, и обычно совершаемая ошибка состоит в смешении присваивания с частной формой записи уравнений (в математическом смысле) или подстановкой. Простой пример, иллюстрирующий трудность проблемы, относится к системе из двух переменных и предыдущему условию

$$[\mathfrak{R}(x, n)]n \leftarrow n + 1[\mathfrak{R}(x, n - 1)]$$

где \mathfrak{R} - отношение, связывающее соответствующие значения двух его аргументов.

1.3. Последовательность

Это алгоритмическая конструкция, заключающаяся в соединении действий в цепочку. Ее часто представляют с помощью знака «;», который служит связкой между последовательными действиями. Эта конструкция самая простая среди тех, которые встречаются в последовательных алгоритмах.

Если A и B — действия, то обозначим через « $A ; B$ » действие, состоящее из последовательности действий A и B в данном порядке. Эта горизонтальная запись обладает некоторыми неудобствами, так что очень часто ту же последовательность действий записывают, располагая вертикально на одной прямой действия одной и той же последовательности (и аннулируя, в случае необходимости, знак «;»).

(2) Аксиома (последовательности).

▷ *Посылки:* A и B — действия, P , Q и R — утверждения

$$[P] A [Q] \text{ и } [Q] B [R]$$

▷ *Заключение:* $[P] A; B [R]$

Эта аксиома просто выражает тот факт, что в последовательности двух действий предусловие второго является постусловием первого. Иными словами, два действия одной и той же последовательности, следующие одно за другим, действуют совместно. Второе использует результаты первого.

Вот очень простой пример, позволяющий конкретизировать предыдущее высказывание. Рассмотрим систему, образованную тремя целочисленными переменными: C , x и n , и предположим, что эти переменные удовлетворяют следующему свойству (которое будет предусловием этой последовательности): $[C \times x^n = x_0^{n_0}]$, где x_0 и n_0 — две целочисленные константы. Последовательность действий, которую при-

меняют к этой системе, представлена справа. Преобразуем сначала исходное утверждение, используя тот факт, что $n > 0$ и $[C \times x^{n-1} \times x = x_0^{n_0} \text{ и } n > 0]$, и применим первое действие последовательности, придав C значение, полученное при умножении его начального значения на x . Это выражается в предыдущем утверждении слиянием двух первых термов левого члена равенства: $[C \times x^{n-1} = x_0 \text{ и } n > 0]$. Потом, рассматривая это новое утверждение как предусловие, применим второе действие, которое состоит в уменьшении значения n на единицу.

$\begin{aligned} C &\leftarrow C \times x \\ n &\leftarrow n - 1 \end{aligned}$

Таким образом, получаем постусловие рассматриваемой последовательности действий $[C \times x^n = x_0^{n_0} \text{ и } n \geq 0]$. Это последнее преобразование является примером типичного случая, упомянутого в конце предыдущего раздела.

1.4. Альтернатива

Альтернатива — это структура управления, соответствующая принятию решения в алгоритме. Ее часто выражают в виде «если ... то ...». Во всех алгоритмах, представленных в книге, структуры управления и примитивные конструкции будут записаны на английском языке, в виде схем, близких к языку Ада. Этот прием имеет два важных преимущества: во-первых, алгоритмы пишутся на алгоритмическом диалекте, приближенном к языку программирования; и потом, выбранный язык Ада чрезвычайно точен (небольшой эвфемизм!) и достаточно понятен для записи алгоритмов.

Нет необходимости объяснять действие структуры управления. Ее семантика тождественна семантике, используемой в разговорном языке.

(3) Аксиома (Альтернатива).

▷ *Посылки:* A и B — действия; C — предикат, P и Q — условия.

$$[P \text{ и } C] A [Q] \text{ и } [P \text{ и } (\text{не } C)] B [Q]$$

▷ *Заключение:* **if** C **then** A **else** B **end if** $[Q]$

Прежде всего эта аксиома выражает тот факт, что какова бы ни была ветвь используемого теста¹, получаем один и тот же результат (благодаря разделению действий A и B).

Если продолжить анализ, то из этой аксиомы можно извлечь некоторую философию (или методологию) программирования. В самом деле, можно субъективно объяснить такой подход и сделать вывод, что хорошая альтернативная структура (т.е. альтернатива, простая для понимания доказательства) — это конструкция, в которой действия, представленные в каждом из ответвлений, очень похожи. Не редко, например, можно встретить в программах начинающих, а также в работах лю дей более опытных (что красноречиво свидетельствует об их опыте!) альтернативные структуры, в которых в зависимости от того, является истинным или ложным предикат, совершается вычисление с внутренними переменными или выполняется действие с промежуточным результатом (обычно на экран выводится сообщение). Конечно, эти програм мы могут быть правильными и, следовательно, вполне доказуемыми, но ценой каких искажений в изложении алгоритма и в формулировках утверждений! Не говоря уже о том, какой будет через несколько месяцев степень понятности программы этого программиста! Зато хороший алгоритм не содержит особенностей, сопровождается простыми и ясными утверждениями и потребует намного меньше затрат во время его возможной доработки.

В качестве примера рассмотрим следующую альтернативу, представляющую небольшую часть дихотомического алгоритма возведения в степень, который мы рассмотрим в дальнейшем. Возьмем снова систему, состоящую из трех целочисленных переменных C , x и n , удовлетворяющую условию:

$$[C \cdot x^n = x_0^{n_0} \text{ и } n > 0] \quad (1)$$

Исходя из этой системы переменных, применим нижеследующий элементарный алгоритм, который в основном записывают в виде:

► **if** n *нечетно* **then** $C := C * x$; **end if**; ◀ Ключевое слово **null**

¹Ветвью теста называют каждую часть альтернативной структуры: часть «то*» или часть «в противном случае»

if (*n* - нечетно)
 $C = C \times x$
else *null*

является примитивом алгоритмического языка, выражающим пустое действие (для структура листов это действие является нейтральным элементом ассоциативной операции последовательной композиции).

Семантика пустого действия такова (для любого утверждения *P*): **null** [*P*].

Итак, необходимо отдельно исследовать влияние действий каждой из двух ветвей теста на исходное утверждение, учитывая значение предиката «*n* нечетно». Сначала можно записать: $n = 2q + r$, деление с остатком n на 2; q — это неполное частное, а r — остаток. Когда n и p — целые числа удобно обозначить через n/p неполное частное от деления с остатком n на p и через $n \bmod p$ — остаток при этом делении; учитывая вышесказанное, получаем: $n = 2(n/2) + n \bmod 2$. Кроме того, значение $n \bmod 2$ выражает четность n . Исходное утверждение может быть преобразовано в

$$[C \times x^{2(n/2)+n \bmod 2} = x_0^{n_0} \text{ и } n > 0]. \quad (2)$$

Теперь рассмотрим две ветви альтернативы. Если n нечетно, то $n \bmod 2$ равно 1 и исходное утверждение переписывается в виде: $[C \cdot x \cdot x^{2(n/2)} = x_0^{n_0} \text{ и } n > 0]$ — выражение, которое легко преобразуется после применения действия $C \leftarrow C \times x$ в

$$[C \cdot x^{2(n/2)} = x_0^{n_0} \text{ и } n > 0]. \quad (3)$$

Аналогичным образом, если n четно, то $n \bmod 2$ равно 0 и без всякого преобразования утверждение (2) становится тождественным (3).

Итак, утверждение (1) преобразуется с помощью алгоритма в утверждение (3) при любом значении предиката «*n* нечетно».

В примере, рассмотренном выше, и в исследовании последовательности, представленной в предыдущем разделе, мы всегда начинали доказательство с преобразования исходного утверждения. Преобразование, которое может показаться магическим и без которого трудно понять результат вычислений. В обычной практике алгоритмики все происходит иначе; действительно, чрезвычайно сложно, если вообще возможно, обосновать уже написанный алгоритм, и эта задача становится еще более сложной для автора доказательства, если только он не автор алгоритма. Чтобы правильно составить алгоритм, нужно сначала формально рассмотреть решаемую задачу, потом вывести несколько утверждений, описывающих основные этапы решения, и, наконец, построить одновременно алгоритм и его доказательство.

Но, разумеется, приведенные примеры являются всего лишь учебными. Их цель — разъяснить аксиомы, у которых, как известно, ясность — не главнейшее достоинство!

1.5. Итерация

Существует много итеративных структур управления во всех языках программирования, а также во всех алгоритмических формализмах. На самом деле, все эти структуры образуются от одной и той же элементарной структуры управления: обобщенной итерации. Обобщенная итерация построена с помощью примитивов **loop**, **end loop** и **exit**. Два первых примитива **loop** и **end loop** являются, соответственно, левой и правой скобками, ограничивающими действие или последовательность действий, которая должна повторяться, априори, бесконечно. Примитив **exit** допускает выход из цикла, в котором он представлен; этот примитив всегда ассоциируется с одним или несколькими условиями выхода из цикла и часто появляется в алгоритмах в виде **exit when**.

Справа представлена наиболее общая форма итерации. Эта запись достаточно ясна и не требует других объяснений. На практике, что бы сохранить четкость алгоритмов (что означает также их понимание), мы ограничимся использованием циклов, имеющих, самое большее, два выхода. Ни теперь, ни в дальнейшем, мы не дадим самой общей аксиомы (правильнее сказать мета-аксиомы) итерации; понимание нескольких частных случаев итераций, изучаемых в следующих разделах, позволяет легко ввести любую нужную аксиому.

Любопытно отметить, что в практическом плане концепции алгоритмов альтернатива, определенная в предыдущем разделе, может быть представлена в виде обобщенной итерации (теоретические последствия см. [25]).

```

while(true)
{
    Действие 1
    if(Условие 1) break;
    Действие 2
    if(Условие 2) break;
    .
    .
    if(Условие i) break;
    .
    .
    if(Условие n) break;
    Действие n+1
}

```

1.6. Итерация со счетчиком или цикл «для»

Конечно, это самая распространенная форма итерации. Она используется, когда точно известно количество повторений действия, необходи-

Итерация со счетчиком

```
for(int  $i = p; i < q; i++$ )
{
    Повторяющееся действие
}
```

мого для решения данной задачи. Схема слева представляет сосчитанную итерацию, в которой i — это переменная, p и q — выражения. Запись $p..q$ обозначает интервал, у которого нижняя и верхняя

границы — соответственно p и q (что приводит, в частности, к возможности обозначения пустого интервала). Результатом этой конструкции является повторение внутреннего действия цикла, называемого **телом цикла**, для всех значений **индекса цикла** i в заданном интервале. Это построение обладает как преимуществами, так и некоторыми ограничениями:

- по своему применению переменная i (индекс цикла) относится к командной части цикла (это семантика индекса цикла в языке Ада),
- следовательно, эта переменная существует и имеет определенное значение только в теле цикла и было бы неправильно и бессмысленно использовать ее предполагаемое значение после выполнения цикла.

(4) Аксиома (цикла «для»).

▷ *Посылки:* $A(i)$ — действия, зависящие от значения i ; $P(i)$ — условие, при котором i — свободная переменная и выполнено:

$$[P(i-1)p \leq i \leq q]A(i)[P(i)]$$

▷ *Заключение:* $[P(i-1)]$ **for** i **in** $p..q$ **loop** $A(i)$ **end loop** $[P(q)]$

Эта аксиома, действительно, выражает в алгоритмике метод, по добрый принципу индукции в математике. Можно также выразить эту аксиому в слегка отличной форме, изменив индексацию утверждений или действий... Все зависит от контекста и от ясности доказательства.

```
 $i = p - 1;$ 
while(true)
{
    if( $i \geq q$ ) break;
     $i++$ ;
     $A(i)$ ;
}
```

Теоретически цикл «для» может быть реализован с помощью обобщенной итерации, как в алгоритме слева. Однако, на практике реализация этим упрощенным методом цикла «для» с помощью компилятора не проводится.

Конструкция, представленная в последнем алгоритме, — это итерация с выходом в начале цикла. Она так часто используется, что все языки программирования, достойные так называться, имеют

особую структуру для ее выражения: цикл «пока».

1.7. Цикл «пока»

Справа дана общая форма цикла «пока»; его действие заключается в следующем: проверяется условие, и, если оно верно, то действие, представленное в теле цикла, выполняется; затем снова проверяется предикат, и процесс повторяется до тех пор, пока проверка условия не даст значение «ложно». Конечно, если предикат ложный с самого начала, никаких действий не производится: тело цикла не выполняется.

Цикл пока
while(Условие)
 {
 Действие
 }

(5) Аксиома (цикла «пока»).

▷ *Посылки:* A — действие; P — утверждение; C — предикат;

$$[P \text{ и } C] A [P]$$

▷ *Заключение:* $[P] \text{ while } C \text{ loop } A \text{ end loop } [P \text{ и } (\text{не } C)]$

Более точно, цикл «пока» является аббревиатурой более общего цикла:

loop exit when not C; Действие; end loop

В аксиоме цикла «пока» заложен более общий принцип рассуждения, чем рекуррентность, — индукция. Из сформулированной аксиомы можно сразу же сделать вывод, что любое свойство P , выполненное до входа в цикл и сохраняющееся при выполнении тела цикла, остается верным после окончания выполнения цикла. Такое утверждение P называется **инвариантом цикла**.

Кроме того, при выходе из цикла предикат C , называемый **условием циклического перехода**, — ложный. Этот последний феномен необходимо рассмотреть более подробно, так как он затрагивает важный этап доказательства алгоритма. До сих пор все изученные алгоритмические конструкции были построениями, выполнение которых ограничивалось во времени: последовательность, альтернатива или итерация со счетчиком заканчиваются за определенное время, если предположить, что составляющие их действия выполняются за конечный отрезок времени. Но это не относится ни к циклу «пока», ни к любой другой итерации, более общей, чем цикл «для». Эти последние структуры управления потенциально бесконечны во времени, а на практике ограничены только проверкой условий выхода (когда они есть!).

Таким образом, обоснование алгоритма, в котором представлены одна или несколько итераций, должно содержать две части: доказательство частичной корректности, цель которого — показать, что если

Гипотеза Сиракуз
while($n \neq 1$)
 {
 if($n \% 2 == 0$) $n = n/2$;
 else $n = (3*n + 1) / 2$;
 }

алгоритм завершен, то он дает ожидаемый результат, и доказательство выхода, под тверждающее, что всякое выполнение алгоритма при любом наборе данных завершается по истечении определенного времени. Эти два этапа иногда очень отличаются по сложности; наиболее простой пример, известный как *гипотеза Сиракуз*, представлен напротив. Доказательство частичной корректности этого алгоритма, принимающего целое положительное число n в качестве исходного данного, является простым: если он завершен, то он преобразует n в 1; однако в настоящее время никто не может доказать, что он завершается при любом положительном значении n (см., например, Терра [169], Лагария [108] или Арзак [7]).

Мы не будем пока приводить примеры алгоритмов (и доказательств), использующих итеративные структуры. Это будет сделано во время исследования первых получисленных алгоритмов.

Мы не будем пока приводить примеры алгоритмов (и доказательств), использующих итеративные структуры. Это будет сделано во время исследования первых получисленных алгоритмов.

1.8. Итерация (продолжение и окончание)

while(true)
 {
 Действие 1
 if(C) *break*;
 Действие 2
 }

Прежде, чем перейти к следующему разделу, в котором будет представлен дихотомический алгоритм возведения в степень — один из основных алгоритмов для алгебраических вычислений — и его реализация на языке Ада, опишем третью особую форму итерации и ее аксиоматику. Основываясь на этом, читатель сможет обобщить и изобрести аксиому, адекватную любой форме итерации. Эта третья форма итерации является итерацией с одним выходом, расположенным в середине цикла.

Вот, без объяснений (предыдущие параграфы в полной мере развили интуитивный аспект аксиом), аксиома, управляющая действием этой структуры.

(6) Аксиома (итерации).
 ▷ *Посылки:* A и B — действия; C — предикат; P и Q — утверждения;

$$[Q] A [P] \text{ и } [P] \text{ и } (\text{не } C)] B; A [P]$$

▷ *Заключение:* $[Q] \text{ loop } A \text{ exit when } C; B \text{ end loop } [P \text{ и } C]$

2. Дихотомический алгоритм возведения в степень

В настоящем разделе рассматривается дихотомический алгоритм возведения в степень. В общем виде он позволяет вычислить n -ю степень элемента в моноиде. Будучи применен к множеству целых чисел с операцией сложения, этот метод позволяет умножать два целых числа и более известен как египетское умножение.

Рассматриваемый алгоритм — один из наиболее употребительных в методах алгебраических вычислений. В самом деле, наряду с алгоритмами Евклида, возведение в степень является основой большинства арифметических вычислений в конечных кольцах $\mathbb{Z}/p\mathbb{Z}$, в частности, для проверки простоты чисел Ферма и Мерсенна, нахождения квадратичных вычетов и т.д.

Будут использованы два совершенно различных подхода для представления этого алгоритма: алгоритмический подход, наиболее распространенный в курсах информатики, и более классический, математический подход. Эти два подхода позволяют дать, с одной стороны, доказательство (корректности) некоторого очень простого алгоритма, а с другой — показать, как на основе явных математических формул можно построить соответствующий вычислительный алгоритм. В алгоритмах, которые мы в дальнейшем представим, будут равным образом использованы оба эти подхода в зависимости от природы решаемой задачи.

Классический алгоритм возведения в степень посредством последовательного умножения характерен, главным образом, своей неэффективностью в обычных обстоятельствах - его время работы линейным образом зависит от показателя степени (упражнение 28 показывает, что в более сложных случаях это не так).

Дихотомическое возведение в степень, несмотря на логарифмическую оценку сложности в зависимости от показателя степени, не является наилучшим из всех известных методов (цепочки сложений иногда дают лучшие результаты, см. Кнут [99]), но оно, вне всякого сомнения, является самым простым из них. Основным его предположением является то, что время вычисления операции умножения является константой независимо от того, к каким элементам эта операция применяется. Если же это не так (например, в кольце полиномов), то существует много других простых и более эффективных методов (см., в частности, упражнение 28).

2.1. Математический (или рекуррентный) подход

Возьмем моноид M^1 с операцией умножения и рассмотрим некоторый элемент x_0 из M , а также произвольное натуральное число n_0 . Для того, чтобы вычислить $x_0^{n_0}$, представим n_0 в двоичной системе счисления:

$$n_0 = b_t 2^t + b_{t-1} 2^{t-1} + \dots + b_1 2^1 + b_0 2^0,$$

предполагая, что n_0 содержит $(t+1)$ двоичных цифр (т.е. что $b_t \neq 0$ и $b_{t+1} = 0$). В этих условиях вычисляемое выражение может быть записано:

$$x_0^{n_0} = \prod_{i=0}^t x^{b_i 2^i} \text{ или же } x_0^{n_0} = \prod_{i=0}^t x^{2^i}.$$

Если задана последовательность: $(x_i)_{0 \leq i \leq t}$, первый элемент которой есть x_0 , и x_i для $i \in [1, t]$ определено соотношением $x_i = x_{i-1}^2$, то можно записать $x_0^{n_0} = \prod \{x_i | 0 \leq i \leq t, b_i \neq 0\}$. Чтобы завершить построение алгоритма и иметь возможность получить значение предыдущего произведения, необходимо вычислить биты b_i , числа n_0 .

Читатель легко сможет проверить для последовательности $(n_i)_{0 \leq i \leq t}$ (с начальным элементом n_0 , определенной соотношением $n_i = [n_{i-1}/2]$ для любого $i \in [1, t+1]$, что бит b_i равен нулю, если n_i четно, и равен единице в противном случае и что первое значение индекса i для которого n_i равно нулю, есть $t+1$. Предыдущее рассуждение позволяет представить «математическую» версию дихотомического алгоритма возведения в степень с исходными данными x_0 (элемент моноида M) и n_0 (некоторое натуральное число) и результатом $P = x_0^{n_0}$.

```
P = 1;
i = 0;
while(n[i] != 0)
{
  if(n[i] % 2 != 0)
  {
    P = P * x[i];
    n[i+1] = n[i] / 2;
    x[i+1] = x[i] * x[i];
    i++;
  }
}
return P;
```

Этот алгоритм представляет собой простой синтез предыдущих вычислений, представленный в алгоритмической форме вместо того, что бы быть выраженным с помощью лемм, свойств и т.п. Заметим, что в процессе построения алгоритма появляется переменная x_{t+1} , которая

¹Напомним, что моноид - это структура, состоящая из множества и определенно го на нем внутреннего ассоциативного закона композиции и содержащая нейтраль ный по отношению к этому закону элемент. Например, множество всех слов, которые можно записать с помощью букв фиксированного алфавита, наделено структурой моноида, если добавить еще операцию конкатенации (закрывающуюся в приписыва нии одного слова в конец другому), которая обладает свойством ассоциативности, и пустое слово, которое является нейтральным элементом для этого закона; в результате получается свободный моноид — структура, имеющая чрезвычайно важное значение в информатике

не встречается в последовательности x_i . В последующих алгоритмах будем избегать бесполезного вычисления этого значения. Естественно, что предложенный алгоритм не очень удобен для использования в той форме, в какой он изложен. В частности, в нем присутствуют переменные с индексами — объекты, которые программисты используют в своих программах очень неохотно и при крайней необходимости. Тем не менее, на базе этого псевдо-алгоритма нетрудно построить

настоящий. Для этого достаточно оптимизировать использование входящих в него переменных. Каждая переменная i появляется только два раза: первый раз, когда определяется ее значение, а второй — в вычислении, использующем это значение. Следовательно, все вхождения переменных i , можно заменить на вхождение одной единственной переменной.

Совершенно аналогична процедура для переменных n_i (заменяемых одной переменной n), что делает ненужной переменную i . Здесь приведена эффективная версия (Алгоритм 1) дихотомического алгоритма возведения в степень. Благодаря использованному методу построения, не требуется никакого доказательства его корректности.

Достаточно лишь проверка последней приведенной записи. Далее мы дадим окончательную версию алгоритма, которая несколько отличается от приведенной здесь, главным образом, возможностью обнаружения ошибок и надежностью алгоритма.

Рассмотрим теперь другой подход, позволяющий конструировать алгоритмы, — алгоритмический подход. Этот метод концептуально сильно отличается от предыдущего, но не более и не менее совершенен. В зависимости от условий тот или иной будет использоваться с большей или меньшей легкостью.

```

 $x = x_0;$ 
 $n = n_0;$ 
 $P = 1;$ 
while ( $n \neq 0$ )
{
  if ( $n \neq 0$ )  $P = P * x;$ 
   $n = (n/2);$ 
   $x = x * x;$ 
}
return  $P;$ 

```

Алгоритм 1.

2.2. Алгоритмический подход

Этот подход основан на следующем замечании: для натурального числа n

$$x^n = \begin{cases} (x^2)^{n/2}, & \text{если } n \text{ четно} \\ x(x^2)^{n/2}, & \text{если } n \text{ нечетно} \end{cases}$$

Он приводит к алгоритмам двух типов. Первый, получае-
мый без вычислений из вы-
шеприведенного определе-
ния, является рекурсивным
алго ритмом и не требует
никакого обоснования, поскольку он — результат непосредственного при-
менения безупречного определения.

```

if( $n == 0$ ) return 1;
else if ( $n \% 2 == 0$ )
{return Exponentiation( $x * x$ ,  $n / 2$ );}
else if ( $n \% 2 == 1$ )
{return  $x * \textit{Exponentiation}(x * x, n / 2)$ ;}

```

Второй является итеративным алгоритмом, который требует дока за-
тельства. Рекурсивный алгоритм по очевидным причинам записан в виде
функции *Exponentiation* (см. предыдущий рисунок).

В этом алгоритме рекурсивные вызовы имеют место во время опре-
деления возвращаемого значения функции и в точности соответствуют ин-
дуктивному определению, приведенному в начале раздела.

В более общем алгоритме использование этой функции осуществлялось
бы операцией типа: $\textit{Resultat} \leftarrow \textit{Exponentiation}(x, n)$.

Замечание. Определение возведения в степень может
быть дано также в следующем виде:

$$x^n = (x^{\frac{n}{2}})^2, \text{ если } n \text{ четно, } x^n = (x^{\frac{n-1}{2}})^2, \text{ если } n \text{ нечетно,}$$

который позволяет построить рекурсивный алгоритм, слег-
ка от личающийся от предыдущего. Однако эта последняя
формулиров ка несколько хуже подходит для конструи-
рования итеративного алгоритма, чем исходная. В качестве
упражнения читатель мо жет попробовать записать соот-
ветствующий алгоритм.

Главная идея, на которой основывается построение итеративного алго-
ритма, представляет собой другую интерпретацию определения, приве-
денного в начале этого раздела:

- чтобы вычислить x^n , где n положительно, достаточно возвести
 x в квадрат и возвести полученный результат в степень $n/2$, до-
множая поправочный множитель на x , если n нечетно;
- кроме того, когда n принимает значение 0, искомый результат
должен быть равен поправочному множителю, что и дает началь-
ное значение поправочного множителя.

Таким образом, алгоритм состоит в вычислении поправочного множи те-
ля так, чтобы произведение текущего значения x^n и поправочного множи-
теля было все время равно искомому результату $x_0^{n_0}$, где x_0 и n_0 являются
начальными значениями x и n соответственно. Итак, инвари антным со-
отношением алгоритма будет $c \times x^n = x_0^{n_0}$.

Предыдущие замечания приводят нас к искомому итеративному алгоритму (рис. 2-А), вполне аналогичному тому алгоритму, который был получен в результате рекуррентного подхода к решению проблемы. В этом алгоритме до и после каждого шага рассматриваются соотношения, которые позволяют доказать справедливость метода. Проверка корректности этих утверждений становится детской игрой для того, кто понял смысл обозначений. Чтобы облегчить чтение, обозначим *не* ременную, которая позволяет вычислять *поправку*, буквой *c*.

В реальной программе или же алгоритме, в котором, возможно, представлены не все утверждения, использовалось бы, разумеется, более выразительное имя.

Всюду в дальнейшем входящие в алгоритмы утверждения будут табуированы, как на рис. 2, для того, чтобы отличать утверждения от действий, совершаемых алгоритмом.

Документирование алгоритма посредством утверждений доказывает его частичную корректность. Остается показать, что алгоритм останавливается при любых данных, поданных на вход, только бы они соответствовали спецификациям.

Общим принципом доказательства остановки алгоритма является введение некоторой положительной величины, которая строго убывает с каждой итерацией, — величины, которую можно назвать *длиной* алгоритма (или же *расстоянием* до точки окончания). В этом частном случае такая работа чрезвычайно проста, ибо длина алгоритма изменяется на самом деле самым же целым числом *n*. Это целое число положительно, строго убывает с каждой итерацией — оно делится каждый раз пополам — и, следовательно, через конечный отрезок времени становится нулем, делая тем самым условие возобновления цикла ложным, и алгоритм останавливается за конечное время.

Внимательное изучение последнего алгоритма показывает, что даже если с теоретической или алгоритмической точки зрения он корректен, в процессе его буквальной реализации могут появиться определенные проблемы. Некоторые вычисления могут оказаться бесполезными и даже опасными: так, возведение *x* в квадрат может вызвать переполнение во время выполнения последней итерации, даже если искомый результат вполне вычислим.

Аналогичным образом, описанный алгоритм не способен корректно вычислить результат, когда показатель отрицательный или не является целым числом. Проверка природы целого числа по обычно является функцией среды реализации языка. И хотя можно снова опираться на язык, если речь идет о языке Ада, будет удобнее, по крайней мере в экспериментальной версии, приступить самому к проверкам первого типа.

Алгоритм 2. Дихотомическое возведение в степень

Мы видим (алгоритм 2-В) окончательную версию дихотомического

А. Алгоритмы с полным доказательством	В. Окончательный вариант
<pre> с ← 1; x ← x₀; n ← n₀ с × xⁿ = x₀^{n₀} while n ≠ 0 loop с × xⁿ = x₀^{n₀} и n > 0 if n mod 2 ≠ 0 then с ← с × x; end if; с(x²)^[n/2] = x₀^{n₀} n ← [n/2]; с × (x²)ⁿ = (x²)^[n/2] x ← x × x; с × xⁿ = x₀^{n₀} end loop; с × xⁿ = x₀^{n₀} и n = 0 return с; </pre>	<pre> x ← x₀; n ← n₀; if n mod 2 ≠ 0 then с ← x; else с ← 1; end if; loop n ← [n/2]; с × (x²)ⁿ = x₀^{n₀} exit when n = 0; x ← x × x; if n mod 2 ≠ 0 then с ← с × x; end if; end loop; return с; </pre>

алгоритма возведения в степень. Версия эта очень близка к разработке в языке Ада. Во избежание усложнения алгоритма (и это правило будет применено к большинству представляемых алгоритмов) единственное утверждение, фигурирующее еще в тексте алгоритма, — это ивариант цикла позволяющий правильно понять используемый метод.

2.3. Изучение сложности алгоритма

Чтобы завершить изучение алгоритма, остается оценить его сложность, т.е. сосчитать с достаточной точностью число элементарных операций, необходимых для его выполнения при фиксированных данных. Ясно, что число итераций, необходимых для выполнения алгоритма, зависит только от показателя. Упростим обозначения, предположив, что n (вместо n_0) — показатель, и предположим, в соответствии с обозначениями, введенными при исследовании рекуррентного алгоритма, что число цифр в двоичной записи n равно $t + 1$. Это равносильно тому, что

$$2^t \leq n < 2^{t+1} \text{ или } t \leq \log_2 n < t + 1$$

Первая часть этого свойства может быть выражена следующим образом: $[n/2^{(t+1)}] = 0$ и $[n/2^t] \neq 0$ что позволяет точно определить число совершаемых делений n , равное числу итераций алгоритма при заданном значении n . Очевидно, нужно совершить $t + 1$ итераций, чтобы выполнить алгоритм, т.е. $[\log_2 n] + 1$ итераций. Следовательно, тру-

доемкость алгоритма есть $O(\log n)$. Порядок величины — достаточная оценка, начиная с момента, когда константа, замаскированная обозначением O , находится в разумных пределах, как в данном примере. Можно было бы уточнить число операций, но это не представляет большого интереса.

Замечание . Как было уже сказано в начале раздела, дихотомический алгоритм возведения в степень не всегда является оптимальным методом для вычисления n -й степени элемента моноида, даже если базовая операция выполняется за постоянное время. Например, чтобы сосчитать x^{15} , используя дихотомическое возведение в степень, требуется 6 умножений: сосчитать x^2 , x^4 , x^8 и умножить вместе и эти три величины. Однако, если начать с вычисления x^5 (посредством x^2 и x^4), возводя последнюю величину в куб, считаем x^{15} за 5 умножений. Метод, использованный в вычислении, известен как «цепочка сложений» и идея его следующая: цепочка сложений для n — это последовательность целых чисел $a_0 = 1, a_1, a_2, \dots, a_r = n$, которая удовлетворяет следующему свойству:

$$\forall i \in [1, r], \exists j, k \quad \text{такие, что} \quad 1 \leq k \leq j < i \quad \text{и} \quad a_i = a_j + a_k.$$

Цель изучения цепочек сложения в том, чтобы найти при любом n как можно более короткую цепь сложений. Затем можно применить полученный результат для открытия самого эффективного метода, позволяющего вычислить x^n с помощью умножений. Не вдаваясь в детали метода, проиллюстрируем основной принцип использования цепочек сложения. Предположим, что требуется вычислить x^{54} . Чтобы произвести расчет дихотомическим методом, нужно сосчитать следующие степени x : $x, x^2, x^4, x^8, x^{16}, x^{32}$. Потом выполнить необходимые умножения этих чисел друг на друга. Если выполнить сразу только одно умножение и расположить полученные члены в порядке возрастания степеней, то получим последовательность $x, x^2, x^4, x^6, x^8, x^{16}, x^{22}, x^{32}, x^{54}$. Эта последовательность, совершенно очевидно, соответствует цепи сложений (1, 2, 4, 6, 8, 16, 22, 32, 54), которая не минимальна, потому что 54 можно получить при помощи цепи (1, 2, 3, 6, 9, 18, 27, 54), являющейся минимальной. Это означает, что вычисление 54-й степени x произойдет очень быстро при выполнении расчетов:

$$x_0 = x, x_1 = x_0 * x_0 = x^2,$$

$$x_2 = x_0 \cdot x_1 = x^3, x_3 = x_2 \cdot x_1 = x^6, x_4 = x_2 \cdot x_2 = x^9, \\ x_5 = x_4 \cdot x_1 = x^{18}, x_6 = x_4 \cdot x_3 = x^{27}, x_7 = x_6 \cdot x_6 = x^{54}.$$

Заинтересованный читатель может обратиться к работе [99]: «Искусство программирования», откуда были взяты описанные приемы.

3. Введение в программирование на языке Ада

Пришло время представить первые программы на языке Ада. Основным приемом, используемым для этого, заключается в представлении исходного текста¹ на языке Ада, результатов выполнения — чаще всего для определения времени их получения — и комментария к Ада-программе для объяснения новых введенных понятий или специальной техники программирования.

3.1. Программа сравнения двух методов возведения в степень

Эта первая программа — непосредственное воплощение алгоритма, изученного ранее; она позволяет сравнить время вычисления дихотомического алгоритма возведения в степень и алгоритма возведения в степень посредством последовательных умножений. Разумеется, результат не будет неожиданным: первый алгоритм требует $O(\log n)$ умножений, а второй $O(n)$; однако, полученные величины могут удивить того, кто не имеет опыта в этом типе вычислений и кто только с позиций теории знает о различии в росте между линейной и логарифмической функциями. Для измерения времени мы используем стандартную библиотеку *Calendar*, хронометр которой имеет разрешающую способность около 50 мкс (для нашего компилятора).

¹В этом контексте исходный текст обозначает файл, который содержит текст программы на исходном языке (здесь Ада) и который после компиляции порождает файл, записанный на целевом языке (обычно, машинный язык). Для краткости часто просто говорят о программе или исходном модуле в отличие от объектной программы или объектного модуля.

Входные данные			Время вычисления (сек.)		Результат
p	x	n	Послед.умн.	Дих.возв. в ст.	$x^n \bmod p$
181	11	1 024	0.050	0.000	126
	17	4 096	0.221	0.000	39
	23	16 384	0.710	0.000	15
	29	65 536	2.859	0.000	29
	31	262 144	11.370	0.000	59
	37	1 048 576	45.529	0.000	34
	41	4 194 304	182.130	0.000	15
	43	16 777 216	728.090	0.000	43
	47	67 108 864	2912.370	0.000	102

Таблица 1.1: Вычисление $x^n \bmod p$

Несколько комментариев по таблице 1:

- прежде всего, и это не зависит от сделанных измерений, расчеты производились по модулю целого числа p , чтобы можно было оценить большие степени, не сталкиваясь с проблемами переполнения машины целыми числами;
- когда используют таблицу этого типа, в которой представлены измерения времени вычисления, не нужно никогда забывать, с какой точностью были сделаны эти измерения (разрешающая способность таймера); в этом примере, в частности, время выполнения порядка 50 мкс не было значимым;
- в самом деле, заключаем, что сложность возведения в степень методом последовательных умножений является линейной по отношению к показателю: в каждой строке таблицы, показатель, так же, как и время вычисления, умножается на 4; предпоследнему измерению соответствует время вычисления около 12 минут, при вполне разумном суммарном значении показателя, а последнее измерение, при показателе порядка 70 миллионов, дает нам время абсолютно неразумное (около 50 минут); это доказывает, что алгоритм, работающий с помощью метода последовательных умножений не может быть использован в контексте более широком, чем сравнение алгоритмов;
- зато невозможно выявить малейшее изменение времени вычисления для дихотомического алгоритма возведения в степень, при таких *малых* значениях n ;
- число, по модулю которого производятся вычисления, является наибольшим простым числом таким, что $(-1)^2 \leq 2^{15} - 1$; это означает, что умножения по модулю p могут быть сделаны на машине, оперирующей целыми 16-битовыми числами (со знаками),

без риска переполнения (читатель, желающий больше узнать об арифметике компьютеров, может обратиться к Таненбауму [168], Мюллеру [132] или Менадье [126]);

- результат вычисления, $x^n \bmod p$, хотя и приведен в этой таблице, абсолютно не представляет никакого интереса; но программа без результатов еще способна шокировать чувствительные сердца.

Этот первый пример не может служить образцом программирования на языке Ада, в том смысле, что опытный программист не записал бы его в такой форме. В частности, типизация данных чрезвычайно слаба: моноид, на котором действует возведение в степень, — это множество целых чисел по модулю p с обычным умножением, представленное в программе типом *Integer*, показатель — предопределенного типа *Long_Integer*, чтобы иметь возможность использовать относительно большие показатели. Эти два типа еще не очень различаются, но ведь речь пока идет только о написании первой Ада-программы; этот тип программирования больше приближен к программированию на языке Паскаль, с которым читатель может быть более знаком. «Хороший» вариант Ада-программы будет представлен в конце этого раздела после введения основных понятий языка Ада.

*Программа оценки дихотомического возведе-
ния в степень*

```
with Text_IO, Calender;
procedure Exponentiation_First_Version is
  p : constant Integer := 181
  subtype Modular_Integer is Integer ranger 0..p-1
  x, Dichotomic_Result, Sequential_Result : Modular_Integer;
  n : Long_Integer;
  Begin_Time : Calendar.Time;
  Computation_Time : Duration;

  package Long_Integer_IO is new Text_IO.Integer_IO(Long_Integer);
  package Modular_Integer_IO is new Text_IO.Integer_IO(Modular_Integer);
  package Duration_IO is new Text_IO.Fixed_IO(Duration);
  use Calendar, Text_IO, Duration_IO, Modular_Integer_IO, Long_Integer_IO;

  function Dichotomic_Exponentiation(x0 : Modular_Integer; n0 : Long_Integer)
    return Modular_Integer is
    x : Modular_Integer := x0;
    n : Long_Integer := n0;
    Correction : Modular_Intrger;
  begin
    if n mod 2 /= 0 then Correction := x mod p; else Correction := 1; end if;
    loop
      n := n/2;
```

```

exit when n = 0;
  x := (x * x) mod p;
  if n mod 2 / = 0 when Correction := (Correction * x) mod p;
end loop;
return Correction;
end Dichotomic_Exponentiation;

function Sequential_Exponentiation(x0 : Modular_Integer; n0 : Long_Integer)
  return Modular_Integer is
  x_To_i : Modular_Integer := 1;
begin
  for i in 1 .. n0 loop x_To_i := (x_To_i * x0) mod p; end loop;
  return x_To_i;
end Sequential_Exponentiation;

begin — — Exponentiation_First_Version
  Put_Line ("Оценка дихотомического возведения в степень.");

loop
  Get(x);
exit when x = 0
  Get(n); Put(x); Put(n);

  Begin_Time := Clock; Sequential_Result := Sequential_Exponentiation(x, n);
  Computation_Time := Clock — Begin_Time; Put(Computation_Time);

  Begin_Time := Clock; Dichotomic_Result := Dichotomic_Exponentiation(x, n);
  Computation_Time := Clock — Begin_Time; Put(Computation_Time);

  if Dichotomic_Result = Sequential_Result then Put(Dichotomic_Result);
  else raise Numeric_Error;
  end if;
  New_line;
end loop;
end Exponentiation_First_Version

```

Комментарий к этой программе заключается, в основном, в описании структуры Ада-программы. В общих чертах, программа в языке Ада — это процедура (без параметров, в случае используемого нами компилятора), построенная на основе элементарных или сложных действий и объектов, которые можно определить во внешних **блоках компиляции** — блоки могут быть процедурами, функциями или **пакетами** (название, данное тому, что в других языках называется модулями или библиотеками).

Предыдущая Ада-программа образует блок компиляции, в данном случае являющийся главной программой; она состоит, в основном, из двух частей:

- определение контекста компиляции (оператор **with**); этот оператор позволяет компилятору распознать используемые внешние объекты, определенные в другом блоке компиляции, и убедиться в связности между их спецификацией и использованием;
- текст подпрограммы, соответствующий блоку компиляции.

Оператор контекста ► **with** *Text_IO* , *Calendar* ◀ касается двух стандартных пакетов: первый из этих модулей *Text_IO* содержит все обычные процедуры ввода-вывода текстовых файлов (включая, конечно, клавиатуру и экран); под этим понимается считывание и написание знаков, последовательностей знаков, чисел различных типов и т.д. Пакет *Calendar* содержит определения типов, операторов и подпрограмм, которые позволяют произвести измерение времени.

Затем идет определение процедуры, которая играет роль главной программы; с точки зрения структуры, эта процедура полностью аналогична процедуре на языке Паскаль. Она включает, в определенном порядке, последовательность описаний констант, типов и переменных, несколько конкретизаций настраиваемых пакетов, определения функций и, наконец, само тело процедуры.

Рассмотрим, прежде всего, описательную часть данной процедуры. После определения константы *p* находим определение **подтипа** , в котором будут сделаны расчеты: **subtype** *Modular_Integer is Integer range 0..p-1*; ◀. Это определение позволяет ограничить множество допустимых значений, а среди целых значений рассматривать только те, которые заключены между 0 и *p* — 1. Это приводит, в частности, к тому, что во время выполнения программы операционная среда языка Ада будет контролировать значения переменных типа *Modular_Integer* и вызовет внезапную остановку программы, если эти значения выйдут за рамки намеченного интервала. После данного определения типа мы видим описания трех переменных этого подтипа, используемых в ходе вычислений, потом переменную, представляющую показатель. Данная переменная относится к (предопределенному) типу *Long_Integer*, несовместимому с типом *Integer*.

Замечания. Ада имеет несколько предопределенных типов, которые мы не будем здесь подробно рассматривать; их список можно найти в приложении *F* в справочнике, сопровождающем любой компилятор, достойный этого названия. Однако, мы скажем несколько слов о целых типах. Компилятор языка Ада дол-

жен предоставить, как минимум, один предопределенный целый тип, называемый *Integer*, кроме того, он может давать другие целые типы с названиями *Short-Integer*, *Long-Integer*, *Short-Short-Integer*, . . . характеризуемые тем, что, например, тип *Short-Integer* не может быть больше типа *Integer*, который в свою очередь не может быть больше типа *Long_Integer* . . . как скучно все это объяснять!

Это наложение на целые типы позволяет высказать дополнительные критические замечания по поводу предыдущей программы. Действительно, идентификатор *Long_Integer* появляется там отчетливо несколько раз, и это означает, что нужно будет сделать много модификаций при переносе этой программы на машину, компилятор которой не содержит типа *Long_Integer*. Этот недостаток исправлен в последующих Ада-программах. Данная первая версия представляет только первый опыт программиста, привыкшего писать на Паскале.

Переменная *Begin_Time* принадлежит типу *Time*, определенному в пакете *Calendar* и обозначающему набор время-дата. Без дополнительного уточнения после спецификатора контекста этот тип не является непосредственно видимым, и нужно задать компилятору Ада путь, позволяющий обнаружить определение в описанных модулях, что и делается с помощью выражения *Calendar.Time*, обозначающего, что тип *Time* определен в пакете *Calendar*. Во избежание тяжеловесности записи, вызванной этим феноменом, нужно, чтобы объекты пакета *Calendar* стали видимыми, начиная с подпрограммы, что и делается в строке программы, содержащей спецификатор ► **use Calendar** ◀ : с этого оператора все происходит так, с точки зрения видимости, как если бы описания пакета *Calendar* фигурировали в тексте подпрограммы. Последняя переменная программы, *Computation_Time*, типа *Duration*, используется для выражения результатов измерений времени, выполненных в программе. Тип *Duration* — это предопределенный тип, позволяющий найти продолжительность, выраженную в секундах: это реальный тип с фиксированной точкой, т.е. с точностью абсолютной, а не относительной, как в случае реальных типов с плавающей точкой. Язык Ада типизирован до такой степени, что, к примеру, процедуры ввода-вывода отличаются для объектов различных типов; это придает языку большую однородность. Однако, было бы не очень удобно определять такие процедуры для каждого численного типа, который хотят использовать, — на самом деле способ вывода чисел целого типа всегда одинаковый, независимо от того, о каком целом идет речь: обыкновенном, длинном или о любом другом целом типе, определенном

пользователем. Рассуждение совершенно аналогично для определенных пользователем действительных, символьных и других типов. По этим причинам и по многим другим, которые возникнут позднее, язык Ада обладает понятием настраиваемого объекта, позволяющим более легко воплотить абстрактные типы.

Таким образом, стандартный пакет *Text_IO* содержит определенное число настраиваемых подпакетов, позволяющих без труда создать для каждого особого типа процедуры ввода-вывода, соблюдая типовые ограничения, необходимые для связности и точности языка: для целых типов, например, настраиваемый пакет ввода-вывода — *Text_IO.Integer_IO*. В определении настраиваемого модуля некоторые параметры функционирования модуля не уточняются, но прежде, чем использовать такой модуль, нужно явно указать эти параметры. Таким образом, параметры настраиваемого пакета *Integer_IO* определяются целым типом, для которого предполагается реализовать ввод-вывод; во время своей **конкретизации** — фазы, позволяющей вызвать генерацию, — исходя из общей совокупности объектов, предназначенных для использования, нужно уточнить целый тип, который придется обрабатывать. В примере этой программы имеются два целых типа, для которых необходимо зарегистрировать методы ввода-вывода; итак, находим две конкретизации пакета *Integer_IO*. Два новых пакета определены; в дальнейшем можно будет на них ссылаться.

```
package Long_Integer_IO is new Text_IO.Integer_IO (Long_Integer);  
package Modular_Integer_IO is new Text_IO.Integer_IO (Modular_Integer);
```

Затем в исходном тексте находим конкретизацию для типа *Duration* настраиваемого пакета *Fixed_IO*, позволяющего производить ввод-вывод действительных типов с фиксированной точкой. Потом, для облегчения записи последовательности программы, четыре пакета ввода-вывода перечисляются в описанном выше операторе **use** (что делает видимым все объекты внутри пакетов, и при отсутствии конфликтов позволяет использовать эти объекты без упоминания их источника). После всех этих описаний находим в тексте две функции возведения в степень, которые служат для сравнения двух разработанных методов. Эта часть Ада-программы не требует много комментариев потому, что она точно отражает алгоритм, изученный в разделе 2.2 (страница 36); единственная неизвестная команда — это **return**, которая вызывает остановку функции и определение ее значения.

Важно отметить, что все параметры функции в языке Ада являются **входящими**, или в режиме **in**. Таким образом значения фактических параметров вызова функции передаются соответствующим формаль-

ным параметрам во время активации этой функции (с помощью метода, который нет необходимости здесь описывать). Синтаксис языка Ада не позволяет подпрограмме изменять свои формальные параметры режима *in*, рассматриваемые как константы в теле подпрограммы. Это приводит, в частности, к тому, что вызов функции не может привести к изменению параметров вызова (это противоречило бы логике и духу программирования). И, наконец, программу завершает тело главной процедуры,

требу-

ющее лишь небольших комментариев:

- функция *Clock* определена в пакете *Calendar* и возвращает дату и время;
- функции *Put* и *Get* позволяют, соответственно, считывание и запись; однако на этом этапе напрашивается замечание: хотя процедуры считывания различны для целых и вещественных, а также различны для разных целых типов, обычно названия у них одинаковые. Это так называемая перегрузка идентификаторов, которая может использоваться только с идентификаторами функций, операторов или процедур. В зависимости от контекста, в частности, от типа параметров, компилятор устраняет двусмысленности;
- работу программы достаточно легко понять: по очереди считываются два целых: одно x типа *Modular_Integer*, второе n типа *Long_Integer* и вычисляется x^n сначала с помощью простого метода, потом с помощью дихотомического. Во время каждой операции время выполнения измеряется и выводится на экран. Наконец, после оценки результата двумя методами, так как речь идет об экспериментировании, два результата вычисления сравниваются (никто не застрахован от ошибки программирования!), и в случае (очень маловероятном), когда будет расхождение, программа возбуждает предопределенное исключение *Numeric_Error*, что приводит к ее немедленному завершению, как если бы дело касалось ошибки выполнения. Этот механизм исключения является методом, избранным разработчиками языка Ада для обработки ошибок и их устранения. Нам еще представится случай к нему вернуться. Программа останавливается, когда для переменной x берется значение, равное нулю.

Входные данные			Время вычисления (сек.)	Результат
p	x	n	Дих.возв. в ст.	$x^n \bmod p$
46 337	31	262 144	0.000	3951
	37	1 048 576	0.000	43153
	41	4 194 304	0.000	1430
	43	16 777 216	0.000	29703
	47	67 108 864	0.000	26840
	53	268 435 456	0.000	43887
	59	1 073 741 824	0.000	13314
	61	2 147 483 647	0.000	5770

Таблица 1.2: Вычисление $x^n \bmod p$

3.2. Об использовании типов

После этой первой версии, представим теперь другую программу, где дихотомический алгоритм возведения в степень еще фигурирует, но в виде, который больше приближается к тому, что мы считаем программированием на языке Ада. В частности, обрабатываемые объекты там намного более типизированы, чем в первоначальной версии, что позволит впоследствии без больших усилий построить настраиваемую версию дихотомического возведения в степень. Но прежде всего пример выполнения (таблица 2) позволяет констатировать, что при значениях показателя, близких к допустимому максимуму, время вычисления на большинстве машин не всегда будет значимым.

Комментарии будут краткими:

- число 46337 — наибольшее простое число такое, что $(-1)^2 \leq 2^{31} - 1$, следовательно, можно выполнять умножения по модулю без риска переполнения на машине, обладающей целыми 32-битовыми числами со знаками¹,
- как в предыдущей таблице, целые x являются простыми числами, а целые n , все (кроме последнего) — степени двойки (более точно, n имеет форму 2^{4k}),
- целое 2 147 483 647 — это наибольшее положительное целое, которое может быть представлено в машине, оперирующей 32-битовыми словами и использующей целую, знаковую арифметику в дополнительном коде (оно равно $2^{31} - 1$).

¹В языке Ада числа могут задаваться программе с использованием символа подчеркивание () для того, чтобы сделать их более понятными; и возможно даже, что они будут выражены не в десятичной системе счисления! Так, число 46337 может быть задано в тексте программы или при считывании в виде 46-337, или 7#252J044#, когда его хотят выразить в системе с основанием 7.

Основное отличие двух первых версий программы (это мы отметили в предыдущем параграфе) состоит в различии (на уровне используемых типов) между элементами моноида, в котором они работают, и показателями, которые всегда целые; что, впрочем, не нарушает общности изложения. Итак, рассмотрим исходный текст этой программы:

Дихотомическое возведение в степень

```
with Text_IO; use Text_IO; with Calender; use Calender;
procedure Exponentiation_Second_Version is
```

```
  p : constant Integer := 46_337;
  type Modular_Integer is ranger 0..p - 1
  type Exponent is new Long_Integer
```

```
  n : Exponet
  x, Result : Modular_Integer;
  Begin_Time : Calendar.Time;
  Computation_Time : Duration;
```

```
  package Exponent_IO is new Integer_IO(Exponent);
  package Modular_Integer_IO is new Integer_IO(Modular_Integer);
  package Duration_IO is new Fixed_IO(Duration);
  use Exponent_IO, Modular_Integer_IO, Duration_IO;
```

```
  function Multiply(a, b : Modular_Integer) return Modular_Integer is
    -- переопределение умножения для типа Modular_Integer
  begin
    return Modular_Integer((a * b) mod p);
  end Multiply;
```

```
  function Dichotomic_Exponentiation(x0 : Modular_Integer; n0 : Exponent)
    return Modular_Integer is
    x : Modular_Integer := x0;
    n : Exponent := n0;
    Correction : Modular_Integer;
```

```
  begin
    if n mod 2 /= 0 then Correction := x; else Correction := 1; end if;
```

```
  loop
    n := n/2;
    exit when n = 0;
    x := Multiply(x, x);
    if n mod 2 /= 0 then Correction := Multiply(Correction, x); end if;
  end loop;
  return Correction;
end Dichotomic_Exponrntiation;
```

begin – **Exponentiation_Second_Version**

```
Put_Line ("Дихотомический возведение в степень_,_вторая_версия");
```

```
loop
```

```
Get(x);
```

```
exit when x = 0
```

```
Get(n);
```

```
Begin_Time := Clock;
```

```
Result := Dichotomic_Exponentiation(x, n);
```

```
Computation_Time := Clock – Begin_Time;
```

```
Put(x); Put(n); Put(computation_Time); Put(Result); New_Line;
```

```
end loop;
```

```
end Exponentiation_Second_Version;
```

В начале этого текста стоит, как обычно в любой программе, спецификатор контекста, который указывает на два пакета, а за ним сразу же следует соответствующий спецификатор использования **use**. Спецификатор **use** может появиться в контекстной части блока компиляции или в любом месте описательной части: место, где появляется спецификатор **use**, определяет области видимости объектов, содержащихся в пакетах, на которые он указывает.

В описательной части снова видим описание константы p , но в данной версии она недостаточно типизирована: это, так называемая, универсальная константа, что означает ее совместимость со всеми целыми типами. Нормой языка Ада является то, что вычисления, выполняемые на универсальных константах, должны быть точными, и для удовлетворения этой нормы любой компилятор способен производить расчеты на любых больших универсальных константах; в дальнейшем мы увидим это использование. Константа p используется затем, чтобы определить *целый тип*; этот тип определяется абсолютным способом, как интервал натуральных чисел (в терминологии языка Ада целые математические числа называются целыми универсальными). Преимущество этого метода в том, что он делает определения используемых типов независимыми от предопределенных типов, существующих в конкретной используемой реализации. При компиляции программы могут происходить два события, относящиеся к этому определению типа:

- тип не может быть представлен в реализации: в этом случае компилятор делает предупреждение и отказывается компилировать,
- тип может быть представлен, и язык Ада гарантирует, что результаты вычисления будут идентичными, каковы бы ни были

используемая машина или реализация языка, при условии, что не будет переполнения.

После этого определения находим второе определение типа: ► **type Exponent is new Long_Integer** ◀ Это определение производного типа, что означает: тип *Exponent* идентичен по всем пунктам типу *Long_Integer*, но эти два типа несовместимы; тип *Long_Integer* — это родительский тип типа *Exponent*. Во время записи программы, при условии преобразования явных типов, будет совершенно невозможно соединить переменные этих двух типов в одно выражение; эта несовместимость обеспечивает большую надежность программирования, не допуская смешения (часто легкого) идентично представленных объектов (по причинам, связанным с реализацией данной программы), значения которых полностью отличаются друг от друга (не складывают, например, числа, представляющие вольты, с числами, представляющими амперы — техническая версия детской задачки о картошке и моркови).

В этой программе два объектных типа — показатели и множество модулярных целых, на котором действует возведение в степень, — должны быть полностью несовместимыми. Определения типов нас убеждают в этой несовместимости. Разделение типов дает второе немало-важное преимущество: возможность переопределения операций и под-программ, воздействующих на объекты родительского типа; но это уже другой случай, к которому мы обратимся впоследствии.

Когда выводится тип, производный тип наследует, априори, все свойства родительского типа, и, в частности, все происходит так, при отсутствии явного противоположного указания, как если бы все процедуры и функции, в спецификации которых появляется родительский тип, были повторно определены, и в этот раз с упоминанием типа, производного от родительского типа (это не совсем точно, но на данный момент достаточно; чтение справочника, который мы не будем здесь перефразировать, поможет в затруднительных случаях). В частности, это приводит к тому, что обычные элементарные операции, определенные на целых числах (сложение, умножение...) наследуются типами *Exponent* и *Modular_Integer*. К счастью! Если пришлось бы повторно определять все операции каждый раз при определении нового типа, программирование было бы очень трудным!

После нескольких определений переменных (без новшества отношению к предыдущей программе) находим конкретизации настраиваемых пакетов ввода-вывода для всех обрабатываемых типов; созданные таким образом пакеты сразу же применяются в качестве объекта спецификатора **use**, чтобы облегчить использование процедур, которые

они содержат. Затем идет определение функции умножения целых модульных чисел; эта функция использует стандартное умножение целых чисел — умножение, которое унаследовал тип *Modular_Integer* при своем построении.

И, наконец, эта программа очень похожа на предыдущую, и мы теперь сможем направить наши усилия на более общую версию функции дихотомического возведения в степень.

3.3. Итерация в моноиде — Настраиваемые функции

Если мы хотим реализовать метод более общим способом, перед началом программирования нужно задать себе несколько вопросов:

1. В каких границах применяется этот метод? В случае необходимости, в каких особых границах он будет применяться?
2. Каково формальное определение обрабатываемых объектов?
3. Какие ошибки могут возникнуть при использовании модуля, реализующего метод? Будут ли обнаружены эти ошибки во время компиляции или же они появятся при выполнении?
4. Каков тип потенциальных пользователей этого модуля? Существуют ли такие пользователи?
5. В каком виде он будет реализован? И каков его интерфейс?

Когда можно ответить на эти вопросы или, по крайней мере, дать частичные ответы, тогда можно приступать к программированию. В интересующем нас случае представляем некоторые элементы ответа:

1. Метод может применяться в любом моноиде, как уже неоднократно говорилось, хотя никогда до сих пор мы не были поставлены в столь общие рамки. Окончательное наше желание — применить его в любом моноиде, при условии, что такая структура будет определена в программе¹.
2. Рассматриваемый объект — это моноид, математическое определение которого следующее: моноид — это структура, состоящая из некоторого множества, закона внутренней композиции, **ассоциативного** на этом множестве, и выделенного элемента множества, который для этого закона является **нейтральным элементом**. Дополнительно, метод использует множество натуральных чисел — для показателей — и, хотя в математике этот объект универсален, нужно его точно определить в программе.

¹В действительности, в этой версии будет еще существовать ограничение в представлении моноида. Но это уже тонкости программирования на языке Ада.

3. Основные ошибки, которые могут появиться:

- ошибки вычисления, вызванные переполнением разрядной сетки; эти ошибки будут, в принципе, обнаружены средой выполнения Ада-программы (далее мы увидим, что лучше распространять определенные исключения, чем предопределенные исключения языка; но в данный момент и без этого все достаточно сложно);
- ошибки, возникающие вследствие округлений или потерь точности, когда используют метод на неточно представленных структурах, например, числах с плавающей запятой; такие ошибки, если провести исследование числового поведения программы, неизбежны и не поддаются обнаружению, если серьезно не изучать результаты выполнений. Однако, этот тип ошибок не должен появиться при использовании данного метода, поскольку речь идет о точном, алгебраическом вычислении;
- ошибки в использовании метода: либо используемые типы неадекватны, что компилятор обнаружит, либо эти же самые типы не соблюдают определенный диапазон использования; например, если используемый показатель — целое отрицательное число, то в конце программы, вероятно, будет ошибка. Можно, не перегружая слишком алгоритм, исправить этот особый дефект.

Зато использование этого алгоритма с неассоциативным законом — т.е. не в моноиде — среди таких законов есть очень известные (вычитание и деление, например) приведет к получению непредсказуемых результатов, которые невозможно будет исправить; но в этом случае мы выходим за рамки применимости метода.

4. Желание каждого уважающего себя программиста состоит в том, чтобы его продукция использовалась повсеместно. В данном случае наша цель — не увеличивать ограничения, изложенные ранее, помня о том, что спецификация должна быть подтверждена очень точными документальными данными.
5. Учитывая вышеизложенные моменты, мы решили реализовать окончательную версию дихотомического возведения в степень в виде настраиваемой функции. Это позволит определить параметры модуля, построенного с помощью типа элементов обрабаты-

ваемого множества, ассоциативного закона и нейтрального элемента этого закона, т.е. с помощью рассматриваемого моноида.

3.4. Настраиваемая функция дихотомического алгоритма возведения в степень

Настраиваемая функция или, в более общем виде, настраиваемый блок компиляции в основном образован из двух частей: спецификации, содержащей описание **формальных параметров настройки**, и соответствующей реализации. Описание формальных параметров настройки позволяет установить характер и тип объектов, которые очень точно определяют функцию при ее реализации. В нашем случае этих параметров четыре, и первые три в точности соответствуют триплету, определяющему моноид:

- тип элементов моноида,
- закон композиции моноида,
- нейтральный элемент моноида,
- тип, представляющий множество натуральных чисел.

Вот спецификация настраиваемой функции дихотомического возведения в степень:

Спецификация настраиваемой функции

```
generic
type Monoid_Element is private;
with function " * "(a, b : Monoid_Element) return Monoid_Element;
Mooid_Unit : in Monoid_Element;
type Exponent is range <>;
function Dichotomic_Exponentiation(x0 : Monoid_Element; n0 : exponent)
    return Monoid_Element;
```

Спецификация любой настраиваемой функции начинается с ключевого слова **generic**, затем следует список формальных параметров настройки и заголовок функции. Спецификацией называют видимую часть функции: она содержит все необходимые сведения для потенциального пользователя. Способ реализации функции не имеет никакого значения для пользователя (несколько оптимистический взгляд, но мы будем его придерживаться); спецификация играет роль способа употребления, списка указаний и противопоказаний. Формальные параметры настройки представлены в особом виде:

- Тип элементов множества, соответствующий моноиду, является личным типом: ► **type *Monoid_Element* is private**; ◀. Это означает, что при определении функции не нужно знать точную струк-

туру этих элементов. Это позволяет получить стиль программирования, очень независимый от конкретных обрабатываемых объектов. В данном частном случае, например, мы вынуждены явно задать настраиваемому блоку умножение и нейтральный элемент (который не должен быть 1) и впоследствии ничто не помешает использовать эту функцию, чтобы выполнить вычисления в мультипликативном моноиде.

- Формальные параметры типа функции или процедуры вводятся ключевым словом **with** :► **with function** "*" (*a,b: Monoid-Element*) **return**◄, что позволяет их отличить от настраиваемого объекта, которому задается спецификация. Здесь закон моноида обозначен операцией. Это означает, что, если мы хотим, чтобы в определении функции возведения в степень появилось произведение двух элементов моноида, то нужно использовать операцию умножения. Разумеется, это не приводит к переоценке конкретной функции, которая используется как фактический параметр во время реализации и которая может быть стандартным сложением, стандартным умножением или любой другой функцией, определенной пользователем.
- Третий формальный параметр настройки - это нейтральный элемент моноида, который должен быть явно задан при реализации: ► *Monoid.unit*: **in** *Monoid-Element*; ◄ - Ключевое слово **in**, появившееся в этом описании, означает, слегка упрощенно, что объект является константой.
- Наконец, тип показателей — это последний формальный параметр настройки. Этот тип должен быть целым типом машины; это уточнение выражено в виде: ► **type Exponent.Type is range** < > ; ◄ . В момент реализации может быть использован любой целый тип, будет ли это целый предопределенный тип или целый тип, определенный пользователем («целый тип» подразумевается в значении языка Ада).

Последняя часть спецификации настраиваемого блока — это описание самой функции, т.е. описание типа параметров и результата: заголовков функции.

Блок компиляции состоит из спецификации и тела; однако, иногда спецификация интегрирована в тело; в частности, это случай процедуры, которая играет роль главной программы во всех примерах. После того, как мы изучили текст спецификации настраиваемой функции, нам остается изучить ее тело. Тело настраиваемой функции очень похоже на тело функций, которые мы изучали до сих пор, и, за ис-

«««< HEAD

===== >>>> 4f4d0f78d3cc932ba2fb67aca17bd68d8957da6f

ключением некоторых особых моментов, оно потребует не очень много объяснений.

```

function Dichotomic_Exponentiation (x0 : Monoid_Element; n0 : Exponent)
    return Monoid_Element is

begin
    if n0 < 0 then raise Numeric_Error; end if;
    Calculate_x_Power_n : declare
    x : Monoid_Element := x0;
    n : Exponent := n0;
    Correction : Monoid_Element;

begin
    if n mod 2 /= 0 then Correction := x;
    else Correction := Monoid_Unit;
    end if;

loop
    n := n / 2 -- Correction * (x ** 2) ** n = x0 ** n0
exit when n = 0;
    x := x * x;
    if n mod 2 /= 0 then Correction := Correction * x; end if;
end loop;

    return Correction;
end Calculate_x_Power_n;
end Dichotomic_Exponentiation;

```

Первое замечание по тексту программы касается обработки одной из ошибок использования, отмеченной в начале этого раздела: использование функции с отрицательным показателем. Эта ошибка легко поддается обнаружению, и о ней несложно сообщить: для этого используют один из сильных моментов языка Ада, обработку исключений. Команда ► **raise** *Numeric_Error* ◀ позволяет возбудить предопределённое исключение *Numeric_Error*, предусмотренное в начальных спецификациях языка Ада, которое появляется, когда ошибка вычисления (например, вызванная переполнением) возникает в программе¹. Исключение - это конкретизация исключительного события (именно так!), которая может появиться во вре-

¹В современных версиях компилятора языка Ада в таких случаях генерируется не это исключение, а исключение *Constant_Error*. Однако, это позволяет отличить при желании, например, ошибку использования от переполнения разрядной сетки.

мя выполнения программы. При появлении исключения возможны два вида поведения:

- разработчик программы не предусмотрел ничего специального, в этом случае программа просто прекращает работать и сообщает об исключении, которое вызвало остановку программы; именно так и будет при использовании экземпляра этой настраиваемой функции со вторым отрицательным аргументом;
- программист предусмотрел эту проблему и ввёл в программу обработчик исключения, т.е. множество команд, к которым и обращается программа при появлении этого исключения; в этом случае программа может продолжать нормально работать.

Далее мы увиди примеры устранения исключений. Остальное тело функции состоит из *блока*, структуры, включающей описательную часть (множество описаний), последовательность команд и, в случае необходимости, обработчик исключений. Этот тип структуры позволяет описывать локальные переменные только в том случае, когда они необходимы (если не было ошибки в значении показателя). Этот блок вводится ключевым словом **declare** , перед которым стоит (что необязательно) имя блока; затем идёт описательная часть, потом ключевое слово **begin**, список команд и, наконец, ключевое слово **end**, за которым следует, для большей ясности имя блока. Этот блок без всяких хитростей реализует дихотомическое воздействие в степень и не нуждается в дополнительных объяснениях.

3.5. Использование настраиваемой функции

Прежде, чем представить главную программу, показывающую пример использования настраиваемой функции, предлагаем несколько вычислений времени выполнения. В данной таблице больше не фигурируют времена вычислений, так как эти времена совершенно несущественны для показателей, которые могут быть представлены машинным словом (менее 32 итераций). Следует отметить, что в этот раз нас интересует именно результат! Используемые целые числа F_n являются числами Ферма, т.е. чилвами виде $2^{2^n} + 1$; простые числа Ферма - это:

Вычисление $5^k \bmod F_n$ с $k = \frac{F_n-1}{2}$			
n	F_n	k	$5^k \bmod F_n$
2	17	8	16
3	257	128	256
4	65 537	32 768	65 536

$$F_0 = 3, F_1 = 5, F_2 = 17, F_3 = 257, F_4 = 64537, F_5 = 4294967297, \dots$$

Число вида $2^q + 1$ может быть простым, только если q - степень двойки (если m нечётно, то $x + 1 | x^m + 1$). Ферма подтвердил (ручным способом, конечно), что F_0, F_1, F_2, F_3, F_4 являются простыми, и высказал гипотезу, которую ему не удалось доказать даже для F_5 , что числа F_n - все простые. Это предположение оказалось ложным. Действительно, Эйлер доказал, что число F_5 - составное: его два множителя - 641 и 6 700 417.

Теорема Гаусса (см. Мютафиан[133]) гласит, что если число Ферма F_n - простое, то правильный многоугольник с F_n сторонами может быть построен с помощью циркуля и линейки; и этот результат эффективно применялся, потому что построение правильного многоугольника с F_n сторонами было сформулировано для n , равных 0, 1, 2, 3 и 4, - начений, соответствующих только тем числам Ферма, простота которых была доказана [133].

Другой результат, принадлежащий Пепину (см. Рибенбойм [154]), показывает, что *число Ферма F_n является простым тогда и только тогда, когда $5^{(F_n-1)/2} \equiv -1 \pmod{F_n}$* . Как раз этот метод используется в приводимой нами программе. Кроме 3 и 5, простых чисел Ферма, но которые по явным причинам программирования (каким?) не входят в набор данных этой программы, числа F_2, F_3 и F_4 , как показывают результаты, являются простыми. Число F_5 не может быть представлено в данных этой программы (понятно, почему?). Сегодня известно [184], что при $5 \leq n \leq 21$ число Ферма F_n - составное; также известно [154], что F_23471 - число, имеющее более 10^7 000 десятичных цифр, является составным.

Пора, наконец, перейти к изучению программы, использующей настраиваемую функцию, представленную в предыдущем разделе.

```

with Text_IO; use Text_IO;
with Dichotomic_Exponentiation;
    - USE отсутствует, так как он не имеет смысла
procedure Format_s_Numbers is
    type Modulus is new Long_Integer;
    n : Modulus;

    package Modulus_IO is new Integer_IO (Modulus); use Modulus_IO;
    procedure Pepin_s_Test (n : Modulus) is separate;

begin    — Format_s_Numbers
    Put_Line ("Проверка простоты чисел Ферма");

```

```

loop
  Get(n);
exit when n = 0;
  Pepin_s_test (n);
end loop;
end Fermat_s_Numbers

```

Парадоксально, но эта программа является самой сложной из представленных до сих пор. Одно только замечание по поводу оператора контекста: здесь нужно описывать блок компиляции, образованный при помощи настраиваемой функции, так как она входит в контекст компиляции, но нельзя с этим оператором контекста ассоциировать спецификатор *use*, который имеет смысл, только когда применяется с пакетом. В описательной части описываются тип *Modulus* (произвольный тип), пакет ввода-вывода, адаптированный к типу *Modulus*, потом процедура, названная *Pepin_s_Test*, параметр которой обозначает индекс проверяемого числа Ферма.

Эта процедура описана только посредством конструкции ► **procedure** *Pepin_s_Test*(*n* : *Modulus*) **ifseparate**; ◀, ключевое слово **separate** сообщает, что тело процедуры определено в другом месте и скомпилировано отдельно несколько позднее, после компиляции процедуры *Fermat_s_Numbers*. Кроме того, при компиляции отдельного тела, все будет происходить так, как если бы оно компилировалось вну три родительского тела, т.е. контекст (видимые объекты и т.д.) будет таким же. При отладке одной части программы очень полезно отделить эту часть; небольшая модификация данной части приведет к повторной компиляции только отделенной части, а не всей программы, это может значительно уменьшить время компиляции (особенно с таким языком, как Ада). Остальная часть программы — очень простая.

Итак, вся работа, необходимая для применения критерия Пепина, проводится в отделенной процедуре *Pepin_s_Test*, которую мы сейчас будем изучать. Именно в этой процедуре используется настраиваемая функция дихотомического возведения в степень. Вот текст процедуры:

```

separate (Fermat_s_Numbers) procedure Pepin_s_Test (n : Modulus) is
   $F_n$  : constant Modulus := 2 ** (2**Integer(n)) + 1;
  subtype Modular_Integer is Modulus range 0 ..  $F_n$  - 1;

```

```

k : Modulus;
Result : Modular_Integer

function Multiplication (a, b : Modular_Integer) return
Modular_Integer is
  function Plus (a, b : Modular_Integer) return Modular_Integer is
    begin return (a + b) mod  $F_n$ ; end Plus;

  function SOS_Multiplication is new Dichotomic_Exponentiation (
    Exponent  $\Rightarrow$  Modular_Integer, Monoid_Element  $\Rightarrow$  Modular_Integer, "*"
     $\Rightarrow$  Plus, Monoid_Unit  $\Rightarrow$  0);

begin -- Функции Multiplication
  return (a * b) mod  $F_n$ ;
exception
  when Constraint_Error  $\Rightarrow$  return SOS_Multiplication (a, b);
end Multiplication;

function "*" is new Dichotomic_Exponentiation (Exponent  $\Rightarrow$  Modulus,
  Monoid_Element  $\Rightarrow$  Modular_Integer, "*"  $\Rightarrow$  Multiplication,
  Monoid_Unit  $\Rightarrow$  1);

begin -- Проверка Пепина
  k := ( $F_n - 1$ ) / 2; Result := 5 ** k;
  Put ( $F_n$ ); Put(" "); Put("k"); Put(" "); Put (Result); New_Line;

  Put_Line("ЧислоФерма " & Modulus'Image( $F_n$ ));
  if Result =  $F_n - 1$  then Put_Line(" простое");
  else Put_Line (" составное");
  end if;
end Pepin_s_Test;

```

Текст отдельного модуля должен всегда начинаться со ссылки на главный модуль, от которого он был отделен; это нужно для того, что бы компилятор знал точный контекст компиляции этого подмодуля. Команда ► **separate** (*Fermat_s_Numbers*) **procedure**... ◀ обеспечивает эту функцию. Затем находим вычисление числа Ферма F_n , которое использует стандартную операцию возведения в степень. Показатель в языке Ада должен быть предопределенного типа *Integer*. Это объясняет выполненное в выражении преобразование типа. Потом можно определить тип *Modular_Integer*, и останется только конкретизировать настраиваемую функцию дихотомического возведения в степень, а для этого нужно определить умножение целых модульных чисел. Эта операция может быть определена так:

```

function Multiplication (a, b : Modular_Integer)
    return Modular_Integer is
begin
    return (a * b) mod  $F_n$ ;
end Multiplication;

```

где знак операции «*» обозначает обычное умножение целых чисел, унаследованное типом *Modulus* при его определении (выводом). Но в таком способе действия есть и отрицательная сторона. Действительно, при оценке обратного значения выражение $a * b$ должно быть вычислено на компьютере перед тем, как возьмут его остаток по модулю F_n . Однако это не всегда возможно, и в частности, в предварительной версии, реализующей это умножение, вычисление, связанное с числом F_4 , вызвало ошибку. В случае, когда произошло внутреннее переполнение, условиями выполнения Ада-программ допределенного исключения *Constraint-Error*, такое исключение можно устранить и обработать следующим образом:

```

function Multiplication (a, b : Modular_Integer)
    return Modular_Integer is
begin
    return (a * b) mod  $F_n$ ;
exception
    when Constraint_Error  $\Rightarrow$ 
        вычисление возвращённого значения другим способом
end Multiplication;

```

Для этого нужно суметь иначе вычислить произведение двух целых модульных чисел, чтобы при вычислениях не возникало переполнения. Как мы видели в начале изучения дихотомического алгоритма возведения в степень, такой способ существует и известен как египетское умножение — применение дихотомического алгоритма возведения в степень в аддитивном моноиде целых чисел. Все это объясняет относительную сложность модульного умножения, которое должно быть задано при конкретизации. Вот отделенное от своего контекста используемое умножение:

```

function Multiplication (a, b : Modular_Integer)
    return Modular_Integer is
    function Plus (a, b : Modular_Integer) return Modular_Integer is
    begin return (a + b) mod  $F_n$ ; end Plus;
    function SOS_Multiplication is new Dichotomic_Exponentiation (Exponent
         $\Rightarrow$  Modular_Integer, Monoid_Element  $\Rightarrow$  Modular_Integer, "*"  $\Rightarrow$  Plus,
        Monoid_Unit  $\Rightarrow$  0);

```

begin Функции Multiplication**return** (a * b) **mod** F_n ;**exception****when**Constraint_Error \Rightarrow **return** SOS_Multiplication (a, b);**end** Multiplication;

Работа функции заключается в следующем: когда вычисление возможно, она использует стандартные операции языка Ада, а если возникает ошибка, то выполняет дихотомическое вычисление (сложность которого пропорциональна логарифму множителя). Со структурной точки зрения эта функция должна выполнить конкретизацию настраиваемой функции на аддитивном моноиде целых чисел и, следовательно, должна предоставить настраиваемому блоку модульное сложение — функцию, названную *Plus*, простые вычисления которой всегда возможны. Сама конкретизация может выполняться следующим образом:

function Multiply **is new** Dichotomic_Exponentiation

(Modular_Integer, Plus, 0, Modular_Integer);

передавая параметры настройки в том порядке, в каком они представлены в спецификации настраиваемого блока (ассоциация по расположению), но можно также передавать их по имени. В данном случае это позволит перегруппировать два целых типа, чтобы подчеркнуть особенность этой конкретизации. Ассоциации параметров по имени, которые можно также использовать для передачи параметров процедур и функций, соответствуют синтаксису ► *Имя_формального_параметра* \Rightarrow *Фактический_параметр* ◄. Кроме того, очень часто ассоциация по имени позволяет значительным образом повысить удобочитаемость программ.

Последовательность действий программы теперь понятна. Определенное таким образом модульное умножение используется в конкретизации для построения операции возведения в степень в множестве целых чисел по модулю F_n — операции, которая применяется в исполнительной части процедуры.

3.6. Необычное использование компилятора языка Ада

Предлагаем теперь иллюстрацию к одному из самых удивительных моментов языка Ада, в котором компилятор используется для проверки простоты числа Ферма F_5 .

Ведя речь об оценке универсальных выражений, справочник говорит в абзаце 4.10(4) ARM¹:

*"...Furthermore, if a universal expression is a static expression, then the evaluation must be exact."*²

Статическая проверка простоты F_5

```
with Text_IO; use Test_IO;
procedure Test_Fermat_s_Number is
  n : constant := 5; F_n : constant := 2 * (2 * n) + 1; — 4_294_967_297

  x_0 : constant := 5; x_1 : constant := (x_0 * (2 * 6))
    mod F_n; — 5 * (2 * 6)
  x_2 : constant := (x_1 * 6) mod F_n; — 5 * (2 * 12)
  x_3 : constant := (x_2 * 6) mod F_n; — 5 * (2 * 18)
  x_4 : constant := (x_3 * 6) mod F_n; — 5 * (2 * 24)
  x_5 : constant := (x_4 * 7) mod F_n; — 5 * (2 * 31)

begin
  if x_5 = F_{n-1} then Put_Line("F_5 простое");
  else Put_Line("F_5 составное");
  end if;
end Test_Fermat_s_Number;
```

Отличительная черта этой программы заключается в том, что вычисления выполняются в описательной части программы, с использованием универсальных целых констант (а не типизированных целых констант). Именно при компиляции, а не во время выполнения, оцениваются все выражения. По вполне понятным причинам невозможно прямо вычислить $5^{\frac{F_5-1}{2}}$, а потом взять остаток по модулю F_5 (кстати, сколько десятичных цифр содержит число $5^{\frac{F_5-1}{2}}$?). Итак, вычисление разбито на части, используя статическую и улучшенную версию дихотомического возведения в степень. Следовательно, выполнение программы ограничивается обработкой результатов, полученных при компиляции.

Замечание. Помимо прочего, этот пример показывает, что любой компилятор языка Ада обладает встроенным калькулятором произвольно большой точности (с точностью в пределах возможностей конкретной машины), и можно только сожалеть, что этот

¹Ada Reference Manual, имеется перевод на русский язык: см. [186]. — Прим. перев.

²«... Более того, если универсальное выражение — статическое, то выражение также должно быть точным.» см. абзац 4.10.(4) в переводе ARM ([186]). — Прим. перев.

калькулятор не доступен при выполнении программ. Разумеется, таким способом можно проверить простоту некоторых других чисел Ферма, если разлагать вычисления так тщательно, чтобы избежать переполнения разрядной сетки компилятора (числа Ферма растут очень быстро!)

4. 4 Хорошее приближение к бесконечности

Все задачи в информатике можно условно разделить на два класса: задачи, имеющие решение, вычисляемое за разумное время, и задачи, решение которых требует слишком большого времени вычисления. Изучение многочисленных классов программ (или задач) составляет особую, самостоятельную область информатики, названную *теорией сложности* (книга D. Harel, *Algorithmics : The spirit of computing*¹ – это увлекательнейшее введение в эту теорию). Цель этого раздела – доказать, что некоторые решения простых математических задач выходят за рамки возможностей современных машин. Не делая чрезмерных обобщений, можно все же сказать, что эти решения никогда не будут вычисляться на машине, какой бы она ни была! Чтобы проиллюстрировать этот факт, мы решили реализовать вычисление целочисленных определителей при помощи метода, исходящего прямо из математического определения детерминанта². Итак, вот определение детерминанта, которое будет использоваться в этом разделе.

Определение 1

Пусть $A = (a_{ij})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}$ – квадратная матрица с целыми элементами. Детерминант (определитель) матрицы A , обозначаемый $|A|$ или $\det A$ – это число, определенное через $|A| = \sum_{\sigma \in S_n} \epsilon(\sigma) a_{1\sigma(1)} a_{2\sigma(2)} \dots a_{n\sigma(n)}$, где S_n – это множество перестановок интервала $[1, n]$, а ϵ – функция, дающая знак перестановки.

Это определение обладает некоторой инвариантной эффективностью (по крайней мере, внешне): действительно, определитель, кажется, может быть вычислен при помощи конечной суммы, любимой операции в программах. Однако, прежде, чем приступить к программированию этого метода, можно, а любой разумный человек должен, задать себе вопрос: сколько слагаемых имеет эта сумма? Математик тотчас же

¹Д. Харел, Алгоритмика: Дух вычислений (англ.) – Прим. перев.

²Впрочем, метод, вполне применимый (хотя, может быть, правильнее было бы сказать неприменимый) к вычислениям детерминантов, какова бы ни была природа элементов, т.е. для любого базового кольца.

ответит: в этом выражении имеется, очевидно, $n!$ слагаемых (каждое состоит из n множителей)! И обсуждение закончено. Но кто знает, каково значение $n!$, когда n принимает значения 10, 20, ..., т.е. значения очень скромные для любой практической задачи, имеющей матричное решение (например, задачи метеорологии или моделирования течения жидкостей, где нередко приходится оперировать матрицами, имеющими тысячи строк и столбцов)? И снова математик может ответить, используя приближение Стирлинга, что

$$n! \approx n^n e^{-n} \sqrt{2\pi n} \text{ и даже } n! > n^n e^{-n} \sqrt{2\pi n},$$

а любая хорошая система алгебраического вычисления дает точные результаты:

$$10! = 3628800 \text{ и } 20! = 2432902008176640000.$$

Последнее значение производит некоторое впечатление, оно приблизительно равно $2 * 10^{18}$. Зная, что современные микро-ЭВМ способны совершать около миллиона сложений в секунду, естественно спросить: сколько времени затратит такой компьютер, чтобы выполнить $2 * 10^{18}$ сложений?

Этот пример без дополнительных выкладок не убеждает скептиков, которые считают, что стали жертвами фокуса. Вот по этой причине мы и решили практически доказать сложность таких программ, взяв в качестве примера вычисление определителя.

4.1. 4.1 Элементы обработки массивов в языке Ада

Матрицы обычно представлены в алгоритмах и программах с помощью массивов (если речь не идет о разреженных матрицах, для которых в большинстве случаев представление оптимизируют). Кроме того, как мы увидим в дальнейшем, перестановки тоже можно представить с помощью этой структуры. Следовательно, нужно изучить обработку массивов в языке Ада.

Обработка массивов в языке Ада имеет только отдаленное отношение к обработке, которую можно сделать в классических языках, таких как Паскаль, а средства, предоставленные в распоряжение пользователя разработчиками языка, удовлетворяют все его (разумные) ожидания. Хотя в языке Ада и невозможно определить массив переменного размера, но существует понятие, позволяющее обрабатывать массив неизвестного размера во время построения программы: это понятие **шаблона** (*pattern* в англосаксонской терминологии).

В качестве первой иллюстрации принципа шаблонов мы покажем, как в языке Ада построить процедуры или функции, число аргументов которых кажется переменным. Этот метод использования шаблонов не очень принят, но дает блестящие результаты в задачах такого вида (см. работу Буха [27]). Выбранный нами этого пример является построением функции, позволяющей находить ее наибольший аргумент (среди любого их количества). Для начала рассмотрим массив, в котором множество индексов (подынтервал целых положительных чисел) и тип основных элементов произвольны. Массив называется T , а тип элементов массива – *Object*. Так как мы ничего не знаем о типе элементов, то если мы хотим вычислить максимум элементов массива, нужно располагать оператором сравнения; этот оператор обозначен «<». Вычисляемое значение Max должно удовлетворять условию:

при любом индексе i массива T : $T(i) = Max$ или $T(i) < Max$.

В этом выражении внимательный читатель отметит использование двух операторов, «=» и «<», а не единственного оператора «≤»: программистская шероховатость, которая будет сглажена при реализации этой функции в языке Ада. Алгоритм вычисления максимума очень прост, и нет необходимости его подробно объяснять. Итак, предлагаем сразу реализацию этой функции в языке Ада, учитывая тот факт, что единственные допустимые операции на элементах типа *Object* – это: описание, присваивание и сравнение оператором «<».

```
function Max( $T$  : Objects_Array) return Object is
  Temporary_Maximum : Object :=  $T(T'First)$ ;
begin
  for  $i$  in  $T'First + 1..T'Last$  loop
    if Temporary_Maximum <  $T(i)$  then Temporary_Maximum :=  $T(i)$ ;
    end if;
  end loop;
  return Temporary_Maximum;
end Max;
```

Основная характеристика этой части Ада-программы – это полное отсутствие явной ссылки на любое особое свойство обрабатываемого массива: точно не известно, каковы индексы массива, и также не известны элементы, составляющие массив, а единственные используемые операции – это три операции, названные выше. Однако можно обработать массив, обработать его границы благодаря предопределенным функциям в языке Ада, которые называют *атрибутами*. Таким образом, если T – массив, выражение $T'First$ обозначает наименьший индекс массива T , а $T'Last$ обозначает наибольший индекс T . Это доказыва-

ет, что функция, имеющая массив в качестве формального параметра, может каждый раз заново определять некоторые характеристики массива, который ей передан в качестве фактического параметра (длина, границы...): атрибуты позволяют определить эти характеристики *динамически*, т.е. во время выполнения. Хотя это явно не выражено, но в этой функции используется особое свойство индексов: их целочисленный характер, который позволяет записать выражение $T'First + 1$.

Логическое следствие всего этого заключается в том, что текст функции *Max* всегда один и тот же и что обрабатываемые объекты — это целые числа, действительные числа, литералы перечисления или любой другой объект при условии, что пользователь задаст операцию сравнения. Другими словами, очень легко записать на основе всего сказанного настраиваемый алгоритм вычисления максимума. Таким образом, спецификация, обозначения которой слегка отличаются от предыдущих, выглядит так:

Спецификация настраиваемого алгоритма вычисления максимума

```
generic
  type Object is private;
  with Function < "(a,b : Object) return Boolean is <>;
package Maximum is
  type Objects_Array is array (Positive range <>) of Object;
  function Max(T : Objects_Array) return Object;
end Maximum;
```

Рассмотрим сначала формальные параметры настройки для этого пакета. Тип *Object* — это приватный тип; это означает, как мы видели в разделе 3.3 (стр. 53), что единственными операциями, допустимыми неявно внутри пакета на объектах этого типа, являются описание, присваивание, тест на равенство и использование в качестве параметра подпрограммы. Следовательно, разработчик функции *Max* ничего не знает о типе *Object*; он не может знать, идет ли речь о целом типе, последовательности знаков... Второй формальный параметр — это оператор сравнения элементов типа *Object*, необходимый для вычисления максимального значения, но который невозможно прямо ввести из-за незнания типа *Object*. Кроме того, этот формальный параметр настройки имеет значение по умолчанию; описание ► **with function** < "(a,b : *Object*) return Boolean is <>; ◀ означает, что в случае, когда пользователь настраиваемого пакета не задает явно эту функцию в момент конкретизации, то функция, которая должна быть использована, — это функция <, имеющая соответствующий тип параметров и видимая в месте конкретизации. Формальным параметрам настройки

могут быть заданы другие виды значений по умолчанию, в дальнейшем мы еще с ними встретимся.

Имея эти два параметра, разработчик функции *Max* задает пользователю, с одной стороны, тип *Objects_Array*, который, в основном, не будет использоваться, и функцию *Max*, которая нас интересует. Тип *Objects_Array* служит только для описания типа параметров функции *Max*. Как будет видно в применении функции, пользователь может его полностью игнорировать. Описание типа *Objects_Array* – это несколько своеобразное описание массива: ► **type** *Objects_Array* **is array** (*Positive range*<>) **of** *Object*; ◀. Это будет точное описание шаблона, и говорить о типе, имеющем отношение к идентификатору *Object_Array*, значит злоупотреблять языком. Действительно, невозможно описать переменные типа *Object_Array*, потому что Ада, как любой другой язык, должен иметь возможность предоставить в памяти место, необходимое для массива. Впрочем, вполне возможно использовать этот шаблон для описания типа формальных параметров подпрограммы: при фактическом использовании подпрограммы ее параметрами будут описанные переменные (обратное было бы удивительным) или выражения, построенные в момент вызова, характеристики которых хорошо известны, когда контроль идет внутри подпрограммы.

Тело пакета *Maximum* записывается прямо, непосредственно и является, по существу, ни чем иным, как переформулировкой функции *Max*, представленной выше.

Реализация функции *Max*

```
package body Maximum is
  function Max(T : Objects_Array) return Object is
    Temporary_Maximum : Object := T(T'First);
  begin
    for i in T'First + 1..T'Last loop
      if Temporary_Maximum < T(i) then Temporary_Maximum := T(i); end
      if;
    end loop;
    return Temporary_Maximum;
  end Max;
end Maximum;
```

Перейдем теперь к использованию этой функции. Неинтересно подробно объяснять всю программу, использующую эту функцию (в самом деле, длина такой программы была бы огромной по сравнению с необходимым пояснением функции *Max*), однако несколько фрагментов такой программы разъяснят возможности функции.

```

with Maximum; -- Продолжение контекста компиляции
procedure Using_Function_Max is
  type My_Integer is
    ... -- В этом месте уже определен оператор "<"
           на целых числах типа Integer
  package Integer_Maximum is new Maximum(My_Integer);
           use Integer_Maximum;
  x, a, b, c, d : My_Integer; -- Продолжение описаний (переменные ...)
begin
  -- Начало вычислений
  x := Max((a, b, c)); -- Продолжение вычислений
  x := Max((a, b, c, d)); -- Конец вычислений
end Using_Function_Max;

```

Эта программа использует конкретизацию пакета *Maximum* для целых чисел; как видим, она начинается с оператора контекста (может быть неполного в примере), ссылающегося на настраиваемый пакет. Внутри главной процедуры после определения целого типа, который наследует обычные операции на целых числах, и в частности, оператор сравнения «<», находим конкретизацию настраиваемого пакета, в которой задан только фактический параметр, соответствующий типу *Object*; так как второй параметр явно не задан, при конкретизации для него используется значение по умолчанию, т.е. стандартный оператор «<» сравнения целых чисел. Можно уже заметить, что если бы вместо использования оператора «<» передали в качестве второго параметра оператор «>», как в команде

```

package Integer_Minimum is new Maximum(My_Integer, ">");

```

то построенная таким образом функция *Max* вычислила бы наименьший из своих аргументов, а не наибольший.

После спецификатора видимости ► **use** *Integer_Maximum* ◄ и нескольких дополнительных описаний находим тело главной процедуры, которое содержит определенное количество дополнительных вычислений и (что нас интересует) два обращения к функции *Max*:

```

x := max((a, b, c)); x := max((a, b, c, d));

```

Выражения, которые образуют фактические параметры этих двух обращений, являются *агрегатами*, т.е. явными структурными конструкциями. Фактический параметр первого обращения к функции *Max* — это агрегат (*a, b, c*), который вне своего контекста представляет структуру с тремя компонентами типа *My_Integer*. Это выражение (агрегат) совместимо с точки зрения типа как с записью с тремя полями типа *My_Integer*, так и с индексруемым типом с тремя компонентами, индексы которого абсолютно произвольны. Контекст

использования этого агрегата определяет точным образом тип выражения (a, b, c) : так как это параметр функции *Max*, то мы имеем дело с индексруемым типом с тремя компонентами (потому что в агрегате их три), индексированным посредством подынтервала типа *Positive*. О нем больше ничего не известно, и точное определение данного интервала не представляет здесь никакого интереса (за дополнительными сведениями читатель может обратиться к *ARM*). В этих условиях второй вызов функции *Max* существенно не отличается от первого: эта функция может применяться к массивам любой длины, и если бы не перегруженность скобками, то можно было бы сказать, что функция *Max* обладает переменным числом параметров.

Теперь должно быть ясно, что пользователю этой функции нет необходимости представлять тип *Objects_Array* в программе; один из случаев, когда это будет необходимо, соответствует использованию функции *Max*, применяемой к *переменной* индексруемого типа, но это уже другая история...

4.2. 4.2 Работа с матрицами

Следующий этап обучения обработке массивов в языке Ада заключается в изучении представления матриц, базовых операций над матрицами, ввода-вывода матриц – короче, в определении так называемого **абстрактного типа данных** и в построении нескольких средств управления такого типа.

Первая фаза – определение абстрактного типа данных – начинается с распознавания того, что называют простейшими операциями на типе. Сначала можно отметить, что эти операции подразделяются на два вида: **конструкторы** и **селекторы**¹. Конструкторы – это операции, которые модифицируют состояние (значение) данных, тогда как селекторы позволяют только узнать это состояние.

Каковы же будут фундаментальные операции, когда объект – матрица? Прежде всего, имеется конструктор, который позволяет присваивать значение элементу матрицы, а также симметричная операция селектор, позволяющий узнать значение одного из элементов. Например, если A – рассматриваемая матрица, а a_{ij} – интересующий нас элемент, то обычно обозначаем $A(i, j) \leftarrow v$ и $v \leftarrow A(i, j)$ те действия, которые реализуют эти операции. Но такой способ действия имеет мно-

¹Эта терминология кажется повсюду принята программистами, она произошла из разработки технологии программного обеспечения. Можно обратиться к книге Буха [27], которая дает более полные определения с последующим стилем программирования.

го последствий и, в частности, приводит к тому, что матрицы всегда представляются с помощью массивов. Однако хорошо известно, что это не так! Некоторые матрицы – разреженные: матрицы, у которых только небольшое количество элементов не равно нулю; было бы совершенно неразумно, с точки зрения информатики, представлять матрицу больших размеров, например, 1000×1000 , с помощью массива в миллион клеток, если всего лишь 10000 элементов не равны нулю. Следовательно, если все программы, составленные для работы с матрицами, используют понятие индексировемого типа, то эти программы становятся полностью непригодными, когда хотят применить их алгоритмы к разреженным матрицам. Иначе говоря, программы слишком зависимы от внутреннего представления обрабатываемых объектов.

Конкретно, этот тип представления обязывает программиста выражать присваивание и оценку элемента матрицы, соответственно, посредством вызова процедуры и функции, даже если это, возможно, будет тяжелее записывать (что справедливо лишь отчасти). Конструктор обозначен *Set_Coefficient*, а селектор – *Coefficient_Of* со спецификациями, которые мы увидим в пакете, представленном далее.

Этих двух операций недостаточно, чтобы выполнить все возможные манипуляции на матрице; не хватает селекторов, позволяющих узнать размер матриц, чтобы знать, какие элементы существуют. Так как тонкая структура матрицы совершенно не известна пользователю матрицы (но не разработчику, роль которого мы сейчас выполняем), невозможно использовать различные атрибуты массива, как в предыдущем разделе а нужно явно задать функции, позволяющие оценить границы матрицы: *First_Row*, *Last_Row*, *First_Column*, *Last_Column*.

Чтобы модуль был как можно более автономным, нужно предусмотреть и классифицировать возможные ошибки и сопоставить им исключения. В случае матрицы единственная ошибка обработки – это ссылка на несуществующий элемент; эту ошибку символизирует исключение *Index_Out_Of_Bounds*, которое может появиться только при выполнении простейших операций *Set_Coefficient* и *Coefficient_Of*. Вот полная спецификация настраиваемого пакета, реализующего абстрактный тип данных.

Матрица с целыми элементами

```
generic
  type Row_Index is range <>;
  type Column_Index is range <>;
  type Coefficient is range <>;
```



```

package Matrix_Nonsparse_Integer is
  type Matrix(First_Row : Row_Index; Last_Row : Row_Index;
               First_Column : Column_Index; Last_Column : Column_Index) is
    private;
  procedure Set_Coefficient(Of_The_Matrix : in out Matrix;
                             At_Row : in Row_Index;
                             And_Column : in Column_Index;
                             To_The_Value : in Coefficient);
  function Coefficient_Of(The_Matrix : Matrix;
                           At_Row : Row_Index;
                           And_Column : Column_Index) return Coefficient;
  function First_Row(Of_The_Matrix : Matrix) return Row_Index;
  function Last_Row(Of_The_Matrix : Matrix) return Row_Index;
  function First_Column(Of_The_Matrix : Matrix) return Column_Index;
  function Last_Column(Of_The_Matrix : Matrix) return Column_Index;

  Index_Out_Of_Bounds : exception;

private
  type Matrix_Template is
    array(Row_Index range <>, Column_Index range <>) of Coefficient;
  type Matrix(First_Row : Row_Index; Last_Row : Row_Index;
               First_Column : Column_Index; Last_Column : Column_Index)
  is record
    The_Coefficient :
      Matrix_Template(First_Row..Last_Row, First_Column..Last_Column);
  end record;
  pragma Inline(Set_Coefficient, Coefficient_Of,
                 First_Row, Last_Row, First_Column, Last_Column);
end Matrix_Nonsparse_Integer;

```

Первое замечание касается имени пакета, *Matrix_Nonsparse_Integer*. В этом имени соблюдаются простейшие правила построения: первый радикал означает, что абстрактный тип данных, который в нем определен, – матрица, второй, что эти матрицы не разреженные (алгоритмы доступа к разреженным матрицам, в основном, другие), и, наконец, суффикс указывает на тип элементов. Это имя позволяет сразу же узнать примерные характеристики определенного объекта¹. Рассмотрим теперь видимую часть этой спецификации (т.е. все то, что предшествует ключевому слову **private**). Прежде всего, формальных параметров настройки три и их семантика должна быть ясной...

¹Эта проблема может показаться устаревшей, но в программировании выбор идентификаторов – важная и тонкая операция, и некоторые труды по разработке технологии программного обеспечения посвящают этой проблеме целую главу.

Читатель педантичный (и усидчивый, см. раздел 3.3) сможет нас упрекнуть в том, что мы не определили, что элементы — личного типа (т.е. почти любого), это позволило бы обрабатывать одинаковым способом матрицы целых чисел, чисел с плавающей запятой, многочленов и т.д. Этот выбор основан на том, что обычно не совершают один и тот же тип вычислений и на матрицах с целыми и на матрицах с действительными элементами и что алгоритмы вычисления на матрицах многочленов не являются простым применением в кольце многочленов алгоритмов для матриц с элементами из числового кольца. Другими словами, алгоритмы вычисления на алгебраических структурах соответствуют структурному типу, который подлежит обработке. Например, если A — матрица и если a мы хотим вычислить A^n , то для этого можно взять великолепную настраиваемую функцию дихотомического возведения в степень, изученную в разделе 3.3; но намного более экономично, при условии, что n — достаточно большое, вычислить характеристический многочлен (или минимальный, если умеем его вычислять) от A и прежде, чем вычислить произведения матриц, привести A^n по модулю этого многочлена. Все эти примеры показывают, что можно фиксировать тип элементов матрицы: что теряем в общем, то выигрываем в полезном...

Собственно спецификация начинается с определения личного типа *Matrix*, который обладает четырьмя дискриминантами с красноречивыми именами. Когда этот пакет получит начальные значения, тогда будет возможно описать, например, переменную $\blacktriangleright A : Matrix(2, 4, 3, 7); \blacktriangleleft$ это означает, что A — это матрица 3×5 , у которой индексы строчек находятся в интервале $[2, 4]$, а индексы столбцов — в интервале $[3, 7]$. Конкретизация этого пакета порождает не единственный матричный тип, а, скорее, класс матриц, имеющих некоторые общие характеристики (тип индексов и конкретный тип элементов). Преимущество этого способа действия состоит в том, что он позволяет обрабатывать матрицы различных размеров, исходя только из начальных значений.

Перейдем теперь к точному определению типа *Matrix* — определению, которое обычному, среднему пользователю знать необязательно. Так как реализуются «полные» матрицы, внутреннее представление использует двумерные массивы. Но синтаксис языка Ада предписывает, чтобы определение типа, фигурирующее в личной части, имело то же начало, что присутствует у него в спецификации. Следовательно, этот тип должен быть типом дискриминанта записи, его единственное поле (отличное от дискриминантов) — это двумерный массив, о котором

«««< HEAD =====>»»»> 4f4d0f78d3cc932ba2fb67aca17bd68d8957da6f мы говорили. На распечатке спецификации можно увидеть определения типов, которые должны быть представлены в личной части. Все это может показаться невероятно сложным и неэффективным. Это действительно сложно для того, кто не имеет достаточного опыта в программировании. Эта сложность кажется бесполезной тому, кто никогда не пытался повторно использовать свои программы в контекстах, слегка отличных от контекста разработки: с опытом приходит понимание, что эту цену необходимо платить один только раз — лишь во время первого написания. Некоторые могут еще сослаться на то, что теряется четкость: пишем ► *Set-Coefficient* (*A*, *i*, *j*, ...) ◀ вместо ► *A*(*i*, *j*) := ... ◀. И опять, все это вопрос опыта: если стиль — однородный, программы читаются так же легко, как и программы, записанные обычным способом.

В том, что касается эффективности, язык Ада предоставляет нам некоторые средства, позволяющие влиять на скорость выполнения программ или на объем их памяти. Для этого компилятору передаются указания, называемые прагмами. Возможно, что эти указания будут безрезультатны (это зависит от используемого компилятора), но не ставит большого труда их предусмотреть. Как это можно увидеть в личной части, прагма *Inline* — которая указывает, что обращения к подпрограммам *Set-Coefficient*, *Coefficient-Of*, ... нужно развернуть «в строке», т.е. заменить обращения к этим подпрограммам разложением их тела, — применима для всех простейших операций этой спецификации.

Вторая фаза определения типа *Matrix* — его реализация, очень простая операция. Действительно, внутренняя структура представляет собой двумерный массив, реализация примитивов — это перезапись процедур и функций спецификации в терминах массивов. Вместо долгих объяснений проще прямо представить тело пакета.

Реализация матриц с целыми элементами

```
typedef double value_T;
struct _matrix_T {
    int n, m;
    value_T *body;
}
typedef struct _matrix_T *matrix_T;

bool set_coefficient(matrix_T mat,
                    row_T row,
                    column_T col,
                    value_T val) {
    int mat_i = TO_COEFFICIENT(row, col);
    if (mat_i > mat->max_i) {
```

```

        // Exception Handler...
        return false;
    }
    mat->body[mat_i] = val;
    return true;
}

value_T* last_row(matrix_T mat) {
    size_t m = mat->m;
    size_t n = mat->n;
    return mat->body + (n-1)*m;
}

value_T* last_col(matrix_T mat) {
    size_t m = mat->m;
    size_t n = mat->n;
    return mat->body + m;
}

value_T* next_col(matrix_T mat, value_T *cur_col) {
    return cur_col + mat->m;
}

value_T* next_row(matrix_T mat, value_T *cur_row) {
    return cur_row + mat->n;
}

```

Можно сразу отметить простоту кода (нельзя по-настоящему го ворить об алгоритме), но, тем не менее, нужно уточнить несколько моментов. При определении типа мы представили некоторые ошибки, которые могут появиться при использовании этого пакета. Следова тельно, надо выявить эти ошибки в теле имеющих к этому отношение простейших операций. Для этого существует несколько способов:

- В начале каждого модуля, в котором может появиться ошибка, выполняют тест, позволяющий узнать, не произойдет ли ошибка. Для этого нужно уметь предусмотреть все потенциальные ошибки и надо уметь распознавать их предпосылки, что не всегда легко и возможно;
- Другое решение состоит в том, чтобы использовать механизм исключений языка Ада, т.е. дать возможность модулям следовать обычному ходу их выполнения и предусмотреть в конце модуля обработчик исключений, восстанавливающий predeterminedенные исключения и распространяющий одно из исключений, которое было определено для этого типа данных. Это второе решение, заранее привлекательное из-за своей эффективности, — в некоторых реализациях языка

Ада обработка исключений не требует дополнительных издержек, пока не появятся исключения — имеет то т недостаток, что не гарантирует связности данных. В случае матриц, как они были только что реализованы, эта проблема не требует особого внимания.

Исключение представляет собой асинхронное или непредусмотренное событие, которое может вызвать прекращение обработки информации, обеспечивающей связность некоторых данных. Например, при реализации динамических структур, связность которых обеспечивается указателями, очень важно, чтобы ошибки не возникали в момент, когда связи между структурами несогласованы, что часто происходит при модификациях. Если это возможно, нужно обязательно выявить потенциальные ошибки перед их появлением, и следовательно, применить первый метод. Вновь эти проблемы возникают потому, что от пользователя хотят скрыть истинное представление обрабатываемых объектов, при этом маскирование информации доходит до утаивания внутренних ошибок! Итак, находим обработчики исключений в реализации двух элементарных операций `Set-Coefficient` и `Coefficient.Of`: предопределенное исключение `Constraint-Error`, возбуждаемое в случае ошибки выхода индекса матрицы за его пределы, устраняется и далее распространяется исключение `Index-Out-Of-Bounds`. Выполнение четырех небольших примитивов, позволяющих узнать границы матрицы, не может вызвать ошибку, следовательно, там не должно быть никакой обработки ошибки. Мы уже встречали атрибуты, действующие на массивах, в разделе 4.1, и в этих последних функциях снова находим атрибуты `First` и `Last`, это показывает, каким образом данные атрибуты применяются в случае многомерного массива, например, в `return Of-The-Matrix.The.Coefficient'First (1), < параметр атрибута First указывает номер размерности, к которой применяется First.`

Замечание. Хотя только что изученные операции на матрицах являются, действительно, единственными элементарными операциями, может случиться, что в эту спецификацию добавят операции, которые очень часто используются. Это в большинстве случаев очень простые операции, но которые по причинам эффективности и простоты выигрывают за счет информации о точном представлении данных.

4.3. Ввод-вывод матриц

Второй модуль обработки матриц позволит нам научиться использовать один тип абстрактных данных. Для задачи, которая нас сейчас занимает, первым необходимым нам инструментом является множество функций ввода-вывода. Такой аппарат содержит, как минимум, процедуру чтения `Get` и процедуру записи `Put` для терминала, которых пока достаточно. Эти две процедуры предоставляют минимальное удобство для пользователя: `Put` имеет параметр, позволяющий задать размер поля для вывода элементов матрицы, которую хотят вывести; а `Get` имеет параметр, который задает выводимое для ввода одной строки матрицы. У этих двух параметров есть, однако, одна особенность: они имеют значение по умолчанию. Благодаря этому, если выполняется команда ► `Get (Моя-Матрица)`; при отсутствии второго параметра в вызове, программа будет использовать его значение по умолчанию, и никакого приглашения при чтении матрицы не появится. Зато если уточнить ► `Get (Моя-Матрица, ◀`, каждое чтение одной строки будет предваряться вопросительным знаком, позволяя пользователю знать, где он находится. Значение по умолчанию второго параметра процедуры `Put` есть `Coefficient 'Width`, результат применения предопределенного атрибута `Width` к целому типу `Coefficient`: этот атрибут задает максимальное число символов для вывода какой-либо величины типа `Coefficient`. После того как описаны средства, предлагаемые модулем, т.е. услуги, которые он способен предоставить, нужно познакомиться и с услугами, в которых он нуждается, чтобы реализовать все это. Эти услуги появятся в качестве формальных параметров настройки и должны быть обеспечены предыдущим пакетом. Тогда получится следующая полная спецификация:

Ввод-вывод матриц

```
bool set_coefficient(matrix_T mat,
                    row_T row,
                    column_T col,
                    value_T val) {
    int mat_i = TO_COEFFICIENT(row, col);
    if (mat_i > mat->max_i) {
        // Exception Handler...
        return false;
    }
    mat->body[mat_i] = val;
    return true;
}

value_T* last_row(matrix_T mat) {
    size_t m = mat->m;
```

```

    size_t n = mat->n;
    return mat->body + (n-1)*m;
}
value_T* last_col(matrix_T mat) {
    size_t m = mat->m;
    size_t n = mat->n;
    return mat->body + m;
}
value_T* next_col(matrix_T mat, value_T *cur_col) {
    return cur_col + mat->m;
}
value_T* next_row(matrix_T mat, value_T *cur_row) {
    return cur_row + mat->n;
}
bool put(matrix_T mat, size_t row, size_t col, value_T val) {
    int mat_i = TO_COEFFICIENT(row, col);
    if (mat_i > mat->max_i) {
        // ERROR HANDLER
        return false;
    }
    mat->body[mat_i] = val;
    return true;
}
value_T get(matrix_T mat, size_t row, size_t col) {
    int mat_i = TO_COEFFICIENT(row, col);
    if (mat_i > mat->max_i) {
        // ERROR HANDLER
        return mat->body[0];
    }
    return mat->body[mat_i];
}
}

```

«Это не упрощает дела!» — сразу скажет читатель. Еще раз по вторым сказанное: только опыт позволит судить об этом, и во время заключительного построения будет видно, что описанный способ действия имеет, скорее, тенденцию к упрощению архитектуры программы. Однако сделаем несколько уточнений:

- процедуры ввода-вывода нуждаются в информации о типе `Matrix`, и поскольку он определен как личный тип, то здесь он тоже должен быть объявлен личным;
- для считывания и записи должны быть известны границы обрабатываемых матриц, так же как и природа элементов, ибо, в конечном

счете, именно над ними производятся действия; это обясняет присутствие первых четырех параметров настройки;

- наконец, нужно ввести все примитивы, определенные в предыдущем разделе.

Данная спецификация, которая кажется довольно громоздкой, полна и очень мало связана с другими модулями. Это означает, что использовать указанный модуль весьма просто и что модификации описания типа абстрактных данных *Matrix* или других тяготеющих к нему модулей скажутся на этом особом модуле.

Приводим реализацию описанной процедуры, которая пока не представляет никакой особой трудности.

```
bool get_all(matrix_T mat) {
    for (size_t i = 0; i < mat->max_i; ++i)
        put(mat, m, n, get_val());
    return true;
}
```

Отметим способ, с помощью которого здесь используются примитивы работы с матрицами, а также ассоциацию по именам фактических параметров самих этих примитивов. Сделаем последнее замечание, касающееся имени рассмотренного пакета: что хоть и следует избегать этого в общем случае, в данном идентификаторе используют аббревиатуру 10 — общепринятый акроним, означающий Input/Output, как, например, в *TextIO*.

4.4. Описание генератора биекций

Перейдем теперь к вычислению определителя матрицы, пользуясь формулой определения 7 в начале раздела 4 и предполагая, что мы обладаем генератором биекций между двумя целыми интервалами. Реализация самого генератора отнесена в конец главы, чтобы не отвлекаться от нашей первоначальной задачи. Даже если на время не стоит вопрос о деталях построения генератора перестановок, нужно, по крайней мере, знать способ его применения и инструменты, которые он предлагает, — короче, его описание. Первый шаг на этом пути — выбор структуры данных, допускающей представление одной перестановки. Существует несколько стандартных обозначений для перестановок: в форме произведения циклов, в форме таблицы, представляющей граф перестановки, и т.п. Например, перестановка δ интервала $[1, 5]$, определенная равенствами $\delta(1) = 2, \delta(2) = 4, \delta(3)$ представляется в форме произведения циклов как: $\delta = (124)(35)$, а в форме таблицы, как $\delta = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 5 & 1 & 3 \end{pmatrix}$ Этот второй способ представления дает более непосредственный взгляд на перестановку (по крайней мере, на

нашем уровне) и больше подходит для алгоритмической обработки, чем запись через произведение циклов. Обычно это представление упрощают, записывая только вторую строку таблицы и подразумевая первую. По этому соглашению перестановка сг предыдущего примера представляется пятеркой (2,4,5,1,3). Выбрав структуру данных, остается найти процедуру перебора множества перестановок. Канонический метод просмотра конечного множества состоит в полном его упорядочении от минимального (соответственно, максимального) элемента для этого порядка и функции следования (соответственно, предшествования). Вот описание настраиваемого пакета, определяющего такой генератор.

```
typedef enum {  
  \\ ...  
} source;  
  
typedef enum {  
  \\ ...  
} target;  
  
typedef enum {  
  \\ ...  
} signature;  
  
typedef target bijetion[RANGE];  
bijection min_bij(source s_min,  
                 source s_max,  
                 target t_min,  
                 target t_max);  
inline void succesor_of(bijection bij, signature sig);
```

Этот модуль определяет, как указывает его имя, генератор биекций, а не перестановок. Разница между двумя понятиями очень незначительна, но рассмотрение биекций допускает, как увидим во время реализации, более простой подход. Заглавные параметры этого модуля суть типы (целые) интервалов от начальной точки до конечной, на которых действует биекция. В нашем случае эти интервалы будут интервалами изменения индексов строки и столбца обрабатываемой матрицы. С помощью этих двух типов целых программа выдаст нам так же два типа: один, позволяющий представить перестановки в форме таблицы, другой — тип-сигнатуру. Затем, основываясь на принципе перечисления биекций на некотором порядке, используют функцию, определяющую наименьшую биекцию относительно этого порядка, и процедуру вычисления последующей за данной биекцией, а также ее сигнатуру. Когда обрабатывают перестановки, наи-

меньшей перестановкой в выбранном порядке является тождественная, как это увидим ниже, чья сигнатура равна 1; это указывает инициализацию, которую должна осуществить программа вычисления детерминанта. Тогда можно легко перебрать множество биекций между двумя интервалами целых чисел. Возможны две ошибки. `Bijection.Overflow` выполняется, когда пытаются вычислить биекцию, которая следует за самой большой в выбранном порядке; и вторая — если пытаются вычислить минимальную биекцию между двумя множествами с разными мощностями.

4.5. Вычисление определителя

Чтобы достичь нашей цели — вычисление определителя — остается только объединить множество небольших кусков, которые мы построили до сих пор: матричный тип, ввод-вывод, генератор перестановок и написать, собственно, подпрограмму вычисления. Матрицы, которые мы будем обрабатывать, — квадратные, с целыми элементами, и мы будем предполагать, что индексы строк и столбцов имеют тот же тип. Когда этот выбор сделан, нужно конкретизировать оба настраиваемых пакета по оперированию с матрицами и вступить в диалог с пользователем. Именно это делает первый модуль программы, представленный здесь:

Вычисление определителя

```
typedef unsigned int index;
typedef long long int coefficient;

index first_line = matrix.first_line;
index last_line = matrix.last_row;
bijection sigma = minamal_bij(first_line, last_line, first_line,
    last_line);
signature s = 1;
coefficient res = 0, term;

coefficient determ(matrix a)
{
    term = (coefficient) s;
    for (int i = first_line; i <= last_line; i = mat.next_line(i))
    {
        term = term * coef_of_(A, i, sigma);
    }
    return term;
}

coefficient determ(matrix a)
```

```

{
    term = (coefficient) s;
    for (int i = first_line; i <= last_line; i = mat.next_line(i))
    {
        term = term * coef_of_(A, i, sigma);
    }
    return term;
}

```

Начинают, таким образом, с построения матричного типа и его примитивов; тогда можно конкретизировать пакет ввода-вывода матриц (эта последняя конкретизация довольно громоздка для записи) Конкретизация пакета Integer.IO позволяет считывать размер матрицы, подлежащей обработке. Это чтение размера находится в основном цикле, который содержит также блок, в котором объявляют матрицу, считывают ее элементы и проводят вычисление определителя. Исключение `Data.Error`, вставленное в обработчик в конце этой основной процедуры, позволяет изящно остановить программу, задавая нулевую размерность матрицы (так как тип `Index` является производным от типа `Positive`). Функция, осуществляющая непосредственное вычисление определителя, для большей ясности отделена от основной процедуры. Вот запись этой функции, которая ничем особенным не отличается, это лишь переписанное определение 7, данное в начале раздела 4:

Функция вычисления определителя

```

index first_line = matrix.first_line;
index last_line = matrix.last_row;
bijection sigma = minamal_bij(first_line, last_line, first_line,
    last_line);
signature s = 1;
coefficient res = 0, term;

coefficient determ(matrix a) {
    term = (coefficient) s;
    for (int i = first_line; i <= last_line; i = mat.next_line(i))
    {
        term = term * coef_of_(A, i, sigma);
    }
    return term;
}

```

Заметим, однако, что для облегчения записи для пакета `Matrix-Handler` был добавлен спецификатор `use`. Кроме того, во вре-

мя вычисления нужно осуществлять преобразование типа $\blacktriangleright \text{Term} := \text{Coefficient}(s); 4$, чтобы связать элементы и сигнатуру перестановки.

4.6. Вычисление определителя: несколько цифр

Прежде чем продемонстрировать несколько шагов выполнения, и даже прежде чем выполнять что бы то ни было, нужно оценить сложность процедуры, которую мы только что описали. Дорогостоящим элементом в программе, которую мы только что изучили, является функция Determinant, а точнее, самый внешний цикл, который находится в этой функции (тот, который повторяется для всех перестановок). Что видно в этом цикле? Во-первых, вложенный цикл, выполняемый n раз, где n — размерность обрабатываемой матрицы; затем суммирование, далее вычисление последователя какой-либо перестановки, что требует времени (по максимуму), пропорционального n . Внешний цикл, будучи выполненным $n!$ раз, дает сложность вычисления определителя порядка $(n \cdot n!)$. Первые примеры определителей суть определители Майе (см. гл. III). Это определители матриц X , определенных для простого p через:

$$X_{i,j} = \left\lfloor \frac{ij}{p} \right\rfloor - \left\lfloor \frac{(i-1)j}{p} \right\rfloor$$

где i и j находятся в интервале $[3, (p - 1)/2]$. Абсолютная величина определителя Майе для простого p обозначена $h^*(p)$. Эти определители

Эти определители имеют очень тесную связь с большой теоремой Ферма, точнее, если $p \nmid L^*(p)$, то уравнение $x^p + y^p = z^p$ не имеет целых нетривиальных решений. Основываясь на этих

Значение p	Размерность	h * (p)	Время (сек)
11	3	1	0.00
13	4	1	0.00
17	6	1	0.72
19	7	1	5.65
23	9	3	501.58

Таблица 1.3: Определители Майе

примерах, можно констатировать, что большая теорема Ферма справедлива для простых p , заключенных между 11 и 23. С другой стороны можно также констатировать, что время вычисления определителя размера 9×9 немного больше 8 минут (то, что величины даны с двумя знаками после запятой, означает только, что точность составляет 0,01 сек.) Чтобы дополнить измерения времени счета, которые мы только что проделали, и, таким образом, оценить коэффициент прогрессии времени счета, мы также вычислили определители для порядков, которые не фигурируют в предыдущей таблице.

Размерность определителя	Время (сек)
5	0.11
8	52.79
10	5666.83

Установлено, что вычисление определителя размера 10 x 10, независимо от его значения, требует около 1 часа 30 минут; для определителя 11 по рядка нужно при-

мерно один день для определителя 12 порядка потребуется около 10 дней... Таким образом, экспериментально установлено, что когда переходят от определителя порядка p к определителю порядка $p + 1$, время счета приблизительно умножается на $(p + 1)^2 / p$, что подтверждает теоретическую оценку сложности: $O(p \cdot p!)$. Конечно, можно упрекнуть программирование в его неэффективности — маскировка информации, вынуждающая обращаться к подпрограммам там, где обычно присутствует только ссылка на элементы массива, и широкое использование настраиваемых модулей. По этой причине приведены две другие таблицы 4: они показывают времена вычислений, полученные при реализации, одна — без применения настраиваемых модулей, а другая — без маскировки информации.

Размерность определителя	Время (сек)
5	0.11
6	0.77
7	5.88
8	51.51
9	499.93
10	5404.67

Размерность определителя	Время (сек)
5	0.11
6	0.77
7	5.88
8	51.51
9	499.93
10	5404.67

Первая из этих двух таблиц соответствует выполнению программы вычисления определителя по методу, описанному в этой главе, но посредством реализации без применения настраиваемых блоков, кроме стандартных пакетов ввода-вывода. Из этой таблицы видно, что улучшение времени вычислений, полученное данным методом, невелико — улучшение касается, очевидно, только константы, на которую умножается $p \cdot p!$ в оценке сложности — но не является пренебрежимо малым по отношению к полученному времени вычисления. Во всяком случае не меняется скорость возрастания времени счета. Вторая таблица касается измерения времени счета определителей с помощью вполне классической реализации, которая не содержит настраиваемых блоков и в которой представление матриц не замаскировано; в функции вычисления определителя появляются в явной форме такие выражения, как $\triangleright \text{Term} := \text{Term} * \text{Of-The-Matn}(\text{i}, \text{Sigma}(\text{i}))$; <.

Здесь улучшение гораздо больше (примерно на множитель 1,6 по отношению ко всему первому методу), но ясно, что если меняется пред-

ста вление матриц, нужно полностью переписывать функцию вычисления определителя. Таким образом, краткое сравнение издержек показывает, что часто выгоднее купить машину, которая в два раза быстрее по сравнению с имеющейся, чем написать программу, которая действует в два раза быстрее по сравнению с первоначальной. Кроме того, и это здесь убийственный довод, использованный метод очень плохой (так называемый метод «Барейса» — это эффективный метод вычисления определителей в целых числах). Ну, а каково время счета определителя размера 20×20 этим наивным методом?

4.7. Перестановки конечного множества

В пункте 4.4 мы выбрали представление перестановок, которое могло бы, ко всему прочему, подойти для представления отображений одного множества в другое: таблицу, содержащую элементы множества образов (данного отображения), индексированные элементами исходного множества. Это представление в форме таблицы напоминает не которыми аспектами понятие слова (в смысле элемента свободного моноида, см. пункт 3.3). Например, множество перестановок интервала $[1,5]$ может быть идентифицировано с множеством слов, образованных из пяти различных букв и построенных в алфавите 1,2,3,4,5. Тогда в голову естественно приходит мысль наделить множество перестановок индуцированным порядком, который существует для слов и который называется лексикографическим.

Определение 2 (лексикографический порядок на декартовом произведении двух множеств)

Пусть (E, \leq_e) и (F, \leq_f) — два линейно упорядоченных множества. Лексикографический порядок на $E \times F$ определяется для $(a,b), (a', b') \in E \times F$ следующим образом:

$$(a,b) \leq_{E \times F} (a', b') \iff [a = a' \text{ и } b \leq_f b' \text{ или } a \leq_e a']$$

Определенное таким образом отношение дает линейное упорядочение на $E \times F$, которое может удобно обобщаться по индукции на любое декартово произведение линейно упорядоченных множеств.

Замечание. Существуют другие способы, позволяющие определять линейный порядок на декартовом произведении данных множеств. Один из них, в частности, упорядочивает перестановки, чередуя их сигнатуры, чего не делает лексикографический порядок. Заинтересованный читатель может обратиться к упражнениям 17, 23 и 24 в конце главы.

Рассмотрим перестановку σ множества E , представленную пока таблицей из двух строк, и отыщем другую перестановку σ' , непосредственно превосходящую σ в лексикографическом порядке, индуцированном на второй строке таблицы.

$$\sigma = \begin{pmatrix} a_1 & a_2 & \cdots & a_{n-1} & a_n \\ b_1 & b_2 & \cdots & b_{n-1} & b_n \end{pmatrix} \text{ и } \sigma' = \begin{pmatrix} a_1 & a_2 & \cdots & a_{n-1} & a_n \\ b'_1 & b'_2 & \cdots & b'_{n-1} & b'_n \end{pmatrix}$$

выражения, в которых элементы a_i и b_i из E удовлетворяют для всякого $i \in [1, n-1]$ условию $a_i < a_{i+1}$.

Поскольку элементы a_i расположены в порядке возрастания, они не несут никакой полезной информации и могут быть подразумеваемы, что приводит нас к представлению перестановок в форме n -ок. Множество, которое мы пытаемся пронумеровать, есть, следовательно, множество всех n -ок различных элементов из E (мощности n). Предыдущее определение лексикографического порядка без труда распространяется на множество n -ок различных элементов из E , и задача теперь становится такой: считая данной n -ку $b = (b_1, b_2, \dots, b_n)$ элементов из E , найти следующую за ней n -ку $b' = (b'_1, b'_2, \dots, b'_n)$, удовлетворяющую условиям:

$$(i) \{b_1, b_2, \dots, b_n\} = \{b'_1, b'_2, \dots, b'_n\},$$

$$(ii) = (b_1, b_2, \dots, b_n) <_l ex(b'_1, b'_2, \dots, b'_n) = b',$$

(iii) $c > b \Rightarrow c \geq b'$, что можно также записать для линейной упорядоченности через $(b, b') = \emptyset$.

Прежде чем вычислять последователя какой-либо такой n -ки в общем случае, можно поинтересоваться существованием этого последователя.

Свойство 3

Пусть E — линейно упорядоченное множество. Тогда единственная строго убывающая (соответственно, возрастающая) последовательность, образованная из всех элементов E , представляет наибольшую (соответственно, наименьшую) перестановку E и не имеет, таким образом, последователя (соответственно, предшественника).

Это очень простое свойство позволяет применить рекурсию для построения следующего элемента для какой-либо n -ки в общем случае.

Действительно, имея определение лексикографического порядка на E^n для вычисления последователя какого-либо элемента нужно попытаться вычислить последователя в E^{n-1} для $(n-1)$ -ки, образованной последними составляющими первоначальной n -ки; и этот процесс редукции может повторяться до тех пор, пока не достигнем набора из $(n-k)$ элементов, который не имеет последователя. Предыдущее свойство позволит нам охарактеризовать такой набор из $(n-k)$ элементов: он образован убывающей последовательностью элементов. Таким образом, первый этап в вычислении последователя для n -ки элементов из E есть поиск финальной убывающей максимальной части этой n -ки; затем нужно переупорядочить элементы, и следующее свойство указывает, каким образом.

Свойство 4 (последователя для n -ки различных элементов)

Пусть E — множество из n элементов и пусть $b = (b_1, b_2, \dots, b_n)$. Предположим, что существует такое $l \geq 1$, что для всякого i из $(l, n-1]$ имеют место соотношения $b_i > b_{i+1}$ и $b_l < b_{l+1}$; тогда последователь n -ки в множестве n -ок различных элементов из E определяется как

$$(b_1, b_2, \dots, b_{l-1}, \quad b_k, \quad b_n, b_{n-1}, \dots, b_{k+1}, \quad b_l, \quad b_{k-1}, \dots, b_{l+1}),$$

где k — наибольший индекс, превосходящий l и такой, что $b_k > b_l$. Другими словами, чтобы вычислить следующий за b элемент, достаточно обратить финальную убывающую часть b , затем поменять элемент, предшествовавший этой части, с его последователем в этой части.

Доказательство.

- Прежде всего, должно быть ясно, что если индекс l не существует, то мы будем находиться в ситуации, выраженной в свойстве 9: n -ка не имеет последователя.
- Обозначим b' новую n -ку, определяемую этим свойством. Тогда ясно, что $b' >_{lex} b$. Действительно, эти две n -ки совпадают на их $(l-1)$ элементах, и их элементы с индексом l удовлетворяют условию $b'_l = b_k > b_l$.
- Остается показать, что b' является наименьшей мажорантой b в лексикографическом порядке; а это можно сделать, показав, что открытый интервал (b, b') пуст (в силу линейного лексикографического порядка). Предположим, что существует такое c , что $b < c < b'$:

b_1	\dots	b_{l-1}		b_l	$<$	b_{l+1}	$>$	$b_{l+2} \dots$
\parallel		\parallel						
c_1	\dots	c_{l-1}		c_l		c_{l+1}		$c_{l+2} \dots$
\parallel		\parallel						
b'_1	\dots	b'_{l-1}		b'_l	$<$	b'_{l+1}	$<$	$b'_{l+2} \dots$
				b_k		b_n		$b_{n-1} \dots$

Заметим прежде всего, как это подсказывает схема, что части слева от вертикальной черты равны во всех строках, следовательно, нет нужды учитывать их в дальнейшем. Рассмотрим только элементы каждой перестановки, чьи индексы превосходят или равны l . Следующее свойство, которое мы не будем повторять по ходу доказательства, является основным: *части трех перестановок, расположенные справа от вертикальной черты, образованы из одних и тех же элементов, различных между собой.*

Теперь мы покажем, что если предположить, что $b < c < b'$, то $b_l < c_l < b'_l$.

Действительно, предположим $b_l = c_l$; поскольку $b < c$, то из этого следует, что в лексикографическом порядке $b_{l+1...n} < c_{l+1...n}$, но часть $b_{l+1...n}$, будучи строго убывающей, максимальна в лексикографическом порядке, а так как она образована из тех же элементов, что и часть $c_l + 1...n$, то не может быть меньше последней. Следовательно, $b_l \neq c_l$ и, значит, $b_l < c_l$. Рассматривая части $c_{l+1...n}$ и $b'_{l+1...n}$, где последняя является минимальной в лексикографическом порядке, аналогичным образом получаем, что $c_l < b'_l$. Итак, $b'_l = b_k$, и выбор b_k делает невозможным найти среди элементов b_{l+1}, \dots, b_n элемент c_l , заключенный строго между b_l и b'_l , что противоречит первоначальному предположению (существует c , удовлетворяющее $b < c < b'$). Значит, b' есть последователь для b .

Замечание. Предыдущее свойство позволит нам эффективно вычислять последователя n -ки различных элементов из множества E мощности n . Кроме того, как будет видно из записи алгоритма — который уже должен просматриваться из этого рассуждения — нетрудно вычислить отношение между сигнатурами исходной n -ки и новой.

Наконец, перестановка — первоначальный объект нашего изучения — полностью исчезла, а все эти результаты могут при меняться в более общих рамках: для нумерации биекций конечного множества.

Теперь есть возможность разработать алгоритм, который реализует результаты предыдущего свойства. Перестановка будет представлена таблицей элементов из E , индексированной целыми числами, заключенными между p и q . Перестановка, или скорее таблица, которая ее представляет, будет называться *Sigma*. Дальнейшие построения уточняют преобразование *Sigma* в ее последователя в лексикографическом порядке. Эта конструкция осуществляется в три этапа и очень точно следует изложению предыдущего свойства.

Первая фаза

Цикл, который здесь видим, позволяет определить *First-Index*, начало финальной убывающей максимальной подпоследовательности (величину l свойства 10). Кроме того, переменная *Length* сегмента, что позволяет определить, не максимальной ли он длины (что, как это уже было видно, означает отсутствие у перестановки последователя) выражает длину этого сегмента, что позволяет определить, не максимальной ли он длины (что, как это уже было видно, означает отсутствие у перестановки последователя). **Вторая фаза**

Затем нужно обратить конечный сегмент, определенный в первой фазе; алгоритм, который осуществляет эту операцию, тоже очень прост.

```

Ascending_Index = First_Index + 1
Descending_Index = q
while(Ascending_Index < Descending_Index)
{
    tmp_value = Sigma(Ascending_Index)
    Sigma(Ascending_Index) = Sigma(Descending_Index)
    Sigma(Descending_Index) = tmp_value
    Ascending_Index = Ascending_Index + 1
    Descending_Index = Descending_Index - 1
}

```

Нужно заметить, что это последнее преобразование перестановки *Sigma* сделано через транспозиции, и поэтому отсюда можно вывести отношение между сигнатурами начальной перестановки и полученной из нее (число осуществленных перемен $s = Length/2$, и, значит, это отношение равно $(-1)^s$).

Третья фаза

Наконец, нужно отыскать в только что обработанной части таблицы наименьший элемент, превосходящий элемент с индексом *First_Index* (т.е. элемент с индексом *l* из свойства 10), и поменять их местами, что добавляет еще одну дополнительную транс позицию к преобразованию.

```

for (int i = First_Index; i ≤ q; ++i) {
    if (Sigma(i) > Sigma(First_Index))
        swap(Sigma(i), Sigma(First_Index));
}

```

Полный алгоритм генерации биекций формируется очень просто, достаточно склеить все фрагменты, которые мы только что изучили. Целиком это будет видно в следующем разделе, посвященном реализации всего этого.

4.8. Настраиваемый модуль генератора биекций

Мы закончим эту главу реализацией алгоритма перечисления биекций и, таким образом, построим тело цикла, спецификация которого была дана в разделе 4.4. Как это было видно в предыдущем рассуждении, названные биекции действуют на целых интервалах, что вовсе не лишает решение общности; всякая биекция между двумя конечными множествами может трактоваться как целая биекция. Напомним все-таки, что этот пакет, кроме типа биекций между двумя целыми интервалами и процедуры вычисления последующего элемента, дает также наименьшую биекцию в лексикографическом порядке (которая является и единственной строго возрастающей биекцией между двумя рассматриваемыми интервалами). Кроме того, процедура *Successor* имеет два параметра: исходная биекция и ее сигнатура, и эта процедура вычисляет следующую биекцию и ее сигнатуру.

Реализация генератора биекций (Продолжение)

```

.....

typedef Source Source_Interval; //range in SOURCE_MIN ..
    SOURCE_MAX
typedef Target Target_Interval; //range in Target_Min ..
    TARGET_MAX
Bijection Sigma = Create_Bijection(SOURCE_MIN, SOURCE_MAX);
    //range in Source_Interval
Source_Interval Source_Index = SOURCE_MIN;
Target_Interval Target_Index = TARGET_MIN;
if (Sigma.Length != TARGET_MAX - TARGET_MIN + 1) {
    fprintf(stderr, "Not_A_Bijection");
    return BIJECTION_ERROR;
}
for (;;) {
    Set_Index(Sigma, Target_Index);
    if (Source_Index == SOURCE_MAX) {
        break;
    }
    Source_Index++; Target_Index++;
}
return Sigma;
}

void Successor_Of(Bijection The_Bijection,
    Signature With_Signature) {
    Positive Length = 1;
    Source First_Index;
    Source Ascending_Index, Descending_Index;
    for(Source i = The_Bijection.First; i <= The_Bijection.Last;
        i++) {
//Get has signature 'Source * Get(Bijection The_Bijection, Source
Index)'
        if (*Get(The_Bijection, i) > *Get(The_Bijection, i + 1))
        {
            Length++;
        } else {
            First_Index = i;
            break;
        }
    }
}

```

Реализация генератора биекций (Продолжение)

```
.....
if (Length == The_Bijection.Length) {
    fprintf(stderr, "Bijection_Overflow");
    return; }
Ascending_Index = First_Index + 1; Descending_Index =
    The_Bijection.Last;
while (Ascending_Index < Descending_Index) {
    Swap(Get(The_Bijection, Ascending_Index),
        Get(The_Bijection, Descending_Index));
    Ascending_Index++; Descending_Index--;
}
```

Первая функция, включенная в тело цикла, есть функция *Minimal Bijection*. Её реализация требует особой заботы, если хотим избежать переполнения индексов и иметь возможность осуществлять те или иные проверки.

Начинают с определения двух подтипов, которые точно соответствуют целым интервалам исходного и конечного множеств биекции. Эти два типа позволяют очень точно контролировать множество возможных значений локальных переменных, а также проверять прежде всего, что интервалы действительно могут быть подвергнуты биекции. С этой проверки и начинается функция, и, возможно возбуждается исключение *Not_A_Bijection*. Если все указано с этой точки зрения, нужно предпринять ещё другие меры предосторожности; простой цикл, записанный слева не может быть использован в случаях несоответствия типов

```
for (Source i = Sigma.First;
    i <= Sigma.Last; i++){
    *Get(Sigma, i) = i;}
```

Нужно напротив пройти "вручную" по элементам исходного и конечного множеств, внимательно проверяя число осуществленных итераций. Для этого используйте две переменные *Source_Index* и *Target_Index*, придавая им изначально наименьшее соответственно исходного множества и множества конечного и просматривая каждое из этих двух множеств. Цикл заканчивается, когда одна из этих переменных достигнет своего максимального значения (другая переменная достигает своего максимального значения одновременно).

Принимая во внимание сказанное в предыдущем разделе, заключаем, что реализация функции *Successor* не требует дополнительных объяснений. Однако отметим присутствие **Pragma**: в этой процедуре часто бывает необходимо делать обмен значениями двух элементов таблицы, поэтому естественно определить процедуру *Swap*, которая это делает. Пригма *Inline*, которая применена к этой процедуре, означает, как это

показано в разделе 4.3, что нужно заменить обращение к процедуре *Swap* развертыванием "в линию" её тела. Совершенно очевидно, что это указание не может применяться к рекурсивной подпрограмме.

5. Заключение

...as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem. In this sense the electronic industry has not solved a single problem, it has only created them - it has created the problem of using its products. To put it in another way: as the power of available machines grew by a factor of more than a thousand society's ambition to apply these machines grew in proportion, and it was the poor programmer who found his job in this exploded field of tension between ends and means...¹

Edsger W. Dijkstra, The humble programmer (1972 [66])

Сделаем маленькое отступление. Около четверти века назад Дейкстра [65] опубликовал статью *"Go to statement considered harmful"*, в которой он вскрывал недостатки команды безусловного ветвления. Несколько лет спустя Вирт ввел язык Паскаль, который установил затем свою диктатуру в обучении более, чем на десять лет. Этот длительный период ригоризма структур управления позволил структурному программированию завоевать свои титулы и лавры и был совершенно необходим, чтобы умерить беспорядочное разбухание информатики. Однако некоторые авторы, и среди них Кнут (*Structured programming with goto statements*), настаивали на запрещении более гибкого стиля программирования, такого, за который ратовали Дейкстра, Хоар и Дал [58] или Вирт [182]. Но, как говорит Кнут [98]: "... During the

¹...пока не было машин, программирование вовсе не было проблемой; когда у нас появилось некоторое число слабых компьютеров, программирование стало представлять некоторую проблему, а теперь, когда мы имеем гигантские компьютеры, программирование стало столь же гигантской проблемой. В этом смысле электронная индустрия не решила ни одной проблемы, она только создала их - она создала проблему использования своей продукции. Другими словами: как только мощь доступных машин возросла более, чем в тысячу раз, так общество захотело использовать их в той же пропорции, и бедному программисту пришлось трудиться в этой стремительно расширяющейся области напряжения между целями и средствами ...

Э.В.Дейкстра, Смиренный программист (1972)

*programming, because I could' not bear to be found guilty of writing unstructured programs "*², и однако, в течение многих лет никто не осмеливался заявлять миру в лицо, что он не программирует структурным образом (в смысле Паскаля).

Язык Ада, имеющий прямое родство с Паскалем (и, конечно, с другими языками), сделал заметно более гибкой позицию структурного программирования; зато в сравнении с Паскалем Язык Ада ещё крепче закрутил гайки относительно типов данных. С этой точки зрения он действительно суров! Но соображения, которые являлись определяющими при создании языка Ада, уходят гораздо дальше *простых* задач программирования. Речь шла о построении стандартного языка, который, помимо всего прочего, позволил бы пользователям всех уровней обмениваться программами; мы же пока далеки от достижения этой цели. Мы полагаем, однако, что язык Ада обладает выразительной силой и четкостью, достаточными для того, чтобы служить основой для трансляции алгоритмов; но этого недостаточно ...

Гордое восклицание "Пошло с первого раза!" обнаруживает в программировании черты искусства. Это утверждение, вполне понятное, но редко произносимое, означает следующее: написать программу, которая правильно работает с первого раза, возможно, но это не типично. Так как программисты, несомненно, стараются писать программы, которые идут с первого раза, встает вопрос: "если это возможно, почему это необычно?". Ответ на этот вопрос двойной: во-первых, программировать трудно, и во-вторых, имеется имеется очень мало общих методов для разработки и написания хороших программ. Поскольку общих методов мало, каждый программист должен развивать свои собственные, часто с минимальным успехом. Успех этих методов зависит от способа, каким они адаптированы к конкретной задаче. По этой причине качество программ меняется не только от одного программиста к другому, но еще и от одной программы к другой у одного и того же программиста.

Анри Ледгар, Пословицы программирования (1975[115])

Программирование все еще является, в основном, техникой, требующей умений, которые не приобретаются из книг; в нем отсутствует строгость, которая приносит успех в математике. Как мы уже говорили, проверка программы имеющимися методами так же трудна, как и

² "...в 70-е годы я, как и все остальные, был вынужден принять идеи структурного программирования, так как я не мог допустить, чтобы меня обвинили в написании неструктурных программ

математическое доказательство, если не позволять себе никакого отклонения от обозначений в смысле логики; это выше человеческих сил, даже если предположить, что все необходимые для этого средства существуют. Однако представляется вполне возможным убедить разумного человека в том, что программа корректна, при условии ее представления в форме небольших модулей, поведение которых является понятным. Сопровождая эти модули просто проверяемыми утверждениями, мы делаем важный шаг в процессе передачи умений и навыков. Сколько мы изучили алгоритмов, постижение которых потребовало от нас длительных часов потому, что их авторы (которые, быть может, провели еще больше времени, составляя их) опубликовали их без всякого обоснования! Тогда как одно маленькое высказывание в нужном месте позволило бы быстро понять суть алгоритма.

Что касается программирования, были так же сделаны попытки для упрощения передачи понимания программ от разработчика пользователю. Кнут[101], который является для нас объектом постоянных ссылок, работал в этом направлении, создавая свою систему документированного программирования **WEB**. Это направление, даже если оно не решает всех задач программирования — в частности, оно не дает удовлетворительного решения проблемы архитектуры программ — очень помогло нам на этапах программирования, которые мы уже прошли. Конечно, применение методологии программирования требует заметного напряжения, а время получения результата больше, чем обычно, что идет в разрез с философией программистов, для которых верхом изящества является возможность составить и выполнить программу одним нажатием кнопки. Но уже выбор языка Ада четко отделяет нас от этих программистов.

В дальнейшем на странице этой книги читатель найдёт сравнительно мало Ада-программ по сравнению с насыщенной ими первой главой. Причина, в основном, в недостатке места, которое было бы необходимо для представления этих программ. Однако мы реализовали большинство алгоритмов, которые представляем здесь, скорее для того, чтобы *подтвердить* наши утверждения (что не составляет доказательства как такового), нежели для нахождения конкретных примеров или контрпримеров; и все эти алгоритмы представлены в форме, допускающей прямую реализацию. Сверх того, мы строго привержены принципу документирования, и в дальнейшем не найдётся - кроме тривиальных случаев - алгоритма, не сопровождаемого небольшим инвариантом, что и делает жизнь приятной.

Упражнения

1. Вычисление целого квадратичного корня

Если упражнение показывает, что классический метод Ньютона для вычисления квадратичного корня из действительных чисел равно применим в случае вычисления для целых чисел. Пусть a — целое строго положительное число; определим последовательность натуральных чисел (a_n) следующим образом:

$$a_0^2 \geq a \text{ и } a_{n+1} = \left\lfloor \frac{a_n + \lfloor a/a_n \rfloor}{2} \right\rfloor, \forall n \in \mathbb{N}.$$

Можно считать, что если a и b — целые положительные числа, то $\lfloor a/b \rfloor$ есть частное от деления a на b с помощью алгоритма Евклида.

а.Прежде всего, нужно точно сформулировать проблему. Что означает " x есть целый квадратный корень из a "? Для $x \in \mathbb{N}$ доказать, что $\lfloor (x + \lfloor a/x \rfloor) / 2 \rfloor = x$. Для какой формы вычисления наиболее эффективны?

б.Написать программу, которая вычисляет последовательность (a_n) , начиная с некоторого значения a , вводимого с клавиатуры, что позволяет наблюдать формирование последовательности и высказать правдоподобные предположения об условиях её сходимости. Затем можно написать программу, позволяющую проверить предположение на примере целых чисел, гораздо больших, чем в предварительных рассуждениях. Продолжение упражнения состоит в доказательстве этого высказывания.

с.После изучения числовой функции $F(x) = (x + a/x) / 2$, доказать, что последовательность (a_n) обладает начальным строго убывающим сегментом, т.е. существует целое p такое, что для всякого $i > p$ имеет место $a_i > a_{i+1}$ и $a_p \leq a_{p+1}$ (указание: показать, что $a_i^2 > a \Rightarrow a_i > a_{i+1}$).

д.Показать, что $(a_p + 1)^2 > a$. Вывести отсюда алгоритм вычисления целого квадратного корня из a .

е.Изучить так же последовательность, определенную с помощью соотношений $a_0^2 > a$ и $a_{n+1} = \left\lfloor \frac{1+a_n+\lfloor a/a_n \rfloor}{2} \right\rfloor$.

2. Целая часть функций

а. Показать, что $\lfloor \sqrt{\lfloor x \rfloor} \rfloor = \lfloor \sqrt{x} \rfloor$. В более общем случае показать, что если f — возрастающая непрерывная функция и если $f(x) \in N \Rightarrow x \in N$, то $\lfloor f(\lfloor x \rfloor) \rfloor = \lfloor f(x) \rfloor$

б. Если обозначить через $\lceil x \rceil$ наименьшее целое, превосходящее или равное x (его *наибольшая целая часть*), то при каком условии $\lceil \sqrt{\lceil x \rceil} \rceil = \lceil \sqrt{x} \rceil$.

с. Пусть f — строго возрастающая непрерывная функция, такая что $f(x) \in N \Rightarrow x \in N$. Показать, что для всякого целого x выполняется равенство $\lceil f(x+1) \rceil = \lfloor f(x) \rfloor$.

3. Вычисление целого корня n -й степени

Требуется для $a \in N$ вычислить $\lfloor \sqrt[n]{a} \rfloor$, где n — целое ≥ 2 . Метод Ньютона, примененный к уравнению $f(x) = x^n - a = 0$ на множестве действительных чисел, предполагает рассмотрение действительной функции $F(x) = x - \frac{f(x)}{f'(x)} = \frac{(n-1)x^n + a}{nx^{n-1}}$. Тогда естественно для целых чисел рассмотреть последовательность (x_i) , определённую как:

$$(x_{i+1}) = \left\lfloor \frac{(n-1)x_i^n + a}{nx_i^{n-1}} \right\rfloor \text{ для } i \geq 0 \text{ и } x_0 \in N, x_0^n \geq a.$$

Руководствуясь упражнением 1 по вычислению квадратного корня в целых числах, доказать свойства последовательности $(x_i)_{i \geq 0}$ (в случае необходимости — с помощью программы) и вывести оттуда эффективный алгоритм, позволяющий вычислять $\lfloor \sqrt[n]{a} \rfloor$ (нужно особенно внимательно отнестись к выбору первого члена последовательности, если хотим, чтобы все вычисления оказались возможны).

4. Вычисление десятичных знаков числа $\sqrt{2}$

Это упражнение описывает элементарный метод, позволяющий генерировать один за другим десятичные знаки числа $\sqrt{2}$, который был использован Лалем в 1967 году для вычисления 19600 десятичных знаков у $\sqrt{2}$. С тех пор были сконструированы многочисленные другие методы. Читатель может обратиться к главе 5 из [135], где также присутствует вычисление других известных математических констант таких, как π и e .

Пусть $a_0, a_1, a_2, \dots, a_k, \dots$ — разложение $\sqrt{2}$ по основанию 10 ($\sqrt{2} = 1,41421356\dots$). Положим

$$A_k = (a_0 a_1 \dots a_k)_{10} = \lfloor 10^k \sqrt{2} \rfloor \text{ и } R_k = 2 \cdot 10^{2k} - A_k^2 = \left(10^k \sqrt{2}\right)^2 - A_k^2.$$

Это означает, что A_k есть целое число, образованное $k + 1$ первыми цифрами $\sqrt{2}$, тогда как R_k — ошибка, обнаруживаемая при возведении этих двух величин в квадрат.

Показать, что $a = a_{k+1}$ есть наибольшее целое (заключительное между 0 и 1) такое, что $20aA_k + a^2 \leq 100R_k$, затем показать, что $R_{k+1} = 100R_k - (20a_{k+1}A_k + a^2)$. Отсюда вывести алгоритм, позволяющий получить заданное число десятичных знаков $\sqrt{2}$.

Хорошо видно, что A_k — это неограниченное целое; такого ли R_k ? Каким бы ни был ответ, дать порядок величины интервала, в котором находится R_k .

5. Точный корень из целого числа

Для $n \in \mathbb{N}^*$, обозначим $e(n)$ наибольшее $k \geq 1$ такое, чтобы n было точной копией k -й степени (n всегда точная k -ая только для $k = 1$); например, имеем $e(18) = 1$ и $e(248832) = 5$, ибо $248832 = 12^5$. Это упражнение посвящено алгоритму вычисления пары $(r, e(n))$ такой, что $n = r^{e(n)}$. Этот алгоритм часто используется перед алгоритмами разложения и не может поэтому использовать в своей реализации, по крайней мере, каноническое разложение n .

а. Рассматривая каноническое разложение n , установить несколько свойств функции $e : \mathbb{N}^* \rightarrow \mathbb{N}^*$. Показать, в частности, что $e(n^q) = qe(n)$.

б. Используя в качестве примитивов функции $\lfloor \sqrt[m]{} \rfloor$ ($m \geq 2$) и функцию Next_Prime , дающую простое число, следующее за данным целым, написать алгоритм (сопровождаемый доказательством), реализующий вычисление пары $(r, e(n))$ для данного целого n . Каков диапазон целых n , поддающихся обработке с помощью набора простых чисел $< 10^3$?

6. Порождение простых чисел

В этом упражнении устанавливаются алгоритмы, позволяющие последовательно находить нечетные простые числа, начиная с 3, которые будем обозначать $p_1, p_2, p_3 \dots$ (в отличие от общепринятых обозначений, когда $p_1 = 2$).

а. Написать алгоритм, который вычисляет p_1, p_2, \dots, p_n для фиксированного n . Метод: определив все простые числа до p_k , выбирают в качестве кандидата p_{k+2} и пробуют делить его на уже вычисленные p_i . Обязательно ли проверять делимость на все уже вычисленные простые числа?

б. Прежде чем читать формулировку этого вопроса, внимательно рассмотреть доказательство корректности (обеспечиваемое решением) алгоритма из предыдущего вопроса. В одном из циклов этого алгоритма для всякого нечетного q осуществляется проверка $p_i^2 > q$, что сводится к многочисленным бесполезным вычислениям квадратов. Можно без использования таблицы квадратов сократить этот счет, сохраняя две переменные, которые обозначают, соответственно, номер i наименьшего простого p_i , квадрат которого превосходит испытываемые числа, и квадрат этого простого. Это возможно благодаря постулату Бертрана, который утверждает, что $p_{i+1} < 2p_i$. Построить новый алгоритм, реализующий эту оптимизацию.

7. Правильный пятиугольник для всех (без Ферма и Гаусса)

Показать, что $\cos \frac{\pi}{5} = \frac{\phi}{2}$, где ϕ есть число золотого сечения. Вывести отсюда построение правильного пятиугольника с помощью циркуля и линейки.

8. "Двоичное" деление нацело

Цель этого упражнения — вычисление частного и остатка от деления алгоритмом Евклида целого a на целое b , используя только сложение, вычитание, умножение и деление на 2.

а. Выразить частное и остаток от евклидова деления a на b как функцию частного и остатка от евклидова деления a на $2b$

б. Написать алгоритм (рекурсивный) евклидова деления двух целых чисел, основанный на предыдущем свойстве. Этот алгоритм принимает на входе пару целых чисел и дает на выходе пару целых чисел.

с. Очень вероятно, что предыдущий алгоритм, если он прямо реализует свойства пункта *а*, даёт большую ошибку. Для примера попробуйте деление 31001 на 15 и представьте, что произошло бы, если бы все целые числа, с которыми проводятся вычисления, были заключены между -2^{15} и $-2^{15} - 1$.

д. Если алгоритм пункта *б* не вызвал особых сложностей при работе с предыдущим примером написать итеративный алгоритм евклидова "бинарного" деления.

е. В конце упражнения рассматривается случай, наиболее вероятный для возникновения проблемы. В наиболее вероятном случае, когда возникает проблема, конец упражнения посвящен коррективке этой проблемы. Предполагая, что $b_0 \leq a$, показать, что два следующих алгоритма эквивалентны.

```

_b = b[0]; k = 0;
do
{
    _b *= 2;
    k++;
}
while (_a >= _b);

```

```

_b = b[0]; k = 0;
while (_b <= _a / 2) {
    _b *= 2;
    k++;
}
_b *= 2;
k++;

```

Отсюда вывести итеративный алгоритм евклидова деления двух целых чисел двоичными операциями, не содержащий ошибку, указанную в вопросе с.

9. Построение прямых линий методами DDA

DDA —аббревиатура, обозначающая *Digital Differential Analyser*¹, — имя, данное методам построения геометрических фигур на основе дискретного разрешения (экран или графическое печатающее устройство), использующим только простые арифметические операции и целые числа (в частности, эти методы никогда не применяются к вычислениям над числами с плавающей точкой). В дальнейшем предполагается, что “светящиеся” точки имеют целочисленные координаты. В этом упражнении описаны два алгоритма, позволяющие строить отрезки прямых на экране; предполагается, что прямые проходят через начало координат, имеют положительный наклон и находятся в первом квадранте; уравнение прямой имеет тогда вид $y = xv/u$, где $v > 0$ и $u > 0$ (вертикальные и горизонтальные прямые не составляют проблемы).

a. Во всех этих алгоритмах можно считать u и v взаимно простыми. Почему?

b. Предположим сначала, что $u > v > 0$ (т.е. отрезок, который требуется построить, находится в первом октанте); как следствие, рисуемый отрезок будет сформирован из маленьких горизонтальных отрезков. Разрешаются диагональные и горизонтальные движения точки на экране (чисто вертикальные движения бесполезны, так как угловой коэффициент прямой меньше 1). Написать алгоритм, осуществляющий построение по предыдущим правилам. Указание: если (x, y) — точка дискретной прямой, y должен быть наилучшим приближением величины vx/u .

c. Предположим только, что $u > 0$ и $v > 0$; требуется построить прямую в первом квадранте. Разрешаются только вертикальные и горизонтальные движения точки на экране (не диагональные!). Написать алгоритм, который строит сегмент прямой $[0, (u, v)]$, соблюдая

¹Цифровой дифференциальный анализатор.

правило перемещения и ставя точки как можно ближе к действительной прямой. Указание: поддерживать положение точки, аппроксимирующей прямую, в полосе, расположенной между двумя прямыми, параллельными искомой прямой и проходящими через точки $(-1/2, 1/2)$ и $(1/2, -1/2)$ (см.рис.1)

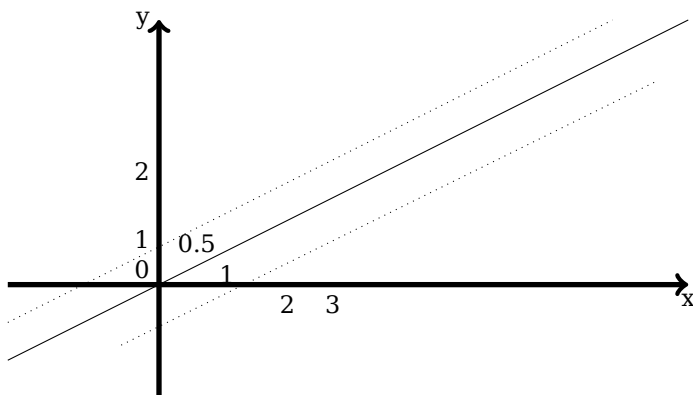


Рис.1. Аппроксимация прямой линии.

10. Построение окружности методами DDA

Речь идёт об ещё одном из методов DDA, который позволяет построить часть окружности в верхнем правом октане(каждой точке, построенной в этом октане, соответствует 1 точка в каждом из 7 других октанов). Ставится задача построить окружность с центром O и радиусом R . Уравнение окружности известно: $x^2 + y^2 = R^2$, и в верхнем правом октане имеем: $y \in [R\sqrt{2}/2, R]$ и $x \in [0, R\sqrt{2}/2]$. Два метода, представленные ниже, строят восьмую часть окружности как кривую $y = f(x)$, а это означает, что на каждом шаге этих алгоритмов отыскиваются точки, расположенные на вертикали и минимизирующие (в смысле, который еще предстоит определить) удаленность по отношению к окружности.

а. Показать, что если (x, y) есть точка на вертикали x , близкая к действительной окружности, то одна из точек $(x+1, y)$ или $(x+1, y-1)$ есть точка с целыми координатами, ближайшая к действительной окружности на вертикали $x+1$. В дальнейшем полагаем $f(x) = \sqrt{R^2 - x^2}$.

б. Первый метод: для целой точки (x, y) , аппроксимирующей точку $(x, f(x))$, полагаем, что $y - 1/2 \leq f(x) < y + 1/2$. Установить алгоритм, который строит восьмую часть окружности.

с. Второй подход: алгоритм Брезенхама (см. [38]) использует для построения дискретной окружности точки с целыми координатами, ближайшие к окружности в следующем смысле: если $P = (x, y)$ — точка дискретной окружности, то из двух точек $P_1 = (x + 1, y)$ и $P_2 = (x + 1, y - 1)$ выбирают ту, которая минимизирует расстояние до окружности.

Это правило могло бы привести к исключительно сложным вычислениям, если бы Брезенхам не доказал (это нужно будет еще доказать) следующее свойство, которое мы принимаем: с предыдущими обозначениями и предложениями минимизировать разность между квадратами длин OP_i и R , что позволит минимизировать разность между этими же длинами. После этого замечания составить алгоритм, который строит окружность Брезенхама во втором октане. Какие замечания можно сделать при сравнении полученного алгоритма с тем, который был составлен в вопросе *б*.

11. Сложность возведения в степень

Цель этого упражнения — показать, что всякий алгоритм вычисления $P = x^n$, использующий только умножение (как алгоритмы, представленные в книге), имеет минимальную сложность $\log_2 n$. В этом исследовании нас интересует только число перемножений; процесс же счета может быть схематично представлен следующим образом:

$$\begin{aligned} P_1 &= y_1 z_1, \text{ где } y_1 z_1 \in \{1, x\}, \\ P_2 &= y_2 z_2, \text{ где } y_2 z_2 \in \{1, x, P_1\}, \\ &\vdots \\ P_t &= y_t z_t, \text{ где } y_t z_t \in \{1, x, P_1, P_2, \dots, P_{t-1}\}; \end{aligned}$$

здесь t есть сложность алгоритма.

12. Последовательность Фибоначчи и золотое сечение

Числа Фибоначчи определены с помощью соотношений: $F_0 = 0$, $F_1 = 1$ и $F_n = F_{n-1} + F_{n-2}$. Они тесно связаны с числом золотого сечения $\phi \approx 1,618$, которое является положительным корнем уравнения $x^2 - x - 1 = 0$; второй корень этого уравнения традиционно обозначается ϕ .

а. Установить соотношения, возможно проще выражающее F_n как функцию от ϕ и $\hat{\phi}$. Указание: установить свойства порождающего ряда $F = \sum_{n \geq 0} F_n \lambda^n$ (или, иначе, производящей функции).

б. Представить $\phi^n + \hat{\phi}^n$ как функцию от чисел Фибоначчи.

с. Записать ϕ^n как функцию от чисел Фибоначчи наиболее простым возможным способом.

13. Соотношения в последовательности Фибоначчи

а. Рассмотрим матрицу $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$. Вычислить A^n . Отсюда вывести соотношения

$$F_{n+1}F_{n-1} - F_n^2 = (-1)^n F_{n+m} = F_n F_{m+1} + F_{n-1} F_m.$$

б. Вычислить $f_n = \sum_{k=0}^n F_k F_{n-k}$. Указание: каков общий член произведения двух формальных рядов?

14. Линейные рекуррентные последовательности k -го порядка

а. Исследовать последовательность целых чисел (x_n) , определяемую соотношением $x_{n+2} = x_{n+1} + x_n$, где первые члены x_0 и x_1 — произвольные фиксированные числа. Какую связь она имеет с последовательностью Фибоначчи?

б. Фиксируется k чисел a_0, a_1, \dots, a_{k-1} и рассматриваются все линейные рекуррентные последовательности k -го порядка, т.е. определяемые соотношениями вида $x_{n+k} = a_0 x_{n+k-1} + \dots + a_{k-1} x_n$. Какая связь существует между этими последовательностями и матрицей Фробениуса

$$A = \begin{pmatrix} a_0 & a_1 & \dots & a_{k-2} & a_{k-1} \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & & 1 & 0 \end{pmatrix} ?$$

Указание: найти базис пространства этих последовательностей.

15. Дихотомия по Горнеру

Классический алгоритм дихотомического возведения в степень основан на том факте, что если $n = n_k 2^k + n_{k-1} 2^{k-1} + \dots + n_1 2^1 + n_0 2^0$, то

$x^n = x^{n_k 2^k} x^{n_{k-1} 2^{k-1}} \dots x^{n_1 2^1} x^{n_0 2^0}$; алгоритм действует, вычисляя x^{2^i} и перемножая те из них, которые соответствуют ненулевым цифрам n_i .

Существует другой способ использования разложения n по основанию 2, который состоит в применении для вычисления n метода Горнера, как многочлена, и использовании этого разложения для вычисления x^n . Например, для x^{11} традиционным дихотомическим способом, берут $n = 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$ и вычисляют

$x, x^2, x^3, x^4, x^8, x^{11}$ возведением в квадрат и перемножением,

тогда как при дихотомии «по Горнеру» записывают $n = (((1) \times 2 + 0) \times 2 + 1) \times 2 + 1$ и вычисляют

$x, x^2, x^4, x^5, x^{10}, x^{11}$ возведением в квадрат и перемножением.

Упражнение посвящено созданию этого алгоритма возведения в степень «по Горнеру».

а. Рассмотрим многочлен $P = \sum_{i=0}^k a_i X^i$ и обозначим $P_i = \sum_{j=1}^k a_j X^{j-i}$. Каковы P_0 и P_k ? Каково соотношение между P_i и P_{i-1} , определяющее традиционный метод Горнера, используемый для вычисления многочлена в некоторой точке? Вывести отсюда алгоритм вычисления многочлена P в некоторой точке и указать число умножений и сложений, осуществляемых алгоритмом.

б. Рассматривая запись величины n по основанию 2 как вычисление некоторого многочлена P в точке 2, вывести отсюда алгоритм вычисления x^n , точно следуя процессу вычисления «по Горнеру» этого многочлена. Для этого следует предполагать, что известны цифры числа n по основанию 2. Полученный алгоритм не должен быть, очевидно, рекурсивным.

с. Как можно действовать в предыдущем алгоритме, не зная бинарного разложения n , но зная, что $2^k \leq n < 2^{k+1}$ (тут есть тонкость!).

д. Оценить сложность этого алгоритма количеством умножений, предполагая затраты на одно умножение постоянными.

16. Вычисление чисел Фибоначчи

а. Написать эффективный алгоритм вычисления чисел Фибоначчи. Напомним, что числа Фибоначчи определяются последовательностью $F_0 = 0, F_1 = 1$ и $F_{n+2} = F_{n+1} + F_n$.

б. Используя соотношение между последовательностью Фибоначчи $(F_n)_{n \geq 0}$ и числом золотого сечения ϕ или матрицей $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$, рассмотренные в упражнении 12, написать эффективный алгоритм вычисления n -го числа Фибоначчи.

с. Прочитав решения предыдущего пункта и используя упражнение 15, придумать другой алгоритм вычисления F_n , основанный на тех же соотношениях. Является он более эффективным, чем предыдущий?

17. ϵ -лексикографический порядок и законопеременный лексикографический порядок

Для заданного отношения порядка \leq условимся использовать следующие обозначения:

$$a \leq^1 b \equiv^{def} a \leq^b \text{ и } a \leq^{-1} b \equiv^{def} b \leq^a$$

Пусть E и F - два упорядоченных множества и ϵ - отображение E в множество $\{-1, 1\}$. Рассматривается отношение порядка, определенное на $E \times F$ посредством

$$(x, y) \leq (x', y') \Leftrightarrow^{def} x < x' \text{ или } x = x' \text{ и } y \leq^{\epsilon(x)} y'.$$

а. Проверить, что это отношение - действительно порядок; оно называется ϵ -лексикографическим порядком. Что можно сказать, если отношение порядка, заданные на E и F , суть отношения линейного порядка? Какой порядок получим на $E \times F$, если функция ϵ тождественно равна 1?

б. Для $x \in E, x \times F$ есть подмножество множества $E \times F$, которое можно рассматривать как "вертикальную плоскость". Сравнить порядок в F и порядок на $x \times F$ (индуцированный на $x \times F$ порядком на $E \times F$)

с. Каждому линейно упорядоченному множеству $E, E = \{x_0 < x_1 < x_2 < \dots\}$, по определению поставим в соответствие функцию ϵ , заданную как $\epsilon(x_i) = (-1)^i$ и называемую *сигнатурой* E . Для данных двух конечных, линейноупорядоченных множеств законочередующимся лексикографическим порядком называется ϵ -лексикографический порядок на $E \times F$, где ϵ есть сигнатура. Каков первый элемент в $E \times F$? Последний? Написать функцию следования в $E \times F$. Выразить сигнатуру $E \times F$ как функцию сигнатур E и F

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 2 & 9 & 4 & 3 & 7 & 6 & 5 & 1 \end{pmatrix}.$$

И обратно, дать перестановки, таблицы инверсий которых следующие:

$$(0 \ 1 \ 1 \ 3 \ 2 \ 4 \ 6), (0 \ 1 \ 1 \ 3 \ 3 \ 4 \ 6 \ 7 \ 5).$$

д. Показать, что массив инерсий a некоторой перестановки σ удовлетворяет условиям: $0 \leq a_k < k = 1, 2, \dots, n$. Каков массив инверсий единственной возрастающей перестановки интервала $[1, n]$? Тот же вопрос для единственной убывающей перестановки.

е. Написать алгоритм, который некоторой биекции α интервала $[1, n]$ ставит в соответствие ее массив инверсий a .

ф. Пусть дана последовательность a целых чисел n , удовлетворяющих условиям: $0 \leq a_k < k = 1, 2, \dots, n$; показать, что существует одна и только одна перестановка σ интервала $[1, n]$, таблица инверсий которого и есть a . Ответ на этот вопрос является не только свойством существования, но и алгоритмом вычисления, позволяющим получить α , исходя из ее таблицы инверсий. Дать элементы обоснования этого алгоритма.

Проверить алгоритм на следующих примерах таблиц инверсий:

$$(0, 0, 2, 1, 3, 4, 6), (0, 1, 0, 3, 4, 2, 1), (0, 0, 0, 2, 4, 0, 5),$$

$$(0, 0, 0, 0, 0, 0, 0) \text{ и } (0, 1, 2, 3, 4, 5, 6).$$

24. Перебор перестановок транспозициями $(i, i+1)$

а. Показать, что можно расположить перестановки интервала $[1, n]$ в последовательность $\sigma_1, \sigma_2, \dots, \sigma_{n!}$ такую, что перестановка (σ_{i+1}) получается из перестановки (σ_i) транспозицией образов двух последовательных чисел. Это упорядочение приводит также к тому, что первая из этих перестановок получается из последней применением в точности того же правила - σ_1 есть результат композиции транспозиции $(i, i+1)$ и перестановки $\sigma_{n!}$.

б. Рассмотрим множество E таблиц инверсий интервала $[1, n]$, которое наделяется знакочередующимся лексикографическим порядком. Пусть $a, b \in E$ и α, β - соответствующие перестановки интервала $[1, n]$; показать, что если b является последующим элементом для a в E , то перестановка β получается из перемтановки α транспозицией двух последовательных элементов.

с. Вывести из предыдущего вопроса алгоритм, реализующий вопрос **а**: этот алгоритм должен быть способен по некоторой данной перестановке вычислить транспозицию, которую необходимо совершить, чтобы получить следующую перестановку.

25. Принцип включения-исключения или формула решета

Это упражнение посвящено формуле перечисления, которая обобщает формулу для мощности объединения двух множеств $|A \cup B| = |A| + |B| - |A \cap B|$. Всюду в упражнении X обозначает конечное множество, а $(X_i)_{i \in I}$ - семейство подмножеств X , индексированное конечным множеством индексов I ; напомним что $\bigcup_{i \in \emptyset} X_i = \emptyset$, $X_i = X$. Если $Y \subset X$, то через \bar{Y} обозначается дополнение Y в X и через $|Y|$ - число элементов Y .

а. Доказать формулы:

$$\left| \bigcup_{i \in I} X_i \right| = \sum_{\emptyset \neq J \subset I} (-1)^{|J|+1} |\bigcap_{i \in J} X_i| \text{ и двойственную}$$

$$|\bigcap_{i \in J} X_i| = \sum_{\emptyset \neq J \subset I} (-1)^{|J|+1} \left| \bigcup_{i \in I} X_i \right|.$$

Отсюда вывести формулу Сильвестера (называемую также формулой решета):

$$\left| \bigcap_{i \in I} \bar{X}_i \right| = \sum_{J \subset I} (-1)^{|J|} |\bigcap_{i \in J} X_i|$$

термин "решето" происходит из того факта, что для перечисления множества элементов, которые не принадлежат ни одному из X_i , достаточно перечислить те, которые принадлежат X_i , затем те, которые принадлежат $X_i \cap X_j$, и т.д.

б. Доказать аналогичные формулы, когда кардинальная функция $P(X) \rightarrow \mathbb{N}$ заменяется весовой функцией p со значениями в некоторой абелевой группе, эта функция p определена на X и продолжена на множество $P(X)$ подмножества X в соответствии с равенством $p(Y) = \sum_{y \in Y} p(y)$

с. Использовать формулу Сильвестра для доказательства, что число σ_n перестановок из n элементов без фиксированной точки дается формулой:

$$\sigma_n = n! \left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots + (-1)^n \frac{1}{n!} \right)$$

d. Для $n \in N$ обозначим через $\varphi(n)$ число целых чисел из интервала $[1, n]$, взаимно простых с n : $\varphi(n)$ называется функцией Эйлера и будет изучена более полно в главе IV; здесь предлагается метод вычисления $\varphi(n)$, использующий формулу решета. Если $p_1^{\alpha_1} \dots p_k^{\alpha_k}$ — разложение на простые множители числа n , доказать, что:

$$\varphi(n) = n \left(1 - \frac{1}{p_1}\right) \dots \left(1 - \frac{1}{p_k}\right).$$

e. Показать, что формула Райзера для вычисления перманента (см. упражнение 21) выводится из формулы решета.

26. Произведение многочленов, заданных массивами

Условимся представлять многочлены массивами, индексированными, начиная с 0, в которых элемент с индексом i означает коэффициент одночлена степени i :

type *Polynome* **is** *array* (*Natural range* <>) **of** *Ring_Element*;

a. Исходя из умножения и сложения элементов базового кольца, реализовать функцию умножения двух таких многочленов.

b. Предполагая, что умножение и сложение элементов базового кольца требует постоянного времени счета (какими бы ни были комбинируемые элементы), дать достаточно точную оценку времени перемножения двух многочленов в зависимости от их степеней.

27. Возведение в степень многочленов, заданных массивами

a. С предположениями предыдущего упражнения — время умножения и сложения постоянно в базовом кольце — оценить сложность алгоритма возведения в степень через последовательные перемножения.

b. Сохраняя те же предположения, определить время вычисления через алгоритм дихотомического возведения в степень: сначала, если n является степенью 2, $n = 2^l$, затем, если n имеет вид $2^l - 1$. Какой можно сделать вывод?

28. Небольшие оптимизации для произведений многочленов

В принципе вычисление произведения двух многочленов степеней n и m соответственно требует $(n+1)(m+1)$ элементарных перемножений.

a. Пользуясь соображением симметрии, написать алгоритм, позволяющий возвести многочлен в квадрат, осуществляя приблизительно половину элементарных перемножений, которые кажутся необходимыми.

б. Найти способ, позволяющий умножать два многочлена первой степени с помощью 3 элементарных перемножений вместо 4 (конечно, при этом число элементарных сложений возрастает). Рекурсивно применить этот способ для получения эффективного алгоритма перемножения двух многочленов, чья степень имеет вид $2^l - 1$. Выразить сложность этого алгоритма через число элементарных умножений и сложений.

29. Высота произведения двух многочленов

Цель этого упражнения — установить некоторые соотношения между высотой (числом ненулевых одночленов) произведения двух многочленов и высотой исходных многочленов. Запись $\#P$ означает высоту многочлена.

а. Если даны два многочлена P и Q с высотой p и q соответственно, какова максимальная высота произведения P на Q ? Существуют ли многочлены, позволяющие достичь этой высоты?

Ответ, полученный на этот вопрос, не удовлетворителен: действительно, оба многочлена, которые позволяют достичь теоретической границы высоты, очень специфичны, и мы вправе спросить, не является ли эта граница исключением и не является ли высота произведения двух многочленов в среднем гораздо более скромной. Последующие вопросы показывают, что даже если ограничиться степенями одного и того же многочлена, высота результата может расти очень быстро в зависимости от показателя.

б. Пусть P — многочлен высоты d ; показать, что $\#P \leq \left(\frac{d+n-1}{n}\right)$.

с. Рассмотрим многочлен $\sum_{i=1}^d X^{F_1+i_n}$, где F_j означает j -тое число Фибоначчи. Показать, что с этим многочленом граница вопроса **б** достигается.

30. Представление многочленов списками

Когда нужно обратиться к разреженным многочленам (т.е. имеющим много нулевых коэффициентов) или же к многочленам, высоту которых трудно ограничить, их обычно представляют не в виде массивов, а, скорее, в форме динамических структур типа списка. Алгоритмы, действующие с многочленами, представленными этим способом, очень специфичны и требуют тщательного изучения, если мы не хотим удешевить их сложность. Чтобы упростить терминологию, ниже всюду будем называть несобственно разреженным многочленом всякий

многочлен, представление которого является цепным списком, независимо от того, действительно ли этот многочлен разрежен или нет (в целом не меняя представления многочлена по ходу вычислений, даже если окажется, что оно не самое оптимальное).

Описать на языке Ада структуру данных, допускающих представление разреженных многочленов списками. Какие исходные данные априори необходимы в списках, чтобы выполнять действия над этими многочленами? Ввести эти исходные данные и обсудить выгоды и неудобства выбранных структур.

31. Сложение многочленов, представленных списками

Написать алгоритм сложения двух разреженных многочленов, используя только исходные данные, описанные в предыдущем упражнении. Оценить сложность этого алгоритма.

32. Умножение многочленов, представленных списками

Написать алгоритм умножения двух многочленов, представленных списками. Рассмотреть для этого сложность различных методов в зависимости от сложности алгоритма сложения, введенного в упражнении 31.

33. Действия с формальными рядами

а. Принцип вычисления квадрата многочлена, показанный в упражнении 28, может также применяться к вычислению квадрата формального ряда. В соответствии с этим принципом написать алгоритм, который допускает на входе последовательность коэффициентов исходного формального ряда и который вычисляет по мере этого чтения коэффициенты квадрата этого же ряда.

Можно заметить, что этот способ вычисления (волной) является фундаментальным для тех, кто хочет действовать с формальными рядами, т.е. по сути, с бесконечными объектами.

б. Рассмотрим формальный ряд $A = \sum a_i X^i$, n -ую степень которого хотим вычислить. Дифференцируя A^n , найти формулу, связывающую A и A^n . Вывести из нее алгоритм, позволяющий вычислить коэффициент одночлена степени i из A^n , зная i первых коэффициентов ряда A и $i-1$ первых коэффициентов A^n .

с. Применить оба предыдущих алгоритма к случаю многочленов. Какие улучшения это может принести? Какую получаем сложность?

34. Определение нулей многочлена по модулю p^α

Целью этого упражнения является разработка алгоритма, позволяющего вычислить для простого числа p нули многочлена в $\mathbb{Z}/p^n\mathbb{Z}$, если они известны в $\mathbb{Z}/p\mathbb{Z}$.

а. Этот вопрос слегка предвосхищает главу II, но разработанный алгоритм необходим для решения следующих вопросов. Указать алгоритм решения уравнения $ax = b$ по модулю n в общем случае.

б. Доказать, что произведение n целых последовательных чисел является кратным $n!$.

с. Показать, что если a и t — целые числа, P — многочлен с целыми коэффициентами и $\alpha \in \mathbb{N}$, то $P(a + tp^\alpha) \equiv P(a) + tp^\alpha P'(a) \pmod{p^{\alpha+1}}$. Отсюда вывести, что если a есть корень P по модулю p^α , $P(a + tp^\alpha) \equiv 0 \pmod{p^{\alpha+1}}$ тогда и только тогда, когда $P(a)/p^\alpha \equiv 0 \pmod{p}$. Построить алгоритм, который по известным корням P по модулю p^α определяет их по модулю $p^{\alpha+1}$.

д. Пусть a является нулем P по модулю p таким, что $P'(a) \not\equiv 0 \pmod{p}$. Вывести из предыдущего вопроса, что существует один и только один нуль P по модулю p^i , который был бы сравним с a по модулю p^i . Если a^i обозначает этот нуль, показать, что $a_{i+1} = a_i - P(a_i)P'(a)^{-1} \pmod{p^{i+1}}$, формулу, в которой $P'(a)^{-1}$ обозначает обратный к многочлену $P(a)$ по модулю p .

е. Можно усилить свойство, доказанное в **б**, и показать, что $P(a + tp^\alpha) \equiv P(a) + tp^\alpha P'(a) \pmod{p^{\alpha+1}}$. С учетом этого свойства, каким становится результат, заявленный по поводу определения корней? Дать новый алгоритм, способный вычислять корни по модулю p^{2n} многочлена, зная его корни по модулю p^n .

ф. Какой синтез двух алгоритмов может быть сделан?

35. Циклическая перестановка элементов массива

В этом упражнении n представляет собой целое, строго положительное число, которое будет предполагаться заранее определенным.

а. Вспомнить алгоритм, который по заданному массиву, индексированному целыми числами от 0 до $n - 1$, осуществляет круговую перестановку элементов массива по одному шагу вправо (т.е. элемент с индексом 0 оказывается в ячейке 1, элемент с индексом 1 в ячейке 2... и элемент с индексом $n - 1$ в ячейке 0).

б. Рассматривается аддитивная группа $G = \mathbb{Z}/n\mathbb{Z}$ целых по модулю n и a — элемент из G . Каков порядок подгруппы H группы G , порожденной элементом a ? Ответ зависит от n и от a .

Показать, что целый полуоткрытый интервал $[0, (n,))$ является системой представителей G/H. Отсюда вывести, что если x и y — два различных элемента этого последнего интервала, то множества $x + ka | k \in \mathbb{Z}$ и $y + ka | k \in \mathbb{Z}$ не пересекаются.

с. Вывести отсюда алгоритм, обобщающий алгоритм вопроса а следующим образом: он осуществляет циклическую перестановку таблицы на p шагов вправо, где p — целое число, заданное в алгоритме.

Очевидно, требуемое решение не состоит в применении p раз алгоритма из вопроса **а**.

Сколько присваиваний элементов массива осуществляется этим алгоритмом? Сравнить это число с числом присваиваний, которое получилось бы при повторении p раз алгоритма из вопроса **а**.

д. Написать алгоритм, который в массиве меняет местами две группы членов, расположенных соответственно в начале массива и его конце и ограниченных индексом k .

е. Написать Ада-программу, реализую алгоритм вопроса с и проводя (в случае необходимости) все этапы счета. Программа должна быть в высшей степени модульной (разделить чтение параметров, фиксирование промежуточных результатов и процедуру циклической перестановки). Можно будет протестировать эту программу, используя массивы целых чисел, автоматически заполненных целыми числами от 0 до $n - 1$; можно предусмотреть отдельную процедуру для такой инициализации.

36. Элементарные операции в арифметике повышенной точности

Располагая фиксированным раз и навсегда основанием для вычислений, которое обозначается b (например, $b = 2$ или $b = 10$, или же степенью одного из этих чисел), рассмотрим положительные числа u , заданные их представлением по основанию b :

$$u = (u_m u_{m-1} \dots u_1 u_0)_b = u_m b^m + u_{m-1} b^{m-1} + \dots + u_1 b + u_0$$

Для всех искомых алгоритмов имеем в своем распоряжении только элементарные арифметические операции над цифрами по основанию b .

а. Написать алгоритм, позволяющий складывать два целых $u = (u_m \dots u_0)_b$ и $v = (v_m \dots v_0)_b$.

б. Написать алгоритм, позволяющий вычислить $u - v$, если и предполагается большим или равным v .

Если больше не предполагать, что $u \geq v$, модифицировать алгоритм таким образом, чтобы он мог сразу вычислять $(u-v) \bmod b^{m+1}$ и реализовывать проверку ($u < v$).

с. Написать алгоритм, позволяющий умножать число $u = (u_m \dots u_0)_b$ веса $\leq m$ на цифру v , т.е. $v < b$, имея результатом число $w = (w_{m+1} \dots w_0)_b$ веса $\leq m+1$

д. Написать алгоритм, позволяющий вычислять остаток и частное от деления числа $u = (u_m \dots u_0)_b$ веса $\leq m$ на **цифру** v .

37. Деление в арифметике повышенной точности

Показать, что деление каких либо двух чисел всегда сводится *обычным делением* вручную к следующей общей ситуации:

- делимое имеет цифру большую, чем делитель,
- отношение делимого к делителю является цифрой, т.е. строго меньше, чем основание.

Более строго, зададимся двумя числами $u \equiv (u_n, \dots, u_1, u_0)_b$, $v = (v_m, \dots, v_1, v_0)_b$ и предположим, что умеем вычислять их частное $[u/v]$, если они удовлетворяют следующим условиям:

$$u = (u_{m+1}, u_m, \dots, u_1, u_0)_b, \quad v = (v_m, \dots, v_1, v_0)_b, \quad u/v < b.$$

(Это означает, что $[u/v]$ есть цифра). Написать алгоритм, позволяющий определять цифры по основанию b частного $q = [u/v]$ и остатка $r = u \bmod v$ при делении u на v .

38. Вычисление частного методом проб и ошибок

Чтобы закончить разработку алгоритма деления каких-либо двух чисел, остается только установить способ деления двух чисел u , v , удовлетворяющих условиям (5) упражнения 37.

Идея этого метода (см. Кнут [99]) состоит в действии с помощью метода проб и ошибок: используя критерии, изучаемые ниже, определяют кандидата в частные $\hat{q} \geq q = [u/v]$, затем вычисляют $u - \hat{q}v$; эта величина должна удовлетворять условию: $u - \hat{q}v < v$. Если это не так, меняют \hat{q} (пробуя $\hat{q} - 1$, затем $\hat{q} - 2$, ...). Теперь вопросы: как определить кандидата в частное наиболее экономным возможным способом и как его менять? Для этого примененный метод использует две наибольшие цифры чисел u и v .

а. Рассмотрим как (первоначальную) оценку величины $q = [u/v]$ частное от деления двух первых цифр числа и на первую цифру числа v , точнее:

$$\hat{q} = \min(\lfloor \frac{u_{m+1}b + u_m}{u_m} \rfloor, b - 1).$$

Как в этой формуле определить простым способом наименьшее из двух чисел? Показать, что $q \leq \hat{q} \leq q + (b + v_m - 1)/(v_m + 1)$. Что нужно предполагать при реализации, использующей эту оценку?

б. В этом вопросе предполагается, что v имеет более одной цифры, т.е. $m \geq 1$. Для некоторого данного целого числа $\hat{q} \geq 0$ положим $\hat{r} = u_{m+1}b + u_m - \hat{q}v_m$, что можно рассматривать как остаток от деления $(u_{m+1}u_m)_b$ на v_m . Показать, что

$$(i) \quad \hat{q}v_{m-1} > b\hat{r} + u_{m-1} \implies q \leq \hat{q} - 1,$$

$$(ii) \quad \hat{q}v_{m-1} \leq b\hat{r} + u_{m-1} \implies q \geq \hat{q} - 1.$$

Вывести из этого и предыдущего вопросов быстрый корректирующий тест, дающий оценку \hat{q} величины q , удовлетворяющую соотношению $q = \hat{q}$ или же $q = \hat{q} - 1$; будем говорить тогда о **скорректированной** оценке. Какова выгода от этого теста по сравнению с тестом $u - \hat{q}v < v$? Дать примеры, в которых скорректированная оценка отлична от точного частного.

с. Написать алгоритм деления u на v , используя оценку из вопроса **а** и быстрый тест корректировки из вопроса **б**.

39. Деление: операция нормализации

Сохраняем обозначения упражнений 37 и 38; в вопросах **а** и **б** располагаем двумя числами u, v , удовлетворяющими предположениям (5).

а. Вывести из упражнения 38, вопрос **а**, что предположение $v_m \geq \lfloor b/2 \rfloor - 1$ влечет за собой $0 \leq \hat{q} - q \leq 2$.

б. Предположим, что v имеет более одной цифры ($m \geq 1$), $v_m \geq \lfloor b/2 \rfloor$ и что скорректированная оценка \hat{q} отлична от $q = [u/r]$. Показать, что остаток $r = u - qv$ от деления u на v удовлетворяет условию $r/v > (1 - 2/b)$ (можно интерпретировать этот результат, говоря, что вероятность для \hat{q} быть отличным от q меньше $2/b$).

с. Теперь рассмотрим два произвольных целых $u = (u_n \dots u_0)_b$ и $v = (v_m \dots v_0)_b$. Доказать, что если положить $d = \lfloor b/(v_m + 1) \rfloor$, $u' = du$, $v' = dv$, то $v'_m \geq \lfloor b/2 \rfloor$. Какова связь между делением u на v и делением u' на v' ? Каков вес u' и v' ? Написать общий алгоритм деления u на v .

40. Самовоспроизводящаяся программа

Во всяком достаточно мощном языке программирования (фактически, который имеет мощность машины Тьюринга) возможно написать программу, единственной целью которой было бы воспроизведение своего текста. Написать такую программу на языке Ада. Указание: прочитать главу о самовоспроизведении у Хофпггадтера [88] или статью Томсона [170].

Решения упражнений

1. Вычисление целого квадратного корня

Подлинная трудность этого упражнения кроется в возможном смешении целого частного от евклидова деления двух целых чисел и рационального числа, определяемого делением этих чисел: $a/b = [a/b] + (a \bmod b)/b$. Результат упражнения состоит в том, что последовательность (a_n) всегда дает квадратный корень целого числа a , а последующее исследование имеет целью показать, каковы условия сходимости этой последовательности.

а. Целый квадратный корень из a есть целое x такое, что $x^2 \leq a < (x+1)^2$. Учитывая неравенство $[y/2] \geq y/2 - 1/2$, справедливое для целого y , можно написать:

$$\left\lfloor \frac{x + \lfloor a/x \rfloor}{2} \right\rfloor \geq \frac{x + \lfloor a/x \rfloor}{2} - \frac{1}{2} > \frac{x + a/x - 1}{2} - \frac{1}{2} = \frac{x + a/x}{2} - 1,$$

и отсюда выводится доказываемый результат. Конечно, начальная форма более эффективна для вычисления.

с. Функция F , соответствующая методу Ньютона, является строго возрастающей в интервале $[\sqrt{a}, +\infty[$, удовлетворяет условию $x > F(x)$ на открытом интервале и $F(\sqrt{a}) = \sqrt{a}$. Предполагаем впредь, что число a отлично от 0 и 1. Если это не так, легко видеть, что последовательность (a_n) постоянна и тогда ее сходимость очевидна. Исследование, проведенное с помощью программ, показывает, что последовательность (a_n) является строго убывающей, поскольку к квадратному корню приближаемся сверху. Если предположить $a_n^2 > a$, имеем $a_n > \lfloor \sqrt{a} \rfloor$, что влечет $a_n > F(a_n) \geq \lfloor F(a_n) \rfloor = a_{n+1}$. Последовательность целых положительных чисел (a_n) является, следовательно, строго убывающей, пока $a_n^2 > a$; но это не может длиться бесконечно: существует целое p такое, что $a_{p-1}^2 > a$ и $a_p^2 \leq a$. Тогда имеем $a_{p+1} \geq a_p$ - свойство, которое мы будем использовать для построения одного из алгоритмов. Упражнение показывает, и это легко можно доказать, что последовательность (a_n) является постоянной или периодической, начиная с номера : интуитивно a_p есть искомый квадратный корень.

$x \leftarrow x_0; x_0^2 \geq a$ while $x \times x > a$ loop $x \leftarrow \lfloor \frac{x + \lfloor a/x \rfloor}{2} \rfloor;$ end loop; return $x;$	$x \leftarrow x_0;$ loop $x' \leftarrow \lfloor \frac{x + \lfloor a/x \rfloor}{2} \rfloor$ exit when $x' \geq x;$ $x \leftarrow x'$ end loop; return $x;$
--	---

d. Покажем, что если $a_{n-1}^2 > a$, то $(a_n + 1)^2 > a$ (свойство, которое, будучи примененным к $n = p$, докажет, что является целым квадратным корнем из a). Имеем $a_n = \lfloor F(a_{n-1}) \rfloor > F(a_{n-1}) - 1$ и, следовательно, $a_n + 1 \geq F(a_{n-1})$. Но по предположению $a_{n-1} > \sqrt{a}$, и, значит, $F(a_{n-1}) > \sqrt{a}$, поскольку функция F строго возрастающая.

Вычисление целого квадратного корня

Ранее было установлено, что $a_p^2 \leq a < (a_p + 1)^2$. Поэтому получаем: последовательность (a_n) обладает строго убывающим начальным сегментом, все члены которого, кроме последнего (с индексом p), строго больше a ; этот последний член является целым квадратным корнем из a . Это приводит нас к указанным выше двум алгоритмам. Выбор первоначальной величины последовательности совершенно произвольный. Во втором алгоритме проверка завершения модифицирована, чтобы избежать в ней ненужных перемножений, правда ценой одного дополнительного деления. Сложность этого алгоритма, конечно, ниже сложности алгоритма в действительных числах, потому что убывание последовательности происходит быстрее (по причине целых частных). При выборе первоначального члена последовательности таким, что $x_0^2 > a$ (если $x_0^2 = a$, то последовательность постоянна), можно доказать следующие свойства:

- последовательность (a_n) строго периодична тогда и только тогда, когда $a + 1$ является полным квадратом; в этом случае период имеет длину 2 и размах колебаний равен 1.
- последовательность стационарна **тогда и только тогда, когда** $a + 1$ не является полным квадратом.

e. Основной результат изучения этой новой последовательности состоит в том, что она стабилизируется на $\lfloor \sqrt{a} \rfloor$ или на $\lfloor \sqrt{a} \rfloor + 1$.

Замечание. Только что изученный алгоритм полностью пригоден для вычисления целого квадратного корня из действительного числа.

2. Целая часть функций

а. Если x — целое число, результат тривиален. Предположим, что $x \notin \mathbb{N}$. Имеем $|x| < \lceil x \rceil$, что влечет $f(|x|) \leq f(\lceil x \rceil)$ и, следовательно, $|f(|x|)| \leq |f(\lceil x \rceil)| < f(\lceil x \rceil)$ (поскольку $\lceil x \rceil$ — не целое, $f(\lceil x \rceil)$ — и подавно). Предположим, что $|f(|x|)| < f(\lceil x \rceil)$, тогда имеем:

$$|f(|x|)| < |f(\lceil x \rceil)| + 1 \leq f(\lceil x \rceil) < f(x),$$

$$\text{и значит } f(|x|) \leq |f(\lceil x \rceil)| + 1 \leq f(\lceil x \rceil) < f(x).$$

Тогда можно применить теорему о промежуточных значениях в интервале $[|x|, \lceil x \rceil]$ к величине $|f(x)|$, и найдем y в этом интервале, удовлетворяющий $f(y) = |f(x)|$. В соответствии с предположениями относительно f имеем $y \in \mathbb{N}$ и, следовательно, $y = \lceil x \rceil$. Снова обращаясь к последней строчке неравенств, получаем $f(y) \leq f(y) + 1 \leq f(\lceil x \rceil)$, что дает нужное противоречие. Следовательно, $|f(|x|)| = |f(\lceil x \rceil)|$.

б. Рассмотрев несколько примеров, быстро замечаем, что обе величины различаются, когда целая часть (меньшая) от x является квадратом. Действительно, легко показать, что $|\sqrt{|x|}| \neq |\sqrt{x}|$ тогда и только тогда, когда $m^2 < x < m^2 + 1$ для целого m . Другими словами, можно утверждать, что $|\sqrt{|x|}| = |\sqrt{x}|$ тогда и только тогда, когда x является целым числом, или же $\sqrt{|x|}$ не является целым.

с. Положим $|f(x)| = \lceil x \rceil$. Имеем $\lceil x \rceil \leq f(x) < \lceil x \rceil + 1$, и поскольку f строго возрастает, она обратима и $f^{-1}(y) \leq x < f^{-1}(y + 1)$. Итак, по предположению относительно f , f^{-1} принимает целые значения от целых, и значит, $f^{-1}(y) < x + 1 \leq f^{-1}(y + 1)$, и применяя f , находим $\lceil x \rceil \leq f(x + 1) < \lceil x \rceil + 1$, что и доказывает требуемое соотношение.

Замечание

Свойство функций f , изученных в вопросах а и с, в действительности эквивалентно тому факту, что функция, обратная (когда она существует) к этим функциям, принимает целые значения для целых аргументов. Например, это случай функций $x \rightarrow \log_{10} x$ или $x \rightarrow \sqrt{x/2} \dots$, что доказывает в итоге, что такие функции весьма распространены.

3. Вычисление целого корня n -й степени

Функция F строго возрастает на интервале $(\sqrt[n]{a}, \infty)$ и удовлетворяет $F(x) > F(\sqrt[n]{a})$ на открытом интервале, а также $F(\sqrt[n]{a}) = \sqrt[n]{a}$. Если $x_i > |\sqrt[n]{a}|$, то $x_i > \sqrt[n]{a}$ (так как $x_i \in \mathbb{N}$) и значит, $x_i > F(x_i) \geq |F(x_i)| = x_{i+1}$. Следовательно,

поскольку $x_i > |^n\sqrt{a}|$, последовательность x_i является строго убывающей. Тогда существует номер ≥ 1 такой, что $x_p \leq |^n\sqrt{a}| < x_{p-1}$.

Теперь покажем, что только предположение $x_i > |^n\sqrt{a}|$ влечет $x_{i+1} + 1 > |^n\sqrt{a}|$. Имеем $x_{i+1} = |F(x_i)| > F(x_i) - 1$, т.е. $x_{i+1} > F(x_i)$. Но $x_i > |^n\sqrt{a}|$ и, значит, $F(x_i) > |^n\sqrt{a}|$, что и дает результат. Применяя его к $i = -1$, получим $x_p \leq |^n\sqrt{a}| < x_{p-1}$; x_p является, следовательно, целой частью искомого корня n -й степени.

Нужно особо позаботиться о величине начального члена x_0 : он должен быть больше, чем $|^n\sqrt{a}|$ и, конечно, можно выбрать $x_0 = a$ (исключая случаи $a = 0$ или $a = 1$), но предпочтительнее из-за члена $x_0 = a$, участвующего в вычислении x_i , взять его как можно меньшим. Можно, например, вычислить целое l такое, что $2^l \geq a$, потом взять $x_0 = 2^{[l/n]}$ (что можно также представить в виде $2^{[(l+n-1)/n]}$, учитывая тот факт, что немногие машины располагают операцией нахождения верхней целой части).

4. Вычисление десятичных знаков числа $\sqrt{2}$

Имеем $A_0 = 1$, $R_0 = 1$ и, конечно, $A_{k+1} = 10A_k + a_{k+1}$; отсюда выводим, что:

$$R_{k+1} = 2 \cdot 10_{k+2}^2 - 100A_k^2 - 20a_{k+1}A_k - a_{k+1}^2 = 100R_k - (20a_{k+1}A_k + a_{k+1}^2).$$

Так как R_{k+1} должно быть наименьшим положительным целым числом, удовлетворяющим данному ему определению, R_k может рассматриваться как *остаток* в смысле «*backward error analysis*», т.е. как сознательно допущенная ошибка.

А. Первоначальный алгоритм	В. Оптимизированный алгоритм
<pre>// A = A_k, R = R_k a ← 0; R ← 100R; while R ≥ 20aA + a² loop a ← a + 1; end loop; R ← R - (20aA + a²); A ← 10A + a; // a = a_{k+1}, R = R_{k+1}, A = A_{k+1}</pre>	<pre>// A = A_k, R = R_k a ← 0; R ← 100R; loop b := 20A + 2a + 1; exit when R < b; R ← R - b; a ← a + 1; end loop; A ← 10A + a; // a = a_{k+1}, R = R_{k+1}, A = A_{k+1};</pre>

Алгоритм 4. Вычисление десятичных знаков

Объясняем: R_k не является ошибкой по отношению к $\sqrt{2}$, но оно позволяет определить число, для которого 10^{-k} является точным квадратным корнем; это число есть $2 - R_k 10^{-2k}$.

Как следствие, a_{k+1} является наибольшей цифрой числа a , такой что $100R_k - (20aA_k + 2)$ положительно. Тогда получаем алгоритм 4-А, прямо используя эти результаты; этот алгоритм вычисляет a_{k+1} , A_{k+1} и R_{k+1} , исходя из a_k , A_k и R_k . Достаточно повторить k раз алгоритм, начиная с $A=1$ и $R=1$, для вычисления k первых десятичных знаков $\sqrt{2}$.

Замечая, что сумма n первых нечетных чисел есть n^2 , можно легко оптимизировать алгоритм (алгоритм 4-В).

Можно записать $10^k \sqrt{2} = A_k + \varepsilon 0 < \varepsilon < 1$ (неравенства строгие, потому что $y/2$ не имеет конечного десятичного разложения), что позволяет оценить R_k :

$$R_k = 2 \cdot 10^{2k} - A_k^2 = 2 \cdot 10^{2k} - (10^k \sqrt{2} - \varepsilon)^2 = \varepsilon(2A_k + \varepsilon) < 2A_k + 1,$$

следовательно, R_k — тоже неограниченное целое число. Предложенный алгоритм легко реализовать на языке формальных вычислений (который содержит неограниченные целые числа), но можно также и на более традиционном языке — операции над большими числами ограничиваются сложением, вычитанием, умножением на 10 (сдвиг на один десятичный знак) и умножением на 2.

5. Точный корень из целого числа

а. Целое число n , разложенное в произведение простых: $n = p_1^{\alpha_1} \dots p_s^{\alpha_s}$ является k -й степенью тогда и только тогда, когда $k \mid \alpha_i$ для $1 \leq i \leq s$ или когда k делит НОД чисел α_i . Из этого выводим, что $e(n) = \text{НОД}(\alpha_i)$ и что n является k -й степенью тогда и только тогда, когда $k \mid e(n)$. Отсюда удобно получить: $e(n^q) = q e(n)$, что позволяет вычислять $e(n^q)$, если известно $e(n)$ (корень r — один и тот же для n и n^q).

б. Идея алгоритма 5 следующая: вычисляем $n_1 = \sqrt[2]{\sqrt[2]{\dots \sqrt[2]{n}}}$; поскольку результат — целый, это позволяет записать $n = n_1^{2e_1}$, что дает $n = n_2^{3e_2}$, откуда $e(n) = 2^{e_1} 3^{e_2} e(n_2)$. и подвергают n_2 такому же преобразованию, $\sqrt[5]{\sqrt[5]{\dots \sqrt[5]{n_2}}}$ и так далее.

Чтобы подтвердить конечность метода, нужно определить для целого r , когда $e(r) = 1$. Ясно, что $e(r) = 1$ тогда и только тогда, когда для всякого простого q $\sqrt[q]{r} \notin \mathbb{N}$; но если $r < 2^q$, всегда имеем $\sqrt[q]{r} \notin \mathbb{N}$. Следовательно, имеем $e(r) = 1$ тогда и только тогда, когда $\sqrt[q]{r} \notin \mathbb{N}$ для всякого простого $q \leq \lfloor \log_2 r \rfloor$.

```

 $e \leftarrow 1; r \leftarrow n; p \leftarrow 2$ 
while  $p \leq \lfloor \log_2 r \rfloor$  loop
   $// r^e = n, p$  — простое,  $\sqrt[q]{r} \notin \mathbb{N}$ ,  $q$  простое  $< q$ 
   $i \leftarrow 0$ 
  loop  $// r^{ep^i} = n$ 
     $r' \leftarrow \sqrt[p]{r}$ 
    exit when  $r'^p \neq r$ 
     $r \leftarrow r'; i \leftarrow i + 1$ 
  end loop
   $e \leftarrow ep^i; p \leftarrow \text{Next\_prime}(p)$ 
end loop
 $// r^e = n, \sqrt[q]{r} \notin \mathbb{N}, q$  простое  $\leq \log_2(r)$ , значит  $e(r) = 1$ 
return  $(r, e)$ 

```

Алгоритм 1.1: Вычисление корня из целого числа

В цикле этого алгоритма изменяются две величины: простое p возрастает и *остаток* r целого n убывает. При выходе из цикла имеем $r = n$, и ни для какого простого $q \leq \lfloor \log_2 r \rfloor$ число r не является корнем q -й степени; как следствие, в соответствии с выводами предыдущего абзаца $e(r) = 1$.

Все числа из интервала $[0, 2^{1009})$ могут быть уменьшены этим алгоритмом с использованием только простых чисел, меньших 1000. Конечно, можно применить алгоритм к числам, расположенным вне этого интервала, но тогда получаемый с помощью алгоритма результат крайне ненадежен при отсутствии дополнительной информации по рассматриваемому числу.

6. Порождение простых чисел

а. Для этого алгоритма используем таблицу целых чисел, которая содержит все уже вычисленные простые числа и которая возникает по мере вычислений. Что касается доказательства корректности этого алгоритма, то можно удовлетвориться рассмотрением того, что происходит в самом внутреннем цикле, который проверяет *Candidate* на простоту.

В этом тесте (алгоритм 6) просматривается последовательность уже вычисленных простых чисел и делается остановка, как только квадрат одного из них превосходит *Candidate*: действительно, в произведении двух нетривиальных множителей один из них меньше квадратного корня из этого числа (обычный критерий остановки решета Эратосфена).

Если обнаруживается простое число, которое делит *Candidate*, то последний объявляется непростым, и цепочка циклов делает переход к проверке следующего нечетного числа. Зато, если *Candidate* преодолевает все простые числа, то он — простое число; в этом случае выходят из цикла *Searching_For_Next_Prime*, как только выполнено условие $p_i^2 > \textit{Candidate}$ в силу предыдущего замечания; найденное простое число помещается в таблицу.

```

 $p_1 \leftarrow 3; \textit{Candidate} \leftarrow 3;$ 
for  $k$  in  $2 \dots n$  loop
  Searching_For_Next_Prime : loop
     $\textit{Candidate} \leftarrow \textit{Candidate} + 2; i \leftarrow 1;$ 
    loop
      exit Searching_For_Next_Prime when  $p_i^2 > \textit{Candidate};$ 
      exit when  $\textit{Candidate} \bmod p_i = 0;$ 
       $i \leftarrow i + 1;$ 
    end loop;
  end loop Searching_For_Next_Prime;
   $p_k \leftarrow \textit{Candidate};$ 
end loop;

```

Алгоритм 1.2: Генерация простых чисел

Несмотря на все аргументы здравого смысла, которые позволили провести неформальное доказательство этого алгоритма, это доказательство неполно. Действительно, условие остановки $p_i^2 > \textit{Candidate}$ имеет смысл, только если отныне и впредь в таблице имеется простое число, удовлетворяющее этому неравенству. Можно избежать этой проблемы, заменяя условие выхода из цикла *Searching_For_Next_Prime* на

exit *Searching_For_Next_Prime* **when** $i \geq k$ **or else** $p_j^2 > \textit{Candidate}$

В этом случае применяется чистый аргумент Эратосфена. Но в действительности эта модификация бесполезна, так как постулат Бертрана говорит нам, что такое p_i уже вычислено. Почему? Ответ находится во втором вопросе, с другими оптимизациями, но здесь нужно немного поразмышлять...

б. Обратимся к переменной Ord , которая определяет номер наименьшего простого числа, квадрат которого превосходит $Candidate$ и вторую переменную $Square$, которая имеет значение этого квадрата. Когда нужно менять эти переменные? Когда кандидат превосходит $Square$! Это означает, что $Candidate \geq p_{Ord}^2$, а поскольку задействованные числа — нечетные, в действительности имеем равенство. Итак, непосредственным следствием постулата Бертрана является соотношение $p_{i+1} < p_i^2$ (что попутно доказывает корректность алгоритма **а**). Следовательно, когда величина первого кандидата достигает p_{Ord}^2 , имеем уже вычисленным $p_{Ord} + 1$, и изменение переменных $Square$ и Ord возможно. Кроме того, поскольку кандидат является квадратом простого числа, он — непустое число. Это приводит нас к модификации (алгоритм 7) метода из вопроса **а**.

```

 $p_1 \leftarrow 3; Candidate \leftarrow 3; Square \leftarrow 9; Ord \leftarrow 1;$ 
for  $k$  in  $2 \dots n$  loop
  Searching_For_Next_Prime: loop
     $Candidate \leftarrow Candidate + 2;$ 
    if  $Candidate = Square$  then
       $Ord \leftarrow Ord + 1; Square \leftarrow p_{Ord}^2;$ 
       $Candidate \leftarrow Candidate + 2;$ 
    end if;
     $i \leftarrow 1;$ 
    loop
      exit Searching_For_Next_Prime when  $i > Ord;$ 
      //  $Candidate < Square = p_{Ord}^2 \wedge i \leq Ord$ 
      exit when  $Candidate \bmod p_i = 0;$ 
       $i \leftarrow i + 1;$ 
    end loop;
  end loop Searching_For_Next_Prime;
   $p_k \leftarrow Candidate;$ 
end loop;

```

Алгоритм 1.3: Генерация простых чисел (оптимизация)

Окончательный алгоритм — с несколькими дополнительными оптимизациями — был, кажется, представлен впервые Далом, Дейкстрой и Хораром [58]; его можно найти также у Кнута [101] и Верса [182]. В заключение заметим, что для нас постулат Бертрана является теперь теоремой.

7. Правильный пятиугольник для всех (без Ферма и Гаусса)

Зная, что

$$-\cos \frac{\pi}{5} = \cos \frac{4\pi}{5} = 2\cos^2 \frac{2\pi}{5} - 1 = 8\cos^4 \frac{\pi}{5} - 8\cos^2 \frac{\pi}{5} + 1,$$

достаточно разложить на множители многочлен $8X^4 - 8X^2 + X + 1$ (очевидными корнями которого являются -1 и $1/2$). Единственное до-

пустимое решение для $\cos \frac{\pi}{5}$ - это $\frac{1 + \sqrt{5}}{4}$. Тогда построение проводится следующим способом (см. рис. 2):

- построить окружность с центром O и радиусом 1,
- построить диаметр AB , радиус OC перпендикулярно AB и окружность радиусом $1/2$ с центром в D - середине OC ,
- прямая AD делит эту окружность в двух точках, из которых наиболее удаленная от A точка E удовлетворяет условию: $AE = \phi$ ($AD = \sqrt{5}/2$) и $DE = 1/2$).

Тогда достаточно построить

на окружности радиуса 1 с центром O точку F такую, что $AF = \phi$, и получаем прямоугольный треугольник, в котором угол (FAB) равен $\pi/5$.

Соответствующий центральный угол (FOB) равен $2\pi/5$ и позволяет построить правильный пятиугольник (Слейе-Мишо[48]).

Существуют и другие решения, и вот одно из них, особенно простое: если z есть корень 5-й степени из единицы, соотношение $z^4 + z^3 + z^2 + z + 1 = 0$ приводит к тригонометрическому уравнению

$$2\cos \frac{2\pi}{5} + 2\cos \frac{4\pi}{5} + 1 = 0, \text{ которое просто решается, давая } \cos \frac{2\pi}{5} = \frac{\sqrt{5} - 1}{4}.$$

Сразу осуществимо построение. Если U — середина OB , строим точку W , расположенную на радиусе OA и удовлетворяющую равенству $UW = UV = \sqrt{5}/2$. Точка X , середина OW , есть ортогональная проекция точки Y такой, что угол (AOY) равен $2\pi/5$.

8. «Двоичное» деление нацело а. Вот евклидово деление a на $2b$: $a = 2b \times q + r$, с $0 \leq r < 2b$. Если $r < b$, то евклидово деление a на b получается сразу: $a = b \times 2q + r$. Зато, если $b \leq r < 2b$, деление таково: $a = b \times (2q + 1) + (r - b)$.

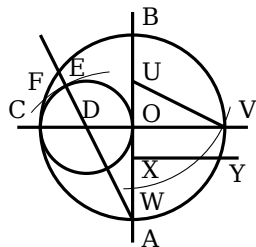


Рис.2 Правильный пятиугольник

b. Предельный случай евклидова деления появляется, когда делитель больше делимого. Вот рекурсивный алгоритм вычисления частного и остатка, представленный в форме функции *Divide*:

```
if (b > a) return (0,a);
else {
  (q,r) ← Divide (a,2b);
  if (r < b) return (2q,r);
  else return (2q + 1,r - b); }
```

d. Недостаток рекурсивного алгоритма следующий: в ходе счета второй параметр функции *Divide*, который удваивается при каждом вызове, может превзойти первоначальные величины переменных *a* и *b*. Это означает, что хотя данные и результат деления поддаются кодированию, может случиться, что в какой-либо частной реализации величины приведут к переполнению.

Например, когда применяют рекурсивный алгоритм к целым числам 31001 и 15, наблюдается ряд рекурсивных вызовов, последний из которых — *Divide*(31001,61440); если целый тип, который используют, записан и закодирован 16 битами, возникает переполнение, тогда как можно очень хорошо вычислить результат (2066,11), если остановить удвоение *b* перед последней итерацией.

<u>A. С переполнением</u>	<u>В. Без переполнения</u>
<pre>k ← 0; if b > a return (0,a); while (b ≤ a/2) { b ← 2 × b; k ← k + 1; } q ← 0; r ← a; for (int i=1; i ≤ k + 1; ++i) { q ← 2 × q; if (r ≥ b) { q ← q + 1; r ← r - b; } b ← b/2; } return (q,r);</pre>	<pre>k ← 0; while (b ≤ a) { b ← 2 × b; k ← k + 1; } q ← 0; r ← a; for (int i = 1; i ≤ k; ++i) { b ← b/2; q ← 2 × q; if (r ≥ b) { q ← q + 1; r ← r - b; } }</pre>

Алгоритм 8. Бинарные деления

Теперь уже можно — это пригодится в дальнейшем — написать итерационный алгоритм (8-А), который есть не что иное, как развитие предыдущего рекурсивного алгоритма (q и r образуют результат алгоритма: частное и остаток от деления). Этот алгоритм яснее выявляет поставленную задачу.

е. Чтобы показать эквивалентность двух алгоритмов, можно начать с доказательства очень простого свойства: $[a/2] < b \Leftrightarrow a < 2b$.

На выходе из цикла первого алгоритма имеем свойство $b/2 \leq a < b = b_0 2^k$ с $k > 0$ - не забудем, что тело цикла используется, по крайней мере, один раз.

Что касается второго алгоритма, если цикл использовался, по крайней мере, один раз, имеем свойство $b/2 \leq a/2 < b = b_0 2^k$ для $k > 1$ и b - четного. После выполнения последних присваиваний обнаруживаем то же свойство, что и для первого алгоритма. Если цикл не выполнен, это означает, что $a/2 < b_0 \leq a$, и после исполнения последних команд получаем $b_0 = b/2 \leq a < b = 2b_0$.

Последний алгоритм (8-В) получен, исходя из первой итерационной версии (8-А) извлечением из первого цикла итерации и ее слиянием с итерацией, взятой из второго цикла (для этого алгоритма, как и для предыдущего, результат - частное и остаток - представлен последними значениями переменных q и r).

9. Построение прямых линий методами DDA

а. Если u и v не взаимно простые числа, отрезок $[0, (u, v)]$ - не что иное, как повторение $d = \text{НОД}(u, v)$ сегментов, идентичных сегменту $[0, (u/d, v/d)]$.

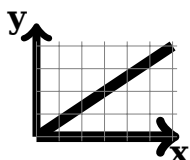
Алгоритм	Построение
<pre> y ← 0; d ← 0; Plot(0, 0); for (int x = 1; x ≤ u; ++x) { 2(vx - uy) = dc - u ≤ d < u. d ← d + 2v; if (d ≥ u) { y ← y + 1; d ← d - 2u; } Plot(x, y); } </pre>	

Рис. 3. Построение прямой в первом октанте

б. Выбрать для y наилучшую целую аппроксимацию величины vx/u - это значит убедиться, что $|vx/u - y| \leq 1/2$, или еще, что $-u \leq 2(vx - uy) < u$. Эта формула является в действительности инвариантом алгоритма. Поскольку нужная прямая имеет тангенс угла наклона меньше 1, есть только один зажигающийся пиксел на данной вертикали (т.е. возрастает при каждой итерации). Этот алгоритм (рис. 3) избегает, следовательно, пересечений маленьких горизонтальных сегментов, что дает красивый результат, когда нужны тонкие линии; было бы совсем по-другому, если нужно «жирное» построение.

Кроме того, если желаем построить этим способом прямую с наклоном, большим 1, нужен второй алгоритм, идентичный первому с точностью до перемены ролями x и y . Наконец, построение вертикальных, горизонтальных или диагональных линий может проводиться наивным способом, который будет всегда более быстрым, чем указанный.

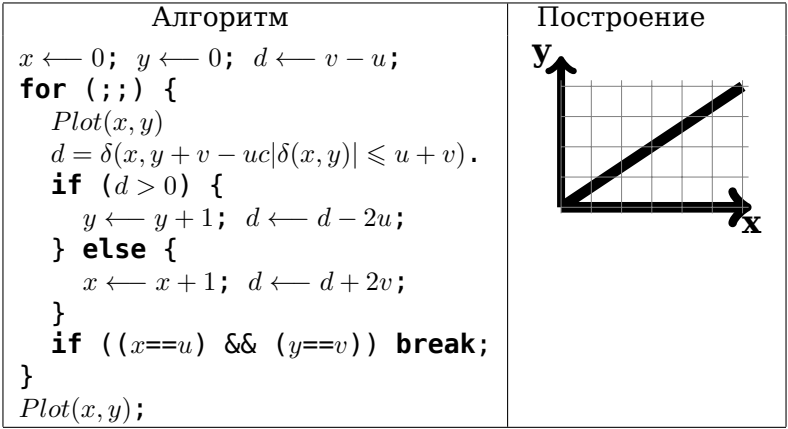


Рис. 4. Построение прямой в первой четверти

с. Парадоксально, но этот алгоритм (рис. 4) - который может дать менее элегантные результаты, чем предыдущий, - оказывается немного более сложным для реализации. Действительно, нельзя больше предполагать, что координата x возрастает после включения точки; на самом деле, каждая координата является *доминантой* в полуквадранте, о котором рассуждаем. Если рассмотреть четыре точки квадрата, сразу замечаем, что невозможно требовать - если не разрешены диагональные перемещения - чтобы вертикальное расстояние правой точки было меньше $1/2$. Если поддерживать включенную точку в полосе фигуры 1, это влечет, между прочим, что одно из расстояний справа, го-

горизонтальное или вертикальное, может быть сохранено меньшим 1, это означает, что величина $|vx - uy|$ меньше, чем u или v . Сказать, что точка расположена в полосе, очень точно означает, что $|2(vx - uy)| \leq u + v$; теперь покажем, что можно подтвердить это положение.

Движение, осуществляемое на экране, будет определяться величиной $\delta(x, y) = 2(vx - uy)$, получаемой после этого движения. Возможны два (не исключительных) случая: увеличивают x , величина становится равной $\delta(x + 1, y) = \delta(x, y) + 2v$, и нужно убедиться, что она остается меньше $u + v$, т.е. что $\delta(x, y) \leq u - v$. Аналогичным образом можно увеличить y , только если $u - v \leq \delta(x, y)$. Видим, что положение $\delta(x, y)$ по отношению к $u - v$ определяет осуществляемые перемещения.

Доказательство сходимости состоит в выявлении, что во время итераций $x \leq u$ и $y \leq v$. Чтобы проверить это, зная, что x или y растут при каждой итерации, можно предположить, что $x = u$ и $y < v$. В этих условиях оценим величину d : $d \geq \delta(u, v - 1) + v - u = u + v > 0$. Как следствие, очередная итерация увеличит y , и так будет до тех пор, пока $y < v$.

Заметим, что по этому алгоритму построенные две вещественные прямые, симметричные относительно первой диагонали, не симметричны относительно этой же диагонали (тест, применяемый к d , не эквивалентен для x и y).

10. Построение окружности методами DDA

а. Функция f убывает на интервале $x \in [0, R\sqrt{2}/2]$, и ее производная здесь меньше 1. Как следствие, $0 \leq f(x) - f(x + 1) \leq 1$, значит, учитывая это, можно ограничиться уменьшением y только на шаг, равный 1.

б. Исходим из сделанного предположения, что $y - 1$ есть лучшее приближение $f(x+1)$ тогда и только тогда, когда $y - 3/2 \leq f(x + 1) < y - 1/2$, т.е. $f(x + 1) < y - 1/2$; второе неравенство вытекает из предположений по поводу $f(x)$ и результата вопроса а.

Следовательно, оцениваем на каждой итерации величину $e(x, y) = 4y^2 - 4R^2 - 4y + 4x^2 + 8x + 5$, и нетрудно доказать, что $y - 1/2 > \sqrt{R^2 - (x+1)^2}$ тогда и только тогда, когда $e(x, y) > 0$, и еще, поскольку оперируем целыми величинами, тогда и только тогда, когда $d(x, y) = e(x, y) - 1 \geq 0$. Кроме того, $e(x + 1, y) = e(x, y) + 8x + 12$ и $e(x + 1, y - 1) = e(x, y) + 8(x - y) + 20$ и, конечно, $e(0, R) = 4 - 4R$.

Алгоритм

Построение $R = 14$

```

 $y \leftarrow R; x \leftarrow 0; s \leftarrow 1 - R;$ 
while ( $x < y$ ) {  $s = s(x, y)$ 
     $\text{Plot}(x, y);$ 
    if ( $s \geq 0$ ) {
         $s \leftarrow s + 2(x - y) + 5;$ 
         $y \leftarrow y - 1;$ 
    } else {
         $s \leftarrow s + 2x + 3;$ 
    }
     $x \leftarrow x + 1;$ 
}
if ( $x == y$ )  $\text{Plot}(x, y);$ 

```

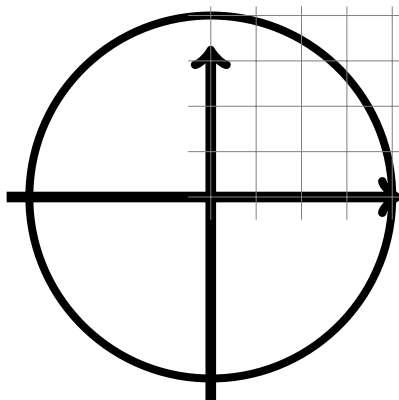


Рис. 5. Построение окружности методом DDA

Рассмотрим теперь более пристально последовательность величин $e_i = d(x_i, y_i) - 1$. Имеем следующие рекуррентные определения:

$$\begin{aligned}
 x_0 &= 0, & y_0 &= R, & e_0 &= 4 - 4R, \\
 x_{i+1} &= x_i, & y_{i+1} &= \begin{cases} y_i - 1, & \text{если } e_i \geq 0, \\ y_i & \text{в противном случае,} \end{cases} \\
 e_{i+1} &= \begin{cases} e_i + 8(x_{i+1} - y_{i+1}) + 20, & \text{если } e_i \geq 0, \\ e_i + 8x_{i+1} + 12 & \text{в противном случае.} \end{cases}
 \end{aligned}$$

Значит, совершенно ясно, что эти соотношения могут упроститься (поскольку только последовательности (x_i) и (y_i) нас интересуют, и их определение зависит от знака (e_i)) следующим образом:

$$\begin{aligned}
 x_0 &= 0, & y_0 &= R, & e_0 &= 1 - R, \\
 x_{i+1} &= x_i, & y_{i+1} &= \begin{cases} y_i - 1, & \text{если } s_i \geq 0, \\ y_i & \text{в противном случае,} \end{cases} \\
 s_{i+1} &= \begin{cases} s_i + 2(x_{i+1} - y_{i+1}) + 5, & \text{если } e_i \geq 0, \\ s_i + 2x_{i+1} + 3 & \text{в противном случае.} \end{cases}
 \end{aligned}$$

Это соотношения, используемые в алгоритме 5.

Когда работа алгоритма заканчивается, $x \geq y$, и не обязательно зажигать точку (x, y) . Действительно, если $x > y$, правила, которые применялись в верхнем октанте, больше недействительны (точка (x, y) - в нижнем октанте). Зато, если $x = y$, зажигать точку нужно.

с. Аргумент Бреэенхама просто выражает тот факт, что если дана точка (x, y) дискретной окружности и $|(x+1)^2 + (y-1)^2 - R^2| \leq |(x+1)^2 + y^2 - R^2|$, то $|\sqrt{(x+1)^2 + (y-1)^2} - R| \leq |\sqrt{(x+1)^2 + y^2} - R|$ (эта импликация верна также, если обратить знаки неравенств), и в этом случае будем брать $(x+1, y-1)$ как ближайшую точку дискретной окружности; в противном случае ближайшей точкой будет $(x+1, y)$.

Обозначим $d(x, y)$ величину (положительную или отрицательную) $x^2 + y^2 - R^2$. Ясно, что $d(x+1, y) > d(x+1, y-1)$. Как следствие, знак суммы $s(x, y) = d(x+1, y) + d(x+1, y-1)$ указывает, какова лучшая (в смысле, определенном условием задачи) точка, аппроксимирующая окружность в этой окрестности: если $s(x, y) \geq 0$, то P_2 является этой точкой, в противном случае - это P_1 . Из этого непосредственно вытекает алгоритм, и он соответствует эквивалентной версии алгоритма, построенного в вопросе **b**: достаточно положить

$$s_{\text{Бреэенхам}} = 2s_{\text{вопрос b}}.$$

11. Сложность возведения в степень

Можно сразу заметить, что P_i есть степень x : $P_i = x^{\alpha_i}$, кроме того, очевидно, что последовательность $(\alpha_i)_{\alpha_i \geq 0}$ является почти цепочкой сложений для n : действительно, нет строгого возрастания α_i , и цепь не обязательно начинается с 1; если исключить 1 из возможных значений y_i и z_i , получили бы настоящую цепочку сложений. Как бы то ни было, легко видеть, что $\alpha_1 \leq 2$, $\alpha_2 \leq 4$, $\alpha_3 \leq 8$ и т.д. Как следствие, $\alpha_t = n \leq 2^t$, а это доказывает, что $t \geq \log_2 n$. Это доказывает также, что невозможно сделать гораздо лучше, чем это делает дихотомический алгоритм возведения в степень, если разрешаются только перемножения; точнее, можно было бы получить улучшенные результаты с оптимизированными цепочками сложений, но от этого не изменится порядок величины сложности.

12. Последовательность Фибоначчи и золотое сечение

а. Запишем равенства:

$$\begin{aligned} F &= F_0 + F_1X + F_2X^2 + F_3X^3 + F_4X^4 + \dots, \\ XF &= F_0X + F_1X^2 + F_2X^3 + F_3X^4 + \dots, \\ X^2F &= F_0X^2 + F_1X^3 + F_2X^4 + \dots, \end{aligned}$$

которые непосредственно (используя определение последовательности Фибоначчи) подтверждают, что $F - XF - X^2F = X$. Формальный ряд

$1 - X - X^2$, будучи обратимым (так как его постоянный член - ненулевой), позволяет отсюда вывести, что $F = \frac{X}{(1 - X - X^2)}$. Остается только разложить эту последнюю рациональную дробь на простые слагаемые. Нулями многочлена $X^2 + X - 1$ являются $-\phi$ и $-\hat{\phi}$, и быстро получается разложение:

$$F = \frac{-\phi}{\sqrt{5}} \times \frac{1}{X + \phi} + \frac{\hat{\phi}}{\sqrt{5}} \times \frac{1}{X + \hat{\phi}} = \frac{1}{\sqrt{5}} \left(\frac{-1}{1 - \hat{\phi}X} + \frac{1}{1 - \phi X} \right),$$

с учетом того факта, что $\phi\hat{\phi} = -1$. Еще раз беря разложения рядов, которые появляются в этом последнем выражении, выводим, что $F_n = \frac{1}{\sqrt{5}}(\phi^n - \hat{\phi}^n)$.

б. $\phi^n + \hat{\phi}^n$ является общим членом ряда

$$\frac{1}{1 - \phi X} + \frac{1}{1 - \hat{\phi}X} = \frac{2 - X}{1 - X - X^2} = \frac{2F}{X} - F.$$

Значит, $\frac{1}{1 - X - X^2} = \frac{F}{X}$ является порождающим рядом последовательности (F_{n+1}) , следовательно, $\phi^n + \hat{\phi}^n = 2F_{n+1} - F_n$.

с. ϕ^n есть общий член ряда

$$\frac{1}{1 - \phi X} = \frac{1 - \hat{\phi}X}{1 - X - X^2} = \frac{F}{X} - \hat{\phi}F.$$

Но $\phi\hat{\phi} = -1$, и тогда, умножая на ϕ равенство $\phi^n = F_{n+1} - \hat{\phi}F_n$, получаем $\phi^{n+1} = \phi F_{n+1} + F_n$.

13. Соотношения в последовательности Фибоначчи

а. Нетрудно установить, что характеристическим многочленом для A является $X^2 - X - 1 = (X - \phi)(X - \hat{\phi})$. Матрица A тогда приводится к диагональному виду, и легко находим базис, образуемый собственными векторами $(\phi, 1)$ и $(\hat{\phi}, 1)$. Следовательно,

$$A^n = \frac{1}{\sqrt{5}} \begin{pmatrix} \phi & \hat{\phi} \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \phi^n & 0 \\ 0 & \hat{\phi}^n \end{pmatrix} \begin{pmatrix} 1 & -\hat{\phi} \\ -1 & \phi \end{pmatrix} = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

Конечно, простое рассмотрение первых величин A^n , вытекающих из рекуррентности, привело бы к тому же результату, но не будем отказывать себе в маленьком удовольствии (обратиться к упражнению 14 для более общей точки зрения).

Требуемые соотношения получаются соответственно вычислением определителя матрицы A^n и использованием равенства $A^{n+m} = A^n A^m$. Эти соотношения обнаруживаются вновь в еще более общей форме в упражнениях главы II, относящимся к непрерывным дробям.

б. Общий член произведения двух формальных рядов является сверткой n первых членов каждого ряда: $\sum_{i \leq n} a_i b_{n-i}$. Для задачи, которой мы занимаемся, имеем: порождающий ряд последовательности f_n есть F^2 (F - порождающий ряд последовательности Фибоначчи).

$$\begin{aligned} F^2 &= \frac{1}{5} \left(\frac{1}{(1 - \phi X)^2} - \frac{2}{(1 - \phi X)(1 - \hat{\phi} X)} + \frac{1}{(1 - \hat{\phi} X)^2} \right) \\ &= \frac{1}{5} \left(\sum (n+1) \phi^n X^n - 2 \left(\sum \phi^n X^n \right) \left(\sum \hat{\phi}^n X^n \right) + \sum (n+1) \hat{\phi}^n X^n \right) \\ &= \frac{1}{5} \left(\sum (n+1) (\phi^n + \hat{\phi}^n) X^n - 2 \sum \left(\sum \phi^i \hat{\phi}^{n-i} \right) X^n \right). \end{aligned}$$

Но мы знаем $\phi^n + \hat{\phi}^n$ (упражнение 12), и общий член второго ряда есть $\frac{\phi^{n+1} - \hat{\phi}^{n+1}}{\phi - \hat{\phi}} = F_{n+1}$. Значит,

$$\begin{aligned} F^2 &= \frac{1}{5} \left(\sum (n+1) (2F_{n+1} - F_n) X^n - 2 \sum F_{n+1} X^n \right) \\ \text{или} \quad \sum_{k=0}^n F_k F_{n-k} &= \frac{2nF_{n+1} - (n+1)F_n}{5}. \end{aligned}$$

14. Линейные рекуррентные последовательности k -го порядка

а. Действуем, выполняя преобразования над порождающими рядами. Пусть \mathcal{F} - порождающий ряд последовательности. Как для последовательности Фибоначчи, можно легко установить, что $(1 - X - X^2)\mathcal{F} = x_0 + X(x_1 - x_0)$. Следовательно,

$$\mathcal{F} = \frac{x_0 + X(x_1 - x_0)}{1 - X - X^2} = x_0 \frac{F}{X} + (x_1 - x_0)F.$$

Значит, $x_n = x_0 F_{n+1} + (x_1 - x_0) F_n = x_1 F_n + x_0 F_{n-1}$.

б. Ясно, что множество рассмотренных последовательностей является векторным пространством размерности k , и существует изоморфизм между этим пространством и K^k : он ставит в соответствие ка-

ждой последовательности ее k первых членов. Полным прообразом канонического базиса K^k является то, что называется фундаментальным базисом пространства последовательностей. Обозначим этот базис (x^{k-1}, \dots, x^0) (x^0 есть полный прообраз вектора $(0, \dots, 0, 1)$). Матрица A , которая действует в K^k , соответствует при этом изоморфизме оператору сдвига последовательностей. Тогда столбцы матрицы A суть выборки (x_k^i, \dots, x_1^i) из последовательностей x^i . Это означает еще, что

$$A^n = \begin{pmatrix} x_{n+k-1}^0 & x_{n+k-1}^1 & \cdots & x_{n+k-1}^{k-1} \\ x_{n+k-2}^0 & x_{n+k-2}^1 & \cdots & x_{n+k-2}^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^0 & x_n^1 & \cdots & x_n^{k-1} \end{pmatrix}.$$

В случае, когда $k = 2$, фундаментальный базис пространства последовательностей образован последовательностью Фибоначчи и ее *сдвигами*. Тут же обнаруживается результат предыдущего вопроса и матричное равенство упражнения 13.

15. Дихотомия по Горнеру

а. Конечно, $P_0 = P$ и $P_k = a_k$. Соотношение между P_i и P_{i-1} , которое определяет метод Горнера, есть $P_{i-1} = XP_i + a_{i-1}$.

б. Если предположить, что $n = \sum b_i 2^i$, то можно рассмотреть многочлен $P = \sum b_i X^i$ и вычислить элементы $x^{P_i(2)}$, используя предыдущее рекуррентное соотношение. Это дает алгоритм:

```
R ← 1;
for (int i = n; i ≥ 0; —i) {
    R ← R2 × xbi; R = xbn2n-i + bn-12n-i-1 + ... + bi = xPi(2)
}
return R;
```

Этот алгоритм может быть оптимизирован исключением ненужных перемножений следующим образом:

```
R ← 1;
for (int i = n; i ≥ 0; —i) {
    R ← R2;
    if (bi == 1) R ← R × xbi;
    R = xbn2n-i + bn-12n-i-1 + ... + bi = xPi(2)
}
return R;
```

с. Если неизвестно двоичное разложение n , то ясно, что нужно будет тем или иным способом вычислить его более или менее неявно по ходу выполнения алгоритма. Если известно, что $2^k \leq n < 2^{k+1}$, то это влечет, что двоичный разряд порядка k числа n равен 1, что позволяет запустить счет. Затем достаточно определить, принадлежит ли $n - 2^k$ интервалу $[2^{k-1}, 2^k]$, чтобы узнать величину двоичного разряда порядка $k - 1$. Все это приводит к следующему алгоритму (который не касается случая $n = 0$):

```

 $\alpha \leftarrow 1$ ;
while ( $\alpha \leq n - \alpha$ )  $\alpha \leftarrow 2\alpha$ ;
Ici  $2\alpha > n \geq \alpha = 2^k$ 
 $R \leftarrow x$ ;  $m \leftarrow n - \alpha$ ;
for (;;) {
     $x^n = R^\alpha x^m$  et  $0 \leq m < \alpha$ ,  $\alpha$  est une puissance de 2
    if ( $\alpha == 1$ ) break;
     $\alpha \leftarrow \alpha/2$ ;  $R \leftarrow R^2$ ;
    if ( $m \geq \alpha$ ) {
         $R \leftarrow R \times x$ ;
         $m \leftarrow m - \alpha$ ;
    }
}
return R;
```

В этом алгоритме все начинается с определения $\alpha = 2^k$ такого, что $2^k \leq n < 2^{k+1}$ (только без того, чтобы результаты промежуточных вычислений превосходили n), затем продолжается вычисление x^n .

d. В этом случае, когда сложность одного перемножения постоянна (независимо от размеров сомножителей), устанавливаем, что имеется самое большее k итераций, каждая из которых содержит 1 или 2 умножения; следовательно, сложность оценивается сверху числом $2 \log_2 n$.

В случае, когда сложность элементарного умножения не постоянна, результат совершенно другой.

16. Вычисление чисел Фибоначчи

a. Несложно написать рекурсивный алгоритм, подобный следующему:

```

if ( $n == 0$ ) return 0;
else if ( $n == 1$ ) return 1;
else return Фибоначчи( $n - 1$ ) + Фибоначчи( $n - 2$ );
```

Число рекурсивных вызовов в действительности равно $2 \sum_{i=1}^{n-1} F_i = 2(F_{n+1} - 1)$ (доказать этот результат), что дает экспоненциальную сложность.

Вдохновившись, напротив, методом, использованным для алгоритма Евклида, состоящим в преобразовании пар последовательных элементов в последовательность: $(F_{n+2}, F_{n+1}) = (F_{n+1} + F_n, F_{n+1})$ (формула, которая имеет связь с матрицей упражнения 14), приходим к алгоритму, который для n вычисляет пару чисел Фибоначчи (F_{n+1}, F_n) :

```

if ( $n == 0$ ) return (1, 0);
else {
    ( $F_n, F_{n-1}$ )  $\leftarrow$  Фибоначчи( $n - 1$ );
    return ( $F_n + F_{n-1}, F_n$ );
}

```

Этот алгоритм, который легко может быть сделан нерекурсивным, имеет сложность, пропорциональную n (линейную, а не экспоненциальную).

Б. Любое из соотношений:

$$F_{n-1} + \phi F_n = \phi^n, \quad \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad F_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}$$

приводит к мысли, что F_n может быть вычислено методом дихотомического возведения в степень. Это действительно так! Обоснуем наше решение с помощью, например, первого соотношения (другие приводят к тому же методу). В несколько научной манере можно сказать, что ϕ принадлежит кольцу $\mathbb{Z}[\phi]$, состоящему из $a + \phi b$ с целыми a, b . Устойчивость относительно произведения в $\mathbb{Z}[\phi]$ вытекает из уравнения $\phi^2 = \phi + 1$:

$$(a + \phi b)(c + \phi d) = ac + \phi(ad + bc) + \phi^2 bd = (ac + bd) + \phi(ad + b(c + d)).$$

Эта формула показывает, к тому же, как считать в $\mathbb{Z}[\phi]$ с помощью целых операций. Следовательно, вычисление ϕ^n может быть осуществлено через дихотомию; как это видим по вышеупомянутой формуле, сложность одного перемножения в $\mathbb{Z}[\phi]$ есть 4 целых перемножения и 3 целых сложения, тогда как сложность возведения в квадрат — 3 целых перемножения и 2 сложения: $(a + \phi b)^2 = (a^2 + b^2) + \phi b(2a + b)$, что приводит к алгоритму 9-А.

Но можно сделать лучше; соотношение

$$(a + bX)(c + dX) = ac + ((a + b)(c + d) - ac - bd)X + bdX^2$$

доказывает, что можно умножать два многочлена первой степени с помощью только 3 перемножений, но 4 сложений (об этом снова пойдет

речь в главе V). Применительно к кольцу $\mathbb{Z}[\phi]$ оно позволяет вычислить произведение с помощью 3 целых перемножений и 4 целых сложений:

$$(a + \phi b)(c + \phi d) = ac + bd + \phi((a + b)(c + d) - ac),$$

и, конечно, использовались при реализации.

с. Можно вычислить ϕ^n через дихотомию, но применяя метод упражнения 15 (использование метода Горнера). Заметим, что одна из компонент аддитивных перемножений в этом способе постоянна (равна элементу, степень которого хотим вычислить): здесь это умножение на ϕ , которое не является настоящим умножением, так как $(a + b\phi)\phi = b + \phi(a + b)$. В алгоритме 9-В последовательные степени ϕ вычислены в паре (g, g') , представляющей $g + \phi g'$.

A. Традиционная дихотомия	B. Дихотомия «по Горнеру»
<pre> (f, f') ← (1, 0); (g, g') ← (0, 1); i ← n; for (;;) { (f + φf') × (g + φg')ⁱ = φⁿ if (i%2 == 1) { (f, f') ← (fg + f'g', fg' + f'(g + g')); } i ← [i/2]; (f + φf') × (g + φg')²ⁱ = φⁿ if (i == 0) break; (g, g') ← (g² + g'², g'(2g + g')); } return f'; f + φf' = φⁿ </pre>	<pre> n = ε_q2^q + ε_{q-1}2^{q-1} + ... + ε₀ (g, g') ← (1, 0); for (int i = q; i ≥ 0; --i) { (g, g') ← (g² + g'², g'(2g + g')); if (ε_i == 1) { (g, g') ← (g', g + g'); } } return g'; </pre>

Алгоритм 9. Вычисление чисел Фибоначчи

17. ϵ -лексикографический порядок и знакопеременный лексикографический порядок

а. ϵ -лексикографический порядок является линейным, если исходные упорядочения линейны. Если функция ϵ тождественно равна 1, получаем тогда обычный лексикографический порядок.

(e_1, f_4)	\rightarrow	(e_2, f_4)	(e_3, f_4)
\uparrow		\downarrow	\uparrow
(e_1, f_3)		(e_2, f_3)	(e_3, f_3)
\uparrow		\downarrow	\uparrow
(e_1, f_2)		(e_2, f_2)	(e_3, f_2)
\uparrow		\downarrow	\uparrow
(e_1, f_1)		(e_2, f_1)	$\rightarrow (e_3, f_1)$

б. Эта история с планкой наводит на мысль о том, что происходит внутри $E \times F$; функция $F \ni y \mapsto (x, y) \in \in x \times F$ является изоморфизмом, если $\epsilon(x) = 1$, в противном случае антиизоморфизмом.

с. Убеждаемся, что $\min_{E \times F} =$

Определение наибольшего элемента немного сложнее:

$\max_{E \times F} = (\max_E, \min_F)$, если мощность множества E - четна, и $\max_{E \times F} = (\max_E, \max_F)$, если мощность множества E - нечетна.

Проиллюстрируем это с помощью примера: пусть $E = \{e_1 < e_2 < e_3\}$ и $F = \{f_1 < f_2 < f_3 < f_4\}$; тогда знакопеременное лексикографическое произведение $E \times F$ представляется так, как это показано в таблице выше.

Последующий элемент для $(x, y) \in E \times F$ вычисляется тогда рекурсивно, например, с использованием алгоритма 10-А.

<p>A. Рекурсивная версия</p> <pre>if ($\epsilon(x) == 1$ && $y < \max_F$) return ($x, \text{succ}_F(y)$); else if ($\epsilon(x) == -1$ && $y > \min_F$) return ($x, \text{предш}_F(y)$); else if ($x < \max_E$) return ($\text{succ}_E(x), y$); else ici, (x, y) = $\max_{E \times F}$;</pre>	<p>B. Итеративная версия</p> <pre>$s \leftarrow \text{sign}_{E_1}(x_1)\text{sign}_{E_2}(x_2)...\text{sign}_{E_n}(x_n)$; for (int $i = n$; $i \geq 1$; $--i$) { $s \leftarrow s \times \text{sign}_{E_i}(x_i)$; $s = \text{sign}_{E_1}(x_1)\text{sign}_{E_2}(x_2)...\text{sign}_{E_{i-1}}(x_{i-1})$ if ($s == 1$ && $x_i < \max_{E_i}$) { $x_i \leftarrow \text{succ}_{E_i}(x_i)$; return x; } else if ($s == -1$ && $x_i > \min_{E_i}$) { $x_i \leftarrow \text{предш}_{E_i}(x_i)$; return x; } } $x = \max_E$; return Successor_{Error};</pre>
---	---

Алгоритм 10. Последующий элемент в знакопеременном лексикографическом порядке

В этом алгоритме замечаем, что единственная составляющая модифицирована и заменена своим последующим или предшествующим элементом. Отсюда следует, что функция $\mu(x, y) = \text{sign}_E(x) \times \text{sign}_F(y)$ со значениями из $\{1, -1\}$ является «знакочередующейся», т.е. удовлетворяет равенству $\mu(\text{succ}(x, y)) = -\mu(x, y)$. Поскольку она равна 1 на

$\min_{E \times F} = (\min_E, \min_F)$, то она совпадает с функцией-сигнатурой на $E \times F$.

d. Пусть E_1, E_2, \dots, E_n — n конечных линейно упорядоченных множеств. Обозначим sign_{E_i} , функцию-сигнатуру на E_i , единственную функцию, отображающую E_i на множество $\{-1, 1\}$, удовлетворяющую условиям $\text{sign}_{E_i}(\min_{E_i}) = 1$ и для $x \in E_i - \{\max_{E_i}\} \text{sign}_{E_i}(\text{succ}_{E_i}(x)) = -\text{sign}_{E_i}(x)$. Обозначим через $\epsilon_i : E \rightarrow \{-1, 1\}$ функцию, определенную через $\epsilon_i(x) = \prod_{j < i} \text{sign}_{E_j}(x_j)$. На декартовом произведении $E = E_1 \times E_2 \times \dots \times E_n$ определим отношение $< : (x_1, \dots, x_n) < (y_1, \dots, y_n)$ тогда и только тогда, когда наименьшее i такое, что $x_i \neq y_i$, удовлетворяет неравенству $x_i \leq_{\epsilon_i(x)} y_i$.

Легко убедиться, что отношение « $x = y$ или $x < y$ » является отношением порядка на E , которое есть не что иное, как отношение порядка, изученное в предыдущем упражнении для $n = 2$. Это отношение линейного порядка, наименьший элемент которого есть $\min_E = (\min_{E_1}, \dots, \min_{E_n})$.

Сигнатура линейно упорядоченного множества E есть «произведение сигнатур каждой компоненты»; это можно доказать с помощью индукции, используя ассоциативность: $(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$, которая вытекает из-того факта, что сигнатура произведения двух конечных линейно упорядоченных множеств есть произведение их сигнатур.

Следующий вопрос показывает, что последующий элемент для (x_1, \dots, x_n) получается изменением *единственной компоненты* x_i и ее заменой своей последующей или предыдущей в E_i . Функция

$$\mu(x_1, \dots, x_n) = \text{sign}_{E_1}(x_1) \dots \text{sign}_{E_n}(x_n)$$

для $x \neq \max_E$ удовлетворяет равенству $\mu(\text{succ}_E(x)) = -\mu(x)$ и, поскольку для \min_E она равна 1, это — сигнатура множества $E = E_1 \times \dots \times E_n$.

e. Как и для лексикографического порядка, последующий элемент для $x = (x_1, \dots, x_n)$ получается в результате *попыток* варьировать последнюю компоненту x_n и ее замены на $\text{succ}_{E_n}(x)$ (если $\epsilon_n(x) = 1$), или на $\text{предш}_{E_n}(x)$ (если $\epsilon_n(x) = -1$). Если это невозможно, т.е. если $\epsilon_n(x) = 1$ и $x_n = \max_{E_n}$, либо $\epsilon_n(x) = -1$ и $x_n = \min_{E_n}$, то пробуем сварьировать предпоследнюю компоненту x_{n-1} по аналогичному правилу..., что приводит к окончательному алгоритму (10-B).

18. Перебор делителей целого числа

Делитель числа $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$ полностью определен данной последовательностью $(\beta_1, \beta_2, \dots, \beta_n)$, которая удовлетворяет соотношению

ям $0 \leq \beta_i \leq \alpha_i$ для $i = 1, 2, \dots, n$. Смена делителя n сводится, значит, к смене показателей простых множителей. Если выбрать порядок, определенный на показателях, т.е. на произведении $[0, \alpha_1] \times \dots \times [0, \alpha_n]$, и вычислить делитель в переменной P , тогда достаточно *одного деления или умножения* для вычисления последователя для P . Действительно, если P записывается: $P = p_1^{\beta_1} p_2^{\beta_2} \dots p_k^{\beta_k}$, и если i является индексом компоненты, которая изменилась в вычислении последующего для $(\beta_1, \beta_2, \dots, \beta_n)$, то $\text{succ}(P) = P \times p_i$ или же $\text{succ}(P) = P/p_i$. Возьмем пример: $n = p^3 q^3 r^2$; последовательные величины для P , т.е. список делителей n , тогда получается так: $p^0 q^0 r^0$, $p^0 q^0 r^1$, $p^0 q^0 r^2$, $p^0 q^1 r^2$, $p^0 q^1 r^1$, $p^0 q^1 r^0$, $p^0 q^2 r^0$, $p^0 q^2 r^1$, $p^0 q^2 r^2$, $p^0 q^3 r^2$, $p^0 q^3 r^1$, $p^0 q^3 r^0$, $p^1 q^3 r^0$, $p^1 q^3 r^1$, ...

19. Код Грея

а. Достаточно рассмотреть множество

$$\{0, 1\}^n = \{0, 1\} \times \{0, 1\} \times \dots \times \{0, 1\}$$

и наделить его знакопеременным лексикографическим порядком. Можно также рассуждать с помощью индукции по n : если s_1, \dots, s_{2^n-1} является кодом Грея для цепочек из n разрядов, то:

$$\begin{aligned} & (0, s_0), \quad (0, s_1), \dots, (0, s_{2^n-2}), \quad (0, s_{2^n-1}), \quad (1, s_{2^n-1}), \\ & \hspace{15em} (1, s_{2^n-2}), \dots, (1, s_1), (1, s_0) \\ \text{и} \quad & (0, s_0), \quad (1, s_0), \quad (1, s_1), \quad (0, s_1), \dots, (0, s_{2^n-2}), \\ & \hspace{15em} (1, s_{2^n-2}), \quad (1, s_{2^n-1}), \quad (0, s_{2^n-1}) \end{aligned}$$

суть два кода Грея для цепочек из $n+1$ разрядов.

б. Используя первый из только что данных примеров кода Грея, можно довольно просто построить требуемую программу. Для этого применяют две взаимно рекурсивные процедуры: одна генерирует код Грея, другая – его зеркальное отражение. Каждая из этих двух процедур *Gray* и *ReverseGray* обладает двумя параметрами, второй обозначает цепочку из 1 или 0, которая должна быть помещена в качестве приставки каждого слова кода Грея, длина которой задается первым параметром. Главный вызов, который должен быть осуществлен для получения вывода на экран кода Грея длины n , есть $\blacktriangleright \text{Gray}(n) \blacktriangleleft$. Вот первая из этих процедур:

```

void Gray(int n, string Prefix) {
  if (n == 0) Put_Line(Prefix);
  else {
    Gray(n - 1, Prefix & "0"); Reverse_Gray(n - 1, Prefix & "1");
  }
}

```

Вторая процедура *Reverse_Gray* получается перестановкой вхождений цепочек "1" и "0".

с. Чтобы написать итерационный алгоритм, достаточно использовать результат упражнения 17, но факт работы с $\{0,1\}$ приносит значительные упрощения. Изучим знакопеременное лексикографическое произведение вида $E \times \{0,1\}$, где E — конечное линейно упорядоченное множество, и рассмотрим последовательность для элемента $(x_1, x_2) \in E \times \{0,1\}$ в случае, когда этот последовательность получается изменением компоненты x_2 ; это проявляется в следующих ситуациях:

```

s := x1 + x2 + ... + xn mod 2;
for (int i = n; i ≥ 1; -- i) {
  if (s == 0) {
    xi ← 1 - xi;
    return x;
  }
  s ← (s + xi) mod 2;
}
return SuccessorError;

```

- $(x_1, x_2) = (x_1, 0)$ и $\text{sign}(x_1) = 1$, значит, $\text{succ}(x_1, 0) = (x_1, 1)$,
- $(x_1, x_2) = (x_1, 0)$ и $\text{sign}(x_1) = -1$, значит, $\text{succ}(x_1, 1) = (x_1, 0)$.

Можно свести эти два случая к одному: $\text{sign}(x_1, x_2) = 1$ и $\text{succ}(x_1, x_2) = (x_1, 1 - x_2)$, что приводит к алгоритму вычисления последовательности для элемента, который приведен слева.

Алгоритм 1.4: Код Грея

Несколько замечаний, однако, к этому алгоритму: прежде всего, вместо использования сигнатуры рассуждаем в терминах четности цепи рассматриваемых разрядов; затем, нормальный выход из цикла (по исчерпанию итераций) происходит только для последнего элемента кода, что объясняет возбуждение исключения.

20. Изменение слов в обратном коде Грея

а. Очевидно, что $T_1 = (1)$. С другой стороны, обратный код Грея таков, что если G_n является таким кодом из n разрядов, то $(0G_n, 1\overline{G_n})$ — обратный код Грея из $n + 1$ разрядов, запись $\overline{G_n}$ означает обращенную последовательность G_n . при этих условиях $T_{n+1} = T_n \cdot (n + 1) \cdot \overline{T_n}$, и тот факт, что $\overline{T_1} = T_1$, доказывает с помощью индукции требуемое свойство.

б. Этот алгоритм, выведенный из тонких методов освобождения от рекурсивности, является искусственным, но обезоруживающе простым.

Очевидно, что при $k = 1$ он дает последовательность T_1 (сведенную к элементу 1). Кроме того, в этом случае выполнена единственная итерация, что влечет при выходе из цикла $Stack(1) = x$ и $Stack(2) = 2$.

Сделаем теперь индуктивное предположение, что этот алгоритм ведет себя так, как описано в условии, для $k = n - 1$. Если $k = n$, то он начинает *создавать* последовательность T_{n-1} (гипотеза индукции), и после этого обнаруживаем, что $Stack(1) = n$ и $Stack(n) = n$, а другие элементы массива стека остаются неизменными по отношению к первоначальной конфигурации. В этой последней ситуации осуществляем итерацию, по ходу которой все начинается с *получения* $Stack(1) = n$, возвращаем $Stack(1)$ к 1, затем переносим значение $x = Stack(n + 1)$ на место n и заменяем $Stack(n + 1)$ на $n + 1$. Тогда оказываемся в ситуации, которая заставляет алгоритм заново создавать последовательность T_{n-1} . Тут остается проверить, что на выходе из этого второго захода содержани массива $Stack$ согласуется с гипотезой индукции; принимая во внимание конфигурацию, предшествующую этому второму заходу ($Stack(n) = n + 1$) и $Stack(n + 1) = n + 1$, видим, что эта проверка является лишь формальностью.

Запись пактеа, реализующего шаг за шагом генерацию слов кода, требует нескольких предостережений, прежде всего, нельзя больше удовлетворяться представлением слова кода только как функции вычисления последующего элемента, нужно получить также таблицу $Stack$, вытекающую из последнего вычисления; затем, когда достигнуто последнее слово кода, исполнение тела цикла будет вызывать ошибку переполнения. Можно сгладить этот недостаток, заменяя последнюю инструкцию тела цикла на $Stack(1 + (t \bmod n)) \leftarrow 1 + (t \bmod n)$; эта модификация позволяет алгоритму бесконечно создавать один и тот же код Грея.

Еще можно заметить, что этот способ генерации кода Грея, благодаря изменениям, которые нужно проделать, позволяет создать цикл в обратном коде Грея, начинающийся с какой угодно конфигурации. Цикл, который проходится таким образом, выводится из *стандартного* кода (того, которому выше дано рекурсивное определение) трансляцией всех слов кода (трансляция означает здесь покомпонентное сложение по модулю 2).

21. Вычисление перманента матрицы

а. Центральный член может быть рассмотрен как однородный многочлен степени n от переменных a_{ij} , который можно выразить суммой $\sum \mu_\alpha a_{1\alpha(1)} \dots a_{n\alpha(n)}$, распространяющейся на все **отображения** α из

$[1, n]$ в $[1, n]$, при этом коэффициент μ_α определяется суммой

$$\mu_\alpha = \sum_{\epsilon_i \in \{0,1\}} (-1)^{\epsilon_1 + \dots + \epsilon_n} \epsilon_{\alpha(1)} \dots \epsilon_{\alpha(n)}.$$

Если α — перестановка, коэффициент μ_α равен $(-1)^n$, поскольку единственный ненулевой член суммы — это тот, для которого все ϵ_i , равны 1. В противном случае можно разложить μ_α следующим образом:

$$\sum_{\epsilon_i, i \neq k} (-1)^{\epsilon_1 + \dots + \epsilon_n} \sum_{\epsilon_k=0,1} (-1)^{\epsilon_k} \epsilon_{\alpha(1)} \dots \epsilon_{\alpha(n)},$$

где k — элемент из $[1, n]$, который не принадлежит образу α , и хорошо видно, что внутренний член (сумма по ϵ_k) равен нулю; это доказывает, что μ_α в этом случае равно нулю. Второе искомое выражение (суммы которого записываются на множествах) просто выводится из первого, исключением ненулевых ϵ_i , появляющихся в сумме.

b. Возвратный код Грея из n

разрядов является выражением линейного порядка на множестве всех частей E из $[1, n]$, наименьший элемент которого есть \emptyset и наибольший, E_{max} , исходное множество. Если E' является последователем для E , то симметрическая разность $E \Delta E'$ сводится к одному элементу, а это означает, что E' получается, исходя из E , добавлением или удалением одного элемента. Введем несколько обозначений: $E' = \text{succ}(E)$, $\sum_E = \sum_{F \leq E} (-1)^{|F|} \prod_{1 \leq i \leq n} S_i(F)$, где $S_i(E) = \sum_{j \in E} a_{ij}$. Вычисляемый перманент является, значит, суммой $(-1)^n \sum_{E_{max}}$. Эта сумма получается вычислением последовательных значений \sum_E и, тогда, итерируя, имеем:

```
S[1] = a[1][succ(n)]; P = S[1];
for (int i = 2; i <= n; ++i)
    S[i] = a[i][succ(n)]; P *= S[i];
Σ = -P; E = succ(n);
while (E != E_max) {
    j = E Δ E'; S[1] ±= a[1][j];
    P = pow(-1, abs(E')) * S[1];
    for (int i = 2; i <= n; ++i)
        S[i] ±= a[i][j]; P *= S[i];
    Σ += P; E = E';
}
return pow(-1, n) * Σ;
```

Алгоритм 1.12: Вычисление перманента

$$\sum_{E'} = \sum_E + (-1)^{|E'|} \prod_{1 \leq i \leq n} S_i(E'). \quad (1.7)$$

Вклад наименьшего элемента \emptyset в эту сумму — нулевой; значит, начнем с его последователя, который может быть $\{1\}$, $\{n\}$ или какое-нибудь одноэлементное множество в соответствии с порядком, заданным на $[1, n]$. Можно заметить, что $S_i(E') = S_i(E) \pm a_{ij}$, где $\{j\} = E \Delta E'$.

В этой записи, как в алгоритме 12, \pm должен пониматься как $+$, если $E \subset E'$ и $-$, если $E' \subset E$.

Чтобы оценить $\sum_{E'}$, исходя из \sum_E с использованием соотношения (7), нужно осуществить n сложений (для подсчета каждого $S_i(E')$), затем $n-1$ перемножений: $S_1(E') \times \dots \times S_n(E')$, и, наконец, 1 сложение. Имеем $2^n - 2$ операций для осуществления (7), при этом первый член $\sum = -\prod_{1 \leq i \leq n} \alpha_{in}$ требует $n-1$ перемножений; это доказывает сформулированный результат о сложности. Сложность $O(n2^n)$ значительна, но остается того же порядка, что и сложность, индуцированная определением ($n!(n-1)$ перемножений и $n!-1$ сложений). Формула Стирлинга позволяет сравнить эти два значения сложности: $n \cdot n! / (n \cdot 2^n) \approx (n/2e)^n \sqrt{2\pi n}$.

22. Перманент матрицы (продолжение)

а. Правая часть может рассматриваться как многочлен (от переменных a_{ij}), равный $\sum_{\alpha} \mu_{\alpha} a_{1\alpha(1)} \dots a_{n\alpha(n)}$, где сумма распространяется на все отображения $[1, n]$ в $[1, n]$. Коэффициент μ_{α} задан формулой

$$\mu_{\alpha} = \sum_{\omega} \mu_{\alpha}(\omega) \quad \text{с} \quad \mu_{\alpha}(\omega) = \omega_1 \dots \omega_{n-1} \omega_{\alpha(1)} \dots \omega_{\alpha(n)},$$

в которой полагаем $\omega_n = 1$. Если α — перестановка, то каждое $\mu_{\alpha}(\omega)$, присутствующее в сумме μ_{α} , равно 1 и, следовательно, $\mu_{\alpha} = 2^{n-1}$. Напротив, если α не является перестановкой, то сумма μ_{α} — нулевая. Действительно, образ α отличен от $[1, n]$, и различаем два случая:

- (i) $\exists k < n$, не принадлежащий образу α ,
- (ii) $\exists k < n$, дважды полученный из α .

В обоих случаях члены $\mu_{\alpha}(\omega)$, присутствующие в сумме, группируются попарно, один соответствуя $\omega_k = 1$, другой — $\omega_k = -1$, и взаимно уничтожаются (в случае (ii) $\mu_{\alpha}(\omega) = \omega_k$).

б. Формула пункта **а** может быть записана в следующем виде:

$$\frac{\text{per} A}{2} = \sum \omega_1 \dots \omega_{n-1} \prod_{1 \leq i \leq n} (a_{in} + \omega_1 a_{i1} + \dots + \omega_{n-1} a_{in-1})/2.$$

Как и в предыдущем упражнении, вычисление перманента получается генерированием перебора при линейной упорядоченности на $\{-1, 1\}^{n-1}$, в которой два последовательных элемента отличаются только одной компонентой. Если для $\omega \in \{-1, 1\}^{n-1}$ и $i \leq n$ положить

$$S_i(\omega) = (a_{in} + \omega_1 a_{i1} + \dots + \omega_{n-1} a_{in-1})/2,$$

то можно, благодаря перебору на $\{-1, 1\}^{n-1}$, вычислить последовательно $\sum_{\omega} = \sum_{\rho \leq \omega} \dots$, используя формулу

$$\sum_{\omega'} = \sum_{\omega} + \omega'_1 \dots \omega'_{n-1} \prod_{1 \leq i \leq n} S_i(\omega'), \quad (1.8)$$

где ω' — последователь для ω в $\{-1, 1\}^{n-1}$.

Если j является индексом, по которому различаются два слова ω и ω' , то сумма $S_i(\omega')$ вычисляется, исходя из $S_i(\omega)$, через $S_i(\omega') = S_i(\omega) - a_{ij}$, если $\omega_j = 1$, и через $S_i(\omega') = S_i(\omega) + a_{ij}$, если $\omega_j = -1$.

```

for (int i = 1; i <= n; ++i)
    S[i] = (a[i][1] + ... + a[i][n]) / 2;
 $\sum$  = S[1] * ... * S[n];  $\epsilon$  = {0, ..., 0}; sgn = 1;
while ( $\epsilon$  !=  $\epsilon_{max}$ ) {
     $\epsilon'$  = succ( $\epsilon$ ); sgn = -sgn; j =  $\epsilon \Delta \epsilon'$ ;
    if ( $\epsilon_j$  == 0)
        for (int i = 1; i <= n; ++i)
            S[i] -= a[i][j];
    else //  $\epsilon_j$  == 1
        for (int i = 1; i <= n; ++i)
            S[i] += a[i][j];
     $\sum$  += sgn * S[1] * ... * S[n];  $\epsilon$  =  $\epsilon'$ ;
}
return 2 *  $\sum$ ;

```

Алгоритм 1.13: Вычисление перманента в кольце где 2 обратимо

С практической точки зрения для генерации адекватного перебора $\{-1, 1\}^{n-1}$, выбираем соответствие между $\{0, 1\}$ и $\{-1, 1\}$ вида $\epsilon \mapsto (-1)^\epsilon$ и классическую генерацию кода Грея на $\{0, 1\}^{n-1}$, что достигается с помощью алгоритма 13. Мультипликативная сложность получается, если заметить, что нужно вычислить $2^n - 1$ членов \sum_{ω} , каждый из которых требует $n - 1$ перемножений (см. формулу (8)), значит, всего $2^{n-1}(n - 1)$ произведений, к которым нужно добавить последнее умножение на 2. С точки зрения сложений первоначальный член $\sum_{(1, \dots, 1)}$ требует $n(n - 1)$ сложений и n делений на 2, тогда как общий член \sum_{ω} вычисляется, исходя из предыдущего, с помощью n сложений; наконец, нужно сложить все эти члены, что требует в целом $n(n - 1) + (2^{n-1} - 1)(n + 1)$ сложений и n делений на 2. Заметим относительно предыдущего упражнения, что сложность была приблизительно разделена на 2.

с. Рассмотрения полностью аналогичны предыдущему пункту, если только невозможно деление на 2; деление (точное) на 2^{n-1} будет иметь место уже в конце. Результатом является алгоритм 14.

```

for (int i = 1; i <= n; ++i)
    S[i] = a[i][1] + ... + a[i][n];
 $\sum$  = S[1] * ... * S[n];  $\epsilon$  = {0, ..., 0}; sgn = 1;
while ( $\epsilon \neq \epsilon_{max}$ ) {
     $\epsilon' = \text{succ}(\epsilon)$ ; sgn = -sgn; j =  $\epsilon \Delta \epsilon'$ ;
    if ( $\epsilon_j == 0$ )
        for (int i = 1; i <= n; ++i)
            S[i] -= 2 * a[i][j];
    else //  $\epsilon_j == 1$ 
        for (int i = 1; i <= n; ++i)
            S[i] += 2 * a[i][j];
     $\sum \text{ += sgn} * S[1] * \dots * S[n]$ ;  $\epsilon = \epsilon'$ ;
}
return  $\sum / (2^{n-1})$ ;

```

Алгоритм 1.14: Вычисление перманента для характеристики отличной от 2

23. Массив инверсий подстановки

d. Массив инверсий перестановки α имеет вид (0, 0, 0, 1, 4, 2, 1, 5, 7). Свойство $0 \leq \alpha_k < k$ легко получается из того, что имеется точно $k-1$ целых чисел, заключенных строго между 0 и k . Массив инверсий возрастающей перестановки интервала $[1, n]$ есть, очевидно, (0, 0, ..., 0), и таблица для единственной убывающей перестановки — (0, 1, 2, ..., n).

e. Можно использовать тот факт, что $a_{\alpha(j)}$ есть число индексов таких, что $i > j$ и $\alpha(i) < \alpha(j)$, что приводит к нижеследующему алгоритму:

```

{ a[1], ..., a[n] } = { 0, ..., 0 }
for (int j = 1; j <= n; ++j)
    for (int i = j + 1; i <= n; ++i)
        if ( $\alpha(i) < \alpha(j)$ )
            a[ $\alpha(j)$ ]++;

```

Сложность полученного способа, конечно, имеет порядок квадрата длины перестановки.

f. Пусть a — элемент из $[0, 1[\times [0, 2[\times \dots \times [0, n[$. Построим перестановку α , для которой a является массивом инверсий, следующим способом:

- элемент n помещаем в массив, индексированный с помощью $[1, n]$, представляющий α , оставляя a_n **пустых ячеек** справа от n ; это означает в точности, что $\alpha^{-1}(n) = n - a_n + 1$;
- затем помещаем $n - 1$ в массив α , оставляя α_{n-1} **пустых ячеек** справа от $n - 1$;
- продолжаем, зная, что на k -м этапе этого процесса $k - 1$ величин уже размещены в массиве, следовательно, в массиве α остается $n - k$ свободных мест, и, с другой стороны, величина α_{n-k} строго меньше, чем $n - k$.

```

{  $\alpha[1], \dots, \alpha[n]$  } = { 1, ..., 1 }
for (int k = 2; k <= n; ++k) {
    j = n; i = 0;
    do {
        if ( $a_j == 1$ ) // (#1)
            i++;
        j--;
    } while (i != 1 +  $a_k$ ); // (#2)
     $\alpha_{j+1} = k$ ;
}

```

Алгоритм 1.15: Генерация перестановок

- #1: i есть число свободных индексов $> j$
 #2: $i = 1 + a_k$, то j свободен

В алгоритме 15 использована оптимизация: свободные места в массиве, представляющем перестановку α , отмечены числами 1, что позволяет не помещать это последнее значение в массив, представляющий α , в конце алгоритма.

24. Перебор перестановок транспозициями $(i, i + 1)$

a. Рассуждаем индукцией по n , при этом случаи $n = 1$ и $n = 2$ очевидны. С помощью перестановки σ интервала $[1, n]$ можно построить $n + 1$ перестановок $\sigma^1, \sigma^2, \dots, \sigma^{n+1}$ интервала $[1, n + 1]$, где перестановка σ^i получается включением в σ элемента $n + 1$ на i -ое место; например, если

$\sigma = (5\ 2\ 4\ 1\ 3)$, то

$$\begin{aligned}\sigma^1 &= (\underline{6}\ 5\ 2\ 4\ 1\ 3), \quad \sigma^2 = (5\ \underline{6}\ 2\ 4\ 1\ 3), \dots, \\ \sigma^5 &= (5\ 2\ 4\ 1\ \underline{6}\ 3), \quad \sigma^6 = (5\ 2\ 4\ 1\ 3\ \underline{6}).\end{aligned}$$

Если $\sigma_1, \sigma_2, \dots, \sigma_{n!}$ и есть такая последовательность перестановок на $[1, n]$, то соответствующую последовательность перестановок интервала $[1, n+1]$ получаем следующим образом:

$$\begin{aligned}\sigma_1^1, \sigma_1^2, \dots, \sigma_1^{n+1}, \quad \sigma_2^{n+1}, \sigma_2^n, \dots, \sigma_2^1 \\ \sigma_3^1, \sigma_3^2, \dots, \sigma_3^{n+1}, \quad \sigma_4^{n+1}, \sigma_4^n, \dots, \sigma_4^1 \text{ и т.д.}\end{aligned}$$

б. Действуем индукцией по n ; знаем, что b — последующий элемент для a в знакопеременном лексикографическом порядке — получается изменением *одной* компоненты. Предположим сначала, что эта компонента — последняя; тогда имеем $b_n = a_n \pm 1$ и $b_j = a_j$ для $1 \leq j \leq n-1$. Если i — индекс n в α (т.е. $\alpha(i) = n$), то имеем $\beta = \alpha \circ (i, i-1)$ в случае $b_n = a_n + 1$, и $\beta = \alpha \circ (i, i+1)$ в случае $b_n = a_n - 1$, при этом запись (j, k) означает транспозицию индексов j и k .

Теперь предположим, что компонента, по которой различаются a и b , не является последней. Поскольку b — последующий элемент для a в знакопеременном лексикографическом порядке, имеем $a_n = b_n = 0$ или $a_n = b_n = n$ и $b_{[1..n-1]}$ есть последующий элемент для $a_{[1..n-1]}$ в лексикографическом знакопеременном произведении $[0, 1[\times [0, 2[\times \dots \times [0, n-1[$, причем компонентой с самым большим индексом массива инверсий является та, которая меняется быстрее всех во время перебора в лексикографическом знакопеременном порядке. Если α' (соответственно, β') означает перестановку $[1, n-1]$, для которой $a_{[1..n-1]}$ (соответственно, $b_{[1..n-1]}$) массив инверсий, то β' получается из α' транспозицией двух последовательных элементов (гипотеза индукции). Тогда утверждение верно также и для α и β , поскольку элемент n находится в этих перестановках либо на месте n (случай $a_n = b_n = 0$), либо на месте 1 (случай $a_n = b_n = n$).

с. Пусть α — перестановка интервала $[1, n]$ и $a = (a_1, a_2, \dots, a_n)$ — ее массив инверсий. По предыдущему сигнатура перестановки α является также сигатурой a в знакопеременном лексикографическом произведении. Алгоритм вычисления последующего элемента в знакопеременном лексикографическом произведении приводит к алгоритму 16-А.

В начале тела основного цикла этого алгоритма делается попытка опустить элемент q (применить транспозицию $(\alpha^{-1}(q) - 1, \alpha^{-1}(q))$ к перестановке α) или же поднять его.

Фактически, бесполезно приниматься за предварительное вычисление массива инверсий a . Достаточно вычислить при необходимости элемент a_q , что может быть реализовано одновременно с вычислением $\alpha^{-1}(q)$ благодаря алгоритму 16-В. В этом втором алгоритме результирующим значением i является $\alpha^{-1}(q)$.

А. Первая версия

```
s = sign( $\alpha$ );
for (int q = n; q >= 1; --q) {
    s *= pow(-1, a[q]);
    // s = sign(a[1]) * ... * sign(a[q - 1])
    i =  $\alpha^{-1}(q)$ ;
    if (s == 1 && a[q] < q - 1)
        swap( $\alpha(i)$ ,  $\alpha(i - 1)$ ); break;
    else if (s == -1 && a[q] > 0)
        swap( $\alpha(i)$ ,  $\alpha(i + 1)$ ); break;
}
```

В. Вторая версия

```
i = n;
a[q] = 0;
while ( $\alpha(i) == q$ ) {
    if (q >  $\alpha(i)$ )
        a[q]++;
    i--;
}
```

Алгоритм 1.16: Генерация перестановок

25. Принцип включения-исключения или формула решета

а. Первая формула удобно получается индукцией по $|I|$, числу элементов I , с использованием хорошо известной формулы $|A \cup B| = |A| + |B| - |A \cap B|$. Вторая получается переходом к дополнениям. Чтобы получить формулу Сильвестра, достаточно записать:

$$\bigcap_{i \in I} \overline{X_i} = \overline{\bigcup_{i \in I} X_i} = X - \bigcup_{i \in I} X_i,$$

затем применить первую формулу.

б. Пусть X — множество всех перестановок на $[1, n]$ и X_i — множество перестановок, имеющих i фиксированной точкой, $1 \leq i \leq n$. Искомое число σ_n :

$$\sigma_n = \left| \bigcup_{i \in [1, n]} \overline{X_i} \right| = \sum_{J \subset [1, n]} (-1)^{|J|} \left| \bigcap_{i \in J} X_i \right|.$$

Множество $\bigcap X_i$ есть множество J перестановок на $[1, n]$ и, следовательно, содержит $(n - |J|)!$ элементов, откуда:

$$\sigma_n = \sum_{J \subset [1, n]} (-1)^{|J|} (n - |J|)! = \sum_{k=0}^n (-1)^k C_n^k (n - k)! = \sum_{k=0}^n (-1)^k \frac{n!}{k!}$$

с. Положим здесь X равным интервалу $[1, n]$ и для $1 \leq i \leq k$ пусть X_i — множество элементов из X , которые кратны p_i . Тогда имеем:

$$\phi(n) = \left| \bigcap_{i \in [1, k]} \overline{X_i} \right| = \sum_{J \subset [1, k]} (-1)^{|J|} \left| \bigcap_{i \in J} X_i \right|$$

Множество $\bigcap X_i$ здесь является множеством элементов из X , кратных $\prod_{i \in J} p_i$; но если d является делителем n , имеется точно n/d элементов из X , кратных d , откуда:

$$\phi(n) = n \sum_{J \subset [1, k]} \frac{(-1)^{|J|}}{\prod_{i \in J} p_i} = n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \dots \left(1 - \frac{1}{p_k}\right).$$

д. Обозначим через X множество всех отображений из $[1, n]$ в $[1, n]$ и для $1 \leq i \leq n$ через X_i — множество отображений из X , не имеющих i в их образе; выберем в качестве весовой функции на X функцию $p(\alpha) = a_{1\alpha(1)} a_{2\alpha(2)} \dots a_{n\alpha(n)}$: тогда нужно вычислить вес множества S_n всех перестановок на $[1, n]$. Заметим, что $S_n = \bigcap_{i \in [1, n]} \overline{X_i}$ и, значит, $\text{per} A = \sum_{J \subset [1, n]} (-1)^{|J|} p(\bigcap_{i \in J} X_i)$. Но $\bigcap_{i \in J} X_i$ есть множество всех отображений, образы которых не встречаются в J , следовательно:

$$p\left(\bigcap_{i \in J} X_i\right) = \sum_{\alpha: [1, n] \rightarrow \overline{J}} a_{1\alpha(1)} \dots a_{n\alpha(n)} = \sum_{j \notin J} a_{1j} \sum_{j \notin J} a_{2j} \dots \sum_{j \notin J} a_{nj}$$

отсюда получаем формулу $\text{per} A = \sum_{J \subset [1, n]} (-1)^{|J|} \prod_{i=1}^n \sum_{j \notin J} a_{ij}$, которая при замене J его дополнением дает формулу Райзера.

26. Произведение многочленов, заданных массивами

а. Алгоритм справа дает функцию умножения двух многочленов P и Q , где многочлен R степени $\deg P + \deg Q$ (который дает результат в конце алгоритма) должен быть предварительно инициализирован нулем.

```

for (int i = 0; i < arrlen(P); ++i)
  for (int j = 0; j < arrlen(Q); ++j)
    R[i + j] += P[i] * Q[j];

```

б. Изучая предыдущий алгоритм, устанавливаем, что его сложность, как по числу перемножений, так и сложений, равна произведению высот двух многочленов: $(\deg P + 1) \times (\deg Q + 1)$ — обычно высотой многочлена

```

for (int i = 0; i < arrlen(P); ++i)
  if (P(i) != 0)
    for (int j = 0; j < arrlen(Q); ++j)
      if (Q(j) != 0)
        R[i + j] += P[i] * Q[j]

```

Алгоритм 1.17: Перемножение
многочленов

называют число его ненулевых коэффициентов, но в этом алгоритме, который не учитывает случай нулевых коэффициентов, можно рассматривать высоту многочлена как число всех коэффициентов. Значит, возможно улучшить предыдущий алгоритм, исключив все

ненужные перемножения: это сделано в алгоритме 17. В противовес тому, что можно было бы подумать, эта оптимизация вовсе не смехотворная и активно применяется при умножении разреженных многочленов.

27. Возведение в степень многочленов, заданных массивами

а. Очень просто вычислить сложность алгоритма возведения в степень последовательными умножениями, если заметить, что когда P — многочлен степени d , то P^i — многочлен степени id . Если обозначить $C_{mul}(n)$ сложность вычисления P^n , то рекуррентное соотношение $C_{mul}(i + 1) = C_{mul}(i) + (d + 1) \times (id + 1)$ дает нам:

$$C_{mul}(n) = (d + 1) \sum_{i=1}^{n-1} id + 1 = \frac{n^2 d(d + 1)}{2} + \frac{n(d + 1)(2 - d)}{2} - (d + 1).$$

б. Что касается возведения в степень с помощью дихотомии (т.е. повторяющимся возведением в квадрат), вычисления несколько сложнее: зная P^{2^i} , вычисляем $P^{2^{i+1}}$ с мультипликативной сложностью $(2^i d + 1)^2$. Как следствие имеем:

$$\begin{aligned} C_{sq}(2^l) &= \sum_{i=0}^{l-1} (2^i d + 1)^2 = \frac{d^2(4^l - 1)}{3} + 2d(2^l - 1) + l = \\ &= \frac{d^2 n^2}{3} + 2nd + \log_2 n - 2d - \frac{d^2}{3} \end{aligned}$$

Предварительное заключение, которое можно вывести из предыдущих вычислений, складывается в пользу дихотомического возведения в степень: если n есть степень двойки (гипотеза ad hoc), этот алгоритм еще выдерживает конкуренцию, даже если эта победа гораздо скромнее в данном

контексте $(n^2d^2/3$ против $n^2d^2/2)$, чем когда работаем в $\mathbb{Z}/p\mathbb{Z}$ ($2\log_2 n$ против n).

Но мы не учли корректирующие перемножения, которые должны быть выполнены, когда показатель не является степенью 2-х. Если $2^{l+1} - 1$, нужно добавить к последовательным возведениям в квадрат перемножения всех полученных многочленов. Умножение многочлена $P^{(2^i-1)d}$ степени $(2^i-1)d$ на многочлен $P^{2^i d}$ степени $2^i d$ вносит свой вклад из $((2^i-1)d+1) \times (2^i d+1)$ умножений, которые, будучи собранными по всем корректирующим вычислениям, дают дополнительную сложность:

$$\begin{aligned} C'_{sqr}(2^{l+1} - 1) &= \sum_{i=1}^l ((2^i - 1)d + 1) \times (2^i d + 1) = \\ &= \frac{d^2 n^2}{3} - \frac{4d^2 n}{3} + 2nd - 2d^2 + d(1 - \lfloor \log_2 n \rfloor). \end{aligned}$$

Теперь можно заключить, что дихотомическое возведение в степень не всегда является лучшим способом для вычисления степени многочлена с помощью перемножений многочленов. Число перемножений базисного кольца, которые необходимы, $C_{sqr}(n)$ — в действительности заключено между $C_{sqr}(2^{\lfloor \log_2 n \rfloor})$ и $C_{sqr}(2^{\lfloor \log_2 n \rfloor}) + C'(2^{\lfloor \log_2 n \rfloor + 1} - 1)$, т.е. между $n^2d^2/2$ и $2n^2d^2/3$, тогда как простой алгоритм требует всегда $n^2d^2/2$ перемножений. В частности, если исходный многочлен имеет степень, большую или равную 4, возведение в степень наивным методом требует меньше перемножений в базисном кольце, чем бинарное возведение в степень, когда n имеет форму $2^l - 1$.

Можно пойти еще дальше без лишних вычислений: можно довольно просто доказать, что если n имеет вид $2^l + 2^{l-1} + c$ (выражения, представляющие двоичное разложение n), то метод вычисления последовательными перемножениями лучше метода, использующего возведение в квадрат (этот последний метод требует корректирующего счета ценой, по крайней мере, $n^2d^2/9$). Все это доказывает, что наивный способ является лучшим для этого класса алгоритмов, по крайней мере в половине случаев.

Действительно, МакКарти [124] доказал, что дихотомический алгоритм возведения в степень оптимален среди алгоритмов, оперирующих повторными умножениями, если действуют с плотными многочленами (антоним к разреженным) по модулю m , или с целыми и при условии оптимизации возведения в квадрат для сокращения его сложности наполовину (в этом

случае сложность действительно падает приблизительно до $n^2 d^2 / 6 + n^2 d^2 / 3 = n^2 d^2 / 2$.

28. Небольшие оптимизации для произведений многочленов

а. Алгоритм состоит просто в применении формулы квадрата суммы:

$$\left(\sum_{0 \leq i \leq n} a_i X^i \right)^2 = \sum_{0 \leq i \leq n} a_i^2 X^{2i} + \sum_{0 \leq i < j \leq n} 2a_i a_j X^{i+j},$$

что дает $n+1$ умножений для первого члена и $n(n+1)/2$ — для второго, или в целом $(n+1)(n+2)/2$ умножений, что близко к половине предусмотренных умножений, когда n большое.

б. Достаточно легко для вычисления произведения двух многочленов $P = aX + b$ и $Q = cX + d$ находим формулы $U = ac$, $W = bd$, $V = (a+b)(c+d)$ и $PQ = UX^2 + (V - U - W)X + W$, в которых появляются только три элементарных умножения, но четыре сложения (само же существование этих формул связано с записью тензорного ранга произведения многочленов, приведенной в главе IV). Можно рекурсивно применить этот процесс для умножения двух многочленов P и Q степени $2^l - 1$, представляя их в виде $P = AX^{2^{l-1}} + B$, $Q = CX^{2^{l-1}} + D$ и применяя предыдущие формулы для вычисления PQ в зависимости от A , B , C и D , где каждое произведение AB , CD и $(A+B)(C+D)$ вычисляется с помощью рекурсивного применения данного метода (это метод Карацубы). Все это дает мультипликативную сложность $M(2^l)$ и аддитивную сложность $A(2^l)$ такие, что:

$$\begin{aligned} M(2^l) &= 3M(2^{l-1}), \dots, M(2) = 3M(1), \quad M(1) = 1, \\ A(2^l) &= 3A(2^{l-1}) + 3 \cdot 2^l, \dots, A(2) = 3A(1) + 6, \quad A(1) = 1. \end{aligned}$$

В этой последней формуле член $3 \cdot 2^l$ представляет собой число элементарных сложений, необходимых, чтобы сделать два сложения многочленов степени $2^{l-1} - 1$ ($a+b$ и $c+d$) и два вычитания многочленов степени $2^l - 1$ ($U - V - W$). Суммируя каждое из этих выражений, находим для n , являющегося степенью двойки:

$$M(n) = n^{\frac{\log 3}{\log 2}} \approx n^{1,585} \quad \text{и} \quad A(n) = 7n^{\frac{\log 3}{\log 2}} - 6n.$$

К сожалению, этот принцип остается теоретическим, и на его основе нужно построить итерационный алгоритм, чтобы получить разумную эффективность (цена управления рекурсией очень велика).

29. Высота произведения двух многочленов

а. Не говоря даже об оценке стоимости умножения двух разреженных многочленов, уже очень трудно — и даже невозможно, потому что эта высота зависит только от высоты каждого из множителей, — выразить высоту произведения двух многочленов как функцию высот исходных многочленов. Однако можно довольно легко ограничить ее произведением высот исходных многочленов. Два многочлена $Q = b_{q-1}X^{q-1} + \dots + b_0$ и $P = a_{p-1}X^{(p-1)q} + a_{p-2}X^{(p-2)q} + \dots + a_0$ показывают, что эта граница может быть достигнута.

б. Положим $P = \sum_{i=1}^d a_i X^{\alpha_i}$. Вычисление P^n дает нам выражение $\sum a_{i_1} \dots a_{i_n} X^{\alpha_{i_1} + \dots + \alpha_{i_n}}$. Эта формула дает такое представление для P^n , в котором не сделано никакого упрощения и никакой группировки членов. Высота P^n дается числом членов, имеющих разные степени, и чтобы оценить ее сверху, нужно сгруппировать члены, которые очевидным образом имеют одинаковые степени. Поскольку сложение показателей неизвестного коммутативно, можно переписать P^n как сумму мономов степеней $\alpha_{i_1} + \alpha_{i_2} + \dots + \alpha_{i_n}$, где последовательность $(\alpha_{i_k})_k$ — возрастающая. Если все эти суммы различны, высота многочлена P^n будет максимальной и равной числу возрастающих отображений (в широком смысле) интервала $[1, n]$ в интервал $[1, d]$. Это число отображений хорошо известно (Берж [19]) и равно $\binom{d+n-1}{n}$.

Этот факт можно легко доказать. Действительно, рассмотрим возрастающее отображение f интервала $[1, n]$ в $[1, d]$; множество $\{f(1), 1+f(2), \dots, n-1+f(n)\}$ является частью из n элементов множества из $n+d-1$ элементов. Обратно, если рассмотреть такую часть $\{x_1 < x_2 < \dots < x_{n-1} < x_n\}$, то последовательность $x_1, x_2 - 1, \dots, x_n - n + 1$ — возрастающая со значениями в $[1, d]$.

с. Прежде чем доказать требуемый результат, сначала сформулируем небольшую лемму (которая легко доказывается индукцией): для $k \geq 1$ и $p \geq 2$ имеем $kF_p < F_{p+k}$. Доказательство максимального характера (в смысле, данном в формулировке условия) многочлена P состоит просто в том, чтобы показать, если положить $S_i = \sum F_{1+ni_k}$, что для различных $i = \{i_1, \dots, i_n\}$ и $j = \{j_1, \dots, j_n\}$ суммы S_i и S_j различны. Предположим, что множества i и j упорядочены в возрастающем порядке; мы покажем, что если $F_{1+ni_n} \neq F_{1+nj_n}$, то две суммы различны. Предположим, что $i_n > j_n$, и пусть $i_n \geq j_n + 1$. Можно применить лемму, сформулированную в начале этого вопроса, и вывести отсюда, что $F_{1+ni_n} > nF_{1+nj_n}$; числа Фибоначчи входят в суммы, расположенные в порядке возрастания, и суммы S_i и S_j содержат только n членов; отсюда непосредственно выводится, что $S_i > S_j$.

30. Представление многочленов списками

```
typedef struct Monomial {
    unsigned int Degree; // Natural
    Ring_Element Coefficient;
} Monomial;
```

```
typedef struct Term {
    Monomial The_Monomial;
    Polynomial Next_Term;
} Term;
```

Разреженный многочлен можно реализовать с помощью списка мономов, где каждый моном представляется парой (степень, коэффициент). Это индуцирует тип, который на языке Ада может быть описан следующей структурой данных. Вот некоторые понятия обработки списков в языке Ада. Мы не будем в де-

талях описывать то, что называется динамической структурой, а только наметим... (для деталей читатель может обратиться к книгам по алгоритмике, например, [181]). Однако укажем, что определение рекурсивной структуры, такой, как список, делается с помощью того, что называется неполным определением типа: ► **type** Term; ◄. Это *описание* позволяет определить тип *Polynomial* как указатель на объект типа *Term* и дополнить затем определением типа *Term*: пара, состоящая из монома и указателя на следующий моном, т.е. классическая структура списка. Так как будут представлены не все коэффициенты многочлена, а только ненулевые, то необходимо ко всякому ненулевому коэффициенту присоединить степень соответствующего монома.

После определения структуры данных остается зафиксировать ее семантику, т.е. смысл данных на языке многочленов, используемых в списках. В данной реализации первый терм списка есть моном наивысшей степени многочлена и все мономы упорядочены в порядке убывания степеней. Кроме того, многочлен нуль представляется пустым списком мономов. Из предыдущего выбора следует, в частности, что просмотр всех коэффициентов многочлена может быть замечательно организован, если начать со старшего члена и заканчивать членом степени 0. Просмотр многочлена в обратном порядке стоит дороже. Можно прибегнуть к компромиссу: если оценить, что алгоритмы действий с многочленами предпочтительнее начинать со старшей степени, то семантика хороша. Если лучше пользоваться обоими способами просмотра, то для разреженного многочлена можно использовать двойные цепочки. Эти «небольшие» несогласования есть цена, которую приходится платить при работе с разреженными многочленами, чтобы избежать разорительной структуры данных, каковой являются массивы для представления разреженных многочленов.

Какие простейшие конструкции мы собираемся использовать при реализации арифметических алгоритмов для многочленов? Очевидно, те,

которые являются основными при действиях со списками: те, что вызываются программами языка Лисп: **car**, **cdr**, **cons** и **null?**. Вот некоторые примитивы с именами и значениями теми же, что и в языке Лисп:

- *Head* дает первый элемент непустого списка,
- *Tail* дает список, полученный пропуском первого элемента непустого списка,
- *Construct* строит новый список, добавляя первый элемент к уже существующему списку,
- *Is-Null* — булевская переменная, истинная, если список пуст.

Представление многочленов списками приводит к тому, что обычное присваивание переменной типа *Polynomial* влечет за собой то, что носит название *раздел структуры*. Последнее означает, что если P — переменная типа *Polynomial*, представляющая многочлен $X^9 + X^5 + X^2$, то после выполнения действия $\triangleright Q := P \triangleleft$ не только переменная Q будет представлять формально тот же многочлен, что и P , но, кроме того, разделит с P ячейки памяти, где расположены мономы P . Если в дальнейшем потребуется прицепить к многочлену Q моном $4X$, то надо будет изменить границы многочлена P , который ввиду равенства присоединит тот же моном!

Эти функции кажутся опасными, но они используют методы контроля, допускающие эффективную реализацию. Мы исследуем это чуть позже.

В действительности эти и другие ограничения на реализуемые с вышеописанными примитивами действия делают невозможным испортить структуру (цепочки связи или компоненты), как мы увидим после описания реализации.

Вот возможная спецификация для управляющего списком настраиваемого пакета, содержащего примитивы, которые были только что представлены:

```
#ifndef LIST_HANDLER_H
#define LIST_HANDLER_H

#include <stdbool.h>

typedef struct Element;
typedef struct Item;
typedef struct List;

const List Null_List = NULL;
```

```

Element Head(List Of_The_List);
List Tail(List Of_The_List);
List Construct(Element The_Element, List And_The_List);
bool Is_Null(List The_List);

#endif

```

К этой спецификации добавлена константа *Null_List*, обозначающая пустой список. Определение этой константы раскрывается в предыдущей части. На первый взгляд есть одно исключение *List_Is_Null*, которое указывает при случае, что требуемая операция невозможна в случае отсутствия списка. С другой стороны, в предыдущей части производилась работа с определением неполного типа (которое пополнялось в теле пакета с помощью определения ► **type** *Item* **is** **record** *Element* : *Element*; *Next* : *List*; **end** **record** ◀).

Теперь реализация трех первых примитивов показывает, как использовать раздел структур. Всегда вначале идет функция *Head*, тело которой содержит только инструкцию ► **return** *Of_The_List.Element* ◀.

Функция *Tail* приводит к разделению структур, ее тело также состоит из единственной инструкции ► **return** *Of_The_List.Next* ◀. Следовательно, можно заметить, что список, возвращаемый этой функцией, скорее окончание списка, используемого в аргументе, а не копия хвоста списка. Поэтому в результате выхода инструкции такой, что ► *L2* := *Tail* (*L1*) ◀, два списка *L1* и *L2* разделяют один и тот же набор ячеек памяти. Следовательно, всякая несвоевременная модификация конца списка *L1* скажется на состоянии списка *L2*. С другой стороны, время выполнения этой функции постоянно, каков бы ни был размер исходного списка, что не имело бы места, если бы этот список копировался.

Процедура *Construct* также вызывает разделение структуры. Вот ее тело, не содержащее ничего, кроме инструкции ► **return** **new** *Item*'(*The_Element*, *And_The_List*) ◀. Эта инструкция провоцирует размещение (по **new**) элемента списка типа *Item*, затем инициализирует его, используя значение следующего агрегата. Полученный этим способом список все еще разделяет часть компонент с исходным списком.

Структура памяти, получаемой с помощью допустимых операций этого вида, представляет собой ориентированный граф, вершинами которого служат части памяти, и в котором существует дуга от одной вершины к другой, если первая вершина содержит логический указатель на вторую

вершину. Вершина этого графа будет внешней, если она не достижима из любой другой вершины. Внешние вершины соответствуют в программе **переменным**-указателям, тогда как внутренние — ячейкам списка. Ясно также, что до тех пор, пока напрямую не изменяется информация, содержащаяся во внутренних вершинах (т.е. в терминологии списков, ни одна из компонент списка не изменяется), информация не изменяется и для существующих связей и вершин, и разделение структур не является опасным. Единственные операции, которые, следовательно, разрешены, это присоединение дуг, выходящих из новых вершин и входящих в уже существующий граф — философия, которой следуют три реализованные примитива.

31. Сложение многочленов, представленных списками

Теперь можно построить алгоритм сложения двух многочленов, используя примитивы, описанные в упражнении 30, так как эти примитивы полны (и поэтому позволяют реализовать всякую функцию, вычислимую с помощью списков), хотя сами детали этого алгоритма мало интересны. Напротив, возможно, было бы наиболее интересным построить эффективный алгоритм сложения разреженных многочленов, свободный от добавления новых примитивов для списков.

Сложение двух рассматриваемых многочленов в каком-то смысле есть слияние двух списков, упорядоченных по убыванию степеней одночленов. Следовательно, надо перебрать каждый из двух списков, представляющих многочлены, и, как только мономы данной степени окажутся в каждом из представленных многочленов, их надо сложить. Как только будет исчерпан один из многочленов, ничего не останется, как прицепить к концу результирующего многочлена хвост оставшегося. Этим вводится структурное разделение между многочленом-результатом и тем из двух многочленов-операндов, порядок которого (низшая степень одночлена) является наименьшим. Но так как используются примитивы из упражнения 30, то такое разделение не опасно.

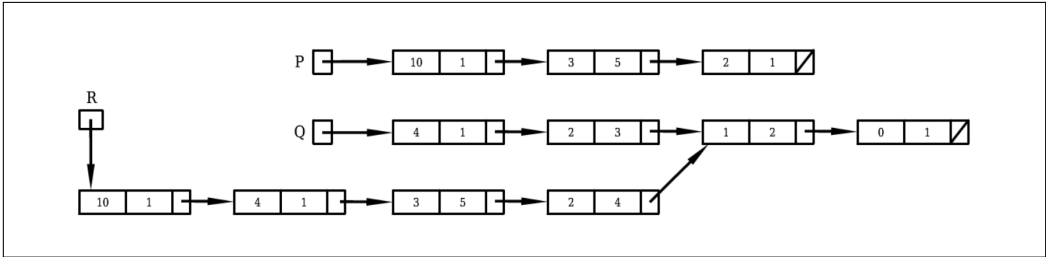


Рис 1.6: Сложение двух многочленов, представленных списками

На рис. 6 видим схему памяти, представляющую три многочлена: $P = X^{10} + 5X^3 + X^2$, $Q = X^4 + 3X^2 + 2X + 1$ и $R = P + Q$, вычисленный с помощью предыдущего метода. Каждый элемент списка на этом рисунке представлен в виде тройки (степень, коэффициент, указатель). Этот метод реализован в алгоритме 18.

Построение списка, содержащего частичные последовательные результаты, осуществляется по возрастанию степеней. Следовательно, если один из списков исчерпался, нужно перевернуть промежуточный результат прежде, чем присоединить хвост другого списка. Это делает подпрограмма *Reverse_Append*, которую можно поместить в пакет, разработанный в упражнении 30. Эта функция реализована, начиная с примитивов управления списком: *Head*, *Tail*, *Construct* и *Is_Null*.

```
List Reverse_Append(List The_List, List And_The_List) {
    List x = The_List;
    List y = And_The_List;
    while (!Is_Null(x)) {
        y = Construct(x->Element, y);
        x = Tail(x);
    }
    return y;
}
```

```
R = Null_List;
while (1) {
    if (Is_Null(Q)) return Reverse_Append(R, P);
    else if (Is_Null(P)) return Reverse_Append(R, Q);
    if (Head(Q)->Degree > Head(P)->Degree) {
        R = Construct(Head(Q), R);
        Q = Tail(Q);
    } else if (Head(P)->Degree > Head(Q)->Degree) {
        R = Construct(Head(P), R);
        P = Tail(P);
    } else {
        x = Head(P)->Coefficient + Head(Q)->Coefficient;
        if (x != 0)
            R = Construct(New_Element1(Head(P)->Degree, x), R);
        P = Tail(P);
        Q = Tail(Q);
    }
}
```

Алгоритм 1.18: Сумма двух разреженных многочленов

Порядок многочлена есть его наименьшая степень, а его высота, обо-

¹*New_Element()* - "конструктор" объекта *Element*. (Прим. редактора)

значаемая через $\#P$, — число ненулевых мономов в P . Если порядок P больше порядка Q , то сложность алгоритма сложения прямо пропорциональна выражению:

$\#P + \text{число мономов многочлена } Q \text{ степени большей, чем порядок } P$.

В этих вычислениях смешаны сложность просмотра списков и сложность вычисления арифметических операций — это вполне разумно в случае многочленов с небольшими целыми коэффициентами, но нереалистично, если арифметика коэффициентов более сложна.

32. Умножение многочленов, представленных списками

Пусть $P = \sum a_i X^{\alpha_i}$, где $\alpha_i < \alpha_{i+1}$ и Q — два многочлена, которые требуется перемножить. Их произведение может вычисляться по формуле $R = \sum a_i X^{\alpha_i} Q$, если предполагается осуществлять умножение, следуя представлению P в виде суммы одночленов. Легко видеть, что было бы удобно использовать функцию, реализующую умножение одночлена на многочлен. Эта функция легко и эффективно реализуется, если воспользоваться более общей функцией, оперирующей списками таким образом, что на основании данного списка и некоторого преобразования его элементов она выдает новый список, полученный дублированием и преобразованием исходного списка.

```
Element Transform(Element The_Element);
List Map_List(List The_List);
```

Реализация этого несколько особенного итерационного цикла — каково должно быть тело пакета *List_Handler*, если стремиться к эффективной реализации? — предлагается читателю. Чтобы получить функцию умножения одночлена на многочлен, конкретизируем функцию *Map_List* с помощью следующей процедуры преобразования¹:

```
function Multiply (M : Monomial, P : Polynomial)
  return Polynomial is
  function Multiply_M_By (M_1 : Monomial) return Monomial;
  function Result is new Map_List (Transform => Multiply_M_By);
  function Multiply_M_By (M_1 : Monomial) is
  begin
    return (Degree => M.Degree + M_1.Degree,
            Coefficient => M.Coefficient * M_1.Coefficient);
  end Multiply_M_By
begin
  return Result(P);
end Multiply;
```

¹Трудноконвертируемый в Си код. Приведен оригинал на Ada для лучшего понимания.
(Прим. редактора)

Первая функция *Multiply-M-By* позволяет умножать произвольный многочлен на параметр M функции *Multiply*, вторая функция — конкретизация на этом преобразовании данного итеративного цикла — дает искомый результат. Следует заметить, что в этом контексте не возможно выполнить конкретизацию вне функции *Multiply*, потому что она зависит от одного из аргументов этой функции. Это цена, которую приходится платить за модулярность множества. Кроме того, реализация функции *Multiply-M-By* существенно использует целостность исходного кольца; без этого предположения было бы необходимо отдельно рассматривать случаи, когда произведение двух коэффициентов равно нулю, что значительно усложнило бы алгоритм.

Имея эту элементарную функцию перемножения, можно теперь реализовать умножение многочленов в общем случае. Но, учитывая сложность алгоритма сложения, рассмотренного ранее, представляется логичным сделать так, чтобы в процессе последовательных сложений структурное разбиение было бы как можно более протяженным. Рассмотрение умножения двух многочленов, представленных в упражнении 29, показывает, что многочлен P , по которому разлагают произведение в ряд последовательных умножений, должен пробегаться в порядке возрастания степеней одночленов, тогда как представление в виде списка облегчает скорее противоположное направление чтения. Так, максимизируя структурное разделение, минимизируют сложность прохода списков, представляющих отдельные многочлены-слагаемые. Надо заметить, что нижеприведенная функция будет встроена в тело пакета управления многочленами и что, как следствие, она может использовать внутреннюю структуру многочленов.

```

function Multiply (P : Polynomial; Q : Polynomial)
    return Polynomial is
    RP : Polynomial := Reverse (P); R : Polynomial := Zero;
begin
    while not Is_Null (RP) loop
        R := R + Multiply (Head (RP).The_Monomial, Q);
        RP := Tail (RP);
    end loop;
    return R;
end Multiply

```

Можно дать оценку сложности этого алгоритма умножения двух многочленов в нескольких частных случаях, рассматривал процесс ум-

$$\begin{array}{r}
 \frac{a_1 X^{\alpha_1} Q}{a_2 X^{\alpha_2} Q} \\
 \dots\dots\dots \\
 \frac{(a_2 X^{\alpha_2} + a_1 X^{\alpha_1}) Q}{a_3 X^{\alpha_3} Q} \\
 \dots\dots\dots \\
 \frac{(a_3 X^{\alpha_3} + a_2 X^{\alpha_2} + a_1 X^{\alpha_1}) Q}{a_4 X^{\alpha_4} Q} \\
 \dots\dots\dots \\
 \frac{(a_4 X^{\alpha_4} + a_3 X^{\alpha_3} + a_2 X^{\alpha_2} + a_1 X^{\alpha_1}) Q}{\dots\dots\dots}
 \end{array}$$

монов $a_1 X^{\alpha_1} Q$ и $a_2 X^{\alpha_2} Q$; для этого нужно лишь пробежать список, представляющий второй моном, и сложение завершается присоединением конца первого монома. Затем это повторяется, когда добавляют многочлен $a_3 X^{\alpha_3} Q$ к только что полученному результату; при этом пробегают только этот последний моном, и т.д. Следовательно, каждое сложение имеет сложность, заключенную между $\#Q$ и $2\#Q$. Так как это повторяется для всех мономов из P , в целом для алгоритма умножения получается сложность, пропорциональная $\#P \times \#Q$.

Можно рассмотреть произведение левого многочлена $P = \sum_{i=0}^{p-1} a_i X^i$ на правый $Q = \sum_{j=0}^{q-1} b_j X^j$, которое даст нам максимальную сложность произведения произвольного многочлена на плотный многочлен. Здесь на каждой итерации (или промежуточном сложении) получают сложность пробега $2q-1$; с учетом повторений для всех мономов из P это дает общую сложность пробега $(-1)(2q-1)$. Пример, дающий нижнюю границу сложности для произведения этого типа, получается вычислением RQ , где $R = \sum_{k=0}^{r-1} c_k X^{kq}$. На каждой итерации сложность пробега равна q , что дает общую сложность пробега $(r-1)q$.

В первом примере для произведения PQ , если бы разлагать многочлен P по убывающим степеням, то последовательно получали бы: $q + (q-1)$, $(q+1) + (q-1)$, ..., $(q+p-2) + (q-1)$ и в целом $(p-1)(2q-1) + (p-2)(p-1)/2$. Для второго примера последовательные пробеги оценивались бы $q, 2q, 3q, \dots, (r-1)q$, и в целом $r(r-1)q/2$.

Другой типичный случай: многочлен, стоящий в произведении справа, является разреженным, а тот, который слева, — плотным (пример произведения QR). Принцип пробега, описанный в схеме, остается в силе, с той лишь разницей, что высота промежуточного результата удваивается на каждой итерации, что дает сложность пробега, равную $r + (r-1) + r + (2r-2) + r + (3r-3) + \dots + r + (q-1)(r-1)$, т.е. $(q-1)r + q(q-1)(r-1)/2$. Разложение левого многочлена Q по убыванию степеней даёт точно такой же результат. Оценим сложность алгоритма: в дальнейшем будем пренебрегать умножениями на целые константы и будем рассматривать только операции над

ножения на приведенной слева схеме. В этих примерах так называемый левый многочлен является тем, по которому осуществляется разложение в произведение двух многочленов. Первый случай соответствует умножению P на Q , когда последний многочлен, является плотным, т.е. образован из мономов с последовательно идущими друг за другом степенями. Начинают со сложения двух моно-

коэффициентами формальных рядов. Очевидно, что для вычисления i -го термина ряда B необходимо $2i$ умножений (i умножений скалярных) и $i - 1$ сложений. Знание $i - 1$ членов этого ряда и i членов формального ряда A также необходимы, что дает сложность: $i(i + 1)$ умножений и $i(i - 1)$ сложений в базовом кольце.

с. При применении к многочленам изученных в двух предыдущих задачах принципов некоторое упрощение может быть достигнуто для многочленов с конечным числом членов. Имеется также возможность реализовать эти алгоритмы для многочленов, представимых массивами (если иметь дело с плотными или не очень разреженными многочленами). Обсудим теперь модификации алгоритма возведения в степень в случае действий с многочленами, с учетом того, что алгоритм возведения многочленов в квадрат был изучен в одном из предыдущих упражнений. В сумме для i -го коэффициента, встречающейся при вычислении многочлена \mathcal{P}^n , число слагаемых, которые находятся под знаком суммы, зависит не только от n , но и от степени многочлена, над которым производится действие. Точнее, при $i > 0$:

$$b_i = \frac{1}{a_0 i} \times \sum_{j=1}^{\min(i, \deg \mathcal{P})} ((n+1)j - i) a_j b_{i-j},$$

и сложность вычисления существенно уменьшается. Действительно, суммы, позволяющие вычислить коэффициент, имеют самое большее $\deg \mathcal{P}$ членов. Следовательно, вычисление коэффициентов результирующего многочлена требует: n умножений для вычисления b_0 , i умножений в базовом кольце и $i - 1$ сложений в базовом кольце, если $i \leq \deg \mathcal{P}$, умножений и $\deg \mathcal{P}$ сложений в базовом кольце, если $i > \deg \mathcal{P}$. Таким образом, получаем общую оценку сложности, не превышающую $d^2 n$ умножений и сложений.

34. Определение нулей многочлена по модулю p^α

а. Решение этого уравнения тривиально в случае, когда a обратим по модулю n . Если a не обратим, то вычислим $d = \text{НОД}(a, n)$ и рассмотрим два случая: в лучшем b делится на d , в этом случае разделим уравнение на d и решим уравнение $a'x = b'$ по модулю n' , с обратимым a' .

```

\\ Пусть  $d \geq 0$ ,  $u$  и  $v$  такие, что  $ua + vn = d = \text{НОД}(a, n)$ 
if ( $b \% d \neq 0$ ) return NULL;
else return ( $ub/d + kn/d$ ) \\  $0 \leq k < d$ 

```

(Если b не делится на d , то уравнение не имеет решений.)

Итак, разложение по убыванию степеней многочлена, стоящего в произведении слева, дает меньшую сложность, нежели если разлагать тот же самый многочлен в обратном порядке: квадратичную в лучшем случае (плотный многочлен справа), или кубическую в худшем случае (плотный многочлен слева и разреженный справа). В любом случае разложение в порядке убывания степеней дает большую сложность.

33. Действия с формальными рядами

а. Формула $(\sum a_i X^i)^2 = \sum a_i^2 X^{2i} + \sum_{i < j} 2a_i a_j X^{i+j}$ позволяет констатировать, что вычисления квадрата ряда возможно и, что для вычисления n -го коэффициента формального ряда, являющегося результатом, необходимо n начальных коэффициентов исходного ряда. Более точно, вот список итераций для осуществления чтения i -го коэффициента:

(i) вычислить

$$S_i = a_i X^{2i} + 2 \sum_{0 \leq j < i} a_i a_j X^{i+j}$$

(тогда $\sum^2 = S_0 + S_1 + S_2 + \dots$),

(ii) прибавить S_i к многочлену (вычисленному с использованием $i - 1$ первых слагаемых), аппроксимирующему формальный результирующий ряд,

iii вывести i -й коэффициент результирующего ряда; этот коэффициент больше не будет участвовать в последующих вычислениях.

То, что было описано, есть теоретический алгоритм, показывающий выполнимость вычисления квадрата формального ряда. Осталось предложить программу, осуществляющую эти вычисления: структуру данных, управление, переменные и т.д.

Как ясно из предыдущего описания, исходный формальный ряд, коэффициенты которого читаются последовательно, надо представить динамической структурой - например, в виде списка, состоящего из пар коэффициент-степень (в действительности, в стек помещается не исходный ряд, а его копия, эффективность обязывает!). По причине однородности, коэффициенты вычисляемого ряда размещают списком, аналогичным коэффициентам, частично вычисленным. Для манипулирования этими списками используются примитивы, введенные в упражнении 30. Каждый список, участвующий в алгоритме, представляет многочлен, аппроксимирующий формальный ряд, мономы которого упорядочены по возрастанию (начиная со слагаемых наименьшей степени в списке).

```

input a;
Square_Of_A  $\leftarrow a^2 X^0$ ; Double_A  $\leftarrow 2aX^0$ ; aux  $\leftarrow 0$ ; i  $\leftarrow 0$ ;
output  $a^2 X^0$ ;
loop
  //deg(Double_A)  $\leq i$ , deg(Square_Of_A)  $\leq i$ , deg(aux)  $\leq 2i$ , ord(aux)  $\geq i$ 
  input a;
  i  $\leftarrow i + 1$ ; aux  $\leftarrow aux + (aX' \times Double\_A) \oplus a^2 X^{2i}$ ;
  output First(aux);
  Double_A  $\leftarrow Double\_A \oplus 2aX'$ ;
  Square_Of_A  $\leftarrow Square\_Of\_A \oplus First(aux)$ ;
  aux  $\leftarrow Tail(aux)$ ;
end loop;

```

Алгоритм 1.19: Квадрат формального ряда

В алгоритме 19 символ «ord» обозначает порядок формального ряда (т.е. степень монома наименьшей степени). Символ \oplus обозначает формальное сложение многочлена и монома степени, превосходящей степень многочлена или не превосходящей порядка многочлена, что с практической точки зрения означает добавление элемента к началу или к концу списка. Кроме того, обозначение $2aX^i \times Double_A$ означает список, в котором элемент с номером j является произведением элемента с тем же номером в списке $Double_A$ на моном $2aX^i$. Прimitives **input** и **output** — это основные примитивы алгоритма, работающего с потоком данных: входные данные, полученные благодаря примитиву **input**, в принципе, порождают результат, выводящийся с помощью примитива **output**. Заметим в заключение, что запоминание результирующего формального ряда $Square_Of_A$ не является необходимым для алгоритма.

b. Определим производную формального ряда $B = A^n$, полученную по правилу $B' = nA'A^{n-1}$. Если это выражение умножить на A , то получим $AB' = nA'B$. Теперь можно почленно сопоставить оба формальных ряда: AB' и $A'B$. Для члена степени $i - 1$ получим $ia_0b_i + \sum_{j=1}^i (i - j)a_jb_{i-j} = n \sum_{j=1}^i ja_jb_{i-j}$, что дает нам формулу для вычисления b_i

$$b_i = \frac{1}{a_0 i} \sum_{j=1}^i ((n+1)j - i)a_j b_{i-j};$$

она позволяет вычислить очередное значение. Детали эффективного алгоритма, осуществляющего вычисление A^n , оставляются читателю.

Можно видеть, что в любом случае алгоритм должен начинаться с вычисления НОД и коэффициентов Везу для a и n . В случае, когда a не обратим и b делится на d , каждому решению по модулю n' отвечают d решений по модулю n , отличающихся слагаемым, кратным n' . Заметим, что этот алгоритм неявно, но конкретно, обрабатывает случай, когда $a = 0$.

б. Для доказательства этого факта с помощью использования двойной индукции заметим, что $(p + 1)(p + 2) \dots (p + n - 1)(p + n) = p \cdot (p + 1)(p + 2) \dots (p + n - 1) + n \cdot (p + 1)(p + 2) \dots (p + n - 1)$, где каждое слагаемое, удовлетворяющее тому или иному предположению индукции, делится на $n!$. К тому же, в данном рекуррентном соотношении нас интересует целочисленность величины $\binom{n+p}{p}$.

с. Формула Тейлора

$$P(a + tp^\alpha) = P(a) + tp^\alpha P'(a)/1! + \dots + t^d p^{ad} P^{(d)}(a)/d!$$

доказывает первое свойство, так как вопрос а позволяет показать, что $P^{(i)}$ делится на $i!$. Доказательство необходимого и достаточного условия очень просто, оно опирается на то факт, что если разделить предыдущую формулу на p^α , то получим доказательство в одну сторону, а если умножить полученную формулу на p^α , то получится доказательство в другую сторону. Алгоритм получается немедленно:

```

E ← ∅; for a in {нули P по модулю pα} loop
  for t in {решение P(a)/pα + tP'(a) = 0 mod p1} loop
    E ← E ∪ {a + tpα};
  end loop;
end loop;

```

д. Предположим, что P имеет единственный корень a_i по модулю p^i , который поднимает корень a . Сравнение $P(a_i + tp^i) \equiv P(a_i) + tp^i P'(a_i) \equiv P(a_i) + tp^i P'(a) \pmod{p^{i+1}}$ доказывает существование и единственность такого t по модулю p , что $P(a_i + tp^i) \equiv 0 \pmod{p^{i+1}}$. Кроме того, tp^i дается формулой $tp^i = -P(a_i)P'(a)^{-1}$. Это доказывает, что P имеет один и только один корень a_{i+1} по модулю p^{i+1} , поднимающий корень a . Дальше достаточно воспользоваться индукцией.

е. В действительности сформулированное в предыдущем вопросе свойство может быть обобщено и передоказано по модулю $p^{k+\alpha}$ для корней P по модулю p^α для всякого $k \leq a$. Необходимое и достаточное условие выглядит тогда так: $P(a + tp^\alpha) \equiv 0 \pmod{p^{k+\alpha}}$ тогда и только тогда, когда $P(a)/p^\alpha + tP'(a) \equiv 0 \pmod{p^k}$. Единственное изменение

состоит в том, что надо заменить p^1 в предыдущем алгоритме на p^k и рассмотреть k как параметр алгоритма.

f. Используя алгоритмы восстановления в $\mathbb{Z}_{p^{a+k}}$ для нулей по модулю p^a , можно сконструировать алгоритм подъема корней из \mathbb{Z}_p в \mathbb{Z}_{p^n} , минимизирующий количество промежуточных этапов. Как и выше, определим корни по модулю p , что делается очень просто, если p — простое число, — подстановкой последовательно всех точек из \mathbb{Z}_p . Это первичное исследование заканчивается, когда будут найдены все корни многочлена P , их число равно степени НОД($P, X^p - X$). Далее поднимаем эти корни в $\mathbb{Z}_{p^2}, \mathbb{Z}_{p^4}, \dots$, каждый раз удваивая показатель p . Этот процесс заканчивается, когда будет получена достаточно большая степень p , не превосходящая p^n , и далее надо применить алгоритм для значения k , которое не обязательно максимально. Например, для перехода от \mathbb{Z}_p к $\mathbb{Z}_{p^{57}}$ вычисляются решения по модулям $p^2, p^4, p^8, p^{16}, p^{32}$ и, наконец, p^{57} при $k = 25$.

35. Циклическая перестановка элементов массива

a. Алгоритм является простым обобщением классического алгоритма обмена значениями двух переменных.

```
Aux ← T(n - 1)
for (unsigned int i = n - 1; n-- > 1;) {
    T(i) ← T(i - 1)
}
T(0) ← Aux;
```

b. Можно показать, что если $i + kp = i + k'p \bmod n$, то $k - k'$ делится на n/d , где d есть НОД n и p . Следовательно мощность множества индексов есть n/d .

Ротация (т.е. циклическая перестановка) элементов массива это, в действительности, сдвиги индексов и, по доказанному выше, эти сдвиги определяют *орбиты*, являющиеся общими инвариантами трансляций, длины n/d , элементы которых находятся на расстоянии p . Следовательно, осуществление ротации массива проходит по-орбитно. Это дает нам алгоритм:

```
Aux ← T(n - 1)
for (unsigned int Orbit = 0; Orbit < n; Orbit++) {
    Index ← (Orbit - p) mod n; Aux ← T(Index);
    for (unsigned int k = 1; k < n/d; k++) {
        T(Index) ← T((Index - p) mod n); Index ← (Index - p) mod n;
    }
    T(Index) ← Aux;
}
```

Сложность полученного алгоритма, следовательно, составляют $n + \text{НОД}(n, p)$ присваиваний элементов массива, что дает pn или $(n - p)n$ необходимых присваиваний, если повторно применять алгоритм из вопроса а.

с. Обмен двух участков (дополнительных) массива сводится к повороту таблицы в лучшую сторону в смысле некоторого числа позиций. Теперь достаточно применить предыдущий алгоритм.

36. Элементарные операции в арифметике повышенной точности

а. Алгоритм 20 – А получается непосредственно. Важным моментом является управление переносом.

А. Сложение

В. Вычитание

```

carry ← 0;
for (unsigned i = 0; i <= m; i
  ++ ) {
  aux ← ui + vi + carry;
  if (aux < b) {
    wi ← aux; carry ← 0;
  } else {
    wi ← aux - b; carry ← 1;
  }
}
if (carry == 1) { // u + v ≥ bm+1
  wm+1 ← 1
}

```

```

carry ← 0;
for (unsigned i = 0; i <= m; i
  ++ ) {
  aux ← ui - vi - carry;
  if (aux < 0) {
    wi ← aux + b; carry ← 1;
  } else {
    wi ← aux; carry ← 0;
  }
}
if (carry == 1) {
  u < v  и w = bm+1 + u - v
}

```

Алгоритм 20. Вычисление $(u_m \dots u_0)_b \pm (v_m \dots v_0)_b$

б. В полном алгоритме 20 – В, если $u \geq v$, то легко доказать по индукции, что $\text{carry} = 0$ или 1 и $0 \geq (u_i - v_i - \text{carry}) + b < 2b$.

Что делать, если $u < v$? В этом случае можно проверить, что вычисляемое число представляет $b^{m+1} + u - v$ и то, что $u < v$ может быть обнаружено после цикла проверкой текущего значения; до того, как цикл будет проверять текущее значение.

с. Алгоритм 21 основывается на следующем соотношении:

$$(u_{i+1} \dots u_0) \times v = (u_i \dots u_0) \times v + u_{i+1} v b^{i+1}.$$

Можно доказать, что в этом алгоритме являются цифрами ввиду соотношений: $0 \leq \text{carry} < b$, $\text{aux} \leq (b-1)b < b$. Этот алгоритм легко распространяется на (наивное) умножение двух чисел произвольной длины.

```

carry=0;
for (i=0; i <= m; ++i){
    // (w_{i-1} + carry * b' = (u_{i-1} ... u_0)_b * v
    aux = v*u_i+carry; w_i=aux%b; carry=aux/b;
}
w_{m+1}=carry;

```

Алгоритм 1.20: Вычисление $v \times (u_m \dots u_0)_b$, $0 \leq v < b$

d. Остаток $r = (u_n, \dots, u_0)_b \bmod v$, где v — цифра, вычисляется по алгоритму Горнера, где полагается $r_{n+1} = 0$ и $r_i = (r_{i+1} \times b + u_i) \bmod v$ при $i = n, n-1, \dots, 0$. Искомое значение есть r_0 , что доказывается с помощью $r_i = (u_n \dots u_i)_b \bmod v$. Можно видеть, что алгоритм 22-А вычисляет частное $\lfloor u/v \rfloor$, а алгоритм 22-В вычисляет остаток $u \bmod v$. Распространение на числа произвольной длины этих двух операций не является таким про-

<u>А. Вычисление частного</u>	<u>В. Вычисление остатка</u>
<pre> r = 0; for (j = n; j >= 0; j--){ aux = r*b + u_n; q_j = aux/v; r = aux % v; } </pre>	<pre> r = 0; for (j = n; j >= 0; j--){ r = (r*b + u_j) % v; } </pre>

Деление $(u_m \dots u_0)_b$ на v при $0 \leq v < b$

стым, как в случае умножения. В действительности, это тема дальнейших упражнений.

37. Деление в арифметике повышенной точности

Предположим, что v имеет *нормализованную* запись (т.е. $v_m \neq 0$) и, кроме того, $n \geq m$ (запись не обязательно нормализованная). Надо найти частное и остаток с помощью массива $q = (q_{n-m} \dots q_1 q_0)$ и $r = (r_m \dots r_1 r_0)$. Заметим, что условие $u/v < b$ можно записать также в виде $(u_{m+1} \dots u_1)_b < (v_m \dots v_1 v_0)_b$. В самом деле,

$$(u_{m+1} \dots u_0)/v < b \Leftrightarrow (u_{m+1} \dots u_0)/b < v \Leftrightarrow \lfloor (u_{m+1} \dots u_0)/b \rfloor < v$$

и $\lfloor (u_{m+1} \dots u_0)/b \rfloor = (u_{m+1} \dots u_1)$. Вот общий алгоритм вычисления пары (частное, остаток):

```

(rn+1rn...r1r0 = (0un...u1u0;
for (j = n-m; j >= 0; j--){
  \ \ u = (qn-m...qj+1) × vbj+1 + r и r < vbj+1
  qj = (rm+j+1rm+j...rj)/v;
  (rm+j+1...rj = (rm+j+1...rj) - qjv;
}
```

Частное, используемое в цикле, является частным двух чисел, удовлетворяющих порождающему условию (5) (и, следовательно, каждое вычисляемое q_j является цифрой!). Чтобы убедиться в этом, достаточно заметить, что перед присваиванием q_j имеем:

$$r < vb^{j+1} \iff (r_{m+j+1} \dots r_j)_b < v,$$

что, по предыдущему замечанию, равносильно $(r_{m+j+1} \dots r_j)/v < b$

38. Вычисление частного методом проб и ошибок

а. Предположения (5) включают неравенство $(u_{m+1} \dots u_1)_b < (v_m \dots v_1 v_0)_b$, откуда $u_{m+1} \leq v_m$. Читатель может проверить, что тогда:

$$\min(\lfloor (u_{m+1}b + u_m)/v_m \rfloor, b-1) = \begin{cases} b-1, & \text{если } u_{m+1} = v_m, \\ \lfloor (u_{m+1}b + u_m)/v_m \rfloor & \text{иначе,} \end{cases}$$

и проверка для определения наименьшего из двух чисел сводится к проверке $u_{m+1} = v_m$

Докажем сначала, что $q \leq \hat{q}$. Так как $q \leq b-1$ (предположение (5)), можно очевидно предположить, что $\hat{q} = \lfloor (u_{m+1}b + u_m)/v_m \rfloor$. Запишем:

$$u = u_{m+1}b^{m+1} + u_mb^m + \alpha, \quad v = v_mb^m + \beta,$$

с $\beta < b^m$ и $\alpha < b^m$. Тогда получим $u - \hat{q}v = (u_{m+1}b + u_m - \hat{q}v_m)b^m + \alpha - \hat{q}\beta$. По определению \hat{q} $u_{m+1}b + u_m < v_m(\hat{q}+1)$ и, следовательно, $u_{m+1}b + u_m - u_m\hat{q} \leq v_m - q$, что влечет

$$u - \hat{q}v \leq (v_m - 1)b^m + \alpha - \hat{q}\beta \leq (v_m - 1)b^m + \alpha \leq v_mb^m + (\alpha - b^m) < v_mb^m \leq v,$$

и неравенство $u - \hat{q}v < v$ дает $q \leq \hat{q}$.

Для доказательства второго неравенства введем $\delta = \hat{q} - q$. Используя неравенство для \hat{q} : $\hat{q} \leq \frac{u_{m+1}b + u_m}{v_m} \leq \frac{u}{v_mb^m}$ и для q : $q > \frac{u}{v} - q$,

получаем:

$$\delta = \hat{q} - q \leq \frac{u}{v_m b^m} - \frac{u}{v} + 1 = \frac{u v - v_m b^m}{v v_m b^m} + 1 \leq \frac{u}{v} \frac{b^m}{v_m b^m} + 1,$$

откуда $\delta v_m \leq \frac{u}{v} + v_m$ и, следовательно,

$$\delta v_m \leq \left\lceil \frac{u}{v} \right\rceil + v_m = \hat{q} - \delta + v_m \leq b - 1 - \delta + v_m.$$

Отсюда получаем окончательно $\delta \leq (b + v_m - 1)/(v_m + 1)$.

Эффективная реализация деления требует уже использования деления числа из двух цифр на число из одной цифры.

б. Записав: $u = u_{m+1}b^{m+1} + u_m b^m + u_{m-1}b^{m-1} + \alpha, v = v_m b^m + v_{m-1}b^{m-1} + \beta$ с $\alpha < b^{m-1}$ и $\beta < b^{m-1}$, получаем:

$$\begin{aligned} u - \hat{q}v &= ((u_{m+1}b + u_m - \hat{q}v_m)b + u_{m-1} - \hat{q}v_{m-1})b^{m-1} + \alpha - \hat{q}\beta = \\ &= (\hat{r}b + u_{m-1} - \hat{q}v_{m-1})b^{m-1} + \alpha - \hat{q}\beta. \end{aligned}$$

В случае (i) $\hat{r}b + u_{m-1} - \hat{q}v_{m-1} \leq -1$, что дает:

$$u - \hat{q}v \leq -b^{m-1} + \alpha - \hat{q}\beta \leq -b^{m-1} + \alpha < -b^{m-1} + b^{m-1} = 0,$$

а неравенство $u - \hat{q}v < 0$ дает $q \leq \hat{q} - 1$. В случае (ii) $\hat{r}b + u_{m-1} - \hat{q}v_{m-1} \geq 0$ и, следовательно,

$$u - \hat{q}v \geq \alpha - \hat{q}\beta \geq -\hat{q}\beta \geq -b \times b^{m-1} \geq -v_m b^m \geq -v.$$

Тогда неравенство $u - (\hat{q} - 1)v \geq 0$ дает $q \geq \hat{q} - 1$. Алгоритм, изобра-

женный справа, показывает, как интерпретировать случаи (i) и (ii) вопроса **б**: они позволяют быстро корректировать (вычисления, которые здесь участвуют, относятся к числам из двух цифр) **начальную** оценку способом получения **исправленной**

```

 $\hat{q} \leftarrow \min \left( \left\lceil \frac{u_{m+1} + u_m}{v_m} \right\rceil, b - 1 \right);$ 
 $\hat{r} \leftarrow u_{m+1}b + u_m - \hat{q}v_m;$ 
while ( $q \leq \hat{q}$ ) {
    if ( $\hat{q}v_{m-1} \leq b\hat{r} + u_{m-1}$ ) break;
     $// q \leq \hat{q} - 1$ 
     $\hat{q} \leftarrow \hat{q} - 1; \hat{r} \leftarrow \hat{r} + v_m;$ 
}
 $// q = \hat{q}$  или даже  $q = \hat{q} - 1$ 

```

оценки \hat{q} , где $q \in \{\hat{q}, \hat{q} - 1\}$. К примеру, зафиксируем $b = 10, m = 2, u = (u_3 u_2 00)_{10}$ и $(q + 1)v_1 \leq 10(10u_3 + u_2 - (q + 1)v_2)$, тогда исправленная оценка равна начальной оценке, которая равна $q + 1$ (вместо q). Например, если $u = 5000$, то возможные значения v , для которых исправленная оценка равна $q + 1$, следующие:

626, 627, 628, 629, 715, 716, 717, 718, 719, 834, 835, 836, 837, 838, 839.

с. Если r обозначает остаток от деления u на v , то имеем $0 \leq r < b^{m+1}$, откуда следует, что достаточно определить $r \bmod b^{m+1}$. Упражнение 36, для чисел u и \hat{q} веса $\leq m+1$, позволяет реализовать одновременно вычисление проверки $u \geq \hat{q}v$ и вычисление $(u - \hat{q}v) \bmod b^{m+2}$, следовательно и вычисление $(u - \hat{q}v) \bmod b^{m+1}$. Кроме того, хотя участвующие числа удовлетворяют соотношениям для весов $\leq m+1$, внимательное изучение показывает, что можно вычислить $(u - \hat{q}v) \bmod b^{m+1}$ в массиве $\rho = (\rho_m, \rho_{m-1}, \dots, \rho_0)$ и реализовать проверку $u \geq \hat{q}v$. Тогда (q, r) можно определить следующим образом

$$\begin{aligned} \text{если } u \geq \hat{q}v, & \quad q = \hat{q} & \quad r = \rho; \\ \text{если } u < \hat{q}v, & \quad q = \hat{q} - 1, & \quad r = (\rho + v) \bmod b^{m+1}. \end{aligned}$$

Получили алгоритм 23.

```

if  $u_{m+1} = v_m$   $\hat{q} \leftarrow b - 1$ ; else  $\hat{q} \leftarrow \left\lfloor \frac{u_{m+1}b + u_m}{v_m} \right\rfloor$ ;
 $\hat{r} \leftarrow u_{m+1}b + u_m - \hat{q}v_m$ ;
for (;;;) {
  if  $(\hat{q}v_{m+1} \leq b\hat{r} + u_{m-1})$  break;
   $\hat{q} \leftarrow \hat{q} - 1$ ;  $\hat{r} \leftarrow \hat{r} + v_m$ ;
}

```

Одновременное вычисление $\rho = (u - \hat{q}v) \bmod b^{m+1}$ и проверка $u \geq \hat{q}v$

```

if  $(u \geq \hat{q}v)$ 
   $(q, r) \leftarrow (\hat{q}, \rho)$ ;
else
   $(q, r) \leftarrow (\hat{q} - 1, (\rho + v) \bmod b^{m+1})$ ;

```

Алгоритм 1.21: Частные и остатки от деления $(u_{m+1} \dots u_0)$ на $(v_m \dots v_0)$, где $u/v < b.n \geq 1$

39. Деление: операция нормализации

а. Известно, что $0 \leq \hat{q} - q \leq (b + v_m - 1)/(v_m + 1)$. Для получения $\hat{q} - q \leq 2$ достаточно иметь $(b + v_m - 1)/(v_m + 1) < 3$, что, после проверки эквивалентно $v_m \geq [b/2] - 1$.

б. Исправленная оценка \hat{q} удовлетворяет соотношению $\hat{q}v_{m-1} \leq b\hat{r} + u_{m-1}$. т.е.:

$$\hat{q}v_{m-1} \leq b(u_{m+1}b + u_m - \hat{q}v_m) + u_{m-1},$$

что влечет: $(v_mb + v_{m-1})\hat{q} \leq u_{m+1}b^2 + u_mb + u_{m-1} \leq u$. Отсюда следует:

$$u - \hat{q}v \geq u(1 - \frac{v}{v_mb^m + v_{m-1}b^{m-1}}).$$

Однако,

$$1 - \frac{v}{v_m b^m + v_{m-1} b^{m-1}} = \frac{v_m b^m + v_{m-1} b^{m-1} - v}{v_m b^m + v_{m-1} b^{m-1}} = -\frac{v_{m-2} b^{m-2} + \dots + v_0}{v_m b^m + v_{m-1} b^{m-1}}$$

$$> \frac{-b^{m-1}}{v_m b^m + v_{m-1} b^{m-1}} \geq \frac{-b^{m-1}}{v_m b^m} = \frac{-1}{v_m b}$$

и, следовательно, $\frac{u - \hat{q}v}{bv_m} > -u$. Оценка \hat{q} по предположению отлична от q , так что $q = \hat{q} - 1$ и поэтому

$$u - qv = v + u - \hat{q}v > v - \frac{u}{v_m b} = v(1 - \frac{u}{v} \frac{1}{v_m b}),$$

и, наконец, используя то, что $u/v \leq q + 1 = \hat{q} \leq b - 1$, и предположение $v_m \geq [b/2]$, имеем:

$$u - qv > v(1 - \frac{b-1}{v_m b}) \geq v(1 - \frac{2}{b}).$$

с. Ясно, что u' имеет вес $\leq n + 1$, так как $[b/(v_m + 1)]$ — цифра, и осталось доказать, что v' имеет вес m . Рассмотрим сначала частный случай $m = 1$:

$$1 \leq v \leq b - 1 \implies [b/2] \leq v[b/(v + 1)] \leq b - 1 \text{ (границы достижимы).}$$

Из $v[b/(v + 1)] < (v + 1)[b/(v + 1)] \leq b$ получаем правое неравенство. Докажем левое неравенство. Так как всегда $v[b/(v + 1)] \geq v$, то можно предположить, что $v < [b/2]$. Тогда: $v[b/(v + 1)] > v(\frac{b}{v+1} - 1) = f(v)$. Но $f(v) - f(1) = (v - 1)\frac{b/2 - v - 1}{v + 1}$. Поэтому:

$$v < [b/2] \implies v \leq b/2 - 1 \implies f(v) - f(1) \geq 0 \implies f(v) \geq f(1) = b/2 - 1,$$

откуда $v[b/(v + 1)] > [b/2] - 1$, что и требовалось доказать.

Перейдем к общему случаю. С одной стороны:

$$v' = [b/(v_m + 1)]v < [b/(v_m + 1)](v_m b^m + b^m) = [b/(v_m + 1)](v_m + 1)b^m < b^{m+1},$$

что и доказывает, что старшая цифра у v' не меньше $[b/2]$.

Из деления u' на v' можно вывести деление u на v . Действительно, если $u' = v'q' + r'$ и $u = vq + r$, то $q = q'$ и $r = r'/d$ (имеется ввиду точное деление числа на цифру). Окончательный алгоритм деления получается объединением алгоритмов, изученных в предыдущих упражнениях.

40. Самовоспроизводящаяся программа

Решением задачи является следующая программа:

```
with Text_IO; use Text_IO;
procedure R is
  procedure P (S : String) is
    use Ascii;
  begin
    Put_Line (S);
    Put_Line ('(' & Quotation & S & Quotation & %); end R; %);
  end P;
begin
  — специальная строка
end R;
```

В этой программе 10 содержит весьма длинную инструкцию (из которой можно удалить все ненужные пробелы):

```
P("WITH Text_IO;USE Text_IO;PROCEDURE R IS PROCEDURE
P(S:String)IS USE ASCII;BEGIN Put_Line(S);
Put_Line('(' & Quotation & S & Quotation & %);end R; %);
END P; BEGIN P");
```

Если нужна программа с действительно более короткими строками (строка "10" содержит около 160 символов), то это, разумеется, можно сделать, хотя программа станет чуть длиннее. Символ " не может просто появиться в цепочке - аргументе *P* в строке 7, там была использована константа *Quotation*, определенная в пакете *Ascii*. Наконец, была использована одна из возможностей языка Ада, которая позволяет разграничивать цепочки символов со знаком % если не располагаем кавычками. Все использованные уловки только укорачивают программу, но не делают ее возможной. Можно заметить, что эта программа не воспроизводит знак за знаком оригинальный исходный текст. Она порождает программу, которая эквивалентна. И пусть этот недостаток кажется неприемлемым, зато идея проста: эта программа порождает программу *P*, которая порождает *P*..

6. Глава II

7. Евклид и основная теорема арифметики

Вычислить наибольший общий делитель двух целых чисел a и b . Такое задание иногда получают лицеисты на уроке математики. Они начинают с разложения чисел a и b в произведение степеней простых чисел (используя при необходимости нулевые показатели для степеней простых чисел, чтобы выровнять количество простых чисел в разложениях):

$$a = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_m^{\alpha_m} (\alpha_i \geq 0), \quad b = p_1^{\beta_1} p_2^{\beta_2} \dots p_m^{\beta_m} (\beta_i \geq 0) \quad (p_i \text{ простое}),$$

Потом они применяют хорошо известную формулу, которая дает наибольший общий делитель (НОД):

$$\text{НОД}(a, b) = p_1^{\inf(\alpha_1, \beta_1)} \times p_2^{\inf(\alpha_2, \beta_2)} \times \dots \times p_m^{\inf(\alpha_m, \beta_m)}.$$

Легко используемый в работе с малыми числами (например, $a = 84 = 2^2 \times 3 \times 7$, $b = 198 = 2 \times 3^2 \times 11$, что дает $\text{НОД}(84, 198) = 2 \times 3 = 6$), этот метод быстро становится неприемлемым для больших чисел. Рассмотрим, к примеру $a = 1100005423$ и $b = 1100000077$. Разложение этих двух чисел в произведение простых множителей с помощью обыкновенного калькулятора, требующее примерно 10^4 делений, убеждает, что приведенная выше формула совершенно бесполезна.

К счастью, 22 века назад греческий математик Евклид открыл эффективный метод вычисления НОД. Этот метод, известный как *алгоритм Евклида*, настолько фундаментальный, что слово *алгоритм* используется математиками (помимо своего обычного смысла в информатике) для выявления делимости в некоторых кольцах. Можно с полным основанием считать Евклида предшественником алгоритмической алгебры.

Мы продемонстрируем этот алгоритм на примере, рассмотрение которого не привело лицейстов к успеху.

Выполним евклидово деление (вводимое в начальной школе и состоящее в нахождении частного и остатка) числа $a = 1100005423$ на число $b = 1100000077$, т.е. запишем $a = bq_1 + r_2$, где $q_1 = 1$ и $r_2 = 5346$. Осуществим аналогичный шаг с b и r_2 , что приводит к $b = r_2q_2 + r_3$ с $q_2 = 205761$ и $r = 1771$. Продолжим затем таким же образом с r_2 и r_3 и т.д. В результате получим следующую таблицу евклидовых делений:

$$\begin{array}{ll} r_0 = 1100005423, & r_1 = 1100000077, \\ r_1 = 1100000077, & r_2 = 5346, \\ r_2 = 5346, & r_3 = 1771, \\ r_3 = 1771, & r_4 = 33, \\ r_4 = 33, & r_5 = 22, \\ r_5 = 22, & r_6 = 11, \end{array}$$

$$\begin{array}{ll} 1100005423 = 1100000077 \times 1 + 5346 & (q_1 = 1), \\ 1100000077 = 5346 \times 205761 + 1771 & (q_2 = 205761), \\ 5346 = 1771 \times 3 + 33 & (q_3 = 3), \\ 1771 = 33 \times 1 + 11 & (q_4 = 53), \\ 33 = 22 \times 1 + 11 & (q_5 = 1), \\ 22 = 11 \times 2 + 0 & (q_6 = 2). \end{array}$$

Алгоритм заканчивает работу после получения нулевого остатка при делении 22 на 11. Последний полученный ненулевой остаток $r_6 = 11$ является НОД чисел a и b . Чтобы убедиться в этом, надо, с одной стороны, увидеть общее равенство (становящееся однородным с обозначением $r_0 = a, r_1 = b, r_7 = 0$): $r_{i-1} = r_i q_i + r_{i+1}$ для $0 \leq i \leq 6$ и, с другой стороны, воспользоваться свойством натуральных чисел: $d|bq + r$ и $d|b$ тогда и только тогда, когда $d|bid|r$. Эта последняя эквивалентность приводит, в частности, к равенству $\text{НОД}(bq + r, b) = \text{НОД}(b, r)$. Из этого следует, что величина $\text{НОД}(r_i, r_{i+1})$ не зависит от i . При $r = 0$ она равна $\text{НОД}(a, b)$, а при $i = 6$ — $\text{НОД}(r_6, 0) = r_6 = 11$, что подтверждает результат, полученный выше.

Замечание. Бели необходимо применить первый метод вычисления НОД $a = 1100005423$ и $b = 1100000077$, то надо разложить эти два числа в произведение простых множителей. Поиск простых делителей для разложения a и b и найденный общий простой делитель 11 приводят к констатации, что числа $a/11$ и $b/11$ оба являются простыми. Эта последняя проверка требует при использовании наивного метода лицейстов приблизительно $2(\sqrt{10^8}/2) = 10^4$ делений (каждое из двух чисел $a/11$ и $b/11$ имеют порядок 10^8). Разложение на простые множители $a = 11100000493, b = 11100000007$ снова дает $\text{НОД}(a, b) = 11$,

но как это далеко от итераций алгоритма Евклида! Зато первый метод дает те сведения, которые не дает второ

Вот второй аргумент, доказывающий, что $r_6 = 11$ есть НОД(a, b). Хотя он очень похож на первый, приведенный выше аргумент, но имеет одно преимущество. Он выделяет незамеченное в прошлом понятие, а именно *соотношение Безу*. Заметим, что $bq +$ и b — линейные комбинации b и r и наоборот (речь идет о линейных комбинациях с коэффициентами из \mathbb{Z}).

Если обозначить через $\mathbb{Z}(bq + r) + \mathbb{Z}b$ множество линейных комбинаций $bq +$ и b с коэффициентами из \mathbb{Z} , то получим равенство множеств $\mathbb{Z}(bq + r) + \mathbb{Z}b = \mathbb{Z}b + \mathbb{Z}r$. Множества $\mathbb{Z}_{r_i} + \mathbb{Z}_{r_{i+1}}$ являются одними и теми же. В частности, имеем $\mathbb{Z}_{r_i}a\mathbb{Z}a + \mathbb{Z}_{r_i}b = \mathbb{Z}_{r_i}r$, что соответствует отношениям $r_6 \mid a$, $r_6 \mid b$ и $r_6 \in \mathbb{Z}a\mathbb{Z}b$. Используя существование целых чисел u и v (которые мы и не старались вычислить), получаем $r_6 = ua + vb$. Легко проверить, применяя последнее равенство, что $\delta \mid r_6$ равносильно $\delta \mid a$ и $\delta \mid b$. Это снова доказывает, что $r_6 = \text{НОД}(a, b)$

Замечание. Сложение оставляет на месте множество $\mathbb{Z}b + \mathbb{Z}r$. То же верно для умножения на элементы из \mathbb{Z} . Математики называют такое множество идеалом. Это основное понятие, к которому мы будем неоднократно обращаться в дальнейшем.

Надо отметить последний важный пункт, который позволяет убедиться, что все лицеисты Франции и Наварры находят тот же самый результат, когда вычисляют НОД с помощью первого метода. Правильность их метода основывается в действительности на следующем результате (и это еще надо доказать):

(1) Теорема основная теорема арифметики

Всякий элемент из \mathbb{N}^ разлагается на простые множители. Это разложение однозначно с точностью до порядка простых сомножителей.*

Теперь декорации готовы. В последующих сценах мы введем и свяжем различные концепции: евклидово деление, НОД, соотношение Безу, разложение на простые множители, идеалы...

8. Обобщение арифметики целых чисел

Мы коснемся теперь общих понятий теории делимости. Это предполагает введение точных определений основных понятий, без которых математик не может работать, и выявление их основных свойств. Что бы избежать появления длинного списка определений/утверждений, мы выбрали

в этом разделе конкретный пример для изучения — кольцо целых чисел Гаусса, сообщая предварительно минимальное количество сведений, позволяющих работать с этим объектом. Другие результаты, относящиеся к свойствам делимости, будут изложены в следующем разделе.

Сразу же уточним, что определения, которые последуют, ориентированы на теорию делимости, отдающую предпочтение элементам. В общих чертах будут рассмотрены алгебраические структуры, в которых **основная теорема арифметики** справедлива для их **элементов**. Существуют и другие теории делимости (кольца Дедекинда), в них основная теорема арифметики справедлива для **идеалов**.

8.1. Делимость и неприводимые элементы

(2) Определение Элемент x коммутативного и унитарного¹ кольца A называется **единицей** A и л и обратимым в A , если найдется такой элемент \in , что $= = 1$. Множество всех единиц в A является **мультипликативной группой** и обозначается $U(A)$

(3) Определение

(i) Элемент a кольца A делит b (в A), если существует $c \in A$ та кой, что $=$. Будем говорить также, что b кратно a , и отмечать этот факт в виде $a \mid b$; или, если хотим уточнить кольцо A , $a \mid_A b$. Это свойство делимости может быть эквивалентным образом выраже но в терминах идеалов (обратим внимание на перевернутость « \subset » по отношению к « \mid ») следующим образом: $a \mid b$ эквивалентно $Ab \subset Aa$.

(ii) Заметим относительно свойства делимости « \mid », что два элемента a и b , удовлетворяющие равенству $Aa = Ab$, неразличимы. Это отноше ние между a и b является эквивалентностью. В этом случае мы говорим, что a и b ассоциированные элементы, и применяем запись $a \sim b$, или, если требуется уточнить кольцо A , $a \sim_A b$. Если кольцо A без делителей

¹В этой книге (за исключением беглого упоминания других ситуаций) все рас сматриваемые кольца предполагаются коммутативными и унитарными (т.е. с еди ницей. — Прим. ред). Добавим, что для «хорошей» теории делимости кольца долж ны быть целостными. Мож но было бы попытаться ограничиться только целостными кольцами, но такая точка зрения чересчур стеснительна в отношении таких поня тий, как нётеров характер, простой идеал или максимальный идеал... Использо вание свойства целостности (без делителей нуля) или необязательно целостности будет уточняться по мере необходимости.

нуля, то оба элемента равны с точностью до обратимого элемента, т.е. существует $\varepsilon \in U(A)$, такой, что $a = \varepsilon b$.

(iii) Элемент $p \in A$ называется **неприводимым**, если он не является ни нулевым, ни обратимым и если его единственными делителями являются 1 и p . Более точно: p неприводим тогда и только тогда, когда $d \mid p \Rightarrow d \sim 1$ или $d \sim p$. Это свойство равносильно тому, что идеал Ap является максимальным в множестве **однопорожденных** (т.е. главных) идеалов в A , отличных от A и $\{0\}$.

(iv) Если кольцо A без делителей нуля, то p — неприводимый элемент тогда и только тогда, когда из $p = de$ следует, что d или e — обратимый элемент в A .

9. 1.2 Что такое факториальное кольцо?

Теперь можно дать точное определение структуры колец, обобщающей в какой-то мере структуру кольца \mathbb{Z} целых чисел.

Определение 5 (факториальность)

Кольцо A называется **факториальным**, если оно не имеет делителей нуля и если оно удовлетворяет основной теореме арифметики, что, формально, выражается следующим образом:

(i) Наличие разложения на неприводимые множители:

$$\forall a \in A^* \setminus U(A), \exists p_1, p_2, \dots, p_n, \text{ неприводимые} \\ \text{и такие, что } a = p_1 p_2 \dots p_n.$$

(ii) Единственность такого разложения:

$$p_1 p_2 \dots p_n \sim_A q_1 q_2 \dots q_m \text{ (} p_i, q_j \text{ неприводимые)} \Rightarrow \\ n = m \text{ и } \exists \sigma, \text{ перестановка на } [1, n], \text{ такая, что} \\ p_i \sim_A q_{\sigma(i)} (1 \leq i \leq n).$$

В дальнейшем мы рассмотрим критерии, позволяющие распознать факториальность кольца. В настоящий момент читатель должен поверить, что это понятие не «бессодержательное», т.е. класс факториальных колец сравнительно широк. Прежде, чем продолжать дальнейшее изучение делимости, укажем одно арифметическое приложение факториальности.

9.1. 1.3 Обобщать арифметику целых чисел: зачем?

Абстрагирование от конкретной ситуации позволяет математику отследить влияние основных параметров, влияющих на ситуацию, и диапазон их изменения. Более того, часто происходит так, что свойства обобщенной ситуации оказываются полезными для изучения базовой ситуации. Вот пример, когда арифметика подкольца кольца \mathbb{Z} оказывает влияние на арифметику в \mathbb{Z} .

Рассмотрим для этого кольцо целых гауссовых чисел $\mathbb{Z}[i] = \mathbb{Z} + i\mathbb{Z} = \{x + iy | (x, y) \in \mathbb{Z} \times \mathbb{Z}\}$. Очевидно, $\mathbb{Z}[i]$ — подкольцо поля \mathbb{C} комплексных чисел, выдерживающее операцию сопряжения $z \rightarrow \bar{z}$. Поэтому операция сопряжения индуцирует инволютивный автоморфизм в $\mathbb{Z}[i]$, в котором \mathbb{Z} является множеством неподвижных точек, и позволяет оснастить $\mathbb{Z}[i]$ мультипликативной нормой $N : \mathbb{Z}[i] \rightarrow \mathbb{N}$, определяемой через $N(z) = z\bar{z}$, для $z \in \mathbb{Z}[i]$. Другими словами, $N(x + iy) = x^2 + y^2$ для всех $x, y \in \mathbb{Z}$.

Предложение 6

- (i) Кольцо $\mathbb{Z}[i]$ целых гауссовских чисел факториально.
- (ii) Пусть $z \in \mathbb{Z}[i]$ — неприводим в \mathbb{Z} и не ассоциирован с элементом из \mathbb{Z} . Тогда $N(z)$ является простым числом в \mathbb{Z} .

Доказательство.

Пункт (i) следует из евклидовости $\mathbb{Z}[i]$ (см. раздел 3.1) и того, что всякое евклидово кольцо без делителей нуля факториально (см. раздел 3.5). Что касается (ii), то мы покажем, что разложение rs числа $N(z)$ в \mathbb{Z} тривиально. Сначала предположим, что ни r , ни s не обратимы в $\mathbb{Z}[i]$. Пусть n (соответственно, m) — длина разложения на простые множители для r (соответственно, для s). Так как $z\bar{z}$ является простым разложением для $N(z)$ (напомним, что z и \bar{z} неприводимы в $\mathbb{Z}[i]$), то из единственности разложения на неприводимые множители следует, что $m + n = 2$, откуда $n = m = 1$, т.е. r и s — неприводимые в $\mathbb{Z}[i]$. Из единственности следует, что $z \sim_{\mathbb{Z}[i]} r$ или $z \sim_{\mathbb{Z}[i]} s$, что противоречит предположению относительно z (z не ассоциирован с элементом из \mathbb{Z}). Следовательно, хотя бы один из двух элементов r и s является обратимым в $\mathbb{Z}[i]$. Но из $r \in U(\mathbb{Z}[i])$ следует, что $N(r) \in U(\mathbb{Z})$. Так как $N(r) = r^2$ и $N(s) = s^2$, то имеем $r = \pm 1$ или $s = \pm 1$.

■

Замечание. В доказательстве, приведенном выше, мы использовали (кроме факториальности кольца) только наличие мультипликативной нормы со значениями в \mathbb{Z} (фактически, в \mathbb{N} , хотя это

и не важно). В то же время вид единиц в $\mathbb{Z}[i]$ не был использован. Кроме того, существование мультипликативной нормы основано, главным образом, на существовании инволютивного автоморфизма $z \rightarrow \bar{z}$, для которого \mathbb{Z} является множеством неподвижных точек. Такой тип доказательства применим и в других ситуациях.

Предложение 7

Элемент вида $x^2 + y^2$ из \mathbb{N} , где $x \in \mathbb{N}$ и $y \in \mathbb{N}$ взаимно просты, назовем точной суммой двух квадратов. Тогда всякий делитель (из \mathbb{N}) точной суммы двух квадратов сам является точной суммой двух квадратов.

Доказательство.

Заметим сначала, что «общая» сумма двух квадратов может иметь делители, о которых ничего сказать нельзя (достаточно рассмотреть сумму виде $(\lambda x)^2 + (\lambda y)^2$ и тогда λ — такой делитель). Поэтому утверждение о взаимной простоте $x + iy$ в произведение неприводимых в $\mathbb{Z}[i] : x + iy = \pi_1 \dots \pi_n$, где π_j неприводимы. Это разложение индуцирует разложение $x^2 + y^2$:

$$x^2 + y^2 = N(x + iy) = N(\pi_1)N(\pi_2) \dots N(\pi_n). \quad (2.9)$$

Заметим, что ни одно из π_j не ассоциировано с элементом из \mathbb{Z} (так как x и y взаимно просты). Согласно предшествующей лемме, $N(\pi_j)$ является простым элементом \mathbb{Z} . Следовательно, разложение (1) есть разложение (в \mathbb{Z}) $x^2 + y^2$ на простые множители. Если $d \in \mathbb{N}$ — делитель $x^2 + y^2$, то существует такое подмножество $J \subset [1, n]$, что

$$d = \prod_{j \in J} N(\pi_j) = N\left(\prod_{j \in J} \pi_j\right) = u^2 + v^2, \text{ где } u + iv = \prod_{j \in J} \pi_j.$$

Откуда следует, что d есть сумма двух квадратов. Так как $u + iv$ является делителем в $\mathbb{Z}[i]$ числа $x + iy$, то эта сумма точная. ■

Примечание. Зная, что кольцо $\mathbb{Z}[\sqrt{-2}] = \mathbb{Z} + i\sqrt{2}\mathbb{Z} = \{x + \sqrt{-2}y, (x, y) \in \mathbb{Z} \times \mathbb{Z}\}$ факториально, можно доказать небанальный результат, касающийся сумм $x^2 + 2y^2$, где $(x, y) \in \mathbb{N} \times \mathbb{N}$. Это обобщенная точка зрения математика...

Кстати: существует аналогичный результат для сумм вида $x^2 - 2y^2$, связанный с кольцом $\mathbb{Z}[\sqrt{2}] = \mathbb{Z} + \sqrt{2}\mathbb{Z}$. Предположив, что кольцо факториально, можно доказать, для взаимно простых $x, y \in \mathbb{Z}$ следующий результат:

$$d \in \mathbb{Z}, d \mid x^2 - 2y^2 \Rightarrow \exists u \in \mathbb{Z}, v \in \mathbb{Z} \text{ такие, что } d = u^2 - 2v^2.$$

Начнём с определения инволютивного автоморфизма $\mathbb{Z}[\sqrt{2}]$ (которое в этом случае уже не связано с операцией сопряжения в \mathbb{C}), затем введём норму (со значением в \mathbb{Z} на этот раз)..., остерегаясь обратимого элемента $-1 \in \mathbb{Z}$. Новая ступень преодолена.

Однако аналогичного феномена для сумм вида $x^2 + 3y^2$ в кольце $\mathbb{Z}[\sqrt{-3}]$ не наблюдается. Не путайте кольцо $\mathbb{Z}[j]$ целых чисел и кольца $\mathbb{Q}[\sqrt{-3}]$. Оно не факториально!

Пусть $p \in \mathbb{N} \setminus \{2\}$ — простое число, являющееся суммой двух квадратов. Легко проверить, рассматривая $\mathbb{Z}/4\mathbb{Z}$, что $p \equiv 1 \pmod{4}$. Вот обратное утверждение:

Следствие 8

Пусть $p \in \mathbb{N}$ — простое число, удовлетворяющее соотношению $p \equiv 1 \pmod{4}$. Тогда p является суммой двух квадратов.

Доказательство.

Сравнение Вильсона для всякого простого p :

$$(p-1)! \equiv -1 \pmod{p}$$

получается при помощи перегруппирования в произведении $1 \times 2 \times \dots \times (p-1)$ каждого сомножителя со своим обратным по модулю p . Если $p \neq 2$, то, перегруппировывая в том же произведении вместе числа i и $(p-i)$, получаем формулу

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

В частности, если $p \equiv 1 \pmod{4}$ и если $x = \left(\frac{p-1}{2} \right)!$, то $x^2 \equiv -1 \pmod{p}$ или, что то же самое, $p \mid (x^2 + 1)$. Простое p делит точную сумму двух квадратов и, следовательно, само является суммой двух квадратов.

Можно заметить, что главный этап доказательства состоит в утверждении, что -1 является квадратом по модулю p , где p — число вида $4n+1$. Этот результат, прямое доказательство которого было только что получено, будет углублен позже в части, посвященной квадратичным вычетам.

Закончим раздел сводкой результатов, относящихся к группе единиц и неприводимым элементам в $\mathbb{Z}[i]$ (результаты, содержащие, в частности, пункт (ii) теоремы 5). Указанные результаты позволяют лучше понять новое кольцо, удовлетворяющее основной теореме арифметики, обобщающее некоторые кольца, являющиеся квадратичными расширениями, например $\mathbb{Z}[\sqrt{2}]$ или $\mathbb{Z}[\sqrt{-2}]$, но не все!

Пусть $p \in \mathbb{Z}$. Если p неприводимо в $\mathbb{Z}[i]$, то оно тем более неприводимо в \mathbb{Z} (это простое число из \mathbb{Z}). Следующее предложение содержит одну из возможных обратных теорем:

Предложение 9

- (i) $U(\mathbb{Z}[i]) = \{z \in \mathbb{Z}[i] \mid N(z) = 1\} = \{+1, -1, +i, -i\}$.
- (ii) Если p — простое в \mathbb{N} , то p неприводимо в $\mathbb{Z}[i]$ тогда и только тогда, когда оно не является нормой $N(z)$ для $z \in \mathbb{Z}[i]$ (что эквивалентно $p \equiv 3 \pmod{4}$).
- (iii) Пусть $z \in \mathbb{Z}[i]$. Элемент z является неприводимым в $\mathbb{Z}[i]$ и не ассоциированным с элементом из \mathbb{Z} тогда и только тогда, когда $N(z)$ — простое число в \mathbb{Z} .

Несколько примеров

1. Число 5 — простое в \mathbb{Z} , не неприводимое в $\mathbb{Z}[i]$, так как существует нетривиальное разложение $5 = 1^2 + 2^2 = (1 + 2i)(1 - 2i)$
2. В $\mathbb{Z}[i]$ 2 разлагается в сумму двух равных квадратов и почти является квадратом: $2 = 1^2 + 1^2 = (1 + i)(1 - i) = -i(1 + i)^2 \sim_{\mathbb{Z}[i]} (1 + i)^2$.
3. В $\mathbb{Z}[i]$ еще неприводимо $2 + i$, так как $N(2 + i) = 13$ — простое число в \mathbb{Z} .
4. Простые числа 181, 281 и 601, сравнимые с 1 по модулю 4, имеют представление в виде суммы двух квадратов: $181 = 9^2 + 10^2$, $281 = 5^2 + 16^2$, и $601 = 5^2 + 24^2$.

Доказательство предложения 8.

Пункт (i) вытекает из равенства

$$U(\mathbb{Z}[i]) = \{z \in \mathbb{Z}[i] \mid N(z) \text{ обратим в } \mathbb{Z}\} = \{+1, -1, +i, -i\}.$$

Докажем (ii), рассматривая простое число из \mathbb{N} . Если p является нормой $N(z)$, то имеется нетривиальная факторизация: $p = z\bar{z}$, которая показывает, что p не является неприводимым в $\mathbb{Z}[i]$.

Обратно, если p не является неприводимым в $\mathbb{Z}[i]$, то имеется его разложение на неприводимые в $\mathbb{Z}[i]$: $= \pi_1 \pi_2 \dots \pi_n$ с $n \geq 2$. Применяя норму, получаем: $p^2 = N(\pi_1)N(\pi_2) \dots N(\pi_n)$ дает нетривиальное разложение p^2 в \mathbb{Z} . Отсюда $n = 2$, $N(\pi_1) = p$ и $N(\pi_2) = p$.

Второй результат о равносильности — следствие первого.

(iii) Пусть $z \in \mathbb{Z}[i]$ такое, что $N(z)$ — простое число в \mathbb{N} . Из разложения $z = z_1 z_2$, где z_1 и z_2 принадлежат $\mathbb{Z}[i]$, получаем, применяя норму, что $N(z) = N(z_1)N(z_2)$. Это — разложение простого числа $N(z)$ в \mathbb{Z} . Оно непременно тривиально. Следовательно, тому, что исходное разложение является разложением неприводимого элемента z из $\mathbb{Z}[i]$ (и подтверждает его неассоциированность с элементом из \mathbb{Z}). Обратное уже было доказано в предложении 5.

■

10. 2 Элементарные свойства теории делимости

Мы займемся нахождением общих критериев факториальности колец. Эти критерии устанавливаются совершенно естественным путем и базируются на основных понятиях теории делимости: нётеровости кольца и свойстве Гаусса.

10.1. 2.1 Существование и единственность разложения на простые множители

10.2. (9) Предложение.

Пусть любая бесконечно неубывающая последовательность главных (т.е. порожденных одним элементом) идеалов кольца A без делителей нуля стабилизируется (т.е. становится постоянной, начиная с некоторого ее члена). Тогда кольцо A удовлетворяет условию существования разложения на неприводимые множители.

Доказательство.

Допустим, что условие на главные идеалы выполнено. Предположим противное: существует $x_0 \in A^* \setminus U(A)$, не являющийся произведением неприводимых. Элемент x_0 тем более не является неприводимым и поэтому имеет нетривиальное разложение $x_0 = x_1 y_1$, где x_1 и y_1 необратимы. Один из двух элементов x_1 или y_1 не представим в виде произведения неприводимых (иначе x_0 обладает разложением в произведение неприводимых). Пусть это будет x_1 . Значит имеем строгое включение $Ax_0 \subsetneq Ax_1$. Тогда это рассуждение можно

повторить для x_1 , что дает другое точное включение того же типа, что и первое: $Ax_1 \subsetneq Ax_2$ с x_2 , не являющимся произведением неприводимых. Ясно, что можно построить бесконечную последовательность главных идеалов, которая не может стабилизироваться.

■

Следующее предложение связано с единственностью разложения на неприводимые множители. Доказательство является простой формальностью и предоставим его читателю.

Предложение 10

Пусть A — кольцо без делителей нуля, удовлетворяющее условию существования разложения в произведение неприводимых. Тогда A обладает свойством единственности разложения (т.е. является **факториальным**), если для любого неприводимого p выполнено следующее свойство: если p делит ab , то $p \mid a$ или $p \mid b$ (значит p может рассматриваться как простой элемент).

10.3. 2.2 НОД и взаимно простые элементы

Определение 11

(i) Элемент $d \in A$, являющийся общим делителем a и b , называется НОД a и b , если всякий другой делитель δ элементов a и b является делителем d . В кратком виде это выражается следующей эквивалентностью:

$$d \text{ есть НОД } a \text{ и } b \text{ тогда и только тогда, когда } [\delta \mid d \iff \delta \mid a \text{ и } \delta \mid b].$$

В этом случае идеал Ad является единственным: действительно, сказать, что $d = \text{НОД}(a, b)$ равносильно тому, что идеал Ad есть верхняя грань для множества $\{Aa, Ab\}$ главных идеалов.

В дальнейшем будем использовать обозначения $d = \text{НОД}(a, b)$ или $a \wedge b$, помня, что в общем случае НОД — не единственный элемент.

(ii) Элементы a и b будут называться **взаимно простыми**, если 1 является НОД a и b (равносильным образом, это означает, что только единицы A — суть общие делители для a и b).

Существуют кольца, обладающие парами элементов, не имеющими наибольшего общего делителя. Покажем это.

Кольцо $A = \mathbb{Z}[\sqrt{-5}]$

Рассмотрим кольцо $A = \mathbb{Z}[\sqrt{-5}] = \{x + iy\sqrt{5}, x, y \in \mathbb{Z}\}$. Это множество есть подкольцо поля \mathbb{C} с обычными операциями сложения и

умножения и потому без делителей нуля. Операция сложения в \mathbb{C} индуцирует инволютивный автоморфизм $\sigma\mathbb{Z}[\sqrt{-5}] : \sigma(x + y\sqrt{5}) = x - y\sqrt{5}$, и дает возможность определить в $\mathbb{Z}[\sqrt{-5}]$ мультипликативную алгебраическую норму со значениями в \mathbb{N} , определяемую через $N(z) = z\sigma(z)$. Другими словами, $N(x + y\sqrt{-5}) = x^2 + 5y^2$. Группа единиц в $\mathbb{Z}[\sqrt{-5}]$ следующая:

$$U(\mathbb{Z}[\sqrt{-5}]) = \{z \in \mathbb{Z}[\sqrt{-5}] \mid N(z) = 1\} = \{\pm 1\}.$$

Элемент 2 неприводим в этом кольце. Действительно, из соотношения $2 = uv$ следовало бы равенство $4 = N(u)N(v)$. Так как 2 не представимо в виде $x^2 + 5y^2$, то 2 не является нормой. Из этого ясно, что $N(u) = 1$ или $N(v) = 1$: u или v обратим, а, следовательно, 2 неприводим. Таким же образом можно показать, что элементы 3, $1 + \sqrt{5}$ и $1 - \sqrt{5}$ неприводимы. Однако для них выполнено равенство $2 \times 3 = (1 + \sqrt{5}) \times (1 - \sqrt{5})$. Как грустно: 4 сомножителя, встречающихся здесь, неприводимы! Понятно, что в кольце $\mathbb{Z}[\sqrt{-5}]$ основная теорема арифметики не имеет места.

Если положим $a = 1 + \sqrt{-5}$, $b = 1 - \sqrt{-5}$, то это равенство имеет другое необычное следствие: $2 \mid ab$, 2 неприводимо, но 2 не делит ни a , ни b ! Беда не приходит одна. В результате предыдущей патологии элементы $2a$ и ab не имеют НОД в $\mathbb{Z}[\sqrt{-5}]$. Допустим, что d — общий делитель $2a$ и ab . Тогда $d \mid 2a$ и $d \mid ab$, что приводит к заключению $(\frac{d}{a} \mid 2)$. Так как 2 неприводимо, то $d \sim_A a$ или $d \sim_A 2a$. Но a не может быть НОД (он не делится на общий делитель с 2), а $2a$ — тем более (b не делится на 2, ab не делится на $2a$), что приводит к противоречию.

10.4. 2.3 Выделение понятия

Мы уже рассмотрели два важных свойства. Одно связано с наличием разложения на неприводимые множители, другое — с единственностью разложения. Это дает повод к следующим определениям:

Определение 12 (свойство Гаусса)

Элемент p кольца A **называется простым**, если он отличен от нуля, не является обратимым и удовлетворяет условию: $p \mid ab \rightarrow p \mid a$ или $p \mid b$, что можно выразить по-другому: факторкольцо A/Ap не имеет делителей нуля.

Определение 13 (упорядоченное нётерово множество)

Упорядоченное множество X называется **нётеровым**, если всякая бесконечная возрастающая последовательность стабилизируется.

Определение 14 (нётерово кольцо)

Кольцо A называется нётеровым, если множество идеалов этого кольца, упорядоченное по включению, является нётеровым. Другими словами, всякая бесконечная возрастающая последовательность идеалов постоянна, начиная с некоторого номера.

По определению (или почти по определению) для факториального кольца понятия простого и неприводимого элемента совпадают. Между прочим, в кольце без делителей нуля имеет место следующее свойство:

Определение 15

В кольце без делителей нуля простой элемент неприводим. Обратное, вообще говоря, неверно.

Определение 16 (принцип нётеровой индукции)

Упорядоченное множество X нётерово тогда и только тогда, когда всякое непустое подмножество Y в X обладает максимальным элементом, содержащимся в Y (элемент $y \in Y$ называется максимальным в Y , если не существует $z \in Y$, для которого $z > y$).

Доказательство.

Допустим, что X нётерово и $Y \subset X$ — непустое подмножество X без максимального элемента. Пусть $x_0 \in Y$. Так как x_0 не максимальный в Y , то найдется $x_1 \in Y$, такой, что $x_0 < x_1$. Так как x_1 не максимален в Y , то можно построить x_2 , такой, что $x_1 < x_2$, $x_2 \in Y$. Это вам ничего не напоминает?

Обратно, пусть $(x_n)_{n \in \mathbb{N}}$ — возрастающая последовательность элементов из X . Если x_m — максимальный элемент множества $\{x_n, n \in \mathbb{N}\}$, то последовательность $(x_n)_{n \in \mathbb{N}}$ стабилизируется, начиная с номера m .

■

Определение 17 (связь между нётеровостью и конечностью колец)

Кольцо A является нётеровым, тогда и только тогда, когда всякий идеал в A конечного типа (т.е. порожден конечным числом элементов).

Доказательство.

Допустим, что всякий идеал конечного типа, и пусть $(I_n)_{n \in \mathbb{N}}$ — возрастающая последовательность идеалов. Объединение $I = \bigcup_{n \in \mathbb{N}} I_n$ есть идеал, порожденный конечным подмножеством \mathcal{F} . Так как $(I_n)_{n \in \mathbb{N}}$ возрастающая, то существует такое $m \in \mathbb{N}$, что $\mathcal{F} \subset I_m$. Отсюда получаем, что $I = I_m$, а, следовательно, начиная с номера m , последовательность $(I_n)_{n \in \mathbb{N}}$ стабилизируется.

Обратно, предположим, что кольцо A нётерово, содержащее идеал I , не являющийся идеалом конечного типа. Так как I не является идеалом конечного типа, то $x_0 \in I$, затем $x_1 \in I \setminus Ax_0$, затем $x_2 \in I \setminus (Ax_0 + Ax_1) \dots$ Это приводит нас к наличию строго возрастающей последовательности идеалов:

$$\begin{aligned} Ax_0 \subsetneq Ax_0 + Ax_1 \subsetneq Ax_0 + Ax_1 + Ax_2 \subsetneq \dots \\ \subsetneq Ax_0 + Ax_1 + \dots + Ax_n \subsetneq \dots, \end{aligned}$$

что заканчивает доказательство. ■

Прежде чем перейти к вещам, более конкретным, забежим вперед в главу III и приведем несколько результатов о нётеровых модулях. Конечно, нётеров модуль — это такой модуль, в котором всякая возрастающая последовательность подмодулей стабилизируется, или, иначе, всякий подмодуль является подмодулем конечного типа. Для того чтобы связать сказанное с предыдущим, заметим, что идеалы кольца являются его подмодулями.

Предложение 18

(i) Пусть A — кольцо. Если A -модуль E — нётеров, то для всякого подмодуля F модуля E A -модули E/F и F нётеровы. Обратно, если в A -модуле E есть такой нётеров подмодуль F , что E/F нётеров, то и E — нётеров.

Доказательство.

Доказательство первого результата очень просто. Действительно, всякая возрастающая последовательность подмодулей F является также возрастающей последовательностью подмодулей в E . Заметим также, что подмодули E/F есть образы подмодулей E , содержащие F .

Обратное утверждение доставляет больше забот. Нельзя пользоваться теми же рассуждениями, что в векторном пространстве, и предполагать, например, что $E \simeq E/F \oplus F$.

Рассмотрим отображение, которое каждому подмодулю G в E ставит в соответствие пару $(G + F, G \cap F)$. Это отображение порождает строго возрастающую последовательность. Действительно, рассмотрим два подмодуля $G' \subset G$, такие, что $G' + F = G + F$ и $G' \cap F = G \cap F$. Пусть элемент $x \in G$. Тогда $x = y + z$, где $y \in G'$, $z \in F$. Так как $G' \cap G$, то $y \in G$ и, значит, $z \in G$. Однако $G' \cap F = G \cap F$, откуда $z \in G'$, и потому $x \in G'$. Таким образом, всякая строго возрастающая последовательность модулей в E порождает строго возрастающую последовательность подмодулей в $(E + F)/F$ и $E \cap F$. Догадаемся о продолжении...

Предложение 19

Если E и F — два A -модуля, то $E \times F$ нётеров модуль тогда, и только тогда, когда таковыми являются E и F .

Доказательство.

Проектируя $E \times F$ на каждый из подмодулей E и F , получаем необходимость условия. Для доказательства обратного достаточно заметить, что $E \simeq E \times F / \{0\} \times F$ и применить предыдущее предложение.

10.5. 2.4 Соотношение Безу**Предложение 20**

(i) Пусть a, b и d — элементы кольца A , связанные соотношением $Ad = Aa + Ab$. Тогда $d = \text{НОД}(a, b)$.

(ii) Пусть A — кольцо, в котором сумма двух главных идеалов есть главный идеал. Тогда всякий неприводимый элемент A является простым элементом (т.е. удовлетворяет свойству Гаусса).

Доказательство.

(i) — простая формальность, в которой нет ничего удивительного. Действительно, $Aa + Ab$ есть верхняя грань для множества идеалов $\{Aa, Ab\}$.

Чтобы доказать (ii), предположим, что p неприводим в A , $a \in A$ и $b \in A$ таковы, что $p|ab$. Допустим, к примеру, что p не делит a . Тогда $Ap \subsetneq Ap + Aa$ и, значит, $Ap + Aa$ есть главный идеал. Из максимальности Ap среди главных идеалов следует, что $Ap + Aa = A$, и поэтому выполнено соотношение Безу $1 = up + va$, из которого выводим $b = upb + vab$. Но тогда p делит b .

Подчеркнем роль, сыгранную соотношением Безу, связанную с НОД и свойством Гаусса в доказательстве предложения. Вот результат, приводящий к понятию максимального элемента.

Определение 21 (кольцо Безу)

Кольцо A называется кольцом Безу, если сумма любых двух его главных идеалов является главным идеалом. Это эквивалентно тому, что всякий идеал конечного типа является главным.

Замечание. Введение понятия кольца Безу продиктовано соображениями удобства записи. Никакой теории колец Безу в этой книге не будет!

Предложение 22

Пусть p — неприводимый элемент кольца Безу A . Тогда факторкольцо A/Ar есть тело. Равносильным образом это означает, что идеал Ar максимален среди всех идеалов (отличных от A). По-другому это можно выразить так: для любого $x \in A \setminus Ar$ существуют такие u и v из A , что $1 = ux + vr$.

Доказательство.

Пусть ξ — ненулевой элемент из $A \setminus Ar$, представляющийся элементом $x \in A \setminus Ar$. Тогда идеал $Ax + Ar$ есть главный идеал Ad , строго содержащий Ar . Отсюда $d \nmid r$ и $d \approx r$. Значит, $d \sim 1$, т.е. $1 \in Ax + Ar = A$. Поэтому найдутся такие u и v , что $1 = ux + vr$, откуда, переходя к факторкольцу A/Ar , получаем $1 = \bar{u}\bar{x} = \bar{u}\bar{\xi}$, что доказывает обратимость ξ в A/Ar . Отсюда следует, что A/Ar — тело.

Читатель без труда проверит самостоятельно эквивалентность трех формулировок результата

**Определение 23**

Идеал I кольца называется простым, если он отличен от A и если для всякого произведения $ab \in I$ следует, что $a \in I$ или $b \in I$. Последнее равносильно тому, что факторкольцо A/I не имеет делителей нуля.

Идеал называется максимальным, если он максимален среди всех идеалов, отличных от A , или, что то же самое, факторкольцо A/I является телом.

Замечание. Идеал $\{0\}$ простой тогда и только тогда, когда кольцо не имеет делителей нуля. Он максимален тогда и только тогда, когда кольцо является телом.

В качестве резюме:

Предложение 24

(i) В кольце без делителей нуля всякий элемент, порождающий максимальный идеал, является простым, а каждый простой элемент неприводим. Два последние понятия (простой и неприводимый) совпадают в факториальном кольце, и все три понятия совпадают в кольце Везу.

(ii) Элемент p кольца A простой тогда и только тогда, когда идеал Ap простой и отличен от $\{0\}$.

(iii) В любом кольце всякий максимальный идеал прост.

11. 3 Евклидовы кольца с точки зрения эффективности

Этот раздел посвящен изучению евклидовых колец, т.е. колец, обладающих евклидовым делением. Здесь можно будет найти точное определение данного понятия, элементарные свойства евклидовых колец, в частности, факториальности евклидовых колец без делителей нуля, алгоритм Евклида и его реализацию в языке Ада во всей его общности. Для более глубокого изучения необходимой математики читатель может обратиться к работе Самюэля [159], являющейся очень толковыми легко читаемым руководством.

11.1. 3.1 Что такое евклидово кольцо?

Определение 25

Евклидово деление в кольце есть оператор $A \times A^* \rightarrow A$, обозначаемый \langle / \rangle , для которого существует отображение ϕ кольца A во вполне упорядоченное множество (т.е. множество, всякая непустая часть которого имеет наименьший элемент), удовлетворяющее соотношению:

$$\forall (a, b) \in A \times A^*, \phi(a - b(a/b)) < \phi(b) \quad (2.10)$$

В этом случае говорят, что \langle / \rangle евклидово для ϕ . Отображение ϕ называется **алгоритмом Евклида**, если оно удовлетворяет свойству:

$$\forall (a, b) \in A \times A^*, \exists (q, r) \in A \times A \text{ такая, что } a = bq + r \text{ с } \phi(r) < \phi(b). \quad (2.11)$$

Эквивалентным образом, для всякого b в A^* существует такая система представителей в A/Ab , элементы r которых удовлетворяют неравенству $\phi(r) < \phi(b)$.

Кольцо A (не обязательно без делителей нуля) называется *евклидовым*, если оно допускает евклидово деление «/» (в таком случае A называется *евклидовым для «/»*), а также, что A допускает евклидов алгоритм (в последнем случае говорят, что A *евклидово для ϕ*).

Замечание. Хотя в математике заостряется внимание на понятии евклидова алгоритма, именно эффективность евклидова деления позволяет вычислять НОД, находить коэффициенты Безу, и т.д. В какой-то мере приходится пожалеть о термине «алгоритм Евклида».

Большинство евклидовых алгоритмов, используемых в математических учебниках, базируется на упорядоченности множества \mathbb{N} (или на упорядочении множества, изоморфного \mathbb{N}). Междупрочим, произведение колец $\mathbb{Z} \times \mathbb{Z}$ есть евклидово кольцо (имеющее делители нуля), не допускающее алгоритма Евклида с целыми значениями [159]. Гиблот в [85] нашел аналогичный пример для кольца без делителей нуля.

Определение 26

Пусть A — евклидово кольцо для ϕ . Скажем, что равенство типа (3) есть **евклидово деление** (относительно ϕ) a на b , q — частное евклидова деления, r — остаток. Назовем оператором **взятия остатка** (для ϕ) оператор $\text{mod} : A \times A^* \rightarrow A$, удовлетворяющий соотношениям:

$$\phi(a \bmod b) < \phi(b) \text{ и } a \bmod b \equiv a \pmod{b}.$$

Каждое евклидово деление «/» определяет остаточный оператор $a \bmod b = a - b \times (a/b)$, для $a \in A$ и $b \in A^*$.

Примеры евклидовых алгоритмов на \mathbb{Z}

Кольцо \mathbb{Z} целых чисел является моделью для конструирования евклидовых колец. Вот два примера деления и евклидовых алгоритмов для \mathbb{Z} .

1. Взятие абсолютного значения в \mathbb{Z} дает возможность построить алгоритм Евклида. Для $(a, b) \in \mathbb{Z} \times \mathbb{Z}^*$ можно записать $a = bq + r$ $|r| < |b|$ различными способами. Например, при $(a, b) = (-14, 3)$ можно записать $-14 = 3 \times (-5) + 1$ и получить остаток, равный единице, или $-14 = 3 \times (-4) - 2$ с остатком, равным -2 . В обоих случаях абсолютная величина остатка меньше 3. Следовательно, знание евклидова алгоритма еще не определяет евклидово деление (оно становится единственным, если потребовать, чтобы остаток был положительным или нулевым).

2. Другой, менее известный, евклидов алгоритм на \mathbb{Z} связан с отображением ϕ , которое по целому числу дает количество двоичных цифр в его записи (уславливаются, что $\phi(0) = 0$). Для доказательства того, что ϕ является евклидовым алгоритмом, необходимо показать, что для $b \neq 0$ выполнено соотношение деления: $a = bq + r, |r| \leq |b|/2$. Действительно, при $b \neq 0$ из соотношения $|r| \leq |b|/2$, конечно следует $\phi(r) < \phi(b)$ и обратно.

Наличие такого деления вытекает из того факта, что всегда можно записать соотношение $a = bq + r$ с $|r| < |b|$ и r того же знака, что и b . Это и дает возможность отыскать такое деление, что $|r| \leq |b|/2$. Если $|r| > |b|/2$, то положим:

$$a = bq + r = b(q + 1) + r - b, \quad (\text{ибо } |r - b| \leq |b|/2).$$

Еще раз подчеркнем, что для пары (a, b) пара (q, r) определяется не единственным образом. Например, если $(a, b) = (27, 6)$, то:

$$\begin{cases} 27 = 6 \times 4 + 3 & (r = 3), \\ 27 = 6 \times 5 - 3 & (r = -3), \end{cases} \quad \text{с } |r| \leq |b|/2 \text{ в обоих случаях.}$$

Несмотря на эту неединственность, будем использовать выражение «евклидово деление с самым малым остатком» или «центрированное деление», чтобы обозначить всякое евклидово деление, ассоциированное с данным евклидовым алгоритмом. Примечательно, что указанный евклидов алгоритм является в определенном смысле наименьшим евклидовым алгоритмом на \mathbb{Z} (см. упр. 27).

Пример кольца многочленов.

Для кольца многочленов $K[X]$, где K — поле, функция степени со значениями во вполне упорядоченном множестве $\{-\infty\} \cup \mathbb{N}$ есть евклидов алгоритм. Действительно, хорошо известно, что для $a, b \in K[X], b \neq 0$ можно записать $a = bq + r$ с $\deg r < \deg b$ и такое представление однозначно. Мы вернемся позднее к этому евклидову делению, когда будем рассматривать алгоритм евклидова деления многочленов для вычисления НОД при нахождении неприводимых многочленов над $\mathbb{Z}/p\mathbb{Z}$.

Пример кольца целых гауссовских чисел.

Кольцо $\mathbb{Z}[i]$, введенное в разделе 1.3, евклидово для нормы N . Чтобы это проверить, рассмотрим сначала поле $\mathbb{Q}[i] = \mathbb{Q} + i\mathbb{Q}$, являющееся полем частных для кольца $\mathbb{Z}[i]$. Это поле остается неподвижным относительно операции комплексного сопряжения, что позволяет продолжить

норму с $\mathbb{Z}[i]$ на $\mathbb{Q}[i]$ в той же форме ($N(z) = z\bar{z}, z \in \mathbb{Q}[i]$). Рассмотрение решетки $\mathbb{Z}[i] \subset \mathbb{Q}[i]$ позволяет приблизить каждый элемент $\mathbb{Q}[i]$ следующим образом:

$$\forall c \in \mathbb{Q}[i], \exists q \in \mathbb{Z}[i] \text{ такой, что } \sqrt{N(c - q)} \leq \sqrt{2}/2, \text{ т.е. } N(c - q) \leq 1/2.$$

Если a и b — два элемента из $\mathbb{Z}[i]$, где $b \neq 0$, использование предыдущего неравенства с a/b в качестве c дает нам наличие $q \in \mathbb{Z}[i]$, такого, что $N(a/b - q) \leq 1/2$, или, по-другому, $N(a - bq) \leq N(b)/2$. $a = bq + r$, где $N(r) < N(b)$, что и ожидалось.

3.2 Алгоритм Евклида нахождения НОД

Алгоритм Евклида для нахождения НОД двух целых чисел адаптируется без изменений для всякого евклидова кольца. Вот точная формулировка:

Алгоритм 27 (алгоритм Евклида в евклидовом кольце)

Пусть евклидово деление в кольце A ассоциировано с евклидовым алгоритмом ϕ и a, b — элементы из A . Алгоритм Евклида заключается в том, чтобы осуществить следующие евклидовы деления (в которых положим $r_0 = a, r_1 = b$) до появления нулевого остатка:

$$\left\{ \begin{array}{lll} \text{если } r_1 \neq 0, & \text{то } r_0 = r_1 q_1 + r_2, & \text{где } \phi(r_2) < \phi(r_1) \\ \text{если } r_2 \neq 0, & \text{то } r_1 = r_2 q_2 + r_3, & \text{где } \phi(r_3) < \phi(r_2) \\ \vdots & \vdots & \vdots \\ \text{если } r_i \neq 0, & \text{то } r_{i-1} = r_i q_i + r_{i+1}, & \text{где } \phi(r_{i+1}) < \phi(r_i) \\ \vdots & \vdots & \vdots \\ \text{если } r_{n-1} \neq 0, & \text{то } r_{n-2} = r_{n-1} q_{n-1} + r_n, & \text{где } \phi(r_n) < \phi(r_{n-1}) \\ \text{если } r_n \neq 0, & \text{то } r_{n-1} = r_n q_n + r_{n+1}, & \text{где } \phi(r_n) = 0 \text{ и } \phi(r_{n+1}) < \phi(r_n). \end{array} \right.$$

Предложение 28

(i) В евклидовом кольце A алгоритм Евклида, ассоциированный с парой $(a, b) \in A \times A$, находит элемент $r_n \in A$ такой, что $Aa + Ab = Ar_n$ (в частности, r_n является НОД a и b).

(ii) Всякое евклидово кольцо A является кольцом Безу.

Доказательство.

Оно идентично тому, что имеется во введении. Из равенства $r_{i-1} = r_i q_i + r_{i+1}$ с $1 \leq i \leq n$ выводим, что $Ar_{i-1} + Ar_i = Ar_i + Ar_{i+1}$ и, в частности, $Aa + Ab = Ar_0 + Ar_1 = Ar_n + Ar_{n+1} = Ar_n$.

Пусть mod — оператор нахождения остатка, ассоциированный с ϕ в схеме вычисления алгоритма Евклида и $r_{i+1} = r_{i-1} \text{ mod } r$. Следовательно, можно только с помощью оператора mod выразить пару (r_1, r_2) через пару (r_0, r_1) , затем (r_2, r_3) через $(r_1, r_2) \dots$. Эта рекуррентная последовательность приводит к алгоритму 1, в котором пара (r, s) принимает последовательные значения (r_i, r_{i+1}) .

```

НОД( $a, b \in A$ ) return  $r \in A$  is
  ( $r, s$ )  $\in A \times A := (a, b)$ ; {
    while( $s \neq 0$ ) { НОД( $r, s$ ) = НОД( $a, b$ )
      ( $r, s$ )  $\leftarrow (s, r \text{ mod } s)$ ;
    }
    НОД( $r, s$ ) = НОД( $a, b$ ) и  $s = 0$ , где  $r = \text{НОД}(a, b)$ 
    return  $r$ ;
  }

```

Алгоритм 1. Вычисление НОД в евклидовом кольце

Замечание. Если обозначить НОД результат обсуждаемого алгоритма, то вполне возможно, что найдутся элементы a, b , для которых $\text{НОД}(a, b)$ отличается от $\text{НОД}(b, a)$. Впрочем, даже два последовательных оператора взятия различных модулей могут дать различные НОД, но, конечно, ассоциированные. См. пример в следующем разделе.

11.2. 3.3 Реализация в языке Ада вычисления НОД

Язык Ада, в котором все понятия определены самым строгим образом [68], располагает, в частности, и евклидовым делением в \mathbb{Z} , обозначенным «/», определяемым в первую очередь тем фактом, что для $a \geq 0, b > 0$ число a/b есть обычное евклидово частное, a с другой стороны, свойством симметрии: $(-a)/b = -(a/b) = a/(-b)$.

Язык Ада располагает также двумя остаточными операторами **mod** и **rem**. **mod** (соответственно, **rem**) есть единственный оператор взятия наименьшего по абсолютной величине остатка того же знака, что и то число, на которое делят (соответственно, делимого), удовлетворяющий соотношениям

$$\begin{aligned}
 a \text{ mod } b &\equiv a \pmod{b}, & |a \text{ mod } b| &< |b|, & a \text{ mod } b &\text{ того же знака, что и } b, \\
 a \text{ rem } b &\equiv a \pmod{b}, & |a \text{ rem } b| &< |b|, & a \text{ rem } b &\text{ того же знака, что и } a.
 \end{aligned}$$

В частности, $a \text{ rem } b = a \bmod b$, если a и b одного знака. Следующие соотношения легко проверяются:

$$a \text{ rem } b = a - (a/b) \times b, \quad (-a) \text{ rem } b = -(a \text{ rem } b), \quad a \text{ rem } (-b) = a \text{ rem } b.$$

Вот (программа 1) настраиваемая Ада-функция, упрощающая вычисления НОД в \mathbb{Z} , основанная на ранее изученном алгоритме. Ниже в рамке записаны сгруппированные спецификация функции и ее реализация, что, вообще говоря, является не самым лучшим способом.

```
generic
  type Ring_Element is private;
  Zero : in Ring_Element;
  with function "-"(a,b: Ring_Element) return Ring_Element is <>;
  with function "*" (a,b: Ring_Element) return Ring_Element is <>;
  with function "/" (a,b: Ring_Element) return Ring_Element is <>;
  function GCD (a, b : Ring_Element) return Ring_Element;

function GCD (a, b : Ring_Element) return Ring_Element is
  type Pair is record
    First, Second : Ring_Element;
  end record;
  u : Pair := (a , b);

function "NOD" (a, b : Ring_Element) return Ring_Element is
begin
  return a - b * (a/b);
end;

begin
  while u.second /= Zero loop
    u := (u.second, u.first mod u.second);
  end loop;
  return u.first;
end GCD;
```

Этот кусок программы требует некоторых пояснений. Тип элементов, для которых требуется вычисление НОД, есть тип `private` языка Ада. Это означает, что не предполагается особых ограничений на структуру этих элементов. Единственные операции, которые над ними производятся — это присваивание и проверка на равенство. Для вычисления НОД нам требуются и другие действия: сложение, вычитание, евклидово деление и, хотя и не сразу, но во время реализации, необходимо распознавать нулевой элемент рабочего кольца. Итак, вот, в основном, все понятия, заложенные в спецификации функции.

Для того, чтобы не усложнять работу с текущими значениями параметров, момент, когда нужна работа с этими значениями, откладывается до выполнения трех арифметических операций. Это делается при помощи формулы $\triangleright \mathbf{is} \langle \triangleright \triangleleft$, фигурирующей в настраиваемой части функции. Точная семантика этого обозначения следующая. Если он не оснащен одним из параметров подпрограммы, имеющим значение по умолчанию, то компилятор использует в подпрограмме то же имя, что и настраиваемый параметр, и то же значение параметра. Таким образом, программа, использующая эту функцию, чтобы найти НОД, может быть реализована следующим образом:

```
#define Ring_GCD GCD
```

где передаваемый параметр для действий целого типа и константа 0.

Перейдем к реализации функции *GCD* (т.е. НОД). Для упрощения работы с элементами поля, для того, чтобы сделать эту работу более похожей на исходный алгоритм, определяется тип *Pair*. Надо отметить, что единственное, в чем нуждается функция *GCD*, это оператор **mod**. На практике, в общем случае, все что нужно, — это, прежде всего, оператор деления. Поэтому оператор деления, снабженный настраиваемым параметром, находит функцию остатка, используемую в алгоритме для вычисления НОД, сравнительно просто.

11.3. Сравнение эффективности различных делений

Разумеется, функция, о которой говорилось в предыдущем разделе, достаточна для вычисления НОД двух элементов евклидова кольца. Но нас интересует также эффективность предложенных методов, а для такой оценки одних этих функций недостаточно. Действительно, чтобы сравнить эти методы, можно написать программу вычисления НОД двух целых чисел, параметризовав ее используемым оператором деления (или нахождения остатка). Если эта программа позволит отследить все промежуточные результаты, то можно наблюдать сходимость метода и сделать некоторые предположения в отношении сложности.

Мы не будем заниматься здесь подробно этой функцией (ее выполнение очень просто). Ограничимся приведением нескольких результатов, иллюстрирующих полученную разницу в сложности и в результатах, когда используются различные операторы нахождения остатка. Функции, которые используются (в порядке следования), деление, отвечающее оператору **rem** в языке Ада, деление с использованием оператора **mod** и, наконец, центрированное деление (которое отвечает стандартному оператору, заложенному в большинстве языков программирования, но который легко может привести к неверному результату).

Сложность вычисления НОД сильно отличается в зависимости от используемой функции, как показывает таблица 1.

<i>i</i>	Остаток rem в яз. Ада	Остаток mod в яз. Ада	Центриров. остаток
1	$204 = -126 \times -1 + 78$	$204 = -126 \times -2 + -48$	$204 = -126 \times -2 + -48$
2	$-126 = 78 \times -1 + -48$	$-126 = -48 \times 2 + -30$	$-126 \times 3 + 18$
3	$78 = -48 \times -1 + 30$	$-48 = -30 \times 1 + -18$	$-48 = 18 \times -3 + 6$
4	$-48 = 30 \times -1 + -18$	$-30 = -18 \times 1 + -12$	$18 = 6 \times 3 + 0$
5	$30 = -18 \times -1 + 12$	$-18 = -12 \times 1 + -6$	
6	$-18 = 12 \times -1 + -6$	$-12 = -6 \times 2 + 0$	
7	$12 = -6 \times -2 + 0$		

Таблица 2.4: Последовательность частных для различных евклидовых делений

Центрированное деление дает в действительности наименьшее количество итераций, результат, на котором мы остановимся в разделе 6. Однако точка зрения, заключающаяся в том, что НОД надо вычислять в \mathbb{Z} , используя наименьший остаток, уязвима. Она не учитывает того факта, что оператор центрированного деления реализуется за большее время из-за логики программирования, тогда как два других вообще свободны от этого недостатка и могут быть прямо реализованы в железе. То, что выигрывается в теории алгоритмов, теряется при программировании на машине. Ах, информатика без машин! На этом примере можно также увидеть неединственность НОД. Два простых алгоритма дают значение -6 , в то время как третий дает значение 6 .

11.4. Факториальность евклидовых колец без делителей нуля

Следующая лемма рассматривает поведение алгоритма Евклида φ по отношению к включению идеалов и показывает, что евклидово кольцо нётерово.

Лемма 29

- Пусть φ — евклидов алгоритм кольца A .
- (i) Неравенство $\varphi(0) < \varphi(b)$ верно для любого $b \in A^*$.

(ii) Отображение $\tilde{\varphi}$ (со значениями в области значений φ), определенное на множестве ненулевых идеалов A соотношением $\tilde{\varphi}(I) = \min\{\varphi(a) | a \in I\}$ — строго убывающее отображение множества ненулевых идеалов из A во вполне упорядоченное множество. Значит, A нётерово.

Доказательство.

(i) Пусть $b \in A^*$. Алгоритм Евклида 27, применяемый к $a \neq 0$ и b , дает последовательность $(r_i)_{0 \leq i \leq n+1}$ с $\varphi(r_{n+1}) < \varphi(r_n) < \dots < \varphi(r_2) < \varphi(r_1)$, где $r_{n+1} = 0$ и $r_1 = b$. Следовательно, $\varphi(0) < \varphi(b)$ для любого $b \in A^*$.

(ii) Пусть теперь I и J два ненулевых идеала, для которых $I \subsetneq J$. Пусть $a \in I \setminus \{0\}$ такой, что $\varphi(a) = \tilde{\varphi}(I)$. Идеал Aa , содержащийся в I , строго содержится в J . Выберем $x \in J \setminus Aa$ и осуществим евклидово деление x на a : $x = aq + r$, где $\varphi(r) < \varphi(a)$. Так как $x \notin Aa$, то r — ненулевой элемент из J . Отсюда $\tilde{\varphi}(J) \leq \varphi(r) < \varphi(a) = \tilde{\varphi}(I)$ что заканчивает доказательство леммы. ■

Предложение 30

Всякое евклидово кольцо без делителей нуля является факториальным, нётеровым и кольцом Безу.

Доказательство.

Предыдущая лемма показывает, что всякое евклидово кольцо (без делителей нуля или с ними) нётерово. Согласно предложению 9, всякий элемент из $A^* \setminus U(A)$ является произведением неприводимых элементов. С учетом того, что A — кольцо Безу, предложение доказано. ■

Итак, мы доказали, в частности, что всякий идеал евклидова кольца имеет конечный тип. Следующее предложение еще интереснее.

Предложение 31

Всякий ненулевой идеал евклидова кольца A главный.

Доказательство.

Вот прямое и классическое доказательство этого факта. Пусть I — ненулевой идеал в A и $a \in I \setminus \{0\}$ такой элемент, что $\varphi(a) = \min\{\varphi(x) | x \in I \setminus \{0\}\}$ (такое определение имеет смысл, так как значения φ принадлежат вполне упорядоченному множеству). Элемент a порождает I . В самом деле, для $x \in I$ осуществим евклидово деление x на a : $x = aq + r$, где $\varphi(r) < \varphi(a)$. Принадлежность r к I и минимальность $\varphi(a)$ дают $r = 0$. Следовательно, $x = aq$, что и требовалось доказать.

Внимательный читатель обратит внимание на аналогию с доказательством леммы 29. Действительно, если I — ненулевой идеал и $a \in I \setminus \{0\}$ такой, что $\varphi(a) = \tilde{\varphi}(I)$, то $Aa \subset I$ и $\tilde{\varphi}(Aa) = \tilde{\varphi}(I)$, ввиду строгого убывания $\tilde{\varphi}$. Значит, $Aa = I$.

Наконец, имеется третий довод! Так как кольцо A конечного типа (A нётерово), то его идеал конечного типа и потому главный (A является кольцом Безу)...



Замечание. Можно заметить, что в \mathbb{Z} или в $K[X]$ имеется эффективный метод (раздел 6) вычисления НОД. Однако проблема разложения целых чисел на простые множители (или многочлена в произведение неприводимых) более трудна для решения. Есть, впрочем, проблема, имеющая, по-видимому, промежуточную сложность. Это задача проверки на простоту целого числа (или неприводимость многочлена). Читатель - скептик приглашается для проверки неприводимости для случая факторизации, с одной стороны, числа, имеющего в своей записи 78 десятичных цифр:

$$2^{257} - 1 = 231\,584\,178\,474\,632\,390\,847\,141\,970\,017\,375\,815 \\ 706\,539\,969\,331\,281\,128\,078\,915\,168\,015\,826\,259\,279\,871,$$

а с другой — полинома с коэффициентами из $\mathbb{Z}/2\mathbb{Z} : X^{2^{30}} - X$.

4. Многочлены с коэффициентами из поля

Читатель, несомненно, знаком с некоторыми результатами относительно распределения простых чисел или с теоремой Евклида, утверждающей бесконечность множества простых чисел. Что известно для многочленов? На самом деле, применение аргумента Евклида показывает, что существует бесконечное множество унитарных неприводимых многочленов над всяким полем. Для некоторых полей можно доказать существование неприводимых многочленов *любой заданной степени*. Например, над полем \mathbb{Q} рациональных чисел многочлен $X^n - 2$ является неприводимым для любого натурального числа n (что совершенно неочевидно и требует доказательства, например, для $n = 2$ это равносильно иррациональности $\sqrt{2}$).

Мы докажем, что для всякого простого p и для любого натурального числа n существует неприводимый многочлен степени n по модулю p , без явного указания его, — результат, который можно рассматривать как замечательный. Мы найдем также формулу, позволяющую подсчитать количество таких многочленов, исходя из которой читатель сможет найти плотность множества неприводимых многочленов по модулю p , имеющих данную степень. Но прежде всего, представим в явном виде алгоритм Евклида деления многочленов.

4.1. Евклидово деление в $K[X]$ (K - поле)

Теорема 32 (евклидово деление многочленов)

Пусть A и B два многочлена с коэффициентами в поле K , B — ненулевой. Существует алгоритм, позволяющий вычислить такие многочлены Q и R , что $A = BQ + R$ и $\deg(R) < \deg(B)$. Более того, указанная пара (P, Q) многочленов определяется единственным образом.

Дидактический пример

Рассмотрим сначала пример, который позволит лучше понять доказательство теоремы. Осуществим евклидово деление в $\mathbb{Q}[X]$ многочлена $A = 2X^5 - 3X^4 + 6X^3 - 9X^2 + 7X + 2$ на многочлен $B = 2X^2 - X + 1$ (на самом деле деление производится в $\mathbb{Z}[X]$, но это несущественно).

Разделим $2X^5$ на $2X^2$, старшие одночлены в A и B соответственно. Получим $1 \times X^3$ (старший одночлен частного). Теперь можно вычислить $A - B \times 1 \times X^3$ и получить многочлен степени ≤ 4 . Наконец, заменим B на $A - B \times 1 \times X^3$ и повторим для A и B описанную выше операцию... Оформим действия в виде следующей таблицы, форма которой читателю, возможно, знакома:

$$\begin{array}{rrrrrr|l}
 2X^5 & -3X^4 & +6X^3 & -9X^2 & +7X & +2 & 2X^2 - X + 1 \\
 & 0 & -2X^4 & +5X^3 & & & X^3 - X^2 + 2X - 3 \\
 & & & 0 & +4X^3 & -8X^2 & \\
 & & & & 0 & -6X^2 & +5X \\
 & & & & & 0 & +2X & +5
 \end{array}$$

Частное есть многочлен $Q = X^3 - X^2 + 2X - 3$, в то время как остаток $R = 2X + 5$.

Рассуждения, встретившиеся в описанном выше примере, должны быть формализованы не только ради математической деонтологии, но, главным образом, для того, чтобы показать рекуррентную основу, поддающуюся переделке в алгоритм, а затем в программу. Очевидно, что читатель легко сможет проследить на предыдущем примере доказательство теоремы, которое приводится ниже.

Доказательство теоремы 32

Пусть $n = \deg(A)$, $m = \deg(B)$ и b_m — старший коэффициент полинома B . Определим рекуррентным образом многочлены R_{m+k} для $k = n-m, n-m-1, \dots, 1$. На шаге k предположим, что верно равенство

$$A = B \times (\dots) + R_{m+k}, \text{ где } \deg(R_{m+k}) \leq m+k,$$

$a(\dots)$ — многочлен, название которого несущественно. Начнем рекурсию, положив $k = n - m$ и положив $R_n = A$ (получим $A = B \times 0 + A$). Если r_{m+k} обозначает коэффициент при X^{m+k} многочлена R_{m+k} и $q_k = r_{m+k}/b_m$, то запишем

$A = B \times (\dots + q_k X^k) + (R_{m+k} - B q_k X^k)$. Достаточно определить R_{m+k-1} с помощью равенства $R_{m+k-1} = R_{m+k} - B q_k X^k$ и проверить, многочлен ли это степени $\leq m + K - 1$. Получаем следующую рекуррентную схему:

$$R_n = A \text{ и } q_k = r_{k+m}/b_m, R_{m+k-1} = R_{m+k} - B q_k X^k, k = n - m, \dots, 1, 0,$$

в которой выполняется порождающее равенство ($0 \leq k \leq n - m + 1$):

$$A = B \left(\sum_{j=k}^{n-m} q_j X^j \right) + R_{k+m-1} \text{ и } \deg(R_{k+m-1}) \leq k + m - 1,$$

дающее для $k = 0$: $A = B \sum_{j=0}^{n-m} q_j X^j + R_{m-1}$ с $\deg(R_{m-1}) \leq m - 1 < \deg(B)$, что заканчивает описание полученного алгоритма евклидова деления. Доказательство единственности евклидова деления очень просто и предоставляется читателю.

Следствие 33

Кольцо $K[X]$ многочленов с коэффициентами в поле K является евклидовым кольцом без делителей нуля, евклидовым относительно степени (со значением в множестве $\{\infty\} \cup \mathbb{N}$).

```
(Q, R) ∈ K[X] × K[X] Quotient_Euclidien(A, B ∈ K[X]){
  n = deg(A), m = deg(B),
  R(0 .. n) и Q(0 .. n - m) два массива с компонентами в K.
  R(0 .. n) ← A(0 .. n);
  Q(0 .. n - m) ← 0;
  for(k = n - m; k ≥ 0; k = k - 1){ // A = BQ + R(0 .. m+k), Q(0 .. k) = 0
    Q(k) ← R(m+k)/B(m);
    R(k .. m+k-1) ← R(k .. m+k-1) - Q(k)B(0 .. m-1);
  }
  return (Q, R(0..m-1)); // A = BQ + R(0 .. m-1)
}
```

Алгоритм 2.22: Евклидово деление многочленов над полем

Алгоритм 2, отвечающий полученной рекуррентной схеме доказательства теоремы 32, существенно использует тот факт, что *единственный*

массив $R(0..n)$ достаточен для вычисления полиномов R_{m+k} . Действительно, переход от R_{m+k} к R_{m+k-1} происходит покоеффициентно (см. рекуррентную формулу, фигурирующую в доказательстве).

Этот алгоритм деления, разумеется, позволяет реализовать алгоритм Евклида для многочленов, который в любом случае больше не представляет затруднений.

4.2. Неприводимые многочлены с коэффициентами из $\mathbb{Z}/p\mathbb{Z}$

Согласно общим определениям раздела 1, неприводимым многочленом в $K[X]$ является многочлен степени $n \geq 1$ (т.е. не являющийся константой), не имеющий делителя степени, строго меньшей n (кроме постоянных величин, понятно). Следующий параграф уточняет строение неприводимых многочленов в $\mathbb{F}_p[X]$, где \mathbb{F}_p (для простого p) обозначает поле $\mathbb{Z}/p\mathbb{Z}$ целых чисел по модулю p . Докажем существование (в неявном виде) неприводимого многочлена по модулю p произвольной степени. В качестве извинения за отсутствие эффективности в следующем разделе даются критерий неприводимости многочлена и его применения в нескольких конкретных примерах.

Если Q — неприводимый многочлен, то уже показано, что фактор-кольцо $K[X]/(Q)$ является полем (предложение 22). Обозначение (Q) использовано для идеала, порожденного многочленом Q . Этот результат является фундаментальным, поскольку является ключевым инструментом в конструировании под полей. Вот несколько уточнений структуры $K[X]/(Q)$.

Предложение 34

Пусть Q — многочлен степени n с коэффициентами в поле K . Тогда факторкольцо $K[X]/(Q)$ есть векторное K -пространство размерности n . Если Q неприводим, то это — поле. Если дополнительно K — конечное поле, состоящее из k элементов, то $K[X]/(Q)$ будет полем, состоящим из k^n элементов.

(Основная часть доказательства состоит в том, чтобы убедиться, что $\bar{1}, \bar{X}, \dots, \bar{X}^{n-1}$ является базой $K[X]/(Q)$).

Теорема 35

Пусть p — простое число из \mathbb{N} и $n \in \mathbb{N}^*$. Для **неприводимого** многочлена Q из $\mathbb{F}_p[X]$ выполнено: $Q|X^{p^n} - X \Leftrightarrow \deg(Q)|n$.

Доказательство.

Пусть K — факторкольцо $\mathbb{F}_p[X]/(Q)$ и $d = \deg(Q)$. То, что K — конечное поле характеристики p , состоящее из p^d элементов, проверяется известным способом. Обозначим через \bar{X} образ X в K . Мультипликативная группа K^* состоит из $p^d - 1$ элементов, удовлетворяющих соотношению

$$y^{p^d-1} = 1, \text{ для любого } y \in K^* \quad (2.12)$$

Сначала докажем импликацию « \Leftarrow » («только тогда»), предполагая, что $d|n$. Это последнее свойство приводит к тому, что $p^d - 1$ делит $p^n - 1$. Затем с помощью равенства из предыдущего параграфа, примененного к $y = \bar{X}$, получаем: $\bar{X}^{p^n-1} = 1$, откуда $\bar{X}^{p^n} = \bar{X}$. Это последнее равенство в K интерпретируется в $\mathbb{F}_p[X]$ следующим образом:

$$X^{p^n} - X \equiv 0 \pmod{Q} \Rightarrow Q|X^{p^n} - X,$$

что и дает нужное заключение.

Доказательство импликации « \Rightarrow » требует большего внимания. Предположим, что $Q|X^{p^n} - X$. Тогда $X^{p^n} - X \equiv 0 \pmod{Q} \rightarrow \bar{X}^{p^n} = \bar{X}$. Но всякий элемент $y \in K$ представим в виде $R(\bar{X})$, где R — многочлен с коэффициентами из \mathbb{F}_p . Используя теперь тот факт, что поле K имеет характеристику p , заключаем: для всякого $y \in K$ выполнено: $y^{p^n} = R(\bar{X})^{p^n} = R(\bar{X}^{p^n}) = R(\bar{X}) = y$, откуда:

$$y^{p^n-1} = 1, \quad \forall y \in K^* \quad (2.13)$$

Поделим n на d : $n = dq + r$ с $0 \leq r < d$. Тогда имеем равенство $p^n - 1 = (p^{dq} - 1)p^r + p^r - 1$, которое ввиду равенств (4) и (5) дает: $y^{p^{p^r-1}} = 1$ для любого $y \in K^*$. Многочлен $R(Y) = Y^{p^r-1} - 1 \in K[Y]$ имеет степень $p^r - 1 < p^d - 1$, но обладает $p^d - 1$ корнями в K . Значит, это нулевой многочлен, откуда $p^r - 1 = 0$, т.е. $r = 0$. В этом случае $d|n$, что и требовалось.

■

Теорема 35 дает возможность найти число унитарных неприводимых многочленов степени n над полем \mathbb{F}_p (целых чисел по модулю p).

Следствие 36

Пусть I_p^n — число неприводимых унитарных многочленов в $\mathbb{F}_p[X]$ степени n .

$$(i) p^n = \sum_{d|n} dI_p^d.$$

(ii) $I_p^n \geq 1$ для всякого простого числа p и всякого натурального n . Другими словами, для любого натурального n существует неприводимы над \mathbb{F}_p многочлен степени n .

Доказательство.

Многочлен $X^{p^n} - X \in \mathbb{F}_p[X]$ не имеет сомножителей, являющихся полными квадратами. Действительно, если $X^{p^n} - X = U^2V$, то, находя производную от обеих частей, получаем: $-1 = 2UU'V + U^2V' = U(2U'V + UV')$, откуда следует, что U — константа.

Теорема 35 утверждает, что унитарные неприводимые многочлены Q с $\deg(Q)|n$ являются неприводимыми унитарными делителями $X^{p^n} - X$. Вышеприведенное рассуждение показывает, что эти многочлены являются *ровно один раз* в простом разложении $X^{p^n} - X$. Следовательно,

$$X^{p^n} - X = \prod_{d|n} \prod_{Q \in K_p^d} Q$$

где K_p^d обозначает множество неприводимых унитарных многочленов степени d над \mathbb{F}_p . Приравнявая степени в левой и правой частях равенства, получаем $p^n = \sum_{d|n} dI_p^d$, что доказывает (i).

Для доказательства (ii) заметим, что

$$p^d = \sum_{e|d} eI_p^e = dI_p^d + \sum_{\substack{e|d \\ e \neq d}} eI_p^e \geq dI_p^d.$$

Следовательно,

$$\begin{aligned} p^n &= \sum_{d|n} dI_p^d = nI_p^n + \sum_{\substack{d|n \\ n \neq d}} dI_p^d \leq nI_p^n + \sum_{\substack{d|n \\ d \neq n}} p^d \leq \\ &\leq nI_p^n + \sum_{d=0}^{n-1} p^d \leq nI_p^n + \frac{p^n - 1}{p - 1} \end{aligned}$$

или еще $nI_p^n \geq p^n - \frac{p^n - 1}{p - 1} \geq 1$, что и требовалось.



Вычисление количества неприводимых унитарных
многочленов степени n по модулю p

Из предыдущего следствия немедленно вытекает (для простого p), что $I_p^1 = p, I_p^2 = p(p-1)/2, I_p^3 = p(p^2-1)/3$ и вообще для **простого** n : $I_p^n = p(p^{n-1}-1)/n$.

Из формулы, доказанной в предыдущем следствии, можно получить рекуррентное соотношение, позволяющее найти I_p^n для произвольного n :

$$I_p^n = \frac{1}{n}(p^n - \sum_{\substack{d|n \\ d \neq n}} dI_p^d).$$

В таблице 2 приведены значения I_p^n (полученные, разумеется, с помощью Ада-программы) для $p = 2, 3, 5, 7$ и n от 1 до 10.

p	I_p^1	I_p^2	I_p^3	I_p^4	I_p^5	I_p^6	I_p^7	I_p^8	I_p^9	I_p^{10}
2	2	1	2	3	6	9	18	30	56	99
3	3	3	8	18	48	116	312	810	2184	5880
5	5	10	40	150	624	2580	11160	48750	217000	976248
7	7	21	112	588	3360	19544	117648	720300	4483696	28245840

Таблица 2.5: Число неприводимых многочленов по модулю p

Замечание. 976 248 неприводимых многочленов степени 10 над F^5 дают 976 248 полей вычетов, каждое из которых состоит из $5^{10} = 9\,765\,625$ элементов. В действительности (это классический результат теории конечных полей) все они изоморфны. Другими словами, поле с таким числом элементов всегда одно! Более точно, для каждой степени q простого числа существует единственное конечное поле, состоящее из q элементов (см. упр. 57).

Применение формулы обращения Мёбиуса (раздел 6.3) к формуле (i) следствия 36 сразу же дает

Следствие 37

Если μ — обычная функция Мёбиуса, то, сохраняя введенные выше обозначения, имеем:

$$I_p^n = \frac{1}{n} \sum_{d|n} \mu(d) p^{n/d}.$$

4.3. Простой критерий неприводимости по модулю p

Переформулировка теоремы 35 позволяет дать критерий неприводимости по модулю p . Этот критерий применяется к полиномам степени n , где n превосходит p и число p мало. Существует другой, эффективный критерий неприводимости по модулю p , принадлежащий Берлек- эмпу [21].

Следствие 38

Пусть p — простое натуральное число, $Q \in \mathbb{F}_p[X]$ имеет степень n . Q неприводим тогда и только тогда, когда для любого простого делителя q числа n выполнено:

$$Q | X^{p^n} - X \text{ и } \text{НОД}(X^{p^{n/q}} - X, Q) = 1.$$

Доказательство.

Допустим, что Q неприводим. По теореме 35 $Q | X^{p^n} - X$ и $Q \nmid X^{p^m} - X$, если m — собственный делитель n (т.е. $n \nmid m$), что доказывает первую половину следствия.

Допустим, что выполнены условия доказываемого критерия и R — неприводимый множитель Q . Тогда $R | X^{p^n} - X$ и $R \nmid X^{p^m} - X$ для любого простого делителя q числа n . Используя теорему 35, имеем: $\deg(R) | n$ и $\deg(R) \nmid n/q$ для любого простого делителя q числа n . Отсюда следует, что $\deg(R) = n$ и потому $R \sim Q$. Следовательно, Q — неприводимый многочлен.

■

Замечание. При реализации этого теста советуем работать в кольце $\mathbb{F}_p[X]/(Q)$ (элементы которого могут быть представлены массивами длины n) и вычислять X^{p^i} с помощью дихотомического алгоритма. Можно использовать также тот факт, что отображение $\Phi : x \mapsto x^p$ является линейным отображением $\mathbb{F}_p[X]/(Q)$ (допускающим кодирование при помощи матрицы размера $n \times n$) и тогда $X^{p^i} = \Phi^i(X)$.

Некоторые примеры тестов на неприводимость

1. Пусть многочлен $Q(X) = X^{10} + X^3 + 1$ из $\mathbb{F}_2[X]$. Согласно критерию, приведенному выше ($n = \deg(Q) = 10$ и $q = 2, 5$), имеем $(X^{2^{10}} - X) \bmod Q(X) = 0$, но $\text{НОД}(X^{2^5} - X, Q(X)) = 1$ для $q = 2$ и, наконец, $\text{НОД}(X^{2^2} - X, Q(X)) = 1$ для $q = 5$. Следовательно, полином $Q(X) = X^{10} + X^3 + 1$ неприводим по модулю 2.

2. Пусть многочлен $Q(X) = X^5 + X^4 + X^3 + X^2 + X - 1$ из $\mathbb{F}_3[X]$. Критерий, описанный выше, дает $(X^{3^5} - X) \bmod Q(X) = -X^2 - X - 1$. Этот результат достаточен для того, чтобы утверждать, что многочлен не является неприводимым. Второй шаг проверки, нахождение $\text{НОД}(X^3 - X, Q(X)) = 1$ при $q = 5$ не требуется. Впрочем, тест не позволяет найти ни одного делителя Q . ■

Итак, теперь мы знаем два основных примера евклидовых колец: \mathbb{Z} и $K[X]$ (K — поле), которые имеют, с точки зрения деления, одинаковое поведение. В частности, эти два кольца обладают тем важным свойством, сформулированным в предложении 31, что все их идеалы главные. Действительно, это свойство (как будет показано в следующем разделе) — единственное достаточное условие для выполнения основной теоремы арифметики.

5. Кольца главных идеалов или идеалистическая точка зрения

Работа математика заключается, между прочим, в изучении гипотез, имеющих важные последствия. Примером служит ситуация, встретившаяся здесь. Если читатель внимательно прочитал доказательства результатов раздела 4, то заметил, что свойство «всякий идеал главный» действительно становится главным (кроме, разумеется, предположений о целостности, т.е. отсутствия делителей нуля).

5.1. Идеализация

Математик идеализирует ситуацию следующим образом:

Определение 39 (кольцо главных идеалов)

Кольцо A называется **кольцом главных идеалов** (КГИ), если оно без делителей нуля и всякий его идеал главный.

Понятие кольца главных идеалов определяет новый класс факториальных колец, удовлетворяющих (не эффективным образом) соотношению Безу и строго содержащих класс евклидовых колец без делителей нуля:

Предложение 40

Кольцо главных идеалов A факториально и удовлетворяет соотношению Безу.

Доказательство.

Из того, что всякий идеал в A главный следует, с одной стороны, что A нётерово, а с другой, что A — кольцо Безу. Согласно полученным в разделах 2.1 и 2.4 результатам (предложения 9, 10 и 20), A факториально, что равносильно свойству Безу.

■

Замечание. Из того, что A — КГИ следует, что для любых a и $b \in A$ существует наибольший общий делитель (НОД) d этих элементов (задающийся равенством $Aa + Ab = Ad$) без способа вычисления этого НОД (в отличие от евклидовых колец). Арифметические свойства евклидовых колец и колец главных идеалов, в основном, те же: явное различие между этими двумя категориями заключается в исчислении объектов. Надо все-таки отметить, что существуют неевклидовы кольца главных идеалов, для которых имеются алгоритмы вычисления НОД и даже коэффициентов Безу (например, некоторые кольца квадратичных расширений кольца целых чисел, квадратичных полей, см. упр. 59).

Предложение 41

Классы колец, введенных к настоящему моменту, связаны следующими включениями:

$$\left\{ \begin{array}{l} \text{Евклидовы кольца} \\ \text{без делителей нуля} \end{array} \right\} \subsetneq \left\{ \begin{array}{l} \text{Кольца главных} \\ \text{идеалов} \end{array} \right\} \subsetneq \left\{ \begin{array}{l} \text{Факториальные} \\ \text{кольца} \end{array} \right\}.$$

Доказательство.

Единственная проблема состоит в том, чтобы показать, что указанные тут классы различны. Чуть позже в этой главе, мы покажем, что кольцо многочленов с целыми коэффициентами $\mathbb{Z}[X]$ факториально. Докажем сейчас, что не все идеалы в $\mathbb{Z}[X]$ являются главными. Рассмотрим, например, идеал $(X) + (2)$ и предположим наличие образующего элемента P для $(X) + (2)$. Тогда $2 \in (P)$, и P делит 2. Итак, $P = \pm 2$ или $P = \pm 1$. Возможность $P = \pm 2$ приводит к противоречию, так как $X \notin (2)$. Что касается другой возможности $P = \pm 1$, то она приводит к существованию полиномов $U(X)$ и $V(X)$ из $\mathbb{Z}[X]$, таких, что $1 = U(X) \times X + V(X) \times 2$. Подставив в последнее тождество $X = 0$, получим $1 = 2V(0)$, что абсурдно. Итак, $\mathbb{Z}[X]$ не КГИ. Существование КГИ, не являющегося евклидовым, более деликатная проблема и, как обычно, предоставляется читателю (упр. 21).

Приведем здесь только два известных кольца, являющихся КГИ, но не евклидовыми. Первое — кольцо A целых чисел из $\mathbb{Q}[\sqrt{-19}]$:

$$A = \mathbb{Z} + \frac{1 + \sqrt{-19}}{2} \mathbb{Z} = \left\{ \frac{x + y\sqrt{-19}}{2}, \text{ где } x \equiv y \pmod{2} \right\},$$

| Второе — факторкольцо $R[X, Y]/(X^2 + Y^2 + 1)$.

■

5.2. Частные кольца главных идеалов

Так как КГИ A является кольцом Безу, то всякий идеал, порожденный неприводимым элементом $q \in A$, максимален (т.е. факторкольцо $A/(q)$ есть тело). Это видно из предложения 22. Классические примеры: $\mathbb{Z}/p\mathbb{Z}$ и $K[X]/(P)$. Следующее предложение обобщает этот результат, описывая обратимые элементы всякого факторкольца $A/(q)$, результат совершенно аналогичный описанию известного факторкольца $\mathbb{Z}/n\mathbb{Z}$.

Предложение 42

В кольце главных идеалов понятия максимального, простого или неприводимого элементов совпадают. Пусть q не является ни нулевым, ни обратимым в кольце A .

(i) Пусть x — элемент A . Тогда \bar{x} обратим в $A/(q)$ тогда и только тогда, когда x и q взаимно просты, равносильным образом, существуют такие u и v , что $ux + vq = 1$.

(ii) В частности, $A/(q)$ — тело тогда и только тогда, когда q является неприводимым элементом A .

Доказательство.

Пусть d является НОД q и x , а u и v — коэффициентами Безу; q и x взаимно просты, т.е. $d = 1$. Отсюда следует (i). Эквивалентности:

$$\overline{ux} = 1 \text{ (в } A/(q)) \Leftrightarrow ux \equiv 1 \pmod{q} \Leftrightarrow ux \in 1 + Aq$$

заканчивают доказательство первого из утверждений. Второе теперь становится очевидным.

■

Примеры

Предыдущее предложение может рассматриваться как инструмент, позволяющий строить тела с помощью перехода к частному. Вот несколько элементарных примеров, в основе которых лежит этот фундаментальный принцип.

1. Каждое простое число p порождает поле $\mathbb{Z}/p\mathbb{Z}$ вычетов по модулю p . Кроме того, всякое простое p , сравнимое с 3 по модулю 4, неприводимо в $\mathbb{Z}[i]$ (предложение 8) и, следовательно, порождает поле $\mathbb{Z}[i]/p\mathbb{Z}[i]$, имеющее p^2 элементов. Например, $\mathbb{Z}[i]/7\mathbb{Z}[i]$ есть поле из 49 элементов. Эти элементы записываются в виде $a + \bar{i}b$, где a и b в $\mathbb{Z}[i]/7\mathbb{Z}[i]$ и \bar{i} (элемент факторкольца) удовлетворяет соотношению $\bar{i}^2 = -1$.

2. Рассмотрим многочлен $X^3 + X + 1$ с коэффициентами из $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z}$. Это неприводимый многочлен (можете проверить), что позволяет построить поле $K = \mathbb{F}_2[X]/(X^3 + X + 1)$ порядка 8, элементы которого имеют вид $a\alpha^2 + b\alpha + c$, где $a, b, c \in \mathbb{Z}/2\mathbb{Z}$ и α (элемент факторкольца) удовлетворяет соотношению $\alpha^3 = -\alpha - 1$.

6. Об оптимальных алгоритмах вычисления НОД

В предыдущих разделах мы рассмотрели алгоритм Евклида и различные математические понятия для его выражения. Чтобы закончить обсуждение, необходимо оценить сложность алгоритма Евклида в нескольких особых случаях и наметить способы получения оценок для более общих ситуаций. Первые оценки сложности алгоритма Евклида принадлежат Ламе [109] и являются предметом изучения первой части этого раздела. Эта теория проста и вполне доступна студенту первого курса университета, но заставляет прибегнуть к последовательности Фибоначчи.

Эта теория также провоцирует вопрос: «существуют ли оптимальные алгоритмы вычисления НОД?» и приводит к появлению понятий квазиевклидова кольца и квазиалгоритма. Наконец, теорема Дирихле порождает идею плотности пар целых простых чисел, полезная вещь для тех, кто хочет вычислять НОД более чем двух целых чисел.

6.1. Вычисление НОД двух целых чисел: теорема Ламе

Используемые обозначения те же, что и в предыдущих разделах. Пусть a и b — два натуральных числа, для которых необходимо вычислить НОД. Как обычно, алгоритм Евклида позволяет построить последовательность $(r_i)_{0 \leq i \leq n+1}$:

$$r_0 = a, r_1 = b \text{ и } r_{i-1} = r_i q_i + r_{i+1},$$

где $0 < r_{i+1} < r_i$ для $1 \leq i < n$, а $r_{n+1} = 0$,

последнее деление $r_{n-1} = r_n q_n$ является делением нацело. В этих условиях алгоритм Евклида требует n итераций для своей реализации (используемое деление — обычное деление натуральных чисел).

Определение 43

Последовательностью Фибоначчи $(F_n)_{n \in \mathbb{N}}$ называется последовательность, задаваемая следующим образом: $F_0 = 0, F_1 = 1$ и для $n \geq 0$: $F_{n+2} = F_{n+1} + F_n$.

Замечание. Эта последовательность названа в честь итальянского математика начала XIII века Леонардо Фибоначчи, который использовал ее при подсчете размножения кроликов (1202 г.), см. Кнут [97]. Эта последовательность естественным образом возникает при изучении процесса роста листы некоторых растений. Числа Фибоначчи связаны с золотым сечением через соотношение $F_p = (\phi^p - \hat{\phi}^p)/\sqrt{5}$, где ϕ — положительный корень уравнения (называемый золотым числом) $X^2 - X - 1 = 0$, $\phi = (1 + \sqrt{5})/2$ и $\hat{\phi} = (1 - \sqrt{5})/2$.

В данном исследовании для простоты предполагается, что $a > b$. Если это не выполнено, то легко видеть, что первое деление, выполненное по алгоритму Евклида, меняет ролями a и b и приводит нас именно к этому предположению.

Прежде чем формулировать теорему Ламе, установим несколько простых свойств, позволяющих сделать ее доказательство более естественным.

Свойство 44

Если для того, чтобы вычислить НОД чисел a и b , таких, что $a > b > 0$ алгоритм Евклида требует n итераций, то $a \geq F_{n+2}$ и $b \geq F_{n+1}$. Если к тому же $a = F_{n+2}$ и $b = F_{n+1}$, то алгоритм Евклида требует ровно n итераций для вычисления НОД(a, b) (их наибольший общий делитель равен 1).

Доказательство.

Прежде всего заметим, что $r_n \geq 1$ (в противном случае последнее деление дает нулевой остаток). Кроме того, $r_n < r_{n-1}$ и, следовательно, $r_{n-1} \geq 2$, откуда $r_n \geq F_2$ и $r_{n-1} \geq F_3$ (напомним: $F_2 = 1$ и $F_3 = 2$).

С другой стороны, при делении $r_{i-1} = r_i q_i + r_{i+1}$ частное q_i нулевое и потому $r_{i-1} \geq r_i + r_{i+1}$. С помощью нисходящей индукции можно доказать, что $r_i \geq F_{n+2-i}$ для $0 \leq i \leq n$, что и доказывает сформулированное свойство.

Чтобы доказать вторую часть, достаточно в явном виде продолжить последовательность делений: $F_{n+2} = F_{n+1} + F_n$, $F_{n+1} = F_n + F_{n-1}$, ... и, наконец, $F_3 = F_2 \times 2$ (действительно, $F_2 = F_1 = 1$). Отсюда следует, что

НОД(F_{n+2}, F_{n+1}) = $F_2 = 1$, и что необходимо ровно n итераций для реализации алгоритма Евклида на этой паре чисел.

■

Свойство 45

В предположении, что алгоритм Евклида требует n итераций, справедливы соотношения:

$$n + 2 < \frac{\log_{10} \sqrt{5}(a + 1)}{\log_{10} \phi} \text{ и } n + 1 < \frac{\log_{10} \sqrt{5}(b + 1)}{\log_{10} \phi}.$$

Доказательство.

Для доказательства этих соотношений достаточно сопоставить предыдущее свойство с соотношением, связывающим ϕ и F_n , приведенное в предыдущем примечании, зная, что $|\hat{\phi}| < 1$. Например,

$$\begin{aligned} a \geq F_{n+2} &\Leftrightarrow a \geq \frac{1}{\sqrt{5}}(\phi^{n+2} - \hat{\phi}^{n+2}) \Rightarrow \\ &\Rightarrow a > \frac{1}{\sqrt{5}}(\phi^{n+2} - 1) \Rightarrow a + 1 > \frac{\phi^{n+2}}{\sqrt{5}}. \end{aligned}$$

■

Теорема 46 (Ламе)

Число итераций, необходимых для вычисления НОД двух натуральных чисел a и b , таких, что $a > b > 0$, мажорируется 5-кратным числом десятичных знаков наименьшего из этих двух чисел. Более формально, если n является искомым числом итерации, то

$$n \leq 5(\lfloor \log_{10} b \rfloor + 1) \text{ или } n \leq 5\lceil \log_{10} (b + 1) \rceil.$$

Лемма 47

Пусть p — число десятичных цифр положительного целого числа b . Тогда

$$\left\lfloor \frac{\log_{10} \sqrt{5}(b + 1)}{\log_{10} \phi} \right\rfloor \leq 5p + 1.$$

Доказательство.

По условию $b \leq 10^p - 1$ и $\log_{10}(b+1) \leq p$. Следовательно,

$$\frac{\log_{10} \sqrt{5}(b+1)}{\log_{10} \phi} = \frac{\log_{10}(b+1)}{\log_{10} \phi} + \frac{\log_{10} \sqrt{5}}{\log_{10} \phi} \leq \frac{1}{\log_{10} \phi} p + \frac{\log_{10} \sqrt{5}}{\log_{10} \phi} = \alpha p + \beta$$

так как $\alpha \approx 4,7849$ меньше 5 и $\beta \approx 1,6722$ несколько меньше 2. Итак, можно записать: $\alpha p + \beta = 5p + \beta - (5 - \alpha)p$, формула, в которой число $(\beta - (5 - \alpha)p)$ мажорируется 1, когда $p \geq 4$, что дает

$$\frac{\log_{10} \sqrt{5}(b+1)}{\log_{10} \phi} \leq \alpha p + \beta \leq 5p + 1.$$

Чтобы завершить доказательство, остается рассмотреть случаи, когда $p = 1, 2$ или 3. Простые вычисления показывают, что когда $p \in [1, 3]$, имеем $\lceil \alpha p + \beta \rceil = 5p + 1$, что доказывает лемму.

■

Доказательство (теоремы Ламе).

Свойство 45 вместе с предыдущей леммой позволяет без труда доказать теорему Ламе. Действительно, из этих двух результатов выводим $n + 1 \leq 5p + 1$, т.е. $n \leq 5p$. Кроме того, утверждение, состоящее в том, что b имеет p десятичных цифр в записи, означает $p = \lfloor \log_{10} b \rfloor + 1 = \lceil \log_{10}(b+1) \rceil$.

■

Главный результат в этой теории — сложность алгоритма Евклида для целых чисел логарифмическая по отношению к наименьшему из двух чисел. (Обозначение $O(\log b)$ скрывает весьма существенную константу).

Замечание. В оценке Ламе коэффициент 5 оптимален, так как для следующих пар (13,8), (144,89) и (1597,987), соответствующих парам (F_7, F_6) , (F_{12}, F_{11}) и (F_{17}, F_{16}) , число итераций алгоритма Евклида, соответственно, 5, 10 и 15. Впрочем, это свойство не идет дальше F_{17} и F_{16} . Действительно, для пары

$$(F_{22}, F_{21}) = (17711, 10946)$$

число итераций равно 20, в то время как оценка Ламе 25. Это показывает, что если коэффициент 5 оптимален, то мажорирующая функция таковой не является.

6.2. Квазиевклидовы кольца

Мы только что доказали, что оценка, даваемая теоремой Ламе, оптимальна только в некоторых особых случаях. Но даже если бы выражение, даваемое этой теоремой, было наилучшим, вопрос об оптимальности алгоритма Евклида нахождения НОД двух целых чисел все равно остался бы. Действительно, обычное евклидово деление не приводит к оптимальному алгоритму Евклида, как видно из раздела 3.4. Цель данного раздела — найти деления, которые оптимизируют сложность этого алгоритма. До настоящего момента мы рассматривали, главным образом, три класса колец, связанные следующими включениями:

$$\left\{ \begin{array}{l} \text{Евклидовы кольца} \\ \text{без делителей нуля} \end{array} \right\} \subsetneq \left\{ \begin{array}{l} \text{Кольца главных} \\ \text{идеалов (КГИ)} \end{array} \right\} \subsetneq \left\{ \begin{array}{l} \text{Факториальные} \\ \text{кольца} \end{array} \right\}.$$

Главные их характеристики, которые нас интересуют, следующие:

- *Наличие* алгоритма Евклида и возможность эффективного *вычисления* НОД. Прототипами евклидовых колец являются \mathbb{Z} и множество многочленов над полем.
- *Наличие* НОД и разложения Безу без эффективного метода вычисления. Кольцо целых чисел из $\mathbb{Q}\sqrt{-19}$ является неевклидовым кольцом главных идеалов, в котором, между прочим, существует метод вычисления коэффициентов Безу [141] (кольцо целых $\mathbb{Q}\sqrt{-19}$, состоящее из элементов, которые являются корнями унитарных многочленов с коэффициентами из \mathbb{Z}).
- *Наличие и единственность* разложения на простые множители (основная теорема арифметики). Кольцо $A[X]$ многочленов с коэффициентами из A факториально, если таковым является A , но не является КГИ, если A не является телом.

Хотя понятие евклидова кольца более интересное с точки зрения эффективности, понятие алгоритма Евклида значительно сильнее. На практике оно ограничено методами, которые были представлены выше:

- В КГИ известно существование НОД, но нет, вообще говоря, простой эффективной процедуры его вычисления.
- Как будет видно из следующей главы, посвященной модулям над КГИ, что является основным содержанием главы, важным является не вычисление НОД с помощью деления, а вычисление коэффициентов Безу (что, конечно, приводит к нахождению НОД).
- В кольце главных идеалов наличие НОД зависит не столько от того, что всякий идеал главный, сколько от того, что он конечного типа (этого достаточно).

- Наконец, как показывают нижеследующие результаты, сходимость некоторого метода такого, как алгоритма Евклида, не обязательно связано с существованием алгоритма Евклида в рассматриваемом кольце.

Определение 48

Пусть A — коммутативное унитарное кольцо. **Квазиалгоритм** на A есть отображение φ множества $A \times A$ во вполне упорядоченное множество, обладающее следующим свойством:

$$\forall (a, b) \in A \times A^*, \exists (q, r) \in A \times A \text{ такая, что } a = bq + r \text{ и } \varphi(b, r) < \varphi(a, b)$$

Это равенство называется делением a на b . Кольцо A называется **квазиевклидовым**, если оно допускает квазиалгоритм φ . Говорят также, что A квазиевклидово относительно φ .

Примечание. Так как в случае евклидовых колец, термин *алгоритм* обозначал отображение, а не эффективный метод вычисления, то квазиалгоритм также будет полностью формальным объектом. Эта терминология нежелательна в работе, которая рассматривает алгоритмы в «информатическом» смысле.

Предложение 49

- (i) Всякое евклидово кольцо является квазиевклидовым.
- (ii) В квазиевклидовом кольце всякий идеал конечного типа главный (что приводит к наличию НОД двух элементов). Это означает, что всякое квазиевклидово кольцо является кольцом Везу.
- (iii) В частности, всякое нётерово квазиевклидово кольцо без делителей нуля является КГИ (в нётеровом всякий идеал конечного типа).

Доказательство (только для (ii)).

Пусть идеал A порожден двумя элементами. Допустим, что $(a, b) \in A \times A$ — пара образующих идеала $I, I = Aa + Ab$, где b отличен от нуля, для которой φ имеет наименьшее значение. Можно осуществить квазиевклидово деление a на b и записать $a = bq + r$, где $\varphi(b, r) < \varphi(a, b)$. Это приводит, в частности, к $Aa + Ab = Ab + Ar$ и противоречит выбору пары (a, b) . Следовательно, такой пары с $b \neq 0$ не существует, т.е. $b = 0$. Это запускает механизм индукции, которая не вызывает особых трудностей. Наличие НОД теперь выводится из того, что $Ad = Aa + Ab \Rightarrow [\delta|d \iff \delta|a \text{ и } \delta|b]$.



Замечание. Конечно, лучшее понимание квазиевклидова деления, основного для вычисления НОД, необходимо. Однако оно не позволяет провести эти вычисления. Как показывает предыдущее доказательство, НОД двух элементов кольца определяется единственным образом (это нулевой элемент пары, порождающий сумму идеалов, для которой значение квазиалгоритма минимально, что не является эффективным средством вычисления).

Алгоритм Евклида остается допустимым для квазиевклидовых колец и обладает той же сходимостью, что и в евклидовых кольцах.

Предложение 50

Коммутативное унитарное кольцо A является квазиевклидовым тогда и только тогда, когда оно удовлетворяет следующему условию:

$$\forall (r_0, r_1) \in A \times A, \exists n \in \mathbb{N}, \exists (q_1, \dots, q_n) \in A^n, \exists (r_2, \dots, r_{n+1}) \in A^n, \\ \text{такие, что: } \forall i \in [1, n] : r_{i-1} = r_i q_i + r_{i+1} \quad \text{и} \quad r_{n+1} = 0.$$

Кроме того, отображение φ которое со всякой парой $(r_0, r_1) \in A \times A$ ассоциирует наименьшее целое n , для которого существует цепь псевдоделений, оканчивающаяся нулевым остатком, является самым малым квазиалгоритмом, определенным на A .

Доказательство.

Согласно второму пункту предшествующего замечания, условие необходимо. Поэтому надо показать его достаточность. Итак, пусть A — кольцо, удовлетворяющее условию предложения 50. Ясно, что отображение φ есть квазиалгоритм для A . Действительно, пусть a, b и r такие, что минимальная цепь псевдоделений для пары (a, b) начинается с $a = bq + r$. Тогда, принимая во внимание минимальность φ , имеем $\varphi(b, r) + 1 \leq \varphi(a, b)$ и, следовательно, $\varphi(b, r) < \varphi(a, b)$. Проверка того, что указанный квазиалгоритм минимален, является простой формальностью.

■

Итак, данное предложение позволяет построить квазиалгоритм для квазиевклидова кольца. Это построение (если оно эффективно) определяет самый быстрый метод вычисления НОД через алгоритм «по Евклиду» в квазиевклидовом кольце. Действительно, Лазар [112] доказал следующие свойства:

Теорема 51 (Лазара)

(i) Обычное евклидово деление в $K[X]$ является евклидовым делением, согласно минимальному квазиалгоритму в $K[X]$. Алгоритм Евклида, следовательно, является самым быстрым методом вычисления НОД

двух многочленов с коэффициентами в поле через последовательные деления.

(ii) В \mathbb{Z} всякое евклидово деление с самым малым остатком является делением согласно минимальному квазиалгоритму в \mathbb{Z} .

(iii) Точнее, в \mathbb{Z} деление $a = bq + r$ есть деление по минимальному квазиалгоритму тогда и только тогда, когда $|r| < |b|/\phi$ (ϕ является золотым числом и $1/\phi \approx 0,6180339\dots$).

Доказательство.

Доказательство пункта (i) очень простое и фигурирует в упражнениях 42 и 43, находящихся в конце главы. Пункты (i) и (iii) не очень интересны для доказательства и не представляют больших трудностей. Например, можно проиллюстрировать пункт (iii). Пусть для вычисления НОД даны числа 4215 и 1177. Вот различные этапы алгоритма Евклида. Слева используется деление с наименьшим остатком, а справа деление, для которого отношение остатка к частному может превысить 0,5, все еще не превышая $1/\phi$:

$$\begin{aligned} 4215 &= 4 \times 1177 + (-493), \\ 1177 &= (-2) \times (-493) + 191, \\ -493 &= (-3) \times 191 + 80, \\ 191 &= 2 \times 80 + 31, \\ 80 &= 3 \times 31 + (-13), \\ 31 &= (-2) \times (-13) + 5, \\ -13 &= (-3) \times 5 + 2, \\ 5 &= 3 \times 2 + (-1), \\ 2 &= (-2) \times (-1) + 0, \end{aligned}$$

$$\begin{aligned} 4215 &= 4 \times 1177 + 684, & (*) \\ 1177 &= 2 \times 684 + (-191), \\ 684 &= (-3) \times (-191) + 111, & (*) \\ -191 &= (-2) \times 111 + 31, \\ 111 &= 3 \times 31 + 18, & (*) \\ 31 &= 2 \times 18 + (-5), \\ 18 &= (-3) \times (-5) + 3, & (*) \\ -5 &= (-2) \times 3 + 1, \\ 2 &= 3 \times 1 + 0, \end{aligned}$$

В решении по этому последнему алгоритму 4 деления, отмеченные звездочкой, дают остатки, абсолютное значение которых больше половины делителя, оставаясь в границах теоремы Лазара, а число итераций остается равным 9 (значение минимального квазиалгоритма для этих двух целых чисел).



6.3. Вычисление НОД нескольких целых чисел: теорема Дирихле

Когда необходимо вычислить НОД нескольких чисел, а не только двух, можно применить несколько методов:

- Распространение алгоритма Евклида, базирующегося на следующих свойствах:

$$(i) \text{НОД}(0, \dots, 0, a, 0, \dots, 0) = a,$$

$$(ii) \text{НОД}(u_1, \dots, u_i, \dots, u_n) = \text{НОД}(u_1 \bmod u_i, \dots, u_i, \dots, u_n \bmod u_i) \text{ при } u_i \neq 0.$$

За подробностями этого метода читатель может обратиться к упр. 24.

- Следующий метод заключается в повторном применении алгоритма Евклида для двух целых чисел. Он основывается на следующем свойстве: $\text{НОД}(u_1, \dots, u_n) = \text{НОД}(u_1, \text{НОД}(u_2, \dots, u_n))$, которое порождает рекурсивный алгоритм вычисления НОД. Именно, $\text{НОД}(u_1, \dots, u_n) = \text{НОД}(\text{НОД}(u_1, \dots, u_n), u_3, \dots, u_n)$, что является основой соответствующего итеративного алгоритма.

Этот последний метод не только упрощает реализацию вычисления — действительно, достаточно несколько раз применить уже реализованный алгоритм — но и имеет неоспоримое достоинство с точки зрения эффективности. Неформально говоря, главное заключается в следующем: выбирают два числа в последовательности, для которой надо вычислить НОД. Затем, сделав первое вычисление, заменяют два выбранные числа на их НОД и повторяют алгоритм. Выигрыш в эффективности получается из того, что как только находят НОД, равный единице, вычисление может быть прервано. Неиспользованные числа ничего не могут добавить к полученным результатам. К тому же это явление довольно распространенное, потому что, как утверждает теория Дирихле, более, чем в 60% случаев два целых числа, взятые случайно, взаимно просты. Продолжение этого раздела посвящается двум доказательствам теоремы Дирихле. Одно — эвристическое (короткое и неправильное), а другое — более длинное, но верное. Второе доказательство требует введения функции Мёбиуса, и мы воспользуемся случаем доказать формулу обращения Мёбиуса — ту формулу, которая уже была использована в разделе 4.2.

Теорема 52 (Дирихле)

Если u и v — два натуральных числа, выбранные случайно, то вероятность того, что они взаимно просты, равна $6/\pi^2 \approx 0,607927$. Более формально, если

$$H_1^n = \{(u, v) \in \mathbb{N}^{*2} / 1 \leq u \leq n, 1 \leq v \leq n \text{ и } \text{НОД}(u, v) = 1\}$$

$$\text{то } p = \lim_{n \rightarrow \infty} \frac{\#H_1^n}{n^2} = \frac{6}{\pi^2}, \text{ где } \#H_1^n \text{ означает мощность множества } H_1^n$$

Обозначения, использующиеся в следующих ниже доказательствах, таковы. Для натурального числа d и вещественного положительного x определим:

$$H_d = \{(u, v) \in \mathbb{N}^{*2} / \text{НОД}(u, v) = d\},$$

$$H_d^x = \{(u, v) \in \mathbb{N}^{*2} / 1 \leq u \leq x, 1 \leq v \leq x \text{ и } \text{НОД}(u, v) = d\}.$$

В этих обозначениях x часто будет заменяться на натуральное число, как в формулировке теоремы Дирихле. Множество H_1 необходимо для оценки функции распределения в \mathbb{N}^{*2} .

Эвристическое доказательство (теоремы Дирихле).

Утверждение $\text{НОД}(u, v) = d$ равносильно тому, что u и v кратны d и что $\text{НОД}(u/d, v/d) = 1$ (т.е. $(u/d, v/d) \in H_1$). К тому же для всякого натурального n множества H_d^{nd} и H_1^n имеют одну и ту же мощность. Следовательно, для $x > 0$:

$$\frac{\#H_d^x}{x^2} = \frac{\#H_1^{x/d}}{x^2} = \frac{\#H_1^{x/d}}{(x/d)^2} = \frac{\#H_1^{x/d}}{(x/d)^2} \times \frac{1}{d^2}$$

Переходя к пределу, получим:

$$\lim_{x \rightarrow \infty} \frac{\#H_d^x}{x^2} = \frac{1}{d^2} \times \left(\lim_{x \rightarrow \infty} \frac{\#H_1^x}{x^2} \right) = \frac{p}{d^2},$$

где последнее число — «вероятность» того, что два числа имеют наибольший общий делитель d . С другой стороны, можно записать $\mathbb{N}^{*2} = \cup_{d=1}^{\infty} H_d$, где объединение является объединением непересекающихся множеств (H_d — есть множество пар целых натуральных чисел с НОД равным d). Тогда можно заключить, что

$$1 = \sum_{d=1}^{\infty} p/d^2 = p \sum_{d=1}^{\infty} 1/d^2 = p\pi^2/6.$$

■

Настоятельно просим читателя найти ошибку в этом интуитивном доказательстве.

А сейчас переходим к доказательству теоремы Дирихле. Для этого доказательства нужна функция Мёбиуса вне связи с теорией арифметических функций, где она обычно появляется.

Свойство 53

Назовем функцией Мёбиуса функцию μ , определенную на \mathbb{N}^* следующим образом:

$$\mu(k) = \begin{cases} 1, & \text{если } k = 1, \\ (-1)^r, & \text{если } k = p_1 p_2 \dots p_r, \text{ где } p_i = p_j \text{ при } i \neq j \text{ — простые числа,} \\ 0 & \text{в противном случае.} \end{cases}$$

Эта функция тесно связана с частичным упорядочиванием на множестве целых чисел с помощью отношения делимости. Для целого числа $d > 1$ функция μ удовлетворяет соотношению $\sum_{k|d} \mu(k) = 0$.

Доказательство.

Пусть $d = p_1^{\alpha_1} \dots p_r^{\alpha_r}$ — каноническое разложение d . Достаточно рассмотреть только делители d , свободные от квадратов, так как функция μ на других делителях исчезает, т.е. такие делители числа d , которые разлагаются в произведение простых p_i с показателями 0 или 1. Следовательно, чтобы найти такие делители, достаточно выбрать j различных чисел i и получить

$$\sum_{k|d} \mu(k) = \sum_{j=0}^r (-1)^j \times \binom{r}{j} = (1 + (-1))^r,$$

согласно определению μ .

■

Теперь мы получим формулу обращения Мёбиуса в не совсем привычной форме

Предложение 54

Пусть f и g — две функции от \mathbb{R}_+^* со значениями в некоторой аддитивной абелевой группе. Тогда

$$f(x) = \sum_{k=1}^{|x|} g\left(\frac{x}{k}\right) \iff g(x) = \sum_{k=1}^{|x|} \mu(k) \times f\left(\frac{x}{k}\right).$$

Доказательство.

Используемый метод заключается в перенесении выражения от f , задаваемого первой формулой, во вторую:

$$\begin{aligned} \sum_{k=1}^{|x|} \mu(k) \times f\left(\frac{x}{k}\right) &= \sum_{k=1}^{|x|} \left(\mu(k) \times \sum_{k=1}^{\lfloor x/k \rfloor} g\left(\frac{x/k}{p}\right) \right) \\ &= \sum_{h \leq x} \mu(k) \times f\left(\frac{x}{kp}\right) \end{aligned}$$

Учитывая, что $\lfloor \lfloor x \rfloor / k \rfloor = \lfloor x/k \rfloor$, можно сделать следующую замену переменных: $h = kp$ с $1 \leq h \leq x$ и $k|h$, что дает

$$\sum_{k=1}^{\lfloor x \rfloor} \times f\left(\frac{x}{k}\right) = \sum_{h=1}^{\lfloor x \rfloor} \left(g\left(\frac{x}{h}\right) \times \sum_{k|h} \mu(k) \right)$$

Согласно свойству 53, в этой сумме есть только одно ненулевое слагаемое, именно то, которое отвечает значению $h = 1$ и, следовательно $\sum_{k=1}^{\lfloor x \rfloor} \mu(k) \times f(x/k) = g(x)$.

■

Доказательство теоремы Дирихле будем осуществлять в три этапа.

Лемма 55

Пусть x — вещественное положительное число. Обозначим через q_x число элементов множества H_1^x . Тогда $q_x = \sum_{k \geq 1} \mu(k) \times \lfloor x/k \rfloor^2$, формула, в которой, на первый взгляд, бесконечное число слагаемых, но только конечное их число отлично от нуля.

Доказательство.

Пусть $q_{d,x}$ — множество элементов множества H_d^x . Множество пар натуральных чисел, не превосходящих x , есть, как это было видно в эвристическом доказательстве, теоретико-множественная сумма непересекающих подмножеств H_d^x , где d изменяется от 1 до $\lfloor x \rfloor$: $\lfloor x \rfloor^2 = \sum_{d=1}^{\lfloor x \rfloor} q_{d,x} = \sum_{d=1}^{\lfloor x \rfloor} q_{\lfloor x/d \rfloor}$, так как $\#H_d^x = \#H_1^{\lfloor x/d \rfloor}$. Применим к этому выражению формулу обращения, фигурирующую в предложении 54, отождествляя функцию f с функцией $x \mapsto \lfloor x \rfloor^2$ и функцию g с функцией $x \mapsto q_x$, и получим искомый результат.

■

Следующий этап доказательство требует (это было неизбежно) вычисления пределов и суммы ряда.

Лемма 56

В предшествующих обозначениях для натурального числа n имеем

$$\lim_{n \rightarrow \infty} \frac{q_n}{n^2} = \sum_{k=1}^{\infty} \mu(k)/k^2.$$

Доказательство.

Ряд, фигурирующий в правой части формулы, является, очевидно, абсолютно сходящимся, так как функция Мёбиуса мажорируется по абсолютной величине единицей. Итак, достаточно оценить разность между общими членами этих двух последовательностей и показать, что она стремится к нулю. Имеем:

$$\sum_{k=1}^n \frac{\mu(k)}{k^2} - \frac{q_n}{n^2} = \sum_{k=1}^n \mu(k) \times \left(\frac{1}{k^2} - \left\lfloor \frac{n}{k} \right\rfloor^2 \times \frac{1}{n^2} \right).$$

Кроме того, для всякого вещественного положительного числа x имеем $0 \leq x - \lfloor x \rfloor < 1$ и, следовательно, $0 \leq 1/k - \lfloor n/k \rfloor/n \leq 1/n$. В этих условиях

$$0 \leq \frac{1}{k^2} - \frac{1}{n^2} \times \left\lfloor \frac{n}{k} \right\rfloor^2 = \left(\frac{1}{k} - \frac{1}{n} \times \left\lfloor \frac{n}{k} \right\rfloor \right) \times \left(\frac{1}{k} + \frac{1}{n} \times \left\lfloor \frac{n}{k} \right\rfloor \right) \leq \frac{1}{n} \times \frac{2}{k}.$$

Переносим эти значения в разность для мажорирования, получаем:

$$\left| \frac{q_n}{n^2} - \sum_{k=1}^n \frac{\mu(k)}{k^2} \right| \leq \frac{2}{n} \times \sum_{k=1}^n \frac{1}{k} \leq \frac{2 \log n}{n}$$

величина, стремящаяся к нулю, когда n стремится к бесконечности. Итак, последовательность $(q_n/n^2)_{n \in \mathbb{N}}$ сходится и имеет тот же предел, что и ряд.

■

Для того, чтобы закончить доказательство теоремы Дирихле, остается вычислить сумму ряда с общим членом $\mu(k)/k^2$. Для этого докажем, что $(\sum_{k=1}^{\infty} \mu(k)/k^2) \times (\sum_{k=1}^{\infty} 1/k^2) = 1$.

Оба рассматриваемых ряда — абсолютно сходящиеся, и потому можно изменить порядок суммирования следующим образом:

$$\left(\sum_{k=1}^{\infty} \frac{\mu(k)}{k^2} \right) \times \left(\sum_{m=1}^{\infty} \frac{1}{m^2} \right) = \sum_{k=1}^{\infty} \sum_{m=1}^{\infty} \frac{\mu(k)}{m^2 k^2} = \sum_{d=1}^{\infty} \left(\sum_{k|d} \mu(k) \right) \times \frac{1}{d^2}.$$

По уже доказанному свойству 53 самая внутренняя сумма нулевая, за исключением случая, когда $d = 1$. Сумма ряда с общим членом $1/k^2$ равна $\pi^2/6$. Теорема Дирихле доказана.

Конец этого раздела посвящен классической формуле обращения Мёбиуса.

Рассмотрим множество \mathbb{N}^* , упорядоченное с помощью отношения делимости. В этой структуре 1 — наименьший элемент и всякий интервал $[a, b]$ конечен. $(\mathbb{N}^*, |)$ — упорядоченное множество, являющееся локально конечным.

Поэтому на множестве функций F , определенных на \mathbb{N}^* со значениями в A , можно ввести внутренний закон композиции.

Определение 57

На F определен закон внутренней композиции $*$, называемый арифметическим произведением, по следующему правилу:

$$\forall f \in F, \forall g \in F, \forall n \in \mathbb{N}^*, \quad (f * g)(n) = \sum_{d|n} f(d)g\left(\frac{n}{d}\right).$$

Свойство 58 (произведения $*$)

- (i) Произведение ассоциативно и коммутативно.
- (ii) Функция Кронекера δ , определяемая как $\delta(1) = 1$ и $\delta(n) = 0$, если $n > 1$, есть нейтральный элемент для произведения $*$.
- (iii) Элемент $f \in F$ обратим для операции $*$ тогда и только тогда, когда $f(1)$ обратим.
- (iv) Множество F , наделенное обычным сложением функций и операцией $*$, является коммутативным унитарным кольцом.

Доказательство (только для пункта (iii)).

Пусть $f \in F$ такой, что $f(1) \in U(A)$. Определим g по индукции следующим образом:

$$g(1) = f(1)^{-1} \quad \text{и} \quad g(n) = -f(1)^{-1} \sum_{\substack{d|n \\ d \neq n}} g(d)f\left(\frac{n}{d}\right) \quad \text{при } n > 1$$

Простая проверка показывает, что g — обратный элемент для f . Остальные утверждения теоремы немедленно выводятся из определения арифметического произведения.

■

Свойство 59

Пусть ξ — элемент из F , определяемый по правилу $\xi(n) = l, \forall n \in \mathbb{N}^*$. Тогда в $(F, *)$ функции ξ и μ обратны друг другу.

Проблема, решаемая с помощью формулы обращения Мёбиуса, следующая. Пусть g — функция из \mathbb{N}^* в A и пусть f — функция, определенная по правилу $\sum_{d|n} g(d)$ для $n \in \mathbb{N}^*$. Можно ли в этом случае выразить функцию g через f ? Другими словами, можно ли найти обращения этой формулы? Ответ дает следующая

Теорема 60 (Формула обращения Мёбиуса)

Пусть f и g — две функции в F , такие, что для любого $n \in \mathbb{N}^*$ справедливо соотношение $f(n) = \sum_{d|n} g(d)$. Тогда можно выразить g через функцию f : $g(n) = \sum_{d|n} \mu(d)f(n/d)$, где μ — функция Мёбиуса.

Доказательство (формулы обращения Мёбиуса).

Выражение f через функцию g описывается в терминах умножения $*$: $f = \xi * g$, а так как ξ и μ взаимно простые элементы относительно операции $*$ (свойство 59), то получаем $g = \mu * f$.

■

Более общие сведения по теории арифметических функций можно получить, обратившись к монографии Бержа [19].

7. Расширенный алгоритм Евклида

Этот раздел посвящен изучению эффективного метода получения коэффициентов Безу в евклидовом и квазиевклидовом кольце. Надо отметить, что квазиевклидовым случаем — не единственная возможность для построения таких алгоритмов (известным примером является неевклидово кольцо главных идеалов кольца целых алгебраических чисел в $\mathbb{Q}(\sqrt{-19})$). В следующей главе мы увидим, что наличие коэффициентов Безу является главным ключом к классификации модулей над кольцом главных идеалов (теория инвариантных множителей): кто умеет их вычислять, тот умеет решать эффективным образом задачи линейной алгебры над кольцом главных идеалов.

7.1. Вычисление коэффициентов Безу в квазиевклидовом кольце

Для квазиевклидова кольца A разновидность алгоритма Евклида, предложенная в разделе 3.2, позволяет вычислить коэффициенты Безу. Используемые обозначения те же, что в разделе 3.2. Алгоритм, примененный к паре чисел a, b , порождает последовательность $(r_i)_{0 \leq i \leq n+1}$ такую, что

$$r_{i-1} = r_i q_i + r_{i+1} \text{ для } 1 \leq i \leq n, \text{ где } r_0 = a, r_1 = b, r_{n+1} = 0.$$

Элемент r_{i+1} является линейной комбинацией r_i и r_{i-1} ($r_{i+1} \in Ar_i + Ar_{i-1}$). Так как $r_0 = 1 \cdot a + 0 \cdot b$, $r_1 = 0 \cdot a + 1 \cdot b$, то по предыдущему рекуррентному соотношению для $i < n$ получаем, что $r_n = \text{НОД}(a, b)$ — линейная комбинация a и b . Точнее, предполагая, что $r_i = u_i a + v_i b$, получаем:

$$r_{i+1} = r_{i-1} - q_i r_i = (u_{i-1} - q_i u_i) a + (v_{i-1} - q_i v_i) b.$$

Из этих формул легко получается рекуррентная последовательность:

$$\begin{cases} u_0 = 1, & v_0 = 0, & r_0 = a, \\ u_1 = 0, & v_1 = 1, & r_1 = b, \\ u_{i+1} = u_{i-1} - q_i u_i, & v_{i+1} = v_{i-1} - q_i v_i, & r_{i+1} = r_{i-1} - q_i r_i \end{cases}$$

из которой теперь и следует классический результат: $r_n = \text{НОД}(a, b) = u_n a + v_n b$. Эти соотношения приводят, к тому же, к рекуррентному алгоритму, изображенному ниже, в котором тройка (u, v, r) соответствует (u_i, v_i, r_i) и тройка (u', v', r') соответствует $(u_{i+1}, v_{i+1}, r_{i+1})$. Переменная i , бесполезная для алгоритма, присутствует в комментариях только для того, чтобы придать смысл утверждениям.

```

int Bezout ((a,b)∈ A){
     $\begin{pmatrix} u \\ u' \end{pmatrix} \leftarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \quad \begin{pmatrix} v \\ v' \end{pmatrix} \leftarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix}; \quad \begin{pmatrix} r \\ r' \end{pmatrix} \leftarrow \begin{pmatrix} a \\ b \end{pmatrix};$ 
    i = 0;
    while(r' != 0){ // (u, v, r) = (u_i, v_i, r_i), (u', v', r') = (u_{i+1}, v_{i+1}, r_{i+1})
        i++; // (u, v, r) = (u_{i-1}, v_{i-1}, r_{i-1}), (u', v, r') = (u_i, v_i, r_i)
        q = r/r'; // q = q_i
         $\begin{pmatrix} u & v & r \\ u' & v' & r' \end{pmatrix} \leftarrow \begin{pmatrix} u' & v' & r' \\ q - qu' & v - qv' & r - qr' \end{pmatrix};$ 
    }
    return (u,v); // Коэффициенты Безу для пары (a,b)
}

```

Вот другое доказательство, основанное на эквивалентном представлении того же алгоритма. Для этого все рекуррентные соотношения (6) запишем в матричной форме:

$$\begin{pmatrix} u_i & v_i & r_i \\ u_{i+1} & v_{i+1} & r_{i+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_i \end{pmatrix} \begin{pmatrix} u_{i-1} & v_{i-1} & r_{i-1} \\ u_i & v_i & r_i \end{pmatrix}$$

$$\text{и} \quad \begin{pmatrix} u_0 & v_0 & r_0 \\ u_1 & v_1 & r_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \end{pmatrix}.$$

Эти равенства дают:

$$\begin{vmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{vmatrix} = - \begin{vmatrix} u_{i-1} & v_{i-1} \\ u_i & v_i \end{vmatrix} \Rightarrow \begin{vmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{vmatrix} = (-1)^i$$

Затем:

$$\begin{pmatrix} u_i & v_i \\ u_{i+1} & v_{i+1} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}, \text{ и его «обращение»}$$

$$\begin{pmatrix} a \\ b \end{pmatrix} = (-1)^i \begin{pmatrix} v_{i+1} & -v_i \\ -u_{i+1} & u_i \end{pmatrix} \begin{pmatrix} r_i \\ r_{i+1} \end{pmatrix}.$$

```
int Bezout(a,b ∈ A){
   $\begin{pmatrix} u \\ u' \end{pmatrix} \leftarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \quad \begin{pmatrix} v \\ v' \end{pmatrix} \leftarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix}; \quad \begin{pmatrix} r \\ r' \end{pmatrix} \leftarrow \begin{pmatrix} a \\ b \end{pmatrix};$ 
  while (r' != 0){
     $\begin{pmatrix} u & v \\ u' & v' \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ r' \end{pmatrix}, \quad \begin{vmatrix} u & v \\ u' & u \end{vmatrix} = \pm 1$ 
    q=r/r';
     $\begin{pmatrix} u & v & r \\ u' & v' & r' \end{pmatrix} \leftarrow \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix} \begin{pmatrix} u & v & r \\ u' & v' & r' \end{pmatrix}$ 
  }
  return (u,v);  ua + vb — есть НОД a и b
}
```

Алгоритм 2.23: Алгоритм Безу для квазиевклидова кольца

Следовательно, для $i = n$ имеем: $a = (-1)^n v_{n+1} r_n$, $b = (-1)^{n+1} u_{n+1} r_n$ и $r_n = u_n a + v_n b$. Последние соотношения *явно* показывают, что r_n является, с одной стороны, общим делителем, а с другой — линейной комбинацией a и b . Это порождает новое *эффективное* доказательство, поскольку r_n является НОД a и b .

Этот подход позволяет построить самодостаточный алгоритм (3), в котором больше не фигурирует переменная i .

i	q_i	u_i	v_i	r_i	u_{i+1}	v_{i+1}	r_{i+1}
0		1	0	1292	0	1	798
1	0	0	1	798	1	-1	494
2	1	1	-1	494	-1	2	304
2	1	-1	2	304	2	-3	190
3	1	2	-3	190	-3	5	114
4	1	-3	5	114	5	-8	76
5	1	5	-8	76	-8	13	38
6	2	<u>-8</u>	<u>13</u>	<u>38</u>	21	-34	0

Таблица 2.6: Вычисление коэффициентов Безу

В таблице 3 приведен пример вычисления коэффициентов Безу в \mathbb{Z} . Этот пример может быть полезен для понимания следующего параграфа.

В рассматриваемом примере $a = 1292$, $b = 798$, их НОД $= 38$ и найденные коэффициенты Безу $u = -8$ и $v = 13$ (подчеркнутые числа).

Коэффициенты Безу часто применяются для вычисления обратного элемента в $\mathbb{Z}/n\mathbb{Z}$. Пусть, например, требуется обратить класс 34 в $\mathbb{Z}/235\mathbb{Z}$ (34 взаимно просто с 235, следовательно, обратимо по модулю 235): алгоритм Безу дает соотношение $1 = 11 \times 235 - 76 \times 34$, и обратным к 34 по модулю 235, следовательно, является $-76 = 159$. Операция обращения часто необходима в модулярной арифметике. Иногда эта конструкция требуется и в других кольцах, например, в кольце целых чисел Гаусса. Так, алгоритм Безу, примененный в $\mathbb{Z}[i]$ к числам $23+14i$ и $7+5i$, дает $1 = (-3+2i) \times (23+14i) + (9-7i) \times (7+5i)$ и, следовательно, обратным к элементу $7+5i$ по модулю $23+14i$ будет $9-7i$.

7.2. Мажорирование коэффициентов Безу в \mathbb{Z}

Равенства $ua + vb = (u - kb)a + (v + ka)b$ и для d , делящего a и b , $ua + vb = (u - kb/d)a + (v + ka/d)b$ показывают, что существует много пар (u, v) , для которых $\text{НОД}(a, b) = ua + vb$. Расширенный алгоритм Евклида, полученный в предыдущем разделе, позволяет вычислить такую пару (u, v) , что, за исключением лишь некоторых особых случаев, выполняются неравенства $|u| \leq |b/2d|$ и $|v| \leq |a/2d|$.

Эти оценки являются объектом исследования для следующего предложения. Единственность такой пары (u, v) , удовлетворяющей указанным неравенствам (упр. 33), доказывает, что пара Безу, получаемая алгоритмом Евклида, является самой «красивой».

Предложение 61

(i) Пусть a и b — различные строго положительные целые числа, и пусть d их НОД. Пусть $(u_i)_{0 \leq i \leq n+1}$ — последовательности, полученные расширенным алгоритмом Евклида. В этих условиях последовательности $|u_i|_{1 \leq i \leq n+1}$ и $|v_i|_{0 \leq i \leq n+1}$ являются возрастающими, не переполняют разрядную сетку машины — в предположении, что a и b представимы машинными кодами — и указанный алгоритм дает коэффициенты Безу u, v , удовлетворяющие оценкам:

$$|u| \leq |b/2d|, \quad |v| \leq |a/2d|. \quad (7)$$

(ii) Если (a, b) — пара целых чисел, отличная от $(0, a)$, $(a, 0)$ и $(a, \pm 1)$, то существуют коэффициенты Безу, удовлетворяющие неравенствам (7)

Доказательство.

Напомним классические обозначения:

$$r_{i-1} = r_i q_i + r_{i+1}, \quad u_{i+1} = u_{i-1} - u_i - u_i q_i, \quad v_{i+1} = v_{i-1} - v_i q_i, \text{ и } u_0 = v_1 = 1, \quad u_1 = v_0 = 0.$$

Легко убедиться, что $u_{2i} \geq 0$ и $u_{2i+1} \leq 0$; значит, $|u_{i+1}| \geq |u_i q_i|$, откуда видно (ввиду $q_i > 0$), что последовательность $(|u_i|)$ возрастающая для $i \geq 1$. В конце алгоритма имеем $|u_{n+1}| \geq |q_n u_n|$, где $q_n \geq 2$, что неверно только для случаев, выписанных в явном виде в (ii). Однако $|u_{n+1}| = a/d$, что заканчивает доказательство (для v_i доказательство аналогично).

■

8. Факториальность кольца многочленов

Прежде чем закончить эту главу, «пробежимся» по кольцам многочленов, что позволит построить приемлемые алгоритмы вычисления НОД с эффективными оценками трудоемкости. Но сначала немного теории.

Теорема 62

(i) Если A — унитарное нётерово коммутативное кольцо, то кольцо многочленов $A[X]$ нётерово. То же верно и для кольца $A[X_1, \dots, X_n]$.

(ii) Если K — поле, то $K[X_1, \dots, X_n]$ нётерово.

(iii) Кольцо многочленов $\mathbb{Z}[X_1, \dots, X_n]$ с целыми коэффициентами нётерово.

Для доказательства теоремы нам понадобятся некоторые простые результаты. Пусть n — натуральное число, I — идеал в $A[X]$. Обозначим через $\text{dom}_n(I)$ часть A , состоящую из коэффициентов при старших (доминирующих) членах многочленов из I , имеющих степень в точности равную n , и к которой добавлена константа 0.

Лемма 63

(i) $\text{dom}_n(I)$ — идеал в A .

(ii) Если $n \leq m$, то $\text{dom}_n(I) \subset \text{dom}_m(I)$ (рассмотреть $X^{m-n}P$ для $P \in I$, имеющего степень n).

(iii) Если $I \subset J$, то $\text{dom}_n(I) \subset \text{dom}_n(J)$.

Лемма 64

Пусть $I \subset J$ — два идеала в $A[X]$, такие, что для всякого n : $\text{dom}_n(I) = \text{dom}_n(J)$. Тогда $I = J$.

Доказательство.

Пусть P — элемент J . Если P степени 0, то очевидно (так как $\text{dom}_0(I) = \text{dom}_0(J)$), что $P \in I$. В остальных случаях будем использовать индукцию по степени n полинома P . Пусть $P = aX^n + \dots$ и $a \in \text{dom}_n(J) = \text{dom}_n(I)$. Поэтому существует $Q \in I$, такой, что $Q = aX^n + \dots$. Многочлен $P - Q$ имеет степень, меньшую, чем n , и принадлежит J . По предположению индукции получаем $-Q \in I$, а тогда $P \in I$.

■

Доказательство теоремы 62.

Рассмотрим возрастающую последовательность (I_i) идеалов в $A[X]$. Семейство идеалов $(\text{dom}_n(I_i))_{i,n}$ имеет максимальный элемент (но не максимум на данный момент) $\text{dom}_{n_0}(I_{i_0})$. Следовательно, для всякого $n \geq n_0$ и для всякого $i \geq i_0$ $\text{dom}_n(I_i) = \text{dom}_{n_0}(I_{i_0})$. Рассмотрим таблицу:

$$\begin{array}{ccccccc}
 \text{dom}_1(I_1) & \subset & \text{dom}_2(I_1) & \subset \dots \subset & \text{dom}_{n_0}(I_1) & \subset & \text{dom}_{n_0+1}(I_1) & \subset \dots \\
 \cap & & \cap & & \cap & & \cap & \\
 \text{dom}_1(I_2) & \subset & \text{dom}_2(I_2) & \subset \dots \subset & \text{dom}_{n_0}(I_2) & \subset & \text{dom}_{n_0+1}(I_2) & \subset \dots \\
 \cap & & \cap & & \cap & & \cap & \\
 \vdots & & \vdots & & \vdots & & \vdots & \\
 \text{dom}_1(I_{i_0}) & \subset & \text{dom}_2(I_{i_0}) & \subset \dots \subset & \text{dom}_{n_0}(I_{i_0}) & \subset & \text{dom}_{n_0+1}(I_{i_0}) & \subset \dots \\
 \cap & & \cap & & \cap & & \cap & \\
 \text{dom}_1(I_{i_0+1}) & \subset & \text{dom}_2(I_{i_0+1}) & \subset \dots \subset & \text{dom}_{n_0}(I_{i_0+1}) & \subset & \text{dom}_{n_0+1}(I_{i_0+1}) & \subset \dots \\
 \cap & & \cap & & \cap & & \cap & \\
 \vdots & & \vdots & & \vdots & & \vdots &
 \end{array}$$

Существование i_0 и n_0 означает, что столбцы таблицы, ранг которых превышает n_0 , стабилизируются, начиная с линии i_0 . Более того, стабилизируется всякий столбец с номером $n < n_0$. Поэтому, строки предыдущей таблицы, начиная с некоторого индекса q , совпадают: для всякого $i \geq q$ имеем $\text{dom}_n(I_i) = \text{dom}_n(I_q)$, и по лемме 64 это доказывает, что $I_i = I_q$. Последовательность идеалов в $A[X]$ стабилизируется. Следовательно, $A[X]$ нётерово.

■

Перейдем теперь к свойствам разложения.

Лемма 65 (Гаусса)

Пусть простым элементом p кольца A делит произведение многочленов P и Q над A . Тогда p делит P или Q .

Доказательство.

Пусть p не делит ни P , ни Q . Обозначим через a_i и b_j такие коэффициенты P и Q , соответственно, что i и j — наименьшие номера, для которых $\nmid a_i$ и $\nmid b_j$. Тогда $\nmid \sum_{k+l=i+j} a_k b_l$, так как p делит все слагаемые этой суммы, кроме первого. Противоречие. В действительности лемма Гаусса не утверждает ничего, кроме того, что «если $/()$ без делителей нуля, то это же утверждение верно и для $A[X]/(p)$ ».

■

Определение 66

Пусть A — факториальное кольцо. Назовем **содержанием** многочлена P с коэффициентами из A НОД его коэффициентов и обозначим его через $c(P)$. Многочлен P называется **примитивным**, если его коэффициенты взаимно просты, т.е. если его содержание равно 1. **Примитивная часть** P равна $P/c(P)$, это примитивный множитель.

Следствие 67

- (i) Произведение примитивных многочленов является примитивным.
- (ii) Содержание произведения двух многочленов ассоциировано с произведением содержаний этих многочленов: $c(PQ) \sim c(P)c(Q)$.

Доказательство.

Пункт (i) является непосредственным следствием леммы Гаусса. Что касается пункта (ii), то можно заметить, что $P = c(P)'$ и $Q = c(Q)Q'$, где $'$ и Q' — примитивные многочлены. Отсюда получаем, что $c(PQ) = c(P)c(Q)c(P'Q')$, которое эквивалентно $c(P)c(Q)$, согласно пункту (i).

■

Следствие 68

Пусть P — **примитивный** многочлен с коэффициентами в факториальном кольце A , Q — другой многочлен. Если P делит Q над полем частных кольца A , то он делит Q и в $A[X]$. В частности, если для $a \in A^*$ P делит aQ , то P делит Q .

Теорема 69 (Гаусса)

Пусть A — факториальное кольцо, K — его поле частных, S_a — система представителей неприводимых элементов из A (т.е. такая система, по которой можно единственным образом разложить любой элемент из A). Пусть S' — система представителей неприводимых многочленов в $K[X]$ с коэффициентами из A , являющихся примитивными. Тогда $S' \cup S_a$ — система представителей неприводимых элементов в $A[X]$. В частности, $A[X]$ факториально.

Доказательство.

Пусть $P \in A[X]$ — многочлен положительной степени (в противном случае P раскладывается по системе S_A). В $K[X]$, являющемся кольцом главных идеалов, а следовательно, факториальным, многочлен P можно разложить в произведение неприводимых:

$$P = a \prod_{Q \in S'} Q^{\alpha_Q}, \text{ где } a \in K^* \text{ и } \alpha_Q \in \mathbb{N},$$

a можно записать в виде $a = p/q$, где $p, q \in A$. Следовательно, $qP = p \prod Q^{\alpha_Q}$, что является равенством в $A[X]$. Согласно следствию 68 $\prod Q^{\alpha_Q}$ являющийся примитивным многочленом, делит qP , а потому делит P и $a \in A$ (в силу единственности разложения в $K[X]$). Но тогда получим разложение в $A[X]$!

Разложение в $K[X]$ единственно, так как $K[X]$ факториально. Следовательно, разложение в $A[X]$ единственно.

■

Следствие 70

- (i) Если A — факториальное кольцо, то это же верно и для $A[X_1, \dots, X_n]$ и результат, разумеется, верен, если A — поле.
 (ii) В частности, $\mathbb{Z}[X_1, \dots, X_n]$ — факториальное кольцо.

Примечание.

1. Теорема Гаусса позволяет охарактеризовать неприводимые элементы в $A[X_1, \dots, X_n]$. А именно:

- неприводимые константы в A ,
- неприводимые примитивные многочлены над полем дробей кольца A .

2. Всякое кольцо многочленов над факториальным кольцом факториально. Однако кольцо многочленов над КГИ не является, вообще говоря, КГИ. ($A[X]$ — кольцо главных идеалов тогда и только тогда, когда A — поле).

Идеал $I = (X + 2)\mathbb{Z}[X] + X\mathbb{Z}[X]$ в кольце $\mathbb{Z}[X]$ не является главным, хотя он и максимален, так как это ядро сюръективного морфизма χ из $\mathbb{Z}[X]$ на $\mathbb{Z}/2\mathbb{Z}$, который каждому многочлену ставит в соответствие класс четности его постоянного коэффициента. Итак, $\mathbb{Z}[X]/I$ тело, и I — максимальный.

Мы закончим эту, немного абстрактную, часть критерием неприводимости многочлена в факториальном кольце: критерий Эйзенштейна.

Предложение 71 (критерий Эйзенштейна)

Пусть A — факториальное кольцо и K — его поле дробей. Пусть $P = a_n X^n + \dots + a_1 X + a_0$ — многочлен с коэффициентами из A , такой, что для некоторого неприводимого $p \in A$ выполнено: $p|a_0, p|a_1, \dots, p|a_{n-1}$, но $p \nmid a_n$ и $p^2 \nmid a_0$. Тогда P неприводим в $K[X]$, а следовательно, и в $A[X]$.

Доказательство.

Предположим, что $P = (b_q X^q + \dots + b_0)(c_r X^r + \dots + c_0)$, $q > 0$ и $r > 0$. Так как $p|a_0$ и $p^2 \nmid a_0$, то можно предположить, что $p|b_0$ и $p^2 \nmid c_0$. Кроме того, из $p^2 \nmid a_n$, следует, что $p^2 \nmid b_q$. Итак, можно найти такое целое число $s \leq q$, что $p|b_i$ для всякого $i < s$ и $p^2 \nmid b_s$. Так как $s \leq q < n$, то p делит a_s , которое равно $b_s c_0 + b_{s-1} c_1 + \dots + b_k c_{s-k}$ с $k = \min(s, r)$. Так как p делит все b_i с $i < s$, то он делит первый член суммы $b_s c_0$. Так как он не делит b_s , то p делит c_0 , откуда получаем противоречие, ибо p^2 не делит a_0 .

■

Примеры.

1. Пусть p — простое число и $\Phi_p(X) = X^{p-1} + X^{p-2} + \dots + X + 1$. Тогда Φ_p неприводим в $\mathbb{Q}[X]$. Действительно, $\Phi_p(X+1) = (X+1)^p - 1 = \sum_{i>0} \binom{p}{i} X^i$ — многочлен, удовлетворяющий критерию Эйзенштейна. Φ_p — циклотомический многочлен уровня p .

2. Однако для всякого непростого $n > 1$ многочлен $P_n = X^{n-1} + X^{n-2} + \dots + X + 1$ не является неприводимым. Чтобы это увидеть, достаточно записать, полагая $n = ab$:

$$P_{ab} = \frac{X^{ab} - 1}{X - 1} = \frac{(X^a)^b - 1}{X^a - 1} \frac{X^a - 1}{X - 1}.$$

3. Рассмотрим теперь для простого числа p циклотомические многочлены $\Phi_{p^\alpha} = X^{(p-1)p^{\alpha-1}} + X^{(p-2)p^{\alpha-1}} + \dots + X^{p^{\alpha-1}} + 1 = \Phi_p(X^{p^{\alpha-1}})$. Эти многочлены неприводимы над \mathbb{Z} . Осуществим ту же самую замену переменных $X \rightarrow X + 1$, что и для Φ_p , и без труда докажем, что p — свободный член многочлена $\Phi_{p^\alpha}(X+1)$ и $X^{(p-1)p^{\alpha-1}}$ — его старший член. Затем рассмотрим $\mathbb{Z}/p\mathbb{Z}[X]$, где имеем формальное равенство $(X+Y)^p = X^p + Y^p$. Тогда

$$\Phi_{p^\alpha}(X+1) = \frac{(X+1)^{p^\alpha} - 1}{(X+1)^{p^{\alpha-1}}} = \frac{X^{p^\alpha}}{X^{p^{\alpha-1}}} = X^{(p-1)p^{\alpha-1}}.$$

Следовательно, в $\mathbb{Z}[X]$ имеем $\Phi_{p^\alpha}(X+1) = X^{(p-1)p^{\alpha-1}} + pQ$, где Q — многочлен с целыми коэффициентами, имеющий степень, строго меньшую $(p-1)p^{\alpha-1}$. Критерий Эйзенштейна тогда применяется напрямую (можно посмотреть упражнение 52, которое доказывает неприводимость всех циклотомических многочленов).

Прежде чем вычислять НОД двух многочленов, рассмотрим алгоритм, позволяющий осуществить *псевдоделение* в кольце без делителей нуля.

Напомним, что обычное евклидово деление многочленов возможно в $A[X]$, если старший коэффициент делителя обратим в A . Чтобы избежать этого предположения, умножим делимое на подходящую степень старшего коэффициента делителя и вместо обычного равенства $P = QW + R$ получим, например, тождество $\alpha P = QW + R$.

```
int pseudo_division(vector* A, vector* B, vector* Q, vector* R)
{
    int n = deg(A); int m = deg(B);
    create_vector(R, n + 1); create_vector(Q, n-m + 1);
    int b_m = B->data[0];
    int alpha = 1;
    int i = 0, k = 0;
    for (i = 0; i <= n; i++){
        R->data[i] = A->data[i];
    }
    for (i = 0; i <= n-m; i++){
        Q->data[i] = 0;
    }
    for (k = n-m; k >= 0; k--){
        for (i = 0; i <= k-1 ; i++){
            R->data[i] = b_m * R->data[i];
        }
        Q->data[k] = b_m * R->data[m+k];
        for (i = k; i <= m+k-1 ; i++){
            R->data[i] = b_m * R->data[i] - Q->data[k] * B->data[
                i-k];
        }
        alpha *= b_m;
    }
    return alpha;
}
```

Алгоритм 2.4: “Евклидово псевдоделение многочленов над кольцом без делителей нуля”

При применении алгоритма к многочленам с целыми коэффициентами

(как это и будет сделано далее) коэффициенты промежуточных результатов растут чрезмерно быстро (элемент α равен b_m^{n-m+1}), и при желании уменьшить этот рост можно заменить множитель b_m при умножении коэффициентов на $b_m/\text{НОД}(b_m, r_{m+k})$. Это ничего не изменит в инвариантах, но зато несколько ограничит рост коэффициентов.

Чтобы разработать алгоритм «а la Евклид» для вычисления НОД, надо убедиться, что метод, заключающийся в итерированном повторении псевдоделений, сохраняет НОД в некотором смысле, который предстоит определить.

Рассмотрим два примитивных многочлена A и B и сделаем псевдоделение A на B : $\alpha A = BQ + R$. Очевидно, что делитель A и B является делителем R . Обратно, если D делит B , то D примитивен, так как B является примитивным. Если к тому же D делит R , то он делит и α , а следовательно, делит A . Итак, псевдоделение A на B сохраняет НОД для примитивных многочленов. Для итерирования этой процедуры можно рассматривать впоследствии только примитивную часть R . Действительно, в предыдущих обозначениях $\text{НОД}(A, B)$ и $\text{НОД}(B, R)$ имеют одну и ту же примитивную часть. С другой стороны, содержание НОД двух многочленов есть НОД содержащий.

Алгоритм вычисления НОД двух многочленов с коэффициентами в (факториальном) кольце заключается, следовательно, в вычислении d , НОД содержаний двух многочленов. Затем, в выполнении псевдоделений примитивных частей исходных многочленов. Повторяют операцию, заменяя полученный остаток его примитивной частью, и т.д. Алгоритм останавливается, когда полученный остаток нулевой или постоянный. В этом случае НОД двух многочленов — примитивная часть последнего ненулевого остатка, умноженного на постоянную d , вычисленную в начале алгоритма.

Пример.

Сейчас мы вычислим НОД двух многочленов (пример заимствован у Кнута):

$$P = X^8 + X^6 - 3X^4 - 3X^3 + 8X^2 + 2X - 5 \text{ и } Q = 3X^6 + 5X^4 - 4X^2 - 9X + 21.$$

Эти два многочлена примитивны. Следовательно, никакие поправки после вычислений не должны вноситься в полученный результат. Сделаем первое псевдоделение:

$$27P = Q \cdot (9X^2 - 6) + (-15X^4 + 3X^2 - 9),$$

и сохраним только примитивную часть полученного остатка: $-5X^2 + X^2 - 3$. Затем повторим эти действия и получим следующую последовательность псевдоделений:

$$\begin{aligned}
&27(X^8 + X^6 - 3X^4 - 3X^3 + 8X^2 + 2X - 5) = \\
&= (3X^6 + 5X^4 - 4X^2 - 9X + 21)W_1 + (-15X^2 + 3X^2 - 9), \\
&125(3X^6 + 5X^4 - 4X^2 - 9 + 21) = (5X^4 - X^2 + 3)W_2 + \\
&+ (-585X^2 - 1125X + 2205), \\
&2197(5X^4 - X^2 + 3) = (13X^2 + 25X - 49)W_3 + (-233150X + 307500), \\
&21743569(13X^2 + 25X - 49) = (4663X - 6150)W_4 + 143193869.
\end{aligned}$$

Два исходных многочлена, следовательно, взаимно просты. Отметим, что после каждого псевдоделения сохраняется только примитивная часть остатка. Это приводит к большому количеству вычислений, которые могут существенно замедлить алгоритм. Посмотрим, что стало бы с последовательностью псевдоделений, если не осуществлять этого сокращения на каждом этапе.

$$\begin{aligned}
&27(X^8 + X^6 - 3X^4 - 3X^3 + 8X^2 + 2X - 5) = \\
&= (3X^6 + 5X^4 - 4X^2 - 9X + 21)W_1 + (-15X^2 + 3X^2 - 9), \\
&3375(3X^6 + 5X^4 - 4X^2 - 9X + 21) = \\
&= (-15X^2 + 3X^2 - 9)W_2 + (15795X^2 + 30375X - 59535), \\
&3940568584875(-15X^2 + 3X^2 - 9) = (15795X^2 + 30375X - 59535)W_3 + \\
&+ (1254542875143750X - 1654608338437500), \\
&1573877825573946701583164062500(15795X^2 + 30375X - 59535) = \\
&= (1254542875143750X - 1654608338437500)W_4 + \\
&+ 12593338795500743100931141992187500.
\end{aligned}$$

Это апокалиптично! Но нужно осознать, что вычисления с гигантскими множителями, появляющимися в левой части равенств, не являются необходимыми. В частности, никакие действительные умножения многочлена, который делится, не имеют места. Важны лишь полученные в псевдоделениях остатки.

В действительности, есть другое решение для сокращения полученных коэффициентов, которые не дают все-таки лучшие результаты, чем сокращения на НОД — без всякого вычисления НОД — но это изложение выходит за рамки, которые нами зафиксированы (см. , например, Кнут [99]).

9. Вместо заключения

Начав эту главу с очень простого введения (вычисление НОД двух целых чисел), мы изучили более сложные арифметические понятия, касающиеся колец, отличных от кольца целых чисел, и использовали несколько алгоритмов и методов вычисления. Вот (без всякого доказательства) краткий обзор нескольких приложений, некоторые из которых будут развернуты в следующих главах.

Арифметика кольца $\mathbb{Z}[\sqrt{3}]$ тесно связана с применением критерия простоты Лукаса — Лемера. Этот критерий, касающийся чисел Мерсенна, т.е. чисел вида $2^q - 1$, позволяет найти самые большие известные к настоящему времени простые числа. Например, в 1985 г. доказана простота числа $2^{216091} - 1$ (числа, имеющего 65050 десятичных цифр). Доказательство успешно сочетало арифметику кольца $\mathbb{Z}[\sqrt{3}]$, изобретательность Лукаса и Лемера и, конечно, быстроту самых мощных компьютеров, таких как CRAY.

Однако вычисления, составляющие критерий Лукаса — Лемера, значительно более простые, чем вычисления, применяемые ординарным методом, приводят к необходимости умножения больших чисел (например, 10000 десятичных цифр), делающих невозможным использование наивных умножений. Это удастся сделать использованием результатов о преобразовании Фурье (Кули, Тьюки, Поллард), китайской теоремы об остатках (для которой вычисление коэффициентов Безу играет основную роль), которая позволяет программировать умножения, значительно уменьшающие время вычислений. В этом случае алгебраические структуры, вводимые в игру, есть вычеты $\mathbb{Z}/p\mathbb{Z}$, богатые корнями из единицы. Можно, впрочем, вместо факторов \mathbb{Z} использовать факторы евклидова кольца $\mathbb{Z}[i]$: это объект современных исследований.

Опишем другое поразительное применение арифметики кольца главных идеалов (не евклидова) $\mathbb{Z}[\frac{1+\sqrt{-163}}{2}]$, кольца целых чисел $\mathbb{Q}(\sqrt{-163})$. Индийский математик Рамануджан нашел, в частности, ряды, быстро сходящиеся к $1/\pi$. Вот один из них:

$$\frac{1}{\pi} = 12 \times \sum_{n=0}^{\infty} (-1)^n \frac{(6n)!}{(n!)^3 (3n)!} \frac{13\,591\,409 + 545\,140\,134 \times n}{(640\,320^3)^{n+\frac{1}{2}}},$$

который сходится при точности до 15 десятичных знаков на слагаемое (основное), но дает тысячи десятичных знаков π . В действительности, число 640320^3 связано с арифметикой поля $\mathbb{Q}(\sqrt{-163})$ (механизм этой связи слишком сложен, чтобы быть воспроизведенным здесь). В некоем смысле $\mathbb{Q}(\sqrt{-163})$ есть то, «что можно сделать лучше»: 163 — самое большое натуральное число d , такое, что кольцо целых чисел в $\mathbb{Q}(\sqrt{-d})$ является КГИ. Невероятное применение абстрактной арифметики к конкретным числовым вычислениям.

Упомянем также теорию кодов, обнаруживающих (и исправляющих) ошибки, для которой конечные поля, неприводимые многочлены над этими полями, в особенности над $\mathbb{Z}/2\mathbb{Z}$, — основные инструменты.

Закончим замечанием исторического плана. Уравнение Ферма (1665) было глубокой мотивацией для изучения арифметики в циклотомическом кольце $\mathbb{Z}[\sqrt[n]{1}]$ (кольцо, порожденное корнями n -й степени из единицы в \mathbb{C}) и это может рассматриваться как исходная точка в развитии теории чисел. Куммер, Ламе, Дирихле, затем Дедекин, Кронеккер и многие дру-

гие математики XIX - XX веков изучали знаменитую «теорему» Ферма:

Теорема 72 (Великая "теорема" Ферма.)

Для $n \geq 3$ уравнение $x^n + y^n = z^n$ не имеет нетривиальных решений в целых числах (т.е. решений, отличных от таких, где одно из неизвестных x , y или z равно нулю).

Эта проблема, не решенная до настоящего времени ¹, была (и остается еще) особым мотором в теории чисел. Большой прогресс и эффективность разработанных методов, связанная с использованием мощных компьютеров, позволяют проверить справедливость теоремы Ферма до $n \leq 125000$.

Все сказанное, должно быть, убедило читателя, что изучение абстрактных понятий вовсе не расходится, а находится в согласии с изучением методов эффективных вычислений.

¹В настоящее время проблема Ферма решена [195]. Последнюю точку в длинной и драматичной истории поставил английский математик Эндрю Уайлс. Би Би Си посвятило этому событию телевизионную программу, показанную в США в конце 1997 г. На эту же тему имеется книга [194], излагающая историю открытия, из которой видно, что в подготовке приняли участие многие математики. Для интересующихся читателей можно также рекомендовать книгу [191] на русском языке, а также книгу [193], вопрос о переводе которой на русский язык обсуждался. — Прим. ред.

Although it is not well known, Kummer at one time believed he had found a complete proof of Fermat's theorem... Seeking the best critic for his proof, Kummer sent his manuscript to Dirichlet... After a few days, Dirichlet replied with the opinion that the proof was excellent and certainly correct, provided the numbers in α could not only be decomposed into indecomposable factors, as Kummer proved, but that this could be done in only one way. If however, the second hypothesis couldn't be satisfied, most of the theorem for the arithmetic of numbers in α would be unproven and the proof of Kummer's theorem would fall apart. Unfortunately, it appeared to him that the numbers in α didn't actually possess this property in general ¹.

Kurt Hensel,

Commemoration of the first centennial of Kummer's birth (1910 [153])

¹Хотя это и не очень хорошо известно, Куммер некоторое время был уверен, что нашел полное доказательство теоремы Ферма... Желая найти лучшего критика для своего доказательства, Куммер послал рукопись Дирихле... Вскоре Дирихле ответил, что, по его мнению, доказательство великолепно и без сомнения верно при условии, что числа в α могут быть не только разложены в произведение неразложимых сомножителей, как доказано Куммером, но что это может быть сделано единственным образом. Однако, если второе предположение не выполнено, то большая часть теоремы об арифметике чисел в α останется недоказанной и доказательство теоремы Куммера рухнет. К сожалению, ему кажется, что для чисел в α это свойство, вообще говоря, не выполнено.

Курт Гензель, Празднование столетия со дня рождения Куммера (1910) [153]

Упражнения

1. Десятичные цифры простых чисел

Теорема об арифметической прогрессии Дирихле утверждает, что для каждой пары (a, b) целых простых чисел существует бесконечно много простых чисел вида $an + b$.

Использовать этот результат, чтобы доказать, что какова бы ни была последовательность B десятичных цифр, существует бесконечное множество простых чисел, десятичная запись которых включает B .

Пример.

Выберем $B = 1991$, которое поместим в предпоследнюю позицию. Вот несколько простых чисел, содержащих 1991:

219917, 519917, 1319917, 1919917, 2319917,
2619917, 3219917, 3419917, 3519917, 4719917, ...

2. Вычисление НОД

Пусть a и b — два целых взаимно простых целых числа. Доказать, что

$$\text{НОД}(a^m - b^m, a^n - b^n) = a^{\text{НОД}(m, n)} - b^{\text{НОД}(m, n)}.$$

3. Алгоритм Евклида и непрерывные дроби

Обозначим через $[a_0, a_1, a_2, \dots, a_{n-2}, a_{n-1}, a_n]$ непрерывную дробь

$$x = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_{n-2} + \frac{1}{a_{n-1} + \frac{1}{a_n}}}}}$$

Так как $[0, a_1, \dots, a_n] < 1$ (исключая дробь $[0, 1]$ ($n = 1, a_1 = 1$)), то a_0 можно найти по числу $x : a_0 = [x]$. Другой способ найти значения a_i , содержится в упражнении.

а. Доказать, что всякое рациональное число может быть записано в этой форме с $a_i > 0$ для $i \neq 0$.

б. Доказать, что если $[c_0, \dots, c_m] = [d_0, \dots, d_n]$ и если $c_m > 1, d_n > 1$, то $n = m$ и $c_i = d_i$.

с. Всякая конечная непрерывная дробь, очевидно, представляет рациональное число. Доказать, что всякое рациональное число можно представить непрерывной дробью и притом двумя способами (значения, участвующие в разложении, положительны, кроме, возможно, первого). Каноническая форма представления в непрерывную дробь для рационального числа — самая короткая из этих форм (естественно, предполагается, что последнее частное отлично от 1).

д. Вычислить рациональные числа, разлагающиеся в непрерывную дробь $[a_0, a_1, \dots, a_n]$ для $n = 1, 2, 3, 4$. Вычислить разложения в непрерывную дробь чисел $F_4/F_3, F_5/F_4$, где F_i — числа Фибоначчи.

4. Многочлены континуанты

Определим последовательность формальных полиномов для бесконечного числа переменных следующим образом: $K_{-1} = 0, K_0 = 1$ и $K_n(X_1, \dots, X_n) = X_n K_{n-1}(X_1, \dots, X_{n-1}) + K_{n-2}(X_1, \dots, X_{n-2})$. Многочлены $K_n(X_1, \dots, X_n)$ называются **континуантами**. По определению, континуанта индекса n — это многочлен от n переменных X_1, \dots, X_n .

а. Вычислить первые члены последовательности континуант. Каковы соотношения между континуантами и числами Фибоначчи?

б. Доказать, что многочлен K_n есть сумма всех одночленов, начиная с произведения X_1, \dots, X_n , полученных вычеркиванием всевозможных мономов и непересекающихся пар соседних переменных в этом одночлене.

с. Вывести из этого другое рекуррентное правило, определяющее последовательность K_n и следующее тождество между рациональными дробями:

$$\frac{K_n(X_1, \dots, X_n)}{K_{n-1}(X_2, \dots, X_n)} = X_1 + \frac{K_{n-2}(X_3, \dots, X_n)}{K_{n-1}(X_2, \dots, X_n)}$$

Какое заключение можно отсюда сделать?

5. Континуанты (продолжение)

а. Выразить функцию полиномов-континуант через произведения матриц

$$\begin{pmatrix} X_1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} X_2 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} X_n & 1 \\ 1 & 0 \end{pmatrix}. \quad (2.8)$$

б. Доказать, что

$$K_n(X_1, \dots, X_n)K_n(X_2, \dots, X_{n+1}) - K_{n+1}(X_1, \dots, X_{n+1})K_{n-1}(X_2, \dots, X_n) = (-1)^n.$$

Что можно отсюда вывести, если $K_n(a_1, \dots, a_n)$ и $K_{n-1}(a_2, \dots, a_n)$ являются целыми?

с. Рассматривая произведение матриц

$$\begin{pmatrix} X_{n+2} & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} K_{n+1}(X_1, \dots, X_{n+1}) & K_n(X_2, \dots, X_{n+1}) \\ K_n(X_1, \dots, X_n) & K_{n-1}(X_2, \dots, X_n) \end{pmatrix},$$

доказать соотношение:

$$K_{n-1}(X_2, \dots, X_n)K_{n+2}(X_1, \dots, X_{n+2}) - K_n(X_1, \dots, X_n)K_{n+1}(X_2, \dots, X_{n+2}) = (-1)^{n+1}X_{n+2}.$$

6. Разложение в непрерывную дробь

Если $x \in \mathbb{R}$, то можно определить последовательности (x_n) и (a_n) следующим образом: положим $x_0 = x$, $a_0 = [x_0]$ и для $n > 0$

$$x_{k+1} = \frac{1}{x_k - a_k} \text{ и } a_{k+1} = [x_{k+1}].$$

(в этой последовательности все элементы a_k положительны, кроме, при необходимости, первого). Эта последовательность конечна тогда и только тогда, когда x — рациональное число, и дает разложение x в непрерывную дробь (это понятие, очевидно, совпадает с определением упражнения 3, если x рационально).

Если $f = [a_0, a_1, a_2, \dots]$ — непрерывная дробь (конечная или бесконечная), то n -членный отрезок этой дроби, представляющий рациональное число $[a_0, \dots, a_n]$, называют подходящей дробью для непрерывной дроби f , или приближением f .

а. Выразить n -ую подходящую дробь для f через функцию многочленов континуант.

б. В предыдущих обозначениях доказать, что $x = [a_0, \dots, a_n, x_{n+1}]$. Вывести отсюда, что последовательность подходящих дробей для вещественного числа x сходится к x .

Начиная с этого момента будем отождествлять непрерывную дробь и ее значение.

с. Вывести из вопроса **(б)** упражнения 5 алгоритм вычисления n -й подходящей дроби для x , приводящий ее к неприводимому виду. Выразить алгоритм в явном виде, как последовательность, сходящуюся к непрерывной дроби.

д. Вычислить разложение в непрерывную дробь числа золотого сечения ϕ . Вычислить последовательные подходящие дроби для этого разложения.

7. Аппроксимация вещественных чисел с помощью непрерывных дробей

Пусть $[a_0, a_1, a_2, \dots]$ — разложение в непрерывную дробь иррационального числа x (в действительности все, что доказывается, остается верным и для рациональных чисел, но необходимо учитывать конечность разложения, что излишне усложняет доказательство). Мы не доказываем факт, что разложение в непрерывную дробь единственно. В дальнейшем, чтобы упростить обозначения, мы обозначим через p_n и q_n числитель и знаменатель n -й подходящей дроби (имеем $p_n = K_{n+1}(a_0, \dots, a_n)$ и $q_n = K_n(a_1, \dots, a_n)$).

а. Используя соотношения между континуантами из упражнения 5, доказать, что последовательность q_n строго возрастающая, и вычислить знаки выражений $(p_{n+1}/q_{n+1} - p_n/q_n)$ и $(p_{n+1}/q_{n+1} - p_{n-1}/q_{n-1})$. Вывести из этого, что последовательность упорядоченных четных пар строго возрастающая, а последовательность с нечетными номерами строго убывает и x заключен между этими последовательностями.

Доказать, что подходящие дроби для непрерывной дроби иррационального числа дают наилучшие аппроксимации этого числа в следующем смысле:

$$\forall q, 0 < q \leq q_n, \frac{p}{q} \neq \frac{p_n}{q_n} : |p_n - xq_n| < |p - xq|, \quad (2.9)$$

откуда, конечно, следует, что $|x - p_n/q_n| < |x - p/q|$. Можно предполагать, что p и q взаимно просты в этом случае.

б. Доказать с помощью равенства (11), фигурирующего в решении упражнения 6, следующее утверждение:

$$\frac{1}{q_n q_{n+2}} < |x - \frac{p_n}{q_n}| < \frac{1}{q_n q_{n+1}}.$$

Вывести, что $|p_n - xq_n| < |p_{n-1} - xq_{n-1}|$, так что $|x - p_n/q_n| < |x - p_{n-1}/q_{n-1}|$.

Для того чтобы доказать результат, можно предполагать (начиная с этого момента), что $q_{n-1} < q \leq q_n$.

с. Пусть $q = q_n$. Доказать утверждение (9), опираясь на формулу (12), фигурирующую в решении упражнения 6.

д. Пусть $q_{n-1} < q < q_n$. Используя матричное выражение (8) из упражнения 5, доказать, что если p линейно выражается через p_n и p_{n-1} и q является линейной функцией q_n и q_{n-1} , то коэффициенты для выражения q те же, что и для выражения p . Вывести отсюда предложение (9).

Замечание. Есть и другие результаты по аппроксимации вещественных чисел с помощью непрерывных дробей. В частности, известно, что для двух последовательных подходящих дробей в разложении x хотя бы одна удовлетворяет соотношению $|x - p/q| < 1/2q^2$. Известно также, что иррациональное число обладает бесконечным множеством аппроксимаций, удовлетворяющих соотношению $|x - p/q| < 1/q^2\sqrt{5}$ и $\sqrt{5}$ — наилучшая возможная константа. Кроме того, среди трех последовательных подходящих дробей для x по меньшей мере две удовлетворяют соотношению $|x - p/q| < 1/q^2\sqrt{5}$. Наконец, подходящие дроби для числа x — единственные хорошие аппроксимации x в том смысле, что если $|x - p/q| < 1/2q^2$, то p/q — подходящая дробь для x . Детали можно найти также у Левека [119], Харди и Райта [80], Кнута [99] и [103].

Примеры.

Вот разложения в непрерывную дробь и подходящие дроби для некоторых известных чисел.

е. $\sqrt{2} = [1, 2, 2, 2, 2, \dots] = 1, 414213\dots$; первые подходящие дроби $3/2 = 1, 5, 7/5 = 1, 4, 17/12 \approx 1, 4166\dots, 41/29 \approx 1, 4137$ и т.д. Как и следовало ожидать, таким способом не получим быстро сходящуюся последовательность к $\sqrt{2}$, по крайней мере, по-началу.

ф. $\sqrt{3} = [1, 1, 2, 1, 2, 1, 2, \dots] = 1, 732050$. Первые подходящие дроби $2, 5/3, 7/4$ и т.д.

Заметим, что для последних двух случаев разложение в непрерывную дробь оказалось периодическим. В действительности это общее свойство: разложение в непрерывную дробь квадратичной иррациональности периодично.

г. $\pi = [3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, \dots] = 3, 141592653\dots$ Это разложение уже не периодическое, так как сформулированное выше свойство на самом деле — необходимое и достаточное условие. Если вычислить последовательные подходящие дроби, то найдем хорошо известные аппроксимации: вторая подходящая дробь $22/7 \approx 3, 142$, четвертая — $355/113 \approx 3, 1415929$.

h. Наконец,

$$= [2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, \dots] = 2, 718281828459045\dots$$

Это разложение, кажется, обладает некоторой периодичностью, и действительно доказано, что разложение e есть $[\dots, 2\pi, 1, 1, 2\pi + 2, 1, 1, \dots]$. Однако никакой результат того же типа не доказан для π .

8. Факториал и простые числа

a. Доказать, что если $n > 4$ не является простым, то $(n-1)! \equiv 0 \pmod{n}$.

b. Доказать, что для любого числа n существует n последовательных натуральных чисел, ни одно из которых не является простым. Указание: рассмотреть число $(n+1)! + 2$.

c. Доказать, что для любого натурального n в интервале $[n, n! + 1]$ существует простое число.

d. Предыдущие задачи наводят на мысль рассмотреть последовательность (e_n) , определенную следующим образом: $e_1 = 2$ и $e_n = e_1 e_2 \dots e_{n-1} + 1$. Все ли эти числа просты? Они взаимно просты?

9. Каноническое разложение $n!$

Доказать следующую формулу, называемую **разложением Лежандра**:

$$n! = \prod_{p \text{ простое}} p^{\sum_{i=1}^{\infty} [n/p^i]}.$$

(Несмотря на видимость противного и в сумме и в произведении, входящих в эту формулу, присутствует лишь конечное число членов.)

10. Уравнение Ферма для $n = 2$ и $n = 4$

Найдем все решения в целых числах x, y, z уравнения $x^2 + y^2 = z^2$.

a. Доказать, что x и y не могут быть оба нечетными. Почему можно предполагать, что x и y взаимно просты?

b. Доказать, что в $\mathbb{Z}[i]$ элемент $x + iy$ делится на $1 + i$ тогда и только тогда, когда $x \equiv y \pmod{2}$. Вывести, используя факториальность

кольца $\mathbb{Z}[i]$, что если x и y — решения уравнения Ферма, то существуют два целых взаимно простых числа u и v такие, что

$$(x, y) \text{ или } (y, x) = (u^2 - v^2, 2uv), \quad z = u^2 + v^2,$$

и обратно, каждая такая тройка дает решение уравнения. Дайте примеры, а затем найти форму решений исходного уравнения.

с. Показать, как можно просто перейти в $\mathbb{Z}[i]$.

д. Пусть два целых числа $x, y > 0$. Показать, что $x^4 + y^4$ не является квадратом. В частности, уравнение $x^4 + y^4 = z^4$ не имеет нетривиальных решений в целых числах.

11. Простое диофантово уравнение

Диофантовы уравнения — это уравнения в целых числах.

а. Найти решение уравнения $x^2 - 3y^2 + 1 = 0$ в $\mathbb{Z}[i]$. Доказать, что оно не имеет целочисленных решений.

б. Найти все нетривиальные решения уравнения $x^2 - 5y^2 + z^2 = 0$.

с. Обобщить: при каких условиях уравнение $x^2 - py^2 + z^2 = 0$ имеет решение в целых числах?

12. Кольцо $\mathbb{Z}[\theta]/(a + \theta b)$, θ — квадратичная иррациональность, $\text{НОД}(a, b) = 1$

Пусть a и b — два целых взаимно простых числа.

а. Доказать, что отображение $\varphi : \mathbb{Z} \rightarrow \mathbb{Z}[i]/(a + ib)$, определяемое по правилу $\varphi(m) = [m]_{a+ib}$, сюръективно, выразив в явном виде элемент t , для которого $\varphi(t) = [i]_{a+ib}$.

б. Как связаны равенство $\text{Ker } \varphi = (a^2 + b^2)\mathbb{Z}$ и сравнение $t^2 \equiv -1 \pmod{a^2 + b^2}$, $a + tb \equiv 0 \pmod{a^2 + b^2}$? Справедливы ли они? Можно ли априори найти элемент в $\mathbb{Z}/(a^2 + b^2)$, квадрат которого равен -1 . Как? Сравнить с уже полученным выражением.

с. Доказать, что φ индуцирует изоморфизм кольца $\mathbb{Z}/(a^2 + b^2)$ на $\mathbb{Z}[i]/(a + ib)$. Доказать, что если $a + ib$ делит целое число m (в $\mathbb{Z}[i]$), то $a^2 + b^2$ делит m (в \mathbb{Z}). Найдя в явном виде обратный изоморфизм, представьте линейную форму $\mu : \mathbb{Z}^2 \rightarrow \mathbb{Z}$, удовлетворяющую соотношению:

$$a + ib|x + iy \iff a^2 + b^2|\mu(x, y). \quad (2.10)$$

Примените соотношение (10) к следующим примерам: $a + ib = 1 + i$, $a + ib = 9 + 2i$.

d. Для произвольных a, b (не обязательно взаимно простых) доказать, что $\mathbb{Z}[i]/(a + ib)$ есть кольцо, состоящее из $a^2 + b^2$ элементов.

e. Пусть θ — квадратичное целое, т.е. элемент в $\theta \in \mathbb{C} \setminus \mathbb{Z}$, удовлетворяющий уравнению $X^2 - SX + P = 0$. Обобщить предыдущее на кольцо $\mathbb{Z}[\theta]/(a + \theta b)$. В частности, число элементов кольца — абсолютное значение нормы $(a + \theta b)$.

13. НОД в $\mathbb{Z}[i]$ и суммы двух квадратов

a. Пусть $p \in \mathbb{N}$ — простое число, для которого $p \equiv 1 \pmod{4}$, $x \in \mathbb{Z}$ такой, что $x^2 \equiv -1 \pmod{p}$ (способ нахождения такого x предложен в с). Доказать, что если $u + iv = \text{НОД}(p, x + i)$, то $p = u^2 + v^2$. Зная корень квадратный x из -1 по модулю p , найти эффективный метод представления p в виде суммы двух квадратов. Записать в виде суммы двух квадратов следующие простые числа: 1301, 1000037 и 2000004973, зная, что:

$$51^2 \equiv -1 \pmod{1301}, 320900^2 \equiv -1 \pmod{1000037}, 106540073^2 \equiv -1 \pmod{2000004973}.$$

b. Обобщить результат пункта **a**, рассматривая целое число n , делитель $x^2 + y^2$, где x и y взаимно просты. Если $u + iv = \text{НОД}(n, x + iy)$, то показать, что $n = u^2 + v^2$ (можно использовать результат упражнения 12: если $a + ib$ с $a \wedge b = 1$ делит целое число m , то $N(a + ib) = (a + ib)(a - ib)$ делит m).

c. Пусть $p \equiv 1 \pmod{4}$. Запишем $p - 1 = 2^k q$, где q нечетно и $k \geq 2$. С $y \in [1, p - 1]$ ассоциируем последовательность $(y_i)_{0 \leq i \leq k}$ элементов из $U(\mathbb{Z}_p)$ следующим образом: $y_0 = y^q \pmod{p}$, $y_{i+1} = y_i^2 \pmod{p} = y^{2^i q} \pmod{p}$. Пусть $H \subset U(\mathbb{Z}_p)$ — подмножество тех $y \in U(\mathbb{Z}_p)$, что $y_1 = y^{2^q} \pmod{p} \neq 1$. Для $y \in H$ доказать, что один из y_i есть корень квадратный из -1 . Сколько элементов в H ? Вывести отсюда вероятностный алгоритм вычисления корня квадратного из -1 по модулю p .

14. Делимость сумм $x^2 + 2y^2$

a. Доказать, что кольцо $\mathbb{Z}[\sqrt{-2}]$ (это подкольцо порождено 1 и $\sqrt{-2} = i\sqrt{2}$) евклидово с нормой N , определяемой через $N(x + y\sqrt{-2}) = x^2 + 2y^2$ при $x, y \in \mathbb{Z}$. Указание: порассуждайте в поле $\mathbb{Q}(\sqrt{-2})$ ¹, поле частных $\mathbb{Q}[\sqrt{-2}]$.

b. Доказать результат относительно сумм $x^2 + 2y^2$, где $x, y \in \mathbb{Z}$ взаимно просты.

¹Вообще, если A — кольцо, то $A[x_1, x_2, \dots]$ обозначает над-кольцо (расширение кольца), порожденное x_1, x_2, \dots ; то же обозначение применяется, если A — поле. Если K — поле, то обозначение $K(x_1, x_2, \dots)$ определяет расширение K с помощью элементов (над-поле) поля x_1, x_2, \dots . Здесь, так как $\mathbb{Q}[\sqrt{-2}]$ уже является полем ($\sqrt{-2}$ алгебраический для \mathbb{Q}), имеем совпадение $\mathbb{Q}(\sqrt{-2})$ и $\mathbb{Q}[\sqrt{-2}]$.

с. Пусть p — простое целое число, отличное от 2. Показать, что p может быть записано в виде $x^2 + 2y^2$ тогда и только тогда, когда -2 есть квадрат по модулю p . Заметим, что последнее эквивалентно $p = 1$ или $3 \pmod{8}$.

15. Делимость целых чисел вида $x^2 - 2y^2$

а. Проверить, что кольцо $\mathbb{Z}[\sqrt{-2}]$ евклидово для абсолютного значения нормы $|N|$, определенной соотношением

$$N(x + y\sqrt{-2}) = x^2 - 2y^2, \quad x, y \in \mathbb{Z}.$$

б. Обоснуйте результат, касающийся сумм $x^2 - 2y^2$ и нечетных простых чисел p , выражающихся в этом виде.

16. Кольца $\mathbb{Z}[\sqrt{-3}]$ и $\mathbb{Z}[\sqrt{10}]$ нефакториальны

а. Доказать нефакториальность кольца $\mathbb{Z}[\sqrt{-3}]$, предъявив элемент, разлагающийся двумя различными способами в произведение неприводимых. Указание: найти два целых простых числа, не являющиеся нормами (в данном кольце) и рассмотреть их произведение.

б. Тот же вопрос для кольца $\mathbb{Z}[\sqrt{10}]$.

17. Целозамкнутые кольца

а. Доказать, что рациональный корень унитарного многочлена с целыми коэффициентами есть целое число. Если этот многочлен не имеет целых корней, то он не имеет и рациональных. Вывести отсюда, что если $a \in \mathbb{N}$ не является корнем n -й степени в \mathbb{N} , то $\sqrt[n]{a}$ — иррациональное число.

Пусть A — кольцо целых чисел, K — его поле частных. Говорят, что K целозамкнуто, если элемент из K , являющийся корнем унитарного многочлена с коэффициентами из A , есть элемент из A . Отсюда, кольцо \mathbb{Z} целозамкнуто. Для каких колец указанное свойство (доказанное для \mathbb{Z}) может быть обобщено?

б. Пусть $d \in \mathbb{Z}$ не является квадратичным и удовлетворяет соотношению $d \equiv 1 \pmod{4}$. Доказать, что кольцо $\mathbb{Z}[\sqrt{d}]$ не целозамкнуто (и потому не факториально). Доказать также, что 2 — неприводимый элемент, но не простой.

с. Найти в $\mathbb{Z}[\sqrt{2i}]$ неприводимый элемент, не являющийся простым; элемент, разлагающийся двумя различными способами в произведение неприводимых, и многочлен, неприводимый над $\mathbb{Z}[\sqrt{2i}]$, который не остается неприводимым над полем частных.

18. Целые элементы; целые квадратичности

а. Пусть K — подполе поля, имеющее размерность 2 над \mathbb{Q} . Доказать существование и единственность такого $d \in \mathbb{Z}$, что K — поле, порожденное над \mathbb{Q} элементами 1 и \sqrt{d} :

$$K = \mathbb{Q}(\sqrt{d}) = \mathbb{Q} \oplus \mathbb{Q}(\sqrt{d}).$$

Проверить существование и единственность автоморфизма σ поля K , оставляющего на месте \mathbb{Q} , такого, что $\sigma(\sqrt{d}) = -\sqrt{d}$. Доказать, что автоморфизм σ инволютивный и множество его неподвижных точек совпадает с \mathbb{Q} .

б. Пусть два кольца A и B таковы, что $A \subset B$. Будем говорить, что элемент из B **целый над** A степени $\leq n$, если он является корнем унитарного многочлена степени $\leq n$ с коэффициентами из A . Элементы из \mathbb{C} , целые над \mathbb{Z} , называются целыми алгебраическими. Квадратичные целые являются целыми алгебраическими степени. Каковы целые алгебраические степени 1? При каких условиях рациональное число является целым алгебраическим?

с. Пусть d не имеет нетривиальных множителей, являющихся полными квадратами. Доказать, что $\frac{1+\sqrt{d}}{2}$ является целым квадратичным тогда и только тогда, когда $d \equiv 1 \pmod{4}$.

19. Целые числа в $\mathbb{Q}(\sqrt{d})$

Обозначим через d элемент из \mathbb{Z} , не имеющий нетривиальных множителей, являющихся полными квадратами, и σ — инволютивный автоморфизм $\mathbb{Q}(\sqrt{d})$, переводящий \sqrt{d} в $-\sqrt{d}$.

а. Пусть $z \in K = \mathbb{Q}(\sqrt{d})$. Показать, что z — целая квадратичность тогда и только тогда, когда $z + \sigma(z) \in \mathbb{Z}$ и $z\sigma(z) \in \mathbb{Z}$.

б. Доказать, что множество целых квадратичностей в K является подкольцом K . Указание: если z и z' — целые квадратичности, показать, что $x = z\sigma(z') + z'\sigma(z)$ и $y = zz' + \sigma(zz')$ — элементы из \mathbb{Z} (рассмотреть $x + y$ и xy).

с. Используя свойства кольца $\mathbb{Z}[X]$ из раздела 8, показать, что целые алгебраические числа из $K = \mathbb{Q}(\sqrt{d})$ являются целыми квадратичностями. Множество таких элементов является кольцом целых алгебраических чисел из K .

d. Доказать, что кольцо A целых поля $\mathbb{Q}(\sqrt{d})$ совпадает с:

$$\mathbb{Z}(\sqrt{d}), \text{ если } d \equiv 2 \text{ или } 3 \pmod{4}, \text{ или } \mathbb{Z}\left[\frac{1+\sqrt{d}}{2}\right], \text{ если } d \equiv 1 \pmod{4},$$

соответственно, $\{u + v\sqrt{d} | u, v \in \mathbb{Z}\}$ и $\{(u + v\sqrt{d})/2 | u, v \in \mathbb{Z}, u \equiv v \pmod{2}\}$.

20. Евклидовость квадратичных колец

Норма N для квадратичного расширения $\mathbb{Q}(\sqrt{d})$ является мультипликативной функцией со значениями в \mathbb{Q} и определяется следующим образом: $N(z) = z\sigma(z)$ для $z \in \mathbb{Q}(\sqrt{d})$. Она принимает целые значения на кольце A целых алгебраических чисел в $\mathbb{Q}(\sqrt{d})$ (см. предыдущее упражнение).

a. Доказать, что A является евклидовым для нормы $|N|$ тогда и только тогда, когда для всякого $z \in \mathbb{Q}(\sqrt{d})$ существует такое $q \in A$, что $|N(z - q)| < 1$.

b. Доказать, что A евклидово для нормы $|N|$, когда d имеет одно из следующих значений: $-11, -7, -3, -2, -1, 2, 3, 5$ и 13 .

Замечание.

1. Для $d < 0$ (поля мнимых квадратичностей) нетривиальный результат утверждает, что соответствующее кольцо A является КГИ в точности для следующих значений d :

$$d = -163, -67, -43, -19, -11, -7, -3, -2, -1.$$

Наиболее легкая часть доказательства состоит в проверке того, что эти 9 колец — действительно КГИ и что для 4 простых чисел (4 первых) не евклидовы. Обратное намного труднее. Когда-то, впрочем, считали, что могло существовать десятое d , такое, что $-5 \cdot 10^9 < d < -163$, для которого кольцо целых чисел в $\mathbb{Q}(\sqrt{d})$ является КГИ. Но из этого ничего не получилось, как доказал Старк в 1967 [166]. Об остальных деталях можно проконсультироваться у Харди и Райта [80], Эстерле [138], Дьедонне [62] или Пуату [145].

2. Для $d > 0$ (вещественное квадратичное расширение) ситуация более сложная. Существуют, как в мнимом случае, значения d , для которых соответствующее кольцо есть КГИ, и другие значения, для которых оно таковым не является. Таблицы, предоставляющие этот вид информации (для d , относящегося или нет к определенному типу), существуют в литературе, например, [28]. Там также утверждается, что самое маленькое $d > 0$, для которого соответствующее кольцо не является КГИ, есть $d = 10$. В настоящий момент неизвестно, существует ли бесконечное множество таких положительных d , что кольцо целых чисел в $Q[\sqrt{d}]$ является КГИ.

Евклидовость квадратичных вещественных колец ставит также другие открытые проблемы. Известно, например, что кольца, являющиеся евклидовыми для *абсолютной нормы* $|N|$, исчерпываются кольцами, соответствующими шестнадцати значениям d (сравнить [80] или [159]):

$$d = 2, \quad 3, \quad 5, \quad 6, \quad 7, \quad 11, \quad 13, \quad 17, \quad 19, \\ 21, \quad 29, \quad 33, \quad 37, \quad 41, \quad 57, \quad 73.$$

В статье Уайнбергера [175] доказано, что из **ослабленной гипотезы Римана** следует, что всякое вещественное квадратичное кольцо евклидово коль скоро оно есть **КГИ**. Однако неизвестно, можно ли в явном виде указать вещественное евклидово квадратичное кольцо, не фигурирующее в данном месте. Например, квадратичное кольцо, соответствующее $d = 14$, является КГИ и, согласно сделанному выше примечанию, возможно, евклидово. Однако этот тезис на настоящий момент мы не можем ни доказать, ни опровергнуть. В самом деле, результат Уайнбергера утверждает, что при условии справедливости ослабленной гипотезы Римана, всякое кольцо целых чисел (неважно какого поля), за исключением 4 мнимых квадратичных колец, соответствующих $d = -163, -67, -43, -19$, является евклидовым, **как только оно является КГИ**.

В связи с этим некоторые математики ввели понятия квазиевклидовых колец (см. статью Кука [54] или диссертацию Буажо [31]). Эти авторы выражают в явном виде квазиевклидовы деления в некоторых вещественных квадратичных кольцах, достаточные для вычисления НОД. Вот некоторые значения d , для которых соответствующее кольцо является квазиевклидовым (значения d , для которых соответствующее кольцо является евклидовым по отношению к абсолютному значению нормы, были исключены);

$$d = 14, 22, 23, 31, 38, 43, 46, 47, 53, 61, 69, \\ 77, 89, 93, 97, 113, 129, 133, 137, 181, 253.$$

Для этих значений d , кажется, ничего не известно о евклидовости соответствующих колец.

Кроме того, Васерштейн в [171] доказал, что **вещественное** квадратичное кольцо будет квазиевклидовым, если оно есть **КГИ**. Следовательно, ситуация вполне предсказуема и мотивирует введение класса квазиевклидовых колец. В самом деле, последний результат наиболее общий. Он утверждает, что всякое **КГИ** целых чисел неважно какого числового поля, исключая 4 упомянутых выше квадратичных кольца (с $d = -163, -67, -43, -19$), является квазиевклидовым.

21. Нахождение колец мнимых квадратичностей

Пусть A — кольцо целых чисел мнимого квадратичного расширения $Q(\sqrt{d})$ поля рациональных чисел, где $d < 0$ — целое число и $|d|$ не имеет нетривиальных квадратных множителей.

a. Определить группу единиц кольца A .

b. Пусть B — евклидово кольцо, не являющееся полем, и $\varphi : B \rightarrow \mathbb{N}$ — евклидов алгоритм. Предположим, что все единицы в B исчерпываются ± 1 . Рассматривая φ -минимальный элемент, доказать наличие $b \in B$ такого, что $B/(b)$ поле \mathbb{Z}_2 или \mathbb{Z}_3 .

c. Используя упражнение 20 и то, что число элементов кольца $A/(z)$ есть норма в \mathbb{Z} (см. упражнение 12), доказать, что A — евклидово тогда и только тогда, когда $d \in \{-11, -7, -3, -2, -1\}$.

22. Вычисление НОД с помощью двоичных операций

На машине, располагающей только следующими арифметическими операциями над целыми числами — сложение, вычитание, деление и умножение на степень двойки, а также критерием четности, необходимо написать алгоритм вычисления НОД двух целых чисел.

Если $a = 2^\alpha a'$ и $b = 2^\beta b'$, то какое соотношение связывает $\text{НОД}(a, b)$ и $\text{НОД}(a', b')$? Отсюда просто вывести алгоритм вычисления НОД на рассматриваемой машине. Примените этот алгоритм к 1610 и 1000. Какой вывод можно сделать с точки зрения эффективности?

Можно *объединить* этот алгоритм с алгоритмом Евклида, используя деление четного остатка: $a = bq + r$ с $|r| < |b|$ и четным r . Записать и поэкспериментировать с программой вычисления НОД, основанной на этом делении.

23. Другое соотношение для последовательности Фибоначчи

Пусть F_n обозначает n -ое число Фибоначчи. Доказать, что $F_{n+m} = F_m F_{n+1} + F_{m-1} F_n$ (доказано в упражнении 13 главы I). Вывести отсюда, что $\text{НОД}(F_n, F_m) = F_{\text{НОД}(n, m)}$ (результат, принадлежащий Лукасу).

24. Вычисление НОД n -ки целых чисел

Начиная с соотношения, полученного в начале раздела 6.3: $\text{НОД}(0, 0 \dots 0, a, 0 \dots 0, 0) = a$ и, если $u_1 \neq 0$, то $\text{НОД}(u_1, \dots, u_n) = \text{НОД}(u_1, u_2 \bmod u_1, \dots, u_n \bmod u_1)$ и, используя операцию *Rotate_Left*, определенную соотношением $\text{Rotate_Left}((u_1, u_2, \dots, u_n)) = (u_2, \dots, u_n, u_1)$, можно написать алгоритм вычисления НОД n -ок целых чисел глобальным образом:

```
while(1){
    Number_of_zeros = 0;
    for(int i = 0; i < n; i++){
        Rotate_left(u);
        if(u[1] != 0) break;
        Number_of_zeros = Number_of_zeros + 1;
    }
    if(Number_of_zeros >= n - 1) break;
    for(int i = 2; i < n; i++){
        u[i] = u[i] % u[1];
    }
}
```

a. Доказать, что если число нулей в массиве u меньше или равно $n - 2$, то в каждой итерации, кроме, быть может, первой, одна из компонент u строго уменьшается. Вывести отсюда, что этот алгоритм эффективным образом вычисляет НОД чисел u_i .

b. Перетасовка массива — не очень эффективная операция в программе. Построить алгоритм вычисления НОД n целых чисел, аналогичный предыдущему, но использующий подвижный указатель в массиве вместо его сдвига.

c. Можно легко построить еще более эффективный алгоритм, используя два указателя в массиве вместо одного, и рассмотреть только часть массива, содержащегося между этими двумя индексами.

25. Странный алгоритм Евклида

Доказать, что отображение $\varphi : \mathbb{Z} \rightarrow \mathbb{N}$, определенное через $\varphi(n) = |n|$ при $n \neq 5$ и $\varphi(5) = 100$, является алгоритмом Евклида на \mathbb{Z} .

26. Возрастающие алгоритмы Евклида

Пусть φ — алгоритм Евклида на кольце A . Определим $\bar{\varphi}$ следующим образом: $\bar{\varphi}(0) = \varphi(0)$ и, если $a \neq 0$, то $\bar{\varphi}(a) = \min\{\varphi(b) \mid b \in Aa \setminus \{0\}\}$.

- а. Доказать, что $\bar{\varphi}$ — алгоритм Евклида, мажорируемый φ .
- б. Доказать, что для ненулевых a и b $\bar{\varphi}$ удовлетворяет соотношению: $a \mid b \Rightarrow \bar{\varphi}(a) \leq \bar{\varphi}(b)$.
- с. При тех же предположениях доказать, что если $a \mid b$ и $\bar{\varphi}(a) = \bar{\varphi}(b)$, то $Aa = Ab$.

27. Самый малый алгоритм Евклида на \mathbb{Z}

Доказать, что отображение φ , ставящее каждому целому числу $b \in \mathbb{Z}$ в соответствие количество двоичных цифр в двоичной записи $|b|$, является алгоритмом Евклида на \mathbb{Z} и что этот алгоритм самый малый алгоритм Евклида на \mathbb{Z} (эта ограниченность, удобства ради, рассматривается на алгоритмах со значениями в \mathbb{N}).

28. Реализация кольца целых чисел Гаусса

- а. Записать алгоритм деления Евклида в кольце $\mathbb{Z}[i]$.
- б. Написать стандартный пакет Ада-программ, реализующий кольцо $\mathbb{Z}[i]$ (включая его евклидову структуру).

29. Реализация алгоритма Безу

- а. Написать на языке Ада программу, реализующую вычисление коэффициентов Безу в кольце целых чисел. Предусмотрите процедуру, позволяющую проследить эволюцию вычислений.
- б. Написать стандартный общий пакет на языке Ада в квазиевклидовом кольце. Должна быть предусмотрена процедура, отслеживающая значения промежуточных параметров.
- с. Рассмотреть этот пакет для случая, когда текущим значением кольца является кольцо целых чисел Гаусса.

30. Конечные поля, порядок которых есть квадрат простого числа

Пусть p — простое число, $p \equiv 3 \pmod{4}$. Доказать, что факторкольцо $\mathbb{Z}[i]/(p)$ — конечное поле из p^2 элементов. Проверить, что это поле может быть получено из поля $\mathbb{Z}/p\mathbb{Z}$ присоединением корня квадратного из -1 , аналогично построению поля \mathbb{C} комплексных чисел, исходя из поля \mathbb{R} вещественных чисел.

31. Несколько конечных полей порядка, являющегося степенью двойки

Выпишите в явном виде поля порядков 2, 4, 8, 16, 32, 64 и 128.

32. Получение всех коэффициентов Безу

Пусть a, b — ненулевые целые числа, d — их НОД, u_0, v_0 — коэффициенты Безу: $au_0 + bv_0 = d$.

Доказать, что остальные коэффициенты Безу (u, v) имеют вид:

$$u = u_0 - k\frac{b}{d}, \quad v = v_0 + k\frac{a}{d}, \quad k \in \mathbb{Z}.$$

33. Единственность ограниченных коэффициентов Безу

Пусть a и b — два ненулевых целых числа, d — их НОД. Пусть (u, v) , (u', v') — коэффициенты Безу: $d = ua + vb = u'a + v'b$. Пусть к тому же элементы u и v удовлетворяют неравенствам: $|u| \leq |b/2d|$ и $|v| \leq |a/2d|$.

а. Предполагая, что (u', v') — пара, отличная от (u, v) , доказать, что среди неравенств

$$|\frac{b}{2d}| \leq |u'|, \quad \text{и} \quad |\frac{a}{2d}| \leq |v'|,$$

хотя бы одно строгое.

б. Вывести отсюда, что (u, v) — единственная и минимальная, в некотором (точном) смысле, пара.

34. Мажорирования коэффициентов Безу в $K[X]$

Пусть $A(X)$ и $B(X)$ — взаимно простые одночлены над полем K . Доказать, что обобщенный алгоритм Евклида находит коэффициенты Безу, удовлетворяющие соотношениям: $\deg(U) < \deg(B)$ и $\deg(V) < \deg(A)$.

35. Дерево Штерна — Броко

Цель этого упражнения заключается в изучении метода построения несократимых дробей, приводящего к перечислению рационального интервала $[0, 1]$. Отправляясь от двух рациональных чисел 0 и 1, записанных в несократимом виде как $\frac{0}{1}$ и $\frac{1}{1}$, и повторяя операцию, описываемую далее, получаем все числа отрезка $[0, 1]$. Пусть $\frac{p}{q}$ и $\frac{p'}{q'}$ — две

последовательные дроби, полученные этим приемом. Тогда следующая дробь $\frac{p+p'}{q+q'}$ определяет число между ними.

Вот что дают первые шаги этого процесса. Исходя из $[\frac{0}{1}, \frac{1}{1}]$, получаем медианную дробь $\frac{1}{2}$ что дает $[\frac{0}{1}, \frac{1}{2}, \frac{1}{1}]$. Вводим затем две медианные дроби $[\frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1}]$ и т.д. Этот процесс может быть представлен естественным образом в виде бинарного дерева, называемого деревом Штерна — Броко, что видно на рис. 1.

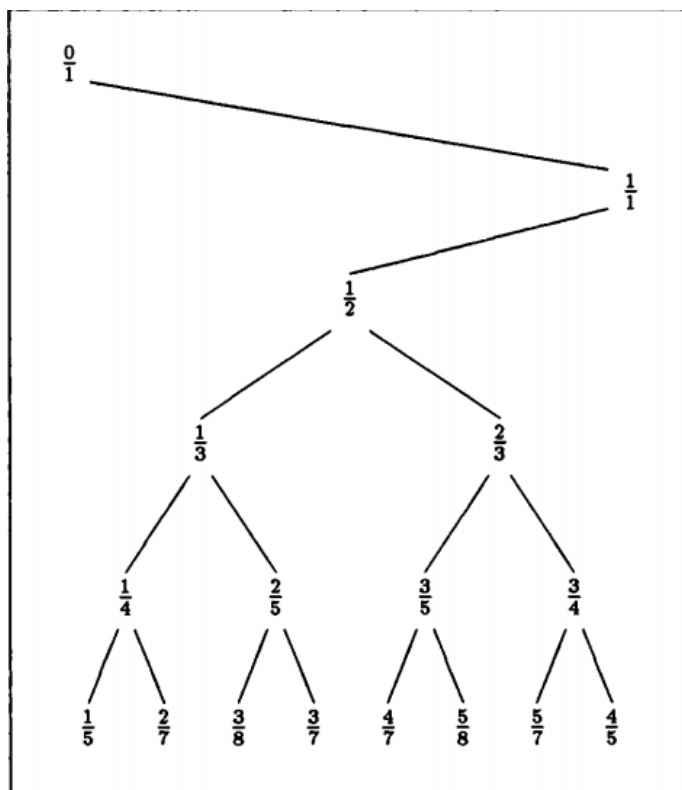


Рис 2.1: Дерево Штерна — Броко

В этом дереве каждый элемент, исключая первые два, получается как медиана его двух самых близких возрастающих: самый первый среди расположенных слева (самый большой минорант среди возрастающих) и самый левый среди расположенных справа (самый малый мажорант среди возрастающих).

Обозначения: Во всем упражнении a, a', a'' обозначают три положительных рациональных числа, представимых в виде $\frac{p}{q}$, $\frac{p'}{q'}$ и $\frac{p''}{q''}$ в несократимой форме, соответственно.

а. Если две дроби a и a' , удовлетворяют соотношению $p'q - pq' = 1$, то доказать, что их медианное число получается в несократимом виде. Вывести отсюда, что в конструкции Штерна — Броко все полученные дроби будут несократимы и что этот процесс приводит к упорядоченной последовательности дробей. Доказать, что к тому же каждое рациональное число из интервала $[0,1]$ достигается некоторой вершиной дерева. Указание: Доказать, что если находится строго между двумя последовательными дробями построения $\frac{p}{q}$ и $\frac{p'}{q'}$, то $c \geq q + q'$.

б. Ряд Фарея порядка N — это последовательность \mathcal{F}_N рациональных чисел из интервала $[0,1]$, представимых в виде несократимых дробей, знаменатели которых не превосходят N . Вывести из предыдущей задачи алгоритм, который представляет \mathcal{F}_N для возрастающих значений N (вычисление \mathcal{F}_N основывается на \mathcal{F}_{N-1}).

с. Если a и a' — два последовательных члена \mathcal{F}_N , то доказать, что $q + q' > N$.

Использовать свойство, доказанное в пункте **а**, что если a, a', a'' — три последовательных члена ряда Фарея (или дерева Штерна — Броко), то

$$\frac{p'}{q'} = \frac{p + q''}{q + q''}, \quad p'q - pq'' = \lfloor (q + N)/q' \rfloor \quad \text{и} \quad \begin{cases} p'' = \lfloor (q + N)/q' \rfloor p' - p, \\ q'' = \lfloor (q + N)/q' \rfloor q' - q. \end{cases}$$

Это последнее свойство позволяет дать алгоритм, который из ничего порождает все элементы \mathcal{F}_N .

д. Пусть p и q — два взаимно простых целых числа, причем $p < q$. Показать, что получение дроби $\frac{p}{q}$ в процессе Штерна — Броко дает не менее двух пар коэффициентов Безу для p и q и что одна из этих пар — самая малая пара коэффициентов Безу для p и q .

36. Вычисление коэффициентов Безу n -ки целых чисел

Предполагая, что известна функция Безу для двух целых аргументов, дающая пару коэффициентов Безу в качестве передаваемых параметров, написать и реализовать алгоритм вычисления коэффициентов Безу n -ки целых чисел.

37. Экспериментальная проверка теоремы Дирихле

Написать программу, иллюстрирующую теорему Дирихле: «вероятность взаимной простоты двух целых чисел равна $6/\pi^2$ ».

38. Произведение рядов и арифметическое произведение

Проверить равенство для двух рядов

$$\sum_{n \geq 1} \frac{f(n)}{n^\alpha} \times \sum_{n \geq 1} \frac{g(n)}{n^\alpha} = \sum_{n \geq 1} \frac{(f * g)(n)}{n^\alpha},$$

где $f * g$ — арифметическое произведение f и g . Что можно отсюда вывести, если f и g взаимно обратные функции (в арифметическом смысле)?

39. Сложность центрированного деления

Изучим в этом упражнении сложность алгоритма Евклида, порожденного центрированным делением, т.е. делением $a = bq + r$, где $|r| \leq |b|/2$. Для этого введем последовательность $(G_n)_{n \in \mathbb{N}}$, определяемую через $G_0 = 0, G_1 = 1, G_{n+2} = 2G_{n+1} + G_n$.

а. Вычислить первые члены этой последовательности. Проверить, что $G_n \leq G_{n+1}/2$. Полагая при $n \geq 1$ $a = G_n + G_{n-1}, b = G_n$, доказать, что $0 < b < a$ и что число делений алгоритма Евклида, вычисляющего НОД(a, b), равно n .

б. Пусть a и b — два целых числа, для которых $0 < b < a$. Предположим, что алгоритм Евклида, применяемый к (a, b) , требует n делений. Доказать, что $G_n \leq b$ и $G_n + G_{n-1} \leq a$. Вывести отсюда, что $n < \log_{1+\sqrt{2}}(b+1)2\sqrt{2}$.

Указание: если r_0, r_1, \dots, r_n и $r_{n+1} = 0$ обозначает последовательность остатков, то показать, что $|r_{i-1}| \geq 2|r_i| + |r_{i+1}|$ при $i = 2, 3, \dots, n$. Кроме того, можно доказать, что

$$G_n = \frac{(1 + \sqrt{2})^n - (1 - \sqrt{2})^n}{2\sqrt{2}}$$

с. Пусть p — число десятичных цифр в записи b . Доказать, что $n \leq \alpha p + \beta$, где α и β следующие: $\alpha + 1/\log_{10} 1 + \sqrt{2} \approx 2,612496$ и $\beta = \log_{10}(2\sqrt{2})/\log_{10}(1 + \sqrt{2}) \approx 1,17965$.

Вывести отсюда, что число центрированных делений, необходимых для вычисления НОД двух положительных целых чисел, не превосходит утроенного числа цифр наименьшего из этих чисел.

40. Порождения простых чисел с помощью многочленов

а. Доказать, что не существует непостоянного многочлена с целыми коэффициентами, который принимает только простые значения. Указание: вычислить $P(n + P(n))$.

Согласно постулату Бертрана, для любого целого $n > 1$ в интервале $]n, 2n[$ существует простое число.

б. Доказать, что если p_i означает i -ое простое число, то

$$p_{n+1} \leq p_1 + \dots + p_n \text{ для } n > 1.$$

с. Использовать постулат Бертрана для доказательства существования такого вещественного числа K , что n -ое простое число есть целая часть от $10^{n^2} K \bmod 10^n$.

41. Соотношение Безу для многочленов

Пусть P и Q — два постоянных многочлена с коэффициентами в факториальном кольце A с полем частных K . Доказать, что P и Q взаимно просты (в $K[X]$) тогда и только тогда, когда найдутся такие два многочлена U и V с коэффициентами в A , что $\deg U < \deg Q$, $\deg V < \deg P$, и элемент $\alpha \in A$, отличный от нуля, что $UP + VQ = \alpha$.

42. Сложность алгоритма Евклида в $K[X]$

Рассмотрим сложность алгоритма Евклида в $K[X]$, где K — нётерово поле. Алгоритм, в котором используется обычное деление: $A = BQ + R$ с $\deg(R) < \deg(B)$. Обозначим через φ отображение в $K[X] \times K[X]$ в \mathbb{N} , которое паре многочленов (A, B) ставит в соответствие число делений, необходимых для получения НОД (A, B) .

а. Доказать, что $\varphi(A, B) \leq 1 + \deg(B)$.

б. Доказать, что эта оценка наилучшая возможная: можно указать последовательность $(F_n(X))_{n \geq 0}$ для которой: $\deg(F_n) = n$ и $\varphi(F_n, F_{n-1}) = n(= 1 + \deg(F_{n-1}))$.

43. Оптимальность алгоритма для $K[X]$ (теорема Лазара)

Цель этого упражнения доказать, что среди всех алгоритмов «à la Евклид» вычисления НОД в $K[X]$ наилучшим является алгоритм, использующий обычное евклидово деление многочленов. Обозначим через $\varphi : K[X] \times K[X] \rightarrow \mathbb{N}$ минимальный квазиалгоритм (см. раздел 6.2) в $K[X]$ и через φ сложность алгоритма с использованием обычного деления. Докажем равенство $\mu = \varphi$.

а. Для A, A', B из $K[X]$ и константы k , отличной от 0, проверить следующие соотношения для μ и φ :

$$\mu(A, B) = 0 \Leftrightarrow B = 0,$$

$$\mu(kA, B) = \mu(A, kB) = \mu(A, B),$$

$$\mu(A, B) + \mu(A', B) \text{ если } A \equiv A' \pmod{B}.$$

b. Пусть A, B в $K[X]$. Доказать, используя индукцию по значениям μ , что $\mu(A, B) = \varphi(A, B)$. Указание: введите остатки R и R' от обычного и оптимального деления A на B . Затем рассмотреть 3 случая: $\deg(R) < \deg(B)$, $\deg(R) > \deg(B)$, $\deg(R) = \deg(B)$.

44. К теореме Лазара

Рассмотрим последовательность $C_0 = 0$, $C_1 = 1$ и $C_{2i} = C_{2i-1} + C_{2i-2}$, $C_{2i+1} = 2C_{2i} - C_{2i-1}$. Доказать, что алгоритм, использующий *центрированное* деление, примененный к (C_{n+1}, C_n) , требует n делений. Доказать также, что используя «одно на двоих» нецентрированное деление, приходим к алгоритму той же сложности (т.е. необходимо n делений, хотя используется $\lfloor \frac{n-1}{2} \rfloor$ нецентрированных делений).

45. (Неэффективное) решето Эратосфена для многочленов

a. Дан список целых чисел от 2 до n . Решето Эратосфена позволяет, вычеркивая кратные числа, оставить в нем лишь те, которые являются простыми. Написать алгоритм, реализующий это решето.

b. Пусть p — целое простое число. Найти в явном виде биективное отображение множества натуральных чисел \mathbb{N} во множество унитарных непостоянных многочленов над $\mathbb{Z}/p\mathbb{Z}$. Если $n \geq 1$, то каким является множество B_n унитарных непостоянных многочленов степени $\leq n$ по модулю p ?

c. С помощью предыдущего кодирования записать алгоритм «решета Эратосфена», который дает список унитарных неприводимых многочленов по модулю p степени $\leq n$. Этот алгоритм является неэффективным по многим причинам. Каким?

d. Написать программу на языке Си, позволяющую найти перечень неприводимых многочленов по модулю 2 степени $\leq n$ для достаточно малого числа n (например, $n \leq 10$).

46. Многочлены $X^n - 1$

Пусть m и n — целые неотрицательные числа, d — их НОД.

a. Показать, что $\mathbb{Z}[X](X^d - 1) = \mathbb{Z}[X](X^n - 1) + \mathbb{Z}[X](X^m - 1)$. Вывести отсюда, что в любом кольце A имеем $A(a^d - 1) = A(a^n - 1) + A(a^m - 1)$ для $a \in A$ и что $\text{НОД}(a^n - 1, a^m - 1) = a^d - 1$. В частности, $\text{НОД}(X^n - 1, X^m - 1) = X^d - 1$.

b. Если $f(X)$ и $g(X)$ — два унитарных многочлена с целыми коэффициентами, а $h(X)$ — их НОД, a — целое число, то верно ли, что $h(a) = \text{НОД}(f(a), g(a))$?

с. Если $m > 0$, то каков остаток от деления $X^n - 1$ на $X^m - 1$?

47. Неприводимые многочлены над \mathbb{Z}

Показать, что многочлены $P_k = X^{2^k} + 1$ и $Q_n = X^n - 2$, неприводимы в $\mathbb{Z}[X]$.

48. Оценка числа неприводимых множителей

Возьмем A — кольцо без делителей нуля.

а. Пусть A — факториальное кольцо. Доказать, что если найдется неприводимый элемент p такой, что показатель a в p , $v_p(a)$ равен 1, то полином $X^n - a$ неприводим ($v_p(a)$ есть показатель p в разложении a в произведение неприводимых).

б. Пусть I — простой идеал в кольце A (не обязательно факториальном) и пусть $P = a_n X^n + \dots + a_0$ такой, что $a_n \notin I$ и для $i \neq n : a_i \in I$. Показать, что всякий непостоянный делитель P имеет свободный член, содержащийся в I .

с. Пусть A — факториально. Доказать, что всякий многочлен, для которого найдется неприводимый элемент p такой, что $p \nmid a_n$ и $p \mid a_i$ для всех a_i имеет не более $v_p(a_0)$ неприводимых сомножителей.

49. Неприводимый многочлен, приводимый по модулю всякого простого числа

а. Проверить, что многочлен $X^4 + 1$ приводим по модулю 2.

б. Пусть p — простое число, отличное от 2. Доказать, что число квадратов в группе обратимых элементов $U(\mathbb{Z}/p\mathbb{Z})$ равно числу не-квадратов. Вывести отсюда, что произведение двух не-квадратов есть квадрат.

с. Записав $X^4 + 1$ в одной из трех форм: $X^4 - (-1)$, $(X^2 + 1) - 2X^2$ или $(X^2 - 1)^2 - (-2X^2)$, доказать, что многочлен $X^4 + 1$ приводим по модулю p . Дайте примеры для разных значений p . Что можно утверждать, если $p \equiv 1 \pmod{4}$?

д. Многочлен $X^4 + 1$ неприводим по модулю 25? А по модулю 15?

50. Нули многочленов с целыми коэффициентами

а. Пусть $P = \sum a_i X^i$ — унитарный многочлен с целыми коэффициентами. Доказать непосредственно, что всякий рациональный корень P является в действительности целым и что он делит a_0 .

б. Пусть $P = \sum_{i \leq n} a_i X^i$ — многочлен, свободный член которого отличен от нуля. Существует ли рациональный корень p/q этого многочлена, где p/q — несократимая дробь, такой, что $p \mid a_0$ и $q \mid a_n$.

51. Расширенный критерий Эйзенштейна

а. Пусть $P = \sum_{i \leq n} a_i X^i$ — многочлен с коэффициентами в факториальном кольце. Пусть p — неприводимый элемент в A и $k \leq n$ такое, что $p^2 \nmid a_0, p \nmid a_k$ и $p \mid a_i$ для всех $i < k$. Доказать, что по меньшей мере один из неприводимых делителей P имеет степень, не меньшую k .

б. Доказать, что многочлен $P = X^4 + 3X^3 + 3X^2 - 5$ неприводим. При каких значениях a многочлен $Q = 5X^4 - 6X^3 - aX^2 - 4X + 2$ неприводим? Дайте возможные разложения.

52. Неприводимость циклотомических многочленов

Определим циклотомический многочлен уровня n как $\Phi_n(X) = \prod (X - \xi)$, где произведение распространено на все первообразные корни n -й степени из единицы в \mathbb{C} . В главе V будет доказано, что коэффициенты этих многочленов — целые числа. В упражнении этот факт будет использоваться как известный результат.

а. Пусть $\alpha \in \mathbb{C}$ корень n -й степени из 1 и P — неприводимый делитель $X^n - 1$, для которого α является корнем. Доказать, что если p — простое число, не делящее n , то $P(\alpha^p) = 0$. Указание: предположите, что $X^n - 1 = PQ$, и показать, что $P \mid Q(X^p)$. Перейдите затем в $\mathbb{Z}/p\mathbb{Z}[X]$.

б. Вывести отсюда, что Φ_n неприводим.

53. Идеалы в $K[[X]]$

Определить все идеалы в $K[[X]]$, кольцо формальных рядов с коэффициентами в поле K .

54. О факториальных кольцах

а. Пусть \mathcal{P} — система представителей неприводимых элементов факториального кольца A (каждый неприводимый элемент из A ассоциирован с одним и только одним неприводимым в \mathcal{P}). Обозначим через $\mathbb{N}^{(\mathcal{P})} \subset \mathbb{N}^{(\mathcal{P})}$ множество семейств целых чисел $(v_p)_{p \in \mathcal{P}}$, таких, что $v_p = 0$ для всех, за исключением конечного числа элементов в \mathcal{P} . Изучите свойства отображения:

$$\varphi : U(A) \times \mathbb{N}^{(\mathcal{P})} \rightarrow A - \{0\}, \quad \varphi(\epsilon, (v_p)_{p \in \mathcal{P}}) = \epsilon \prod_{p \in \mathcal{P}} p^{v_p}.$$

Вывести отсюда, что в факториальном кольце любые два элемента имеют НОД и что множество главных идеалов нётерово по включению.

б. В факториальном кольце выполняются следующие свойства:

$$c \text{ НОД}(a, b) = \text{НОД}(ca, cb), c \text{ НОК}(a, b) = \text{НОК}(ca, cb)$$

$$[a \mid bc \text{ и } \text{НОД}(a, b) = 1] \Rightarrow a \mid c, \quad [a \mid c \text{ и } b \mid c \text{ и } \text{НОД}(a, b) = 1] \Rightarrow ab \mid c.$$

Как доказать? Можно ли дать другое доказательство в случае, когда кольцо является КГИ?

с. Пусть A — кольцо без делителей нуля. Доказать, что A факториально тогда и только тогда, когда некоторые два элемента имеют НОД и множество главных идеалов A является нётеровым. Кроме того, всякий неприводимый элемент является простым и множество главных идеалов нётерово.

55. Условия, при которых факториальное кольцо является кольцом главных идеалов

В пунктах **а** и **б** A означает факториальное кольцо.

а. Пусть всякий неприводимый элемент A порождает максимальный идеал. Доказать, что A есть КГИ. Указание: доказать, что любой простой идеал главный, потом доказать, что $Ax + Ay = Ad$, если $d = \text{НОД}(x, y)$.

б. Допустим, что A удовлетворяет соотношению Безу (т. е. если x и y взаимно просты, то $1 \in Ax + Ay$). Доказать, что A есть КГИ.

с. Доказать, что конечное кольцо без делителей нуля есть поле. Лучше: если R — алгебра без делителей нуля, конечной размерности над полем K , то R — тело (R не предполагается коммутативной). Еще лучше: если R алгебра без делителей нуля, алгебраическая над полем K , то R — тело.

д. Пусть A — подкольцо в \mathbb{C} , целое для \mathbb{Z} (например, кольцо целых числового поля). Доказать, что всякий собственный простой идеал A , отличный от нуля, максимален. Указание: Если I такой идеал, то доказать, что $I \cap \mathbb{Z}$ имеет вид $p\mathbb{Z}$, где p — простое число. Затем рассмотреть $\mathbb{Z}/p\mathbb{Z}$ -алгебру A/I . Вывести отсюда, что если A факториально, то A — КГИ.

56. Произведение упорядоченных нётеровых множеств

Пусть $(E_i)_{i \in I}$ — семейство упорядоченных множеств. Наделим $\prod_{i \in I} E_i$ структурой упорядоченного произведения, положив $x \leq y$ тогда и только тогда, когда $x_i \leq y_i$ для всякого $i \in I$. В частности, если

$E_i = E$ для любого i , то $\prod_{i \in I} E_i = E^I$ есть пространство функций из I в E , наделенное обычным порядком.

а. Доказать, что конечное произведение множеств нётерово тогда и только тогда, когда каждый из множителей нётеров.

б. Рассмотрим подмножество $S \subset E^{\mathbb{N}}$ состоящее из стабилизирующихся последовательностей. Доказать, что в общем случае из нётеровости E не следует нётеровость S (а тем более и $E^{\mathbb{N}}$).

с. Доказать, что подмножество $E^{\mathbb{N}}$, состоящее из возрастающих последовательностей, является нётеровым, если таковым является E .

д. Установите связь между этими результатами и результатами о модулях (нётеровость произведения или частного двух модулей над кольцом многочленов).

57. Конечные (коммутативные) поля

В этом упражнении предполагается доказать несколько вполне классических результатов, касающихся конечных полей. В частности, что для любой степени q простого числа существует одно и только одно (с точностью до изоморфизма) конечное поле из q элементов. Для получения более подробных сведений читатель может обратиться к [111] или [120]. В частности, в [120] можно найти доказательство принадлежащего Веддербарну результата о том, что всякое конечное тело коммутативно.

Тела, рассматриваемые в этом упражнении, **заведомо коммутативны за исключением пункта (а)**. Бели A — унитарное кольцо, не обязательно коммутативное, то $\mathbb{Z} \ni m \rightarrow m \cdot 1 \in A$ есть морфизм. Его ядро, следовательно, есть идеал в \mathbb{Z} вида $m\mathbb{Z}$ для единственного $m \geq 0$. Упомянутое целое число m есть, следовательно, наименьшее целое число (в смысле отношения порядка, но также и в смысле делимости) такое, что $m \cdot 1 = 0$ или по другому $m \cdot x = 0$ для всякого $x \in A$ (такое m называется **характеристикой кольца** A).

а. Доказать, что характеристика кольца без делителей нуля есть или 0, или простое число p . В случае поля K доказать, что отображение $m \mapsto m \cdot 1$ задает морфизм (инъективный) поля $\mathbb{Q} \rightarrow K$ или $\mathbb{Z}/p\mathbb{Z} \rightarrow K$. Конечное поле имеет характеристику $p > 0$. Если $K \subset K'$ два конечных поля, то доказать, что $|K'|$ есть степень $|K|$ (в частности, если K — конечное поле характеристики p , то $|K|$ — степень p).

б. Пусть q — степень простого числа p и Ω — поле характеристики p . Доказать, что если Ω — поле, состоящее из q элементов, то все

корни многочлена $X^q - X$ содержатся в Ω . Доказать обратное, рассматривая автоморфизм Фробениуса $x \rightarrow x^q$ и показывая, что множество его неподвижных точек $\mathbb{F}_q = \{x \in \Omega \mid x^q = x\}$ является *единственным* подполем поля Ω , состоящим из q элементов. Как доказать (неэффективным образом) существование таких полей Ω ?

Замечание. На этом уровне удобно (но не обязательно) рассматривать алгебраически замкнутое поле характеристики p . Напомним, что алгебраически замкнутым называется поле, в котором всякий непостоянный многочлен имеет корень и, следовательно, разлагается в произведение линейных множителей (можно доказать, что всякое поле содержится в алгебраически замкнутом поле, см. например, [111]). В этом смысле все поля \mathbb{F}_q “живут” в алгебраически замкнутом поле Ω .

с. Пусть $K \subset K'$ — два конечных поля. $|K| = q$, $|K'| = q^n$. Поинтересуемся *промежуточными* полями H , т.е. такими, что $K \subset H \subset K'$. Если d — делитель n , то доказать, что $\{x \in K' \mid x^{q^d} = x\}$ промежуточное поле с q^d элементами и установите биекцию между делителями n и промежуточными полями. Эта биекция обладает “хорошим” алгебраическим свойством: каким? Если m — какой-либо делитель n , то $\{x \in K' \mid x^{q^m} = x\}$ — промежуточное поле. Какое у него число элементов? Если K_1 и K_2 — Два промежуточных поля порядков q^{d_1} и q^{d_2} соответственно, то доказать, что $|K_1 \cap K_2| = q^d$, где $d = \text{НОД}(d_1, d_2)$.

d. Пусть p — простое число, $q = p^n$ — степень p . Тогда, как известно, существует унитарный неприводимый многочлен P степени n . Это позволяет построить конкретное поле K из q элементов, взяв $K = \mathbb{Z}_p[X]/(P)$. Пусть K' — другое “абстрактное” поле из q элементов. Всякий многочлен с коэффициентами в \mathbb{Z}_p может рассматриваться как многочлен с коэффициентами в K' . Доказать, что P имеет корень y в K' , так как взятие значения в точке y для полиномов из $\mathbb{Z}_p[X]$ задает гомоморфизм $y: \mathbb{X}_p[X] \rightarrow K'$, индуцирующий изоморфизм K на K' . Отсюда следует, что любые два конечных поля одинакового порядка изоморфны.

e. Пусть K — конечное поле из q элементов и K' — расширение K степени n . Доказать, что отображение $r: x \mapsto x^q$ есть автоморфизм K' порядка n , множество неподвижных точек которого совпадает с K . Доказать, что всякий автоморфизм σ поля K' , оставляющий на месте K , есть степень r .

58. Уравнение $x^{p^k} = x$ в локальном кольце характеристики p

Пусть A унитарное коммутативное кольцо характеристики p , т.е. кольцо, в котором $p \cdot 1 = 0$, и I — идеал в A . Пример для размышлений — $A = K[T]$, где K — поле характеристики p и I — идеал, порожденный неприводимым многочленом. Доказать, что если q — степень p , то пространство S_n решений уравнения $X^q = X$ в A/I^n изоморфно (с помощью применения канонического отображения $\pi : A/I^n \rightarrow A/I$) пространству S_1 решений того же уравнения в A/I . Этот результат будет применен в главе IV, чтобы определить число неприводимых множителей для степени многочлена с коэффициентом в конечном поле.

а. Без всяких предположений относительно A доказать, что если $x \in A$ обратим по модулю I и по модулю J (I и J — любые идеалы), то x обратим по модулю IJ и, в частности, по модулю I^n для любого целого n .

б. Предположим теперь, что I — максимальный идеал, но не будем налагать ограничений на характеристику кольца A (так что q может быть любым). Доказать, что уравнение $x^q \equiv x \pmod{I^n}$ эквивалентно $x \equiv 0 \pmod{I^n}$ или $x^{q-1} \equiv 1 \pmod{I^n}$.

с. Обратимся теперь к предложению, сформулированному во введении. Проверить, что S_n и S_1 — две подалгебры и что $\pi(S_n) \subset S_1$. Вывести отсюда и из пункта **(б)**, что ограничение π на S_n инъективно.

д. Пусть a — элемент в I . Рассматривая потенциально бесконечную сумму $a + a^q + a^{q^2} + \dots + a^{q^i} + \dots$, доказать существование такого $s \in I$, что $s - s^q \equiv a \pmod{I^n}$. Вывести отсюда, что $\pi(S_n) = S_1$.

е. Можно обобщить ситуацию, забыв про идеал I^n , и использовать только кольцо $B = A/I^n$. Доказать, что это кольцо содержит единственный максимальный идеал \mathcal{M} , такой, что $B/\mathcal{M} \simeq A/I$. Утверждается, что это B есть локальное кольцо вычетов B/\mathcal{M} . Как сформулировать результаты пунктов **(б)**, **(с)** в зависимости от B и B/\mathcal{M} ? Чтобы получить конечный результат, надо наложить ограничения на идеал \mathcal{M} . Какое?

59. Эффективное кольцо главных идеалов, не являющееся евклидовым

Кольцо целых чисел в $\mathbb{Q}[\sqrt{-19}]$ есть кольцо $A = \mathbb{Z}[\theta]$, где $\theta = \frac{1+\sqrt{-19}}{2}$.

а. Какому уравнению степени 2 удовлетворяет θ ? Возьмите норму N в кольце A . Какие элементы обратимы в A ?

б. Пусть $a \in A$, $b \in A^*$. Доказать возможность эффективного вычисления таких $q, r \in A$, что $a = bq + r$ или даже $2a = bq + r$ с $N(r) < N(b)$.

с. Доказать явно, что 2 — максимальный элемент в A . Если $a = x + \theta y \notin 2A$, то найти такое \tilde{a} , что $a\tilde{a} \equiv 1 \pmod{2A}$, \tilde{a} — самое простое, по возможности. Указание: элемент принадлежит $2A$ тогда и только тогда, когда его норма четна (в дальнейшем такой элемент будет называться четным).

д. Доказать, что A есть КГИ. Указание: пусть I — ненулевой идеал в A , $b \in I^*$ — элемент с наибольшей нормой. Если $a \in I$, то псевдоделение a на b показывает, что $2a \in (b)$. Использовать теперь максимальность двойки...

е. Написать алгоритм, который для данных $a, b \in A$ находит u и v такие, что $ua + vb = \text{НОД}(a, b)$. Почему не для всех, а для данных a и b ?

В этом случае доказать, что псевдоделение из вопроса (**б**) позволяет выразить в явном виде $u, v, r \in A$, $\alpha \in \mathbb{N}$, такие, что r есть НОД для a и b , удовлетворяющий соотношению $2^\alpha r = ua + vb$.

Если $\alpha \geq 1$, выразить $2^{\alpha-1}r$ как линейную комбинацию от $2^\alpha r$ и a ...

Записать алгоритм вычисления коэффициентов Безу в кольце A .

Решения упражнений

1. Десятичные цифры простых чисел

Пусть l – длина B , т.е. $10^{l-1} \leq B < 10^l$ и $k \geq 0$. Тогда числа вида $n \cdot 10^{k+1} + B \cdot 10^k + c$ являются числами десятичная запись которых содержит последовательность , если $0 \leq c < 10^k$. Если $k > 0$, то можно взять взаимно простым с 10 и тогда числа 10^{k+1} и $B \cdot 10^k + c$ будут взаимно просты. Поэтому можно применить теорему Дирихле. В действительности оказывается, что существует бесконечно много простых чисел, содержащих фиксированную последовательность цифр B в заранее заданных позициях.

2. Вычисление НОД

Допустим, что $m > n$. Тогда $a^m - b^m = (a^n - b^n)a^{m-n} + (a^{m-n} - b^{m-n}b^n)$. Это тождество и условия взаимной простоты a и b дает

$$\begin{aligned}\text{НОД}(a^m - b^m, a^n - b^n) &= \text{НОД}(a^{m-n} - b^{m-n}, a^n - b^n) = \\ &= \text{НОД}(a^{m \bmod n} - b^{m \bmod n}, a^n - b^n),\end{aligned}$$

Повторяя это соотношение как в алгоритме Евклида, получим искомый результат.

3. Алгоритм Евклида и непрерывные дроби

а. Пусть a/b — рациональное несократимое число (с $b > 0$). Тогда последовательность частных, полученных в алгоритме Евклида, примененном к a и b , дает разложение a/b в непрерывную дробь. К тому же все частные, за исключением, быть может, первого, положительны, если используемое деление является обычным делением Евклида.

б. Рассмотрим непрерывные дроби $s_i = [c_i, c_{i+1}, \dots, c_m]$ и $t_i = [d_i, d_{i+1}, \dots, d_n]$. Тогда, очевидно, имеем $s_i = c_i + 1/s_{i+1}$ и аналогичное соотношение для t и d . Отсюда получаем $s_i > 1$ для $i < m$, и, следовательно, $[s_i] = c_i$. кроме того, по предположению, $s_0 = [c_0, c_1, \dots, c_m] = [d_0, d_1, \dots, d_n] = t_0$. Используя предыдущее соотношение, получаем $c_0 = d_0$ (целые части s_0 и t_0) и $s_1 = t_1$. Затем постепенно

доказываем, что $c_i = d_i$. Этот процесс заканчивается на наименьшем из чисел m и n . Допустим, что это m . Тогда $s_m = t_m$ и $c_m = d_m$ через предшествующую рекуррентность. Кроме того, $s_m = c_m$ по определению s . В результате $c_m = d_m$ и $m = n$.

с. Достаточно рассмотреть последние деления алгоритма Евклида:

$$r_{n-2} = r_{n-1}a_{n-1} + 1 \quad \text{и} \quad r_{n-1} = 1 \times r_{n-1} \quad \text{с} \quad r_{n-1} > 1.$$

Последнее деление можно заменить на следующее: $r_{n-1} = 1 \times (r_{n-1} - 1) + 1$ без появления нулевого частного и закончить деление на $1 = 1 \times 1$. Это означает, что

$$[c_0, c_1, c_2, \dots, c_n] = [c_0, c_1, c_2, \dots, c_n - 1, 1], \quad \text{если} \quad c_n > 1.$$

Это эквивалентно факту, что любое целое число a допускает ровно два разложения в непрерывную дробь: $[a]$ и $[a - 1, 1]$.

d. Вот часть ответа (развиваемая потом в упражнении 6):

$$[a_0] = a_0, \quad [a_0, a_1] = \frac{a_0 a_1 + 1}{a_1}, \quad [a_0, a_1, a_2] = \frac{a_0 a_1 a_2 + a_0 a_1 + a_1 a_2}{a_1 a_2 + 1} \dots, \\ F_4/F_3 = [1, 1, 2] \quad \text{и} \quad F_5/F_4 = [1, 1, 1, 2].$$

4. Многочлены континуаты

a. Легко видеть, что $F_n = K_n(1, \dots, 1)$. Это означает, что в K_n имеется F_n слагаемых.

b. Нетрудно проверить, что свойство верно для $n = -1, 0, 1$. Рассмотрим моном $X_1 \dots X_n$. Можно разделить исключения примыкающих пар на две категории: те, которые исключают пару $X_n X_{n-1}$, и те, которые оставляют X_n . По предположению индукции, первая категория дает все одночлены от K_{n-2} , а вторая — все одночлены от K_{n-1} , умноженные на X_n . Отсюда результат (обнаруженный впервые Эйлером).

с. Следовательно, континуанты обладают зеркальной симметрией: $K_n(X_1, \dots, X_n) = K_n(X_n, \dots, X_1)$. Можно определить последовательность K_n следующим образом:

$$K_n = X_1 K_{n-1}(X_2, \dots, X_n) + K_{n-2}(X_3, \dots, X_n).$$

Соотношение, которое будет использоваться в следующих упражнениях.

Требуемое тождество доказывается легко. Оно указывает, что в поле рациональных дробей

$$\frac{K_n(X_1, \dots, X_n)}{K_{n-1}(X_2, \dots, X_n)} = [X_1, X_2, \dots, X_n] = X_1 + \frac{1}{X_2 + \dots}.$$

Если $r_{i-1} = r_i a_i + r_{i+1}$, где $r_n = 1$ и $r_{n+1} = 0$ есть последовательность делений алгоритма Евклида, примененного к целым взаимно простым числам, то $r_i = K_{n-i}(a_{i+1}, \dots, a_n)$.

5. Континуаты (продолжение)

a. Нетрудно доказать, используя индукцию, что рассматриваемое произведение равно матрице

$$\begin{pmatrix} K_n(X_1, \dots, X_n) & K_{n-1}(X_1, \dots, X_{n-1}) \\ K_{n-1}(X_2, \dots, X_n) & K_{n-2}(X_2, \dots, X_{n-1}) \end{pmatrix}$$

Этот результат дает другое доказательство зеркальной симметричности многочленов континуант.

b. Вычислив определитель, находим искомое равенство (с точностью до индексов). Следовательно, значения $K_n(a_1, \dots, a_n)$ и $K_{n-1}(a_2, \dots, a_n)$ дают взаимно простые целые числа.

c. Простая индукция показывает, что искомое произведение равно

$$\begin{pmatrix} K_{n+2}(X_1, \dots, X_{n+2}) & K_{n+1}(X_2, \dots, X_{n+2}) \\ K_n(X_1, \dots, X_n) & K_{n-1}(X_2, \dots, X_n) \end{pmatrix}$$

Определитель этой матрицы (что и требовалось доказать) равен

$$X_{n+2} \left(K_{n-1}(X_2, \dots, X_n) K_{n+1}(X_1, \dots, X_{n+1}) - \right. \\ \left. - K_n(X_1, \dots, X_n) K_n(X_2, \dots, X_{n+1}) \right),$$

т.е. $(-1)^{n+1} X_{n+2}$ по предыдущему пункту

6. Разложение в непрерывную дробь

a. Это решение уже намечено в упражнении 3:

$$[a_0] = a_0, \quad [a_0, a_1] = \frac{a_0 a_1 + 1}{a_1}, \quad [a_0, a_1, a_2] = \frac{a_0 a_1 a_2 + a_0 a_1 + a_1 a_2}{a_1 a_2 + 1} \dots$$

Формула выглядит так:

$$[a_0, a_1, \dots, a_n] = \frac{K_{n+1}(a_0, a_1, \dots, a_n)}{K_n(a_1, \dots, a_n)},$$

и доказывается по индукции. Попутно использовалось соотношение $[a_0, \dots, a_n] = [a_0, [a_1, \dots, a_n]]$ и равенство, выражающее зеркальность многочленов континуант.

б. Требуемое соотношение вытекает прямо из определения последовательностей (a_i) и (x_i) . Запишем

$$\begin{aligned} |x - [a_0, \dots, a_n]| &= |[a_0, \dots, a_n, x_{n+1}] - [a_0, \dots, a_n]| = \\ &= \left| \frac{K_{n+2}(a_0, \dots, a_n, x_{n+1})}{K_{n+1}(a_1, \dots, a_n, x_{n+1})} - \frac{K_{n+1}(a_0, \dots, a_n)}{K_n(a_1, \dots, a_n)} \right| = \\ &= \frac{1}{K_n(a_1, \dots, a_n)K_{n+1}(a_1, \dots, a_n, x_{n+1})} \end{aligned} \quad (2.11)$$

Очевидно, x_{n+1} положительно (по построению) и континуанты возрастают (хорошо видно, каким образом), следовательно, получаем

$$|x - [a_0, \dots, a_n]| \leq \frac{1}{K_n(a_1, \dots, a_n)^2} < \frac{1}{2K_n(a_1, \dots, a_n)}, \quad (2.12)$$

Соотношение, которое впоследствии понадобится. Это доказывает сходимость x . Можно также констатировать, что подходящие вещественные числа дают очень хорошие аппроксимации указанного числа. В действительности, как увидим в дальнейшем, они даже наилучшие.

с. Установленная в пункте **(а)** формула дает значение в неприводимой форме, ибо, согласно упражнению 5, вводимые туда числа взаимно просты. Остается их только вычислить, что и сделаем, применяя рекуррентное определение континуант и значений многочленов. Приводим совмещенные вычисления числителя и знаменателя, чтобы минимизировать необходимые операции.

```

K ← 1; K' ← an;
for (int i=n-1; i>=0; i=i-1){
    (K', K) ← (aiK' + K, K');
}

```

В начале итерации для значения i имеем инвариантное отношение

$$\frac{K'}{K} = [a_{i+1}, \dots, a_n]$$

Этот алгоритм(который оперирует с коэффициентами разложения), разумеется, не должен использоваться для вычисления последовательности подходящих дробей, так как каждый промежуточный этап не дает первые подходящие дроби.

Если необходимо вычислить последовательность подходящих дробей, то можно использовать потоковый алгоритм разложения. Для этого используют две рекуррентные последовательности, определяемые через

$$\begin{aligned} P_0 &= 1, & P_1 &= a_0, & P_n &= a_n P_{n-1} + P_{n-2}, \\ Q_0 &= 0, & Q_1 &= 1, & Q_n &= a_n Q_{n-1} + Q_{n-2}. \end{aligned}$$

Итак, определяемые последовательности удовлетворяют соотношению $[a_0, \dots, a_n] = P_{n+1}/Q_{n+1}$. Это нас приводит к алгоритму 5.

```
int a;
scanf("%d", &a);
P ← 1; P' ← a; Q ← 0; Q' ← 1;
printf("%d %d\n", P', Q');
while(){
    scanf("%d", &a);
    (P', P) ← (aP' + P, P'); (Q', Q) ← (aQ' + Q, Q');
    printf("%d %d\n", P', Q');
}
```

Алгоритм 5. Вычисление текущих подходящих дробей

d. Число золотого сечения удовлетворяет уравнению $x = 1 + 1/x$. Следовательно, его разложение в непрерывную дробь есть $[1, 1, 1, 1, \dots]$. Последовательность подходящих дробей, согласно пункту **(a)** упражнения 4 и предшествующему пункту **(a)** упражнения 6, есть отношения последовательных чисел Фибоначчи: $F_2/F_1, F_3/F_2, F_4/F_3, \dots$

7. Аппроксимация вещественных чисел с помощью непрерывных дробей

a. Выражение $p_{n+1}q_{n-1} - q_{n+1}p_{n-1} = K_{n+2}K_{n-1} - K_{n+1}K_n$ есть знак $(-1)^{n+1}$, откуда следует результат. Выражение $p_{n+1}q_n - q_{n+1}p_n = K_{n+2}K_n - K_{n+1}K_{n+1}$ есть знак $(-1)^n$. Факт, что x находится между двумя сходящимися последовательностями показывает, что эти последовательности сходятся к x .

b. Равенство (11) утверждает, что

$$\left| x - \frac{p_n}{q_n} \right| = |x - [a_0, \dots, a_n]| = \frac{1}{K_n(a_1, \dots, a_n)K_{n+1}(a_1, \dots, a_n, x_{n+1})}.$$

Используя двустороннюю оценку $a_{n+1} < x_{n+1} < a_{n+1}$, а также, что каждая континуата — возрастающая функция любой из этих переменных, получаем

$$\frac{1}{q_n K_{n+1}(a_1, \dots, a_n, a_{n+1} + 1)} < \left| x - \frac{p_n}{q_n} \right| < \frac{1}{q_n q_{n+1}}$$

Но $K_{n+1}(X_1, \dots, X_n, X_{n+1} + 1)$. Действительно, первая континуанта, по определению, равна $K_{n+2}(X_1, \dots, X_{n+1}, 1)$, тогда как вторая равна по определению $K_{n+1}(X_1, \dots, X_{n+1}) + K_n(X_1, \dots, X_n)$, т.е. второй континуанте. Следовательно,

$$K_{n+1}(a_1, \dots, a_n, a_{n+1} + 1) \leq K_{n+2}(a_1, \dots, a_{n+1}, a_{n+2}) = q_{n+2},$$

откуда и получается требуемая оценка. Другие неравенства выводятся немедленно.

с. Зная, что $|x - p_n/q_n| < 1/2q_n$ (соотношение (12)), и что $|p_n/q_n - p/q| \geq 1/q_n$ (так как $q = q_n$), получаем, что $|x - p/q| > 1/2q_n$, что и дает искомое неравенство.

d. Матричные соотношения показывают, что матрица $\begin{pmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{pmatrix}$ обратима над \mathbb{Z} . Следовательно, можно записать $p = \alpha p_n + \beta p_{n-1}$ и $q = \alpha q_n + \beta q_{n-1}$, где α и β — целые. Однако $q < q_n$, откуда следует, что β — не нуль и что α и β имеют противоположные знаки. Кроме того, согласно доказанному в вопросе (а), $p_n - xq_n$ и $p_{n-1} - xq_{n-1}$ тоже имеют противоположные знаки. Отсюда $\alpha(p_n - xq_n)$ и $\beta(p_{n-1} - xq_{n-1})$ одного знака, а потому

$$|p - xq| = |\alpha(p_n - xq_n) + \beta(p_{n-1} - xq_{n-1})| \geq |\beta| \cdot |p_{n-1} - xq_{n-1}| > |p_n - xq_n|$$

8. Факториал и простые числа

a. Пусть $n = p_1^{\alpha_1} \dots p_r^{\alpha_r}$ — разложение n на простые множители. Предположим сначала, что $r > 1$. Тогда $p_i^{\alpha_i} < n$ и, следовательно, делит $(n-1)!$. Если $r = 1$, то так как n — непростое, это приводит к тому, что $\alpha_1 > 1$. Если $\alpha_1 = 2$, то числа p_1 и $2p_1$ встречаются в разложении $(n-1)!$, кроме случая $p_1 = 2$. Если $\alpha_1 > 2$, то числа p_1 и $p_1^{\alpha_1}$ меньше n и, следовательно, появляются в разложении $(n-1)!$.

b. Действительно, очевидно, что всякое целое число k из интервала $[2, n+1]$ делит $(n+1)! + k$. Отсюда мы нашли те целые числа, которые искали.

с. Одно из двух: либо $n! + 1$ — простое число, и тогда задача решена, либо оно составное. В последнем случае оно не может делиться на простое число, меньшее или равное n . Итак, оно делится на простое, большее n , и, разумеется, $< n! + 1$. Это доказательство, в действительности, аналогично доказательству Евклида бесконечности множества простых чисел.

d. Первые числа e_i являются, действительно, простыми: $e_1 = 2$, $e_2 = 3$, $e_3 = 7$, $e_4 = 43$. Однако следующее $e_5 = 1807 = 13 \times 139$ уже нет. В действительности при $5 \leq i \leq 17$ числа e_i составные. Эти числа, очевидно, взаимно простые. Можно также доказать, что существует такая вещественная константа $E \approx 1,26$, что $e_n = \lfloor E^{2n} + \frac{1}{2} \rfloor$, но это похоже на шутку, как и для константы K из упражнения 40.

9. Каноническое разложение $n!$

Если $v_p(m)$ обозначает показатель, с которым простое число p входит в число m , то $v_p(n!) = \sum_{1 \leq m \leq n} v_p(m)$ и остается только доказать, что эта сумма равна $S_n = \sum_{i=1}^{\infty} \lfloor n/p^i \rfloor$. Это можно сделать по индукции, доказывая, что $S_{n+1} = S_n + v_p(n+1)$, и проверяя, что $S_1 = 0$. Если a и b — два целых числа, то записывая $a = bq + r$, где $0 \leq r < b$, легко доказать, что $\lfloor (a+1)/b \rfloor - \lfloor a/b \rfloor$ равно 0, если $b \nmid a+1$, и равно 1, если $b \mid a+1$. Следовательно,

$$\lfloor (n+1)/p^i \rfloor - \lfloor n/p^i \rfloor = \begin{cases} 1, & \text{если } i \leq v_p(n+1), \\ 0 & \text{в противном случае.} \end{cases}$$

Отсюда $S_{n+1} - S_n = \sum_{1 \leq i \leq v_p(n+1)} 1 = v_p(n+1)$.

10. Уравнение Ферма для $n = 2$ и $n = 4$

a. Если x и y нечетны, $x^2 \equiv y^2 \equiv 1 \pmod{4}$ и, следовательно, $z^2 \equiv 2 \pmod{4}$, что невозможно. Если (x, y, z) — решение уравнения, то (dx, dy, dz) — также решение уравнения для любого $d \in \mathbb{Z}$. Обратно, если $d = \text{НОД}(x, y)$, то $d^2 | z^2$, откуда $d | z$ и $(x/d, y/d, z/d)$ — также решения.

b. Если $(x + iy) = (1 + i)(\alpha + i\beta)$, то $x = \alpha - \beta$ и $y = \alpha + \beta$, что дает искомый результат (см. по этому поводу упражнение 12).

В $\mathbb{Z}[i]$ имеем $z^2 = (x + iy)(x - iy)$, где x и y имеют **противоположную четность**. Отсюда ясно, что $x + iy$ и $x - iy$ взаимно просты. Действительно, единственное неприводимое число π , которое могло бы делить одновременно $x + iy$ и $x - iy$, должно делить их сумму $2x$

и разность $2y$, а потому и число 2, являющееся линейной комбинацией $2x$ и $2y$. Но разложение 2 на неприводимые есть $(1+i)(1-i)$, тогда как ни $1+i$, ни $1-i$ не делят $x+iy$ (x и y противоположной четности).

Однако в факториальном кольце (каковым является $\mathbb{Z}[i]$), если произведение двух взаимно простых множителей является квадратом, то это же справедливо и для каждого из сомножителей, являющихся взаимно простыми, с точностью до **обратимых элементов**. Итак, $x+iy = \pm i(\alpha+i\beta)^2$ со взаимно простыми α и β , что отвечает на прямую половину вопроса. Обратное легко проверить непосредственно.

Например, $3^2 + 4^2 = 5^2$ ($u=1$, $v=2$), $5^2 + 12^2 = 13^2$ ($u=2$, $v=3$).

с. Предположим, что x и y взаимно просты. Тогда можно считать, что x нечетно, $y = 2y'$ — четно. Следовательно, z нечетно. Теперь достаточно записать $y^2 = (z-x)(z+x)$, а затем $y'^2 = \frac{z-x}{2} \frac{z+x}{2}$. Два взаимно простых числа $\frac{z-x}{2}$ и $\frac{z+x}{2}$ дающие в произведении квадрат, сами являются квадратами. Поэтому немедленно получаем решение.

d. Можно предположить, что x и y взаимно просты. Запишем $x^4 + y^4 = (x^2)^2 + (y^2)^2$ и, применяя предыдущий результат, предполагая (при необходимости x и y можно поменять местами), что x нечетно, y четно, получим:

$$x^2 = u^2 - v^2, \quad y^2 = 2uv, \quad u, v > 0, \quad \text{НОД}(u, v) = 1.$$

Тогда имеем $x^2 + v^2 = u^2$ с $\text{НОД}(x, v) = 1$. Применим еще раз предыдущий результат, учитывая, что x нечетно:

$$x = a^2 - b^2, \quad v = 2ab, \quad u = a^2 + b^2, \quad a, b > 0, \quad \text{НОД}(a, b) = 1.$$

Отсюда выводим $y^2 = 2uv = 4uab$. Однако u , a и b попарно взаимно просты, а произведение $4uab$ — квадрат. Следовательно, u , a и b — квадраты: $a = c^2$, $b = d^2$. Поэтому

$$c^4 + d^4 = a^2 + b^2 = u \text{ квадрат.}$$

Но $x^4 + y^4 = (u^2 + v^2)^2 > u = c^4 + d^4$. Доказательство закончено, так как в \mathbb{N} не существует бесконечной строго убывающей последовательности (это и есть метод бесконечного спуска, принадлежащий Ферма).

11. Простое диофантово уравнение

а. Пары $(\pm i, 0)$ и $(\pm 2i, \pm i)$ являются решениями в $\mathbb{Z}[i]$.

Если бы существовало целое решение, то оно удовлетворяло бы соотношению $3y^2 = x^2 + 1$, что невозможно, так как 3 не может делить сумму двух квадратов, стоящую справа (более простое рассуждение

заключается в том, чтобы привести уравнение по модулю 3, аналогично тому, как это сделано в следующем вопросе).

б. Приведя уравнение по модулю 5, получаем $x^2 + z^2 = 0$. Однако 2 не является корнем квадратным из -1 по модулю 5. Следовательно, $z \equiv \pm 2 \pmod{5}$, что дает нам решение $(k, \pm k, \pm 2k)$. Однако оно не единственно.

с. Очевидно, что решений нет, если -1 не является квадратичным вычетом по модулю p . Поэтому $p \equiv 1 \pmod{4}$. При этих условиях (мы это уже видели) число p может быть записано в виде суммы двух квадратов, что дает бесконечное множество решений рассматриваемого уравнения.

12. Кольцо $\mathbb{Z}[\theta]/(a + \theta b)$, θ — квадратичная иррациональность, $\text{НОД}(a, b) = 1$

а. Если доказать, что $[i]_{a+ib} = \varphi(t)$, то $[x + iy]_{a+ib} = \varphi(x + ty)$, что доказывает сюръективность φ . Достаточно рассмотреть в $\mathbb{Z}[i]$ уравнение $i - t = (a + ib)(v + iu) = (av - bu) + i(au + bv)$. Если u, v — коэффициенты Безу для a и b и положить $t = bu - av$, то получим искомое соотношение.

б. Если $\text{Ker } \varphi = (a^2 + b^2)\mathbb{Z}$, то φ индуцирует изоморфизм $\mathbb{Z}/(a^2 + b^2)$ на $\mathbb{Z}/(a + ib)$, а так как t переходит в i , то имеем $t^2 \equiv -1 \pmod{a^2 + b^2}$. Из очевидного сравнения $a^2 \equiv -b^2 \pmod{a^2 + b^2}$ получаем $(ab^{-1})^2 \equiv -1 \pmod{a^2 + b^2}$, где b^{-1} — обратный элемент к b по модулю $a^2 + b^2$. Следовательно, если $a + tb \equiv 0 \pmod{a^2 + b^2}$, то $t \equiv -ab^{-1} \pmod{a^2 + b^2}$, откуда $t^2 \equiv -1 \pmod{a^2 + b^2}$. Для выбранного t , $t = bu - av$, известно, что $a + tb \equiv 0 \pmod{a^2 + b^2}$ и элемент $t = bu - av$ и в самом деле является противоположным к ab^{-1} по модулю $a^2 + b^2$.

Примечание

Используя выражение $t = bu - av$ и равенство $au + bv = 1$ с помощью прямых вычислений можно показать, что $t^2 \equiv -1 \pmod{a^2 + b^2}$. Но можно также использовать соотношение $t - i \equiv 0 \pmod{a + ib}$, откуда $t + i \equiv 0 \pmod{a - ib}$, из чего выводим $t^2 + 1 \equiv 0 \pmod{a^2 + b^2}$. ■

с. Положив $\mu(x + iy) = x + ty$ и используя соотношение $t^2 \equiv -1 \pmod{a^2 + b^2}$, покажем, что $\mu : \mathbb{Z}[i] \rightarrow \mathbb{Z}/(a^2 + b^2)$ — морфизм колец. Ясно, что $\mu(m) = m$ для $m \in \mathbb{Z}$ и $\mu(a + ib) = a + tb \equiv 0 \pmod{a^2 + b^2}$. Отсюда следует, что μ дает переход к факторкольцу и $\text{Ker } \varphi = (a^2 + b^2)\mathbb{Z}$, где равенство следует из того, что μ и φ индуцируют взаимно обратные морфизмы.

К примеру, $1 + i$ делит $x + iy$ тогда и только тогда, когда x и y одной четности. Для $a + ib = 9 + 2i$ имеем $t = 38$ и потому $x + iy$ делится на $9 + 2i$ тогда и только тогда, когда $x + 38y$ делится на 85.

d. $\mathbb{Z}[i]/(a + ib)$: Существуют два очевидных случая, когда кольцо $\mathbb{Z}[i]/(a + ib)$ состоит из $(a^2 + b^2)$ элементов. Первый случай — когда $b = 0$, второй — когда a и b взаимно просты и кольцо $\mathbb{Z}[i]/(a + ib)$ изоморфно $\mathbb{Z}/(a^2 + b^2)$. В общем случае, если $d = \text{НОД}(a, b)$, $a = da'$, $b = db'$ каноническое отображение $\mathbb{Z}[i]/(a + ib)$ в $\mathbb{Z}[i]/(d)$ сюръективно и с ядром, изоморфным $\mathbb{Z}[i]/(a' + ib')$ (каноническая инъекция $\mathbb{Z}[i]/(a' + ib') \rightarrow \mathbb{Z}[i]/(a + ib)$ получается умножением на d и образ — в точности это кольцо), откуда следует результат.

e. Кольцо $\mathbb{Z}[\theta]$ снабжено автоморфизмом, который переводит θ в другой корень $\tilde{\theta}$ уравнения $X^2 - SX + P = 0$ и нормой: $N(a + b\theta) = (a + b\theta)(a + b\tilde{\theta}) = a^2 + Sab + Pb^2$. Положив $z = a + b\theta$, докажем, что кольца $\mathbb{Z}[\theta]/(z)$ и $\mathbb{Z}/N(z)$ изоморфны.

Для этого достаточно найти в $\mathbb{Z}/N(z)$ элемент t , играющий роль θ в $\mathbb{Z}[\theta]/(z)$, т.е. $a + bt \equiv 0 \pmod{N(z)}$. Это возможно, так как b взаимно просты с $N(z) = a^2 + Sab + Pb^2$. Следовательно, он обратим по модулю $a^2 + Sab + Pb^2$ и потому $t = -ab^{-1} \pmod{N(z)}$. Элемент t удовлетворяет тому же уравнению, что и θ :

$$t^2 - St + P \equiv 0 \pmod{N(z)}.$$

Докажем теперь, что отображение $\mu : \mathbb{Z}[\theta] \rightarrow \mathbb{Z}/N(z)$, определенное по правилу $\mu(x + y\theta) = x + yt$, является морфизмом колец, который можно пропустить через $\mathbb{Z}[\theta]/(z) \rightarrow \mathbb{Z}/N(z)$. Впрочем, $N(z)$ делится на z и потому имеется канонический морфизм $\varphi : \mathbb{Z}/N(z) \rightarrow \mathbb{Z}[\theta]/(z)$. Тот факт, что эти морфизмы взаимно обратны, равносильно тому, что $t \equiv \theta \pmod{z}$, и это как раз тот случай, так как $bt \equiv -a \equiv b\theta \pmod{z}$. Последнее равенство можно упростить, сократив на b (b обратим по модулю z ввиду $ua + vb = 1$, что приводит к $u(a + b\theta) + (v - u\theta)b = 1$). Отсюда можно сделать вывод, что кольцо $\mathbb{Z}[\theta]/(z)$ состоит из $|N(z)|$ элементов для взаимно простых a и b , а затем перейти к случаю произвольных a и b .

13. НОД в $\mathbb{Z}[i]$ и суммы двух квадратов

a. Специфическое предположение для $\mathbb{Z}[i]$, что p делит $(x+i)(x-i)$. Тогда $z = u + iv = \text{НОД}(p, x + i)$ является **настоящим** делителем p : действительно, z не может быть обратимым (если π неприводимый элемент в $\mathbb{Z}[i]$, делящий p , то он делит $(x + i)(x - i)$). Следовательно, π или π делит p и $x + i$ и z не является ассоциированным

с p (так как $p \nmid x + i$). Поэтому, если z' такое число, что $zz' = p$, то

$$N(z)N(z') = p^2 \Rightarrow N(z) = u^2 + v^2 = p.$$

Используя вычисление НОД в $\mathbb{Z}[i]$, находим:

$$1\,301 = 25^2 + 26^2, \quad 1\,000\,037 = 991^2 + 134^2, \quad 2\,000\,004\,973 = 39\,027^2 + 21\,838^2.$$

б. Если $z = u + iv = \text{НОД}(n, x + iy)$, числа u и v взаимно просты то можно применить результат упражнения 12. Так как z делит n , $z\bar{z}$ делит n . С другой стороны, так как $z \in n\mathbb{Z}[i] + (x + iy)\mathbb{Z}[i]$, то получаем

$$\begin{aligned} z\bar{z} &\in (n\mathbb{Z}[i] + (x + iy)\mathbb{Z}[i])(n\mathbb{Z}[i] + (x - iy)\mathbb{Z}[i]) \subset \\ &\subset n\mathbb{Z}[i] + (x^2 + y^2)\mathbb{Z}[i] = n\mathbb{Z}[i]. \end{aligned}$$

Целые числа n и $z\bar{z}$ делят друг друга, откуда следует их равенство.

с. Имеем $y_k = y^{p-1} = 1$ (малая теорема Ферма). Для $y \in H$ пусть i — наибольшее целое число, такое, что $y_i \neq 1$; $1 \leq i \leq k$, так как, с одной стороны, $y_k = 1$, а с другой, $y_1 \neq 1$ по предположению. Так как $1 = y_{i+1} = y_i^2$, то получаем, что $y_i = -1$ и y_{i-1} — корень квадратный из -1 по модулю p . Чтобы пересчитать H , рассмотрим его дополнение (*плохие элементы*) $G = \{y \in U(\mathbb{Z}_p), y^{2^q} = 1\}$. Это подгруппа **циклической** группы $U(\mathbb{Z}_p)$. Несложное рассуждение показывает, что $|G| = 2q$ и, следовательно, $\frac{|G|}{|U(\mathbb{Z}_p)|} = \frac{1}{2^{k-1}} \leq \frac{1}{2}$. Вероятность случайным образом выбрать m плохих элементов, следовательно, $\frac{1}{2^m}$. Можно запрограммировать вероятностный алгоритм, приведенный ниже (не забывайте вычислять $y^q \bmod p$ при помощи дихотомии!) и сравнить его с детерминистским алгоритмом, который дает $(\frac{p-1}{2})! \bmod p$ как корень квадратный из -1 (который из двух более эффективен?).

```
do{
  Выбрать  $y$  случайно в  $[2...p-2]$ ;  $y \leftarrow y^q \bmod p$ ;  $y' \leftarrow y^2$ ;
}while(  $y' == 1$  );
while(  $y' \neq -1 \bmod p$  )
{
    //  $y' = y^2$ 
     $y \leftarrow y'$ ;
     $y' \leftarrow y^2$ ;
}
//  $y$  — корень квадратный из  $-1$  по модулю  $p$ 
```

14. Делимость сумм $x^2 + 2y^2$

а. Норма N может быть продолжена на поле $\mathbb{Q}[\sqrt{-2}]$, подполе поля \mathbb{C} , натянутое на 1 и $\sqrt{-2}$. $\mathbb{Q}[\sqrt{-2}]$ есть расширение степени 2 с базой

$\{1, \sqrt{-2}\}$, и, конечно, продолженная норма остается мультипликативной.

Если $c = r + t\sqrt{-2} \in \mathbb{Q}[\sqrt{-2}]$ и $x, y \in \mathbb{Z}$ выбраны так, что $|r - x| \leq \frac{1}{2}$ и $|t - y| \leq \frac{1}{2}$, то, обозначая $x + y\sqrt{-2}$ через q , имеем:

$$N(c - q) = (r - x)^2 + 2(t - y)^2 \leq \frac{1}{4} + \frac{2}{4},$$

откуда $N(c - q) < 1$.

Если a и b — два элемента $\mathbb{Z}[\sqrt{-2}]$, где $b \neq 0$, то достаточно применить этот результат с $a/b \in \mathbb{Q}[\sqrt{-2}]$, чтобы найти $q \in \mathbb{Z}[\sqrt{-2}]$ такое, что $N(a/b - q) < 1$ или $N(a - bq) < N(b)$, что доказывает евклидовость $\mathbb{Z}[\sqrt{-2}]$ относительно N .

б. С помощью рассуждений, совершенно аналогичных доказательству леммы 5, докажем, что для неприводимого числа z из \mathbb{Z} выполнено: $N(z)$ — простое число. Копируя доказательство предложения 6, можно показать, что положительный делитель $x^2 + 2y^2$, где x и y взаимно просты, имеет форму $u^2 + 2v^2$.

с. Если p представляется в виде $p = x^2 + 2y^2$, то y не нуль по модулю p и $-2 = (xy^{-1})^2$ в $\mathbb{Z}/p\mathbb{Z}$, так как y обратим в \mathbb{Z}_p ; значит -2 — квадрат по модулю p .

Обратно, если -2 есть квадрат по модулю p , то найдется x такой, что p делит $x^2 + 2$. Согласно предыдущему, p представимо в виде $u^2 + 2v^2$.

15. Делимость целых чисел вида $x^2 - 2y^2$

а. На этот раз кольцо есть $\mathbb{Z}[\sqrt{2}]$. В отличие от уже изученных случаев это кольцо **вещественное**: $\{1, \sqrt{-2}\}$ является \mathbb{Z} -базой $\mathbb{Z}[\sqrt{-2}]$. Кольцо частных его есть поле $\mathbb{Q}[\sqrt{-2}]$, имеющее инволютивный автоморфизм $z \mapsto \tilde{z}$, который $z = r + t\sqrt{2}$ ставит в соответствие $\tilde{z} = r - t\sqrt{2}$. Этот автоморфизм оставляет на месте $\mathbb{Z}[\sqrt{2}]$ и позволяет определить мультипликативную норму N через $N(z) = z\tilde{z}$. Другими словами $N(r + t\sqrt{2}) = r^2 - 2t^2$. Доказательство того, что $\mathbb{Z}[\sqrt{2}]$ является евклидовым для $|N|$, аналогично решению упражнения 14. Заметим, однако, что N может принимать отрицательные значения, откуда и использование абсолютного значения N .

б. Заметим, что -1 является нормой: $-1 = 1^2 - 2 \times 1^2 = N(1 + \sqrt{2})$. Используя рассуждения, аналогичные предложению 6, докажем, что делитель (положительный или нет) числа вида $x^2 - 2y^2$ (x и y взаимно

простые) имеет вид (с точностью до знака ± 1) и $u^2 - 2v^2$. Но равенство

$$-(u^2 - 2v^2) = N((1 + \sqrt{2})(u + v\sqrt{2})) = (u + 2v)^2 - 2(u + v)^2$$

позволяет пренебречь знаком \pm . Следуя соответствующему этапу решения упражнения 13, можно доказать, что нечетное простое число p имеет вид $x^2 - 2y^2$ тогда и только тогда, когда 2 является квадратом по модулю p , что эквивалентно $|p| \equiv \pm 1 \pmod{8}$.

16. Кольца $\mathbb{Z}[\sqrt{-3}]$ и $\mathbb{Z}[\sqrt{10}]$ нефакториальны

а. Простое натуральное число, не являющееся нормой в кольце, остается неприводимым в этом кольце (возьмите норму от произведения). Если p и q таковы и pq — норма ($pq = N(z) = z\bar{z}$), то z и \bar{z} неприводимы (используйте норму) и pq разлагается двумя способами: $pq = z\bar{z}$. Например, $2 \times 5 = (1 + \sqrt{-3}) \times (1 - \sqrt{-3})$. Это доказывает нефакториальность кольца $\mathbb{Z}[\sqrt{-3}]$. Можно проверить, что элементы $2(1 + \sqrt{-3})$ и 10 не имеют НОД в $\mathbb{Z}[\sqrt{-3}]$ и что идеал, порожденный $2(1 + \sqrt{-3})$ и 10, неглавный.

б. Можно использовать тот же тип рассуждений, что и выше, однако в этом случае норма кольца $\mathbb{Z}[\sqrt{10}]$ уже не является неотрицательным числом... Однако все улаживается, так как -1 является значением нормы: $-1 = 3^2 - 10 \times 1^2 = N(3 + \sqrt{10})$... Вот плохое разложение на неприводимые: $2 \times 3 = (4 + \sqrt{10}) \times (4 - \sqrt{10})$ (можно проверить, что ни 2, ни 3 не являются нормами, рассуждая по модулю 10).

17. Целозамкнутые кольца

а. Пусть $F(X)$ — унитарный полином с целыми коэффициентами:

$$F(X) = X^n + a_{n-1}X^{n-1} + \dots + a_2X^2 + a_1X + a_0$$

и p/q рациональное число (p и q взаимно просты), являющееся корнем $F(X)$. Тогда

$$p^n + a_{n-1}p^{n-1}q + a_{n-2}p^{n-2}q^2 + \dots + a_2p^2q^{n-2} + a_1pq^{n-1} + a_0q^n = 0,$$

откуда следует, ввиду того, что p и q взаимно просты и q делит p^n , что $q = \pm 1$. Это доказывает целозамкнутость любого факториального кольца.

б. Нецелозамкнутость $\mathbb{Z}[\sqrt{d}]$ следует из тождества:

$$x^2 - x = \frac{1-d}{4} = 0 \quad \text{с} \quad x = \frac{1+\sqrt{d}}{2}.$$

Заметим, что ± 2 не может быть нормой в $\mathbb{Z}[\sqrt{d}]$, так как равенство $x^2 - dy^2 = \pm 2$ приводит к тому, что в $\mathbb{Z}/4\mathbb{Z}$ $x^2 - y^2 = 2$, что невозможно. Отправляясь от равенства $2 = uv$, получаем $4 = N(u)N(v)$ и, следовательно, $N(u) = \pm 1$ или $N(v) = \pm 1$, что показывает неприводимость числа 2. Но 2 — непростое число:

$$2 \mid (1 + \sqrt{d}) \cdot (1 - \sqrt{d}), \quad \text{хотя} \quad 2 \nmid (1 + \sqrt{d}) \text{ и } 2 \nmid (1 - \sqrt{d}).$$

с. В $\mathbb{Z}[2i]$ число 2 неприводимо. В самом деле, из равенства $2 = zz'$ получаем $4 = N(z)N(z')$ и поскольку $\mathbb{Z}[2i]$ не содержит элемента нормы 2 ($a^2 + 4b^2$ не принимает значение 2), либо $N(z) = 1$, либо $N(z') = 1$, откуда следует обратимость z или z' .

Элемент 4 может быть разложен двумя способами в произведение неприводимых неассоциированных элементов: $4 = 2 \times 2 = -2i \times 2i$. Действительно, неассоциированность $2i$ и 2 следует из того, что единственным числом, которое могло бы их ассоциировать, является i , не являющееся элементом в $\mathbb{Z}[2i]$. Кольцо $\mathbb{Z}[2i]$ не является факториальным.

Разложение числа 4 демонстрирует еще один феномен. Существуют два числа a и b такие, что $a^2 \mid b^2$, но a не делит b . Кольцо $\mathbb{Z}[2i]$, следовательно, не является целозамкнутым. Другая странность, связанная с этим равенством: 2 неприводимо, но не является простым!

Наконец, $\mathbb{Q}[i]$ — поле частных $\mathbb{Z}[2i]$, однако многочлен $X^2 + 1$ неприводим в $\mathbb{Z}[2i][X]$, но не в $\mathbb{Q}[i][X]$. В противном случае его разложение на неприводимые элементы было бы разложением в поле дробей $\mathbb{Z}[2i]$, однако $(X + i)(X - i)$ не является разложением над $\mathbb{Z}[2i]$.

18. Целые элементы; целые квадратичности

а. Пусть $(1, \alpha)$ — база K над \mathbb{Q} . Докажите, что α является корнем квадратного многочлена с коэффициентами из \mathbb{Z} . После этого приведите число, появляющееся под знаком радикала к такому виду, чтобы у него не было множителей, являющихся полными квадратами.

б. Целые алгебраические числа являются элементами \mathbb{Z} . Другими словами, рациональные алгебраические числа являются целыми рациональными числами.

с. Если $\frac{1+\sqrt{d}}{2}$ удовлетворяет уравнению $x^2 - sx + p = 0$, где $s, p \in \mathbb{Z}$, то при $s = 1$, $\frac{1+\sqrt{d}}{2} = \frac{1-d}{4}$. Следовательно, $d \equiv 1 \pmod{4}$. Обратно, если $d \equiv 1 \pmod{4}$, то s и p определены так:

$$s = \frac{1 + \sqrt{d}}{2} + \frac{1 - \sqrt{d}}{2} = 1, \quad p = \frac{1 + \sqrt{d}}{2} \times \frac{1 - \sqrt{d}}{2} = \frac{1 - d}{4},$$

— целые числа, и $\frac{1+\sqrt{d}}{2}$ является корнем уравнения $x^2 - sx + p = 0$.

19. Целые числа в $\mathbb{Q}(\sqrt{d})$

б. Равенства:

$$x + y = (z + \sigma(z))(z' + \sigma(z')), \quad xy = (z^2 + \sigma(z)^2)z'\sigma(z') + (z'^2 + \sigma(z')^2)z\sigma(z)$$

доказывают, что $x + y$ и xy — элементы \mathbb{Z} . Следовательно, x и y — целые алгебраические числа, а так как они принадлежат \mathbb{Q} (они остаются на месте под действием σ), то $x, y \in \mathbb{Z}$. Итак, доказано, что элементы $zz' + \sigma(zz')$, $zz' \cdot \sigma(zz')$ и $(z + z') + \sigma(z + z')$, $(z + z') + \sigma(z + z')$ содержатся в \mathbb{Z} . Следовательно, zz' , $z + z'$ — целые алгебраические числа.

с. Всякий элемент x из $\mathbb{Q}(\sqrt{d})$ является решением квадратного уравнения над \mathbb{Q} . Следовательно, существуют целые a , b и c такие, что $ax^2 + bx + c = 0$, где a и b не равны нулю одновременно. Если, кроме того, x — целый над \mathbb{Z} , то он корень унитарного многочлена с целыми коэффициентами P и тогда НОД($P, aX^2 + bX + c$) есть унитарный многочлен степени, не превосходящей 2, аннулирующий значением x .

д. Пусть $z = a + b\sqrt{d} \in \mathbb{Q}(\sqrt{d})$. Равенства: $z + \sigma(z) = 2a$ и $z\sigma(z) = a^2 - db^2$ показывают, что z — целое квадратичное число тогда и только тогда, когда $2a \in \mathbb{Z}$ и $a^2 - db^2 \in \mathbb{Z}$. Правое условие дает $d(2b)^2 \in \mathbb{Z}$. Затем (так как d не имеет квадратных множителей), обозначив $a = u/2$ и $b = v/2$, предыдущее условие можно переписать так: $u^2 - dv^2 \in 4\mathbb{Z}$. Используя то, что квадратами в $\mathbb{Z}/4\mathbb{Z}$ являются лишь 0 и 1, получаем:

$$z \text{ целое квадратичное} \Leftrightarrow \begin{cases} u \equiv 0 \pmod{2}, & v \equiv 0 \pmod{2}, \\ & \text{если } d \equiv 2, 3 \pmod{4}, \\ u \equiv v \pmod{2}, & \text{если } d \equiv 1 \pmod{4}. \end{cases}$$

20. Евклидовость квадратичных колец

б. Будем различать два случая в зависимости от того, каков остаток от деления d на 4.

Первый случай. Пусть $d \not\equiv 1 \pmod{4}$. Для $z \in \mathbb{Q}(\sqrt{d})$, $z = \alpha + \beta\sqrt{d}$, где $\alpha, \beta \in \mathbb{Q}$ и $q \in A$ с $q = a + b\sqrt{d}$, имеем: $|N(z - q)| = |(\alpha - a)^2 - d(\beta - b)^2|$, откуда извлекаем две оценки:

$$N(z - q) \leq |\alpha - a|^2 + |d||\beta - b|^2, \quad \text{и при } d > 0 \\ |N(z - q)| \leq \max(|\alpha - a|^2, d|\beta - b|^2),$$

Если выбрать $a, b \in \mathbb{Z}$ такие, что $|\alpha - a| \leq 1/2$ и $|\beta - b| \leq 1/2$, то получаем $|N(z - q)| \leq (1 + |d|)/4$ и при $d > 0$: $|N(z - q)| \leq \max(1/4, d/4)$. Отсюда $N(z - q) < 1$, так как $d \in \{-1, -2, 2, 3\}$.

Второй случай. Предположим, что $d \equiv 1 \pmod{4}$. Норма вычисляется следующим образом:

$$N\left(u + v \frac{1 + \sqrt{d}}{2}\right) = \left(u + \frac{v}{2}\right)^2 - d \left(\frac{v}{2}\right)^2.$$

Для $z \in Q(\sqrt{d})$, $z = \alpha + \beta \frac{1 + \sqrt{d}}{2}$ с $\alpha, \beta \in \mathbb{Q}$ и $q \in A$, $q = a + b \frac{1 + \sqrt{d}}{2}$, где $a, b \in \mathbb{Z}$, имеем:

$$|N(z - q)| = \left| \left(\alpha - a + \frac{\beta - b}{2} \right)^2 - d \left(\frac{\beta - b}{2} \right)^2 \right|.$$

Отсюда имеем следующие две оценки $|N(z - q)|$ в зависимости от знака d :

$$d < 0 : \quad \left(\alpha - a + \frac{\beta - b}{2} \right)^2 + |d| \left(\frac{\beta - b}{2} \right)^2, \quad d > 0 : \\ \max \left(\left| \alpha - a + \frac{\beta - b}{2} \right|^2, d \left| \frac{\beta - b}{2} \right|^2 \right).$$

Если выбрать $b \in \mathbb{Z}$ так, что $|\beta - b| \leq 1/2$, а затем $a \in \mathbb{Z}$ так, что $\left| \alpha - a + \frac{\beta - b}{2} \right| \leq 1/2$, то $|N(z - q)| \leq 1/4 + |d|/16$ и при $d > 0$: $|N(z - q)| \leq \max(1/4, d/16)$, что во всех случаях должно давать $|N(z - q)| < 1$. Поэтому $d \in \{-11, -7, -3, 5, 13\}$.

$u + v\theta \stackrel{\text{def}}{=} z'\bar{z};$
 $u \leftarrow \text{LeastRem}(v, |N(z)|); \quad x \leftarrow \text{LeastRem}$
 $// \quad y \equiv v \pmod{N(x)} \text{ и } x \equiv u \pmod{N(z)} \text{ с } |y|, |x| < |N(z)|/2$
 $r \leftarrow x + y\theta; \quad r \leftarrow \frac{rz}{N(z)}; \quad q \leftarrow \frac{(z' - r)\bar{z}}{N(z)};$

Алгоритм 6. Евклидово деление z' на z в $\mathbb{Z}[\theta]$,
где $\theta = \sqrt{d}$ и $d = -1, -2, 2, 3$.

Конечно, возможна реализация евклидова деления, использующая только целочисленные вычисления. Элементы кольца A кодируются своими координатами в базе $\{1, \theta\}$ с $\theta = \sqrt{d}$, если $d \equiv 2$ или $3 \pmod{4}$, $\theta = \frac{1 + \sqrt{d}}{2}$, если $d \equiv 1 \pmod{4}$. Для вычисления частного и евклидова остатка z' для z , вычисляем $z'\bar{z}$ на целое число $N(z)$. Затем делим (точным образом) полученные результаты на \bar{z} . Нам необходима функция

LeastRem: $\mathbb{Z} \times \mathbb{N}^* \rightarrow \mathbb{Z}$, для которой $r = \text{LeastRem}(a, b)$ — наименьший остаток от деления a на b ($|r| \leq b/2$). Алгоритм 6 рассматривает более простой первый случай, когда $d \equiv 2$ или $3 \pmod{4}$.

Второй случай $d \equiv 1 \pmod{4}$ (алгоритм 7) более сложен. Он использует переменные целые числа x и y , предназначенные для установки в них, в соответствии со значениями переменных a и b из предыдущего доказательства, следующим образом: $y = b$, $x = 2a + b$, $a = (x - y)/2$.

```

u+vθ  $\stackrel{\text{def}}{=} z'\bar{z}$ 
y  $\leftarrow \text{LeastRem}(v, |N(z)|)$ ;  $x \leftarrow \text{LeastRem}(2u + y, 2 \times |N(z)|)$ ;
y  $\equiv v \pmod{N(z)}$  и  $2u + y \equiv x \pmod{2N(z)}$ 
  c|y| < |N(z)|/2 и |x| < |N(z)|
r  $\leftarrow (x - y)/2 + y\theta$ ;  $r \leftarrow rz/N(z)$ ;  $q \leftarrow (z' - r)\bar{z}/N(z)$ ;

```

Алгоритм 7. Евклидово деление в $\mathbb{Z}[\theta]$ при $\theta = \frac{1+\sqrt{d}}{2}$ и $d = -11, -7, -3, 5, 13$.

21. Нахождение колец мнимых квадратичностей

а. Элемент $z \in A$ обратим тогда и только тогда, когда $N(z) = 1$. Отсюда следует, что $U(A) = -1, 1$, кроме, возможно, $d = -1, -3$. Для $d = -1$, $A = \mathbb{Z}[i]$ выполнено: $U(A)$ — циклическая группа порядка 4, порожденная i ; для $d = -3$, $A = \mathbb{Z}[i]$ и $U(A)$ — циклическая группа, порожденная элементом j порядка 6.

б. Пусть $b \in B^* = U(B)$, для которого $\varphi(b)$ имеет наименьшее значение. Если $a \in B$, то $a = bq + r$, где $\varphi(r) < \varphi(b)$, что приводит к $r \in U(B)$ или $r = 0$. Но так как $U(B)$ предполагается редуцированной к $-1, 1$, то $r = 0, \pm 1$ и, следовательно, $B/(b) = \{\bar{0}, \bar{1}, \bar{-1}\}$, откуда и заключение.

с. Для упомянутых значений d мы видели (упражнение 20), что A — евклидово кольцо. Докажем обратное. Предположим, что A евклидово, и докажем, что $d \in \{-11, -7, -3, -2, -1\}$. Можно считать, что $d \neq -1, -3$. Это случаи, когда группа единиц редуцируется к $\{-1, 1\}$. Тогда существует такой элемент $z \in A$, что $|A/(z)| = 2$ или 3 . Но $|A/(z)| = |N(z)|$ (упражнение 12) и, следовательно, записав $N(z) = 2$ или 3 . Читатель может сделать вывод самостоятельно, записав $N(z) = a^2 + |d|b^2$ при $d \equiv 2$ или $3 \pmod{4}$, или $4N(z) = a^2 + |d|b^2$ (a и b одной четности) при $d \equiv 1 \pmod{4}$.

22. Вычисление НОД с помощью двоичных операций

Необходимое соотношение, конечно, $\text{НОД}(a, b) = 2^{\inf(\alpha, \beta)} \text{НОД}(a', b')$. Итак, вычисление НОД двух целых чисел приводит после возможного

деления на степени двойки к вычислению НОД для их нечетных частей. Используя к тому же свойство НОД $(a, b) = \text{НОД}(a - b, b)$, получаем следующий алгоритм.

```

int Binary_God( $a, b \in \mathbb{N}^*$ ){
    Reduce( $a, \alpha, a'$ ); Reduce( $b, \beta, b'$ );
    while( $a' \neq b'$ ){ //  $a'$  и НОД( $a, b$ ) =  $2^{\inf(\alpha, \beta)}$  НОД( $a', b'$ )
        if( $a' > b'$ ){  $a' -= b'$ ; Reduce( $a', \gamma, a'$ ); }
        else{  $b' -= a'$ ; Reduce( $b', \gamma, b'$ ); }
    }
    if ( $\alpha > \beta$ ) return  $2^\beta a'$ ;
    else return  $2^\alpha a'$ ;
}

```

в котором процедура $\text{Reduce}(a, \alpha, a')$, начиная с первого аргумента, выдает для a нечетное число a' , такое, что $a = 2^\alpha a'$. Этот алгоритм заканчивается за конечное число шагов, так как количество пар (a', b') , строго положительных целых чисел убывает на каждой итерации (следует заметить, что разность $|a' - b'|$ может не убывать. Например, для $a = 459$ и $b = 309$).

Этот алгоритм, примененный к числам 1610 и 1000, более быстрый (по числу итераций), чем алгоритм Евклида, использующий деление с наименьшим остатком, что не противоречит теореме Лазара, сформулированной в разделе 6.2. Впрочем, этот алгоритм заметно более медленный, чем алгоритм Евклида для пары (509, 3).

Внимание: деление с четным остатком существует только тогда, когда a и b одинаковой четности. В любом случае требуемая программа будет использовать процедуру Reduce , и, следовательно, необходимо наличие этого деления, когда a и b нечетные. Учитывая, что деление на 2 — очень быстрая операция, можно предполагать, что полученный таким образом алгоритм, как правило, эффективнее, чем обычный алгоритм Евклида. Но насколько?

23. Другое соотношение для последовательности Фибоначчи

Требуемое соотношение доказывается с помощью простой индукции. Если вспомнить, что два соседних числа Фибоначчи взаимно просты, то отсюда легко выводится, что $\text{НОД } F_n$ и F_m равен $\text{НОД}(F_n, F_{n+m})$ и тогда можно применять метод Евклида, опираясь на значения индексов.

Соотношений для последовательности Фибоначчи очень много (Кнут [99] приводит их в большом количестве). Можно заметить, что последовательность, используемая в доказательстве, есть немедленное

следствие аналогичного соотношения для многочленов континуант в упражнении 5.

24. Вычисление НОД n -ки целых чисел

б. Вот алгоритм, отвечающий на вопрос:

```

Number_Of_Zeroes = 0;
for (i = 0; i != n; n++){
    Repere = (Repere + 1) % n;
    if (uRepere != 0) break;
    Number_Of_Zeroes += 1;
}
if (Number_Of_Zeroes >= n - 1) break;
Modulus = uRepere;
for (i=0; i != n; i++){
    ui = ui % Modulus;
}
uRepere = Modulus;

```

с. Главное в этом алгоритме состоит в том, чтобы уравнивать два ре пера, называемые левым и правым, ненулевыми элементами массива u . Эти два репера стремятся друг к другу, и, как только они совпадают, вычисления заканчиваются. Избежим в этом алгоритме повторяющихся критериев обращения в нуль компонентов и и некоторых делений (редукция с помощью $Modulus$). Вот несколько строчек алгоритма:

```

while (left < правый && u=0){
    left+=1;
}
if (left > right) break;
for (i=left; i<=right; i++){    ui = ui mod uleft; }
.....

```

Продолжение есть не что иное, как повторение этого фрагмента с переменной ролей *левый* и *правый* и с заменой прибавления единицы на вычитание единицы.

25. Странный алгоритм Евклида

Для всякого элемента $b \in \mathbb{N}$ существует система представителей из $\mathbb{Z}/b\mathbb{Z}$, элементы которой r удовлетворяют соотношению $\varphi(r) < \varphi(r)$.

Это очевидно, если $b \leq 5$. Если $b > 5$, то $5 - b$ представляет 5 по модулю b ...

Можно заметить, что мы располагаем парами (a, b) , где b — делитель a с евклидовыми делениями $a = bq + r$, где r уже не нуль! (Возьмите, например, $40 = 5 \times 6 + 10$.)

27. Самый малый алгоритм Евклида на $\bullet Z$

То, что φ — алгоритм Евклида, содержится в примере (раздел 3.1) из этого курса. Пусть ψ — другой алгоритм Евклида на Z со значениями в N . Докажем с помощью индукции по $n = \psi(b)$, что $\varphi(b) \leq \psi(b)$.

Это верно для $n = 0$, так как из $\psi(b) = 0$ следует, что $b = 0$ (см. лемму 29). Предположив, что это выполняется для всех b , таких, что $\psi(b) < n$, докажем, что оно верно и для $\psi(b) = n$.

Подвергнем ψ -евклидову делению $[|b|/2]$ на число b : $[|b|/2] = bq + r$, где $\psi(r) < \psi(b)$. Так как r представитель $[|b|/2]$ по модулю b , то $|r| \geq [b/2]$. Отсюда имеем:

$$\varphi(r) \geq \varphi([|b|/2]) = \varphi(b) - 1.$$

Так как $\psi(r) < n$, то по индукции $\psi(r) = \varphi(r)$ и поэтому:

$$\varphi(b) \leq \varphi(r) + 1 = \psi(r) + 1 \leq \psi(b),$$

что заканчивает доказательство.

30. Конечные поля, порядок которых есть квадрат простого числа

Согласно изложенному выше (раздел 1.3, предложение 8), ρ — не приводимо в $Z[i]$ и потому порождает максимальный идеал. Итак, $Z[i]/(\rho)$ — поле, так как:

$$x + iy \equiv 0 \pmod{\rho} \iff x, y \equiv 0 \pmod{\rho},$$

аддитивная группа $Z[i]/(\rho)$ изоморфна $\mathbb{Z}/(\rho) \times \mathbb{Z}/(\rho)$, откуда порядок поля, о котором идет речь, равен ρ^2 .

31. Несколько конечных полей порядка, являющегося степенью двойки

Многочлены $P(X)$ следующие:

$$\begin{array}{lll} X^2 + X + 1, & X^3 + X + 1, & X^4 + X + 1, \\ & X^5 + X^2 + 1, & X^6 + X + 1, & X^7 + X + 1, \end{array}$$

неприводимые по модулю 2 (для каждой данной степени n — это первые неприводимые многочлены степени n , в смысле лексикографического порядка, о чем читатель, конечно, догадался). Поля $\mathbb{Z}_2[X]/(P)$ отвечают на вопрос задачи.

32. Получение всех коэффициентов Безу

Так как (a, b) можно всегда заменить на $(a/d, b/d)$, то будем предполагать, что $d = 1$, т.е. a и b взаимно просты. Тогда имеем: $(u_0 - u)a = (v - v_0)b \Rightarrow a \mid (v - v_0)b$. К тому же a и b взаимно просты. Отсюда a делит $v - v_0$. Следовательно, существует $k \in \mathbb{Z}$ такое, что $k = u_0v - v_0u$, так как $a(u_0v - v_0u) = v - v_0$.

33. Единственность ограниченных коэффициентов Безу

С помощью замены (a, b) на $(a/d, b/d)$ можно всегда предполагать, что $d = 1$.

а. Согласно предыдущему упражнению, существует такое $k \in \mathbb{Z}$, что $u' = u - kb$ и $v' = v + ka$. Следовательно, число u' есть представитель по модулю b числа u , отличный от u . Неравенство для $|u|$ показывает, что $|u'| \geq |b|/2$. Равенство возможно только для четного b . Но a и b не могут быть оба четными, так как они взаимно просты.

34. Мажорирование коэффициентов Безу в $K[X]$

Достаточно симитировать доказательство, касающееся коэффициентов Безу в \mathbb{Z} из раздела 7.2. Алгоритм создает последовательность Q_1, Q_2, \dots, Q_n с одной стороны, U_0, U_1, \dots, U_{n+1} и V_0, V_1, \dots, V_{n+1} с:

$$\begin{aligned} U_0 = 1, \quad U_1 = 0, \quad U_{i+1} = U_{i-1} - Q_i U_i, \\ V_0 = 0, \quad V_1 = 0, \quad V_{i+1} = V_{i-1} - Q_i U_i, \quad 1 \leq i \leq n, \end{aligned}$$

и легко проверить, что для $2 \leq i < n$ $\deg(Q_i) > 0$ и:

$$\begin{aligned} \deg(U_2) < \deg(U_3) < \dots < \deg(U_{n+1}), \\ \deg(V_2) < \deg(V_3) < \dots < \deg(V_{n+1}), \end{aligned}$$

что дает искомый результат, так как U_n и V_n — коэффициенты Безу и $U_{n+1} \sim B, V_{n+1} \sim A$.

35. Дерево Штерна — Броко

a. Свойство $p'q - pq' = 1$ верно для всякой пары последовательных дробей, полученных с помощью описанной процедуры. Это доказывает взаимную простоту числителей и знаменателей полученных дробей и что $\frac{p}{q} < \frac{p+p'}{q+q'} < \frac{p'}{q'}$.

Если $\frac{p}{q} < \frac{b}{c} < \frac{p'}{q'}$, то из неравенств $bq - cp \geq 1$ и $cp' - bq' \geq 1$ получаем, что

$$q + q' \leq q(cp' - bq') + q'(bq - cp) = c.$$

b. Рассматривая во время процесса деления дробь $\frac{a}{b}$, можно заметить, что не может быть получена дробь со знаменателем, большим, чем b , легко вывести алгоритм 8, в котором используются первообразные уравнения, введенные в упражнении 30 главы I.

```
float List(float n){
    float Auxiliaire, Resultat;
    if (n == 1){
        return{[ $\frac{0}{1}, \frac{1}{1}$ ]};
    }
    else{
        Auxiliaire ← Farey(n-1); Resultat ← NULL;
        x ← Head(Auxiliaire); Auxiliaire ← Tail{Auxiliaire};
        while(Auxiliaire != null){
            Resultat ← Construct(z, Resultat); y
            ← Head(Auxiliaire); Auxiliaire ← Tail(
                Auxiliaire);
            z ← x ⊕ y;
            if (Denominateur(z) <= n){
                Resultat ← Construct(z, Resultat);
            }
            x ← y;
            Resultat ← Construct(x, Resultat);
        }
        return Reverse(Resultat);
    }
}
```

Алгоритм 2.5: Прохождение ряда Фарей

Список-результат построен в обратном порядке (что не очень важно само по себе). Перед окончанием выполнения его порядок еще раз меняют. В этом алгоритме знак \oplus обозначает странную операцию, которую применяют к дробям в процессе Штерна — Броко.

с. Если две последовательные дроби в \mathcal{F}_N не удовлетворяют соотношению $q + q' > N$, то медиана принадлежит \mathcal{F}_N и должна находиться между двумя простыми, что является противоречием

Начиная с соотношения $p'q - pq' = 1$ и $p''q' - p'q'' = 1$, получаем

$$p'(p''q - pq'') = p + p' \quad \text{и} \quad q'(p''q - pq'') = q + q'',$$

что доказывает первое из свойств. Отправляясь от тех же соотношений, можно доказать, что

$$p''q - pq'' = \frac{q + q''}{q'} = \frac{q + N}{q'} - \frac{N - q''}{q'}.$$

Согласно свойству знаменателей, доказанному выше, последний терм меньше 1 и, следовательно, второе свойство доказано.

Для формул, позволяющих выразить член ряда Фарея с помощью двух предшествующих дробей, достаточно проверить, что два выражения для p'' и q'' взаимно просты и что их отношение эффективно дает a'' . В этом случае запись алгоритма очевидна.

d. Предположим, что p и q отличны от нуля и единицы и что $p < q$. Свойство, сформулированное в первой задаче, позволяет доказать, что обе родительские дроби для $\frac{p}{q}$ дают коэффициенты Безу для p и q . Кроме того, свойства

- всякая дробь $\frac{p}{q}$ на — Брокко с помощью двух дробей, расположенных на той же ветви дерева,
- на ветви дерева, начиная с уровня 2, знаменатели строго возрастают

позволяют доказать, что одна из этих пар коэффициентов минимальна

36. Вычисление коэффициентов Безу n-ки целых чисел

```

u1 ← 1; d ← a1;
for(i = 0; i == 2; i++){
  //d = НОД(a1, ..., ai-1) = Uiai + ... + ui-1ai-1; j > i : Uj = 0
  (α, β) ← Bezout(d, ai); d ← αd + βai;
  for(j = 0; j == (i-1)){
    u2 ← αuj;
    ui ← β;
  }
  if (|d| == 1) break;
}
```

Алгоритм 2.6: Коэффициенты Безу

Пусть требуется вычислить коэффициенты Безу n -ки целых чисел \bar{a} . Алгоритм 9 находит эти коэффициенты и помещает их \bar{u} .

39. Сложность центрированного деления

а. Первые члены последовательности (G_n) выглядят следующим образом: 0, 1, 2, 5, 12, 29, 70, 169, ... Имеется точно n центрированных делений в алгоритме Евклида, примененном к паре (a, b) , ибо:

$$\begin{aligned} a &= 1 \times G_n = G_{n-1}, & |G_{n-1}| &\leq |G_n|/2, \\ G_n &= 2 \times G_{n-2} + G_{n-2}, & |G_{n-2}| &\leq |G_{n-1}|/2, \\ &\vdots \\ G_3 &= 2 \times G_2 + G_1, & |G_1| &\leq |G_2|/2, \\ G_2 &= 2 \times G_1 + 0. \end{aligned}$$

б. Запишем деления в виде $r_{i-1} = r_i q_i + r_{i+1}$ с $|r_{i+1}| \leq |r_i|/2$ для $1 \leq i \leq n$ и $|r_1| < |r_0|$. Отсюда следует, что $|q_i| \geq 2$ для $2 \leq i \leq n$. Чтобы доказать неравенство $|r_{i-1}| \geq 2|r_i| + |r_{i+1}|$ ((для $2 \leq i \leq n$), рассмотрим два случая:

- $|q_i| < 3$: $|r_{i-1}| = |r_i q_i + r_{i+1}| \geq |q_i||r_i| - |r_{i+1}| \geq 3|r_i| - |r_{i+1}| \geq 2|r_i| + |r_{i+1}|$.
- $|q_i| < 3$ лишь при $q_i = \pm 2$. Из того, что $|\pm 2r_i + r_{i+1}| = |r_{i-1}| \geq 2|r_i|$, следует $|r_{i-1}| = |\pm 2r_i + r_{i+1}| = 2|r_i| + |r_{i+1}|$ (если выполняется $|u+v| \geq |u|$ и $|v| \leq |u|$, то $|u+v| = |u| + |v|$).

Из неравенств $|r_{i-1}| \geq 2|r_i| + |r_{i+1}|$ выводим:

$$\begin{aligned} |r_n| &\geq 1 = G_1, |r_{n-1}| \geq 2 = G_2, \\ |r_{n-2}| &\geq 2|r_{n-1}| + |r_n| \geq 2G_2 + G_1 = G_3, \\ &\vdots \\ |r_1| &\geq 2|r_2| + |r_3| \geq 2G_{n-2} = G_n, |r_0| \geq |r_1| + |r_2| \geq G_n + G_{n-1}. \end{aligned}$$

Оставшаяся часть упражнения трудностей не представляет.

40. Порождения простых чисел с помощью многочленов

а. Предположим, что существует такой многочлен. Нетрудно показать, что $P(n)$ делит $P(n + kP(n))$ для любого целого k (достаточно расписать $P(n + kP(n))$). Так как P не постоянен и принимает только простые значения, то приведенное свойство означает, что для всякого $k \in \mathbb{Z}$, $P(n) = \pm P(n + kP(n))$. Следовательно, один из многочленов

$P - P(n)$ или $P + P(n)$ имеет бесконечное множество корней, что противоречит тому, что P не постоянен.

Замечание: натуральное число n называется эйлеровым, если много член $X^2 + X + n$ принимает простые значения для всякого целого числа из интервала $[0, n - 1]$. Доказано, что существует только шесть (уже известных) чисел Эйлера: 2, 3, 5, 11, 17, 41 (см. Ле Лионне [113]).

b. В действительности мы докажем, что это неравенство почти всегда точное. Согласно постулату Бертрана $p_{n+i} < 2p_n$, и если имеется точное неравенство для p_n то оно имеется и для p_{n+1} . Достаточно теперь заметить, что точное неравенство будет при $n = 3$ (так как $7 < 5 + 3 + 2$).

c. Постулат Бертрана позволяет утверждать, что n -е простое число не меньше 2^n и, следовательно, 10^n . Тогда, если через K обозначим сумму ряда $\sum_{k \geq 1} 10^{-k^2}$, то это и будет та постоянная, которую мы ищем. Действительно,

$$10^{n^2} K = 10^{n^2-1} p_1 + 10^{n^2-4} p_2 + \dots + 10^{2n-1} p_{n-1} + p_n + 10^{-2n-1} p_{n+1} + 10^{-4n-4} p_{n+2} + \dots \equiv p_n + x \pmod{10^{2n-1}}, x \in]0, 1[.$$

Так как $p_n < 10^n$, то утверждение верно.

Однако, с практической точки зрения, приведенная выше формула числа K неэффективна: чтобы посчитать по ней число K , необходимо прежде всего знать последовательность простых чисел, а затем еще и просуммировать ряд ... приблизительное значение этой суммы 0,2003. Вообще, приемлемая формула, позволяющая находить простые произвольно большие числа, не известна.

41. Соотношение Безу для многочленов

Если P и Q взаимно простые многочлены из $A[X]$, то они принадлежат и $K[X]$. Из равенства Безу $UP + VQ = 1$, освобождаясь от знаменателей, появляющихся в U и V , получаем соотношение, требуемое в упражнении. Расширенный алгоритм Евклида показывает, что можно выбрать многочлены U и V , удовлетворяющие условиям, наложенным на степени (см. упражнение 34).

42. Сложность алгоритма Евклида в $K[X]$

a. Пусть $(R_i)_{0 \leq i \leq n+1}$ — последовательность остатков в алгоритме Евклида, примененном к паре (A, B) , с $R_0 = A, R_1 = B, R_{n+1} = 0$ и $R_n \neq 0$; тогда $0 \leq \deg R_n < \deg R_{n-1} < \dots < \deg R_1$ и, следовательно, $n - 1 \leq \deg(R_1)$.

б. Подходящей является последовательность $F_0 = 1, F_1 = X + 1$ и $F_{n+2} = XF_{n+1} + F_n$.

43. Оптимальность алгоритма для $K[X]$ (теорема Лазара)

а. Ищем $\mu \leq \varphi$. Запишем оптимальное деление в виде $A = B \times Q + R$ и евклидово деление $A = B \times Q' + R'$ и $\deg(R') < \deg(B)$. Это дает $\mu(A, B) = \mu(B, R) + 1$ и $\varphi(A, B) = \varphi(B, R') + 1$.

1. Предположим, что $\deg(R) > \deg(B)$. Тогда из двух евклидовых делений $B = 0 \times R + B$ и $R = (Q' - Q) \times B + R'$ выводим:

$$\begin{aligned} \mu(A, B) &= \mu(B, R) + 1 = \varphi(B, R) + 1 && \text{(определение, затем рекуррентность).} \\ &= \varphi(R, B) + 2 = \varphi(B, R) + 3 && \text{(1-е, затем 2-е евклидово деление).} \\ &= \varphi(A, B) + 2 && \text{(по определению).} \end{aligned}$$

что противоречит $\mu(A, B) < \varphi(A, B)$.

2. Предположим, что $\deg(R) = \deg(B)$. Тогда $\deg(R') < \deg(R)$, откуда $\deg(R' - R) = \deg(R)$ и равенство:

$$B(Q - Q') = R' - R, \quad \deg(R' - R) = \deg(R) = \deg(B), \quad (13)$$

приводит к тому, что $Q - Q'$ постоянный (не нулевой). Если k обозначает $(1/Q - Q')$, то имеем евклидово деление B на R : $B = -kR + kR'$ с $\deg(kR') < \deg(B)$ откуда:

$$\begin{aligned} \mu(A, B) &= \mu(B, R) + 1 = \varphi(B, R) + 1 && \text{(определение,} \\ & && \text{затем рекуррентность).} \\ &= \varphi(R, kR') + 2 = \varphi(-kR, R') + 2 && \text{(евклидово деление(13),} \\ & && \text{затем вопрос а).} \\ &= \varphi(B, R') + 2 = \varphi(A, B) + 1 && (B \equiv -kR \pmod{R'}) \\ & && \text{затем определение).} \end{aligned}$$

что противоречит $\mu(A, B) \leq \varphi(A, B)$. Поэтому единственно возможный случай $\deg(R) < \deg(B)$, и, согласно единственности обычного частного Евклида, $R = R' \dots$

44. К теореме Лазара

Последовательность $(C_n)_{n \geq i}$ строго возрастающая и, следовательно, состоит из натуральных чисел; ее первые члены 0, 1, 3, 5, 18, 31, 111, 191... Нетрудно проверить неравенства:

$$C_{2i-1} < (1 - \alpha)C_{2i} < 0,5 \times C_{2i} \quad 0,5 \times C_{2i-1} < C_{(2i-2)} < \alpha C_{2i-1}$$

константа a введена в теореме Лазара ($\alpha \approx 0,6180339$ — обратное к золотому сечению).

Предположим, что первый элемент при центрированном делении имеет четный индекс, скажем C_{2n} . Ниже приведена последовательность центрированных делений C_{2n} на C_{2n-1} (записанных в виде $a = qb + r$):

$$\begin{aligned}
 C_{2n} &= 4C_{2n-1} + (C_{2n-2} - C_{2n-1}), \\
 C_{2n-1} &= (-2)(C_{2n-2} - C_{2n-1}) + C(2n-3), \\
 C_{2n-2} - C_{2n-1} &= (-3)C_{2n-3} + (C_{2n-3} - C_{2n-4}), \\
 C_{2n-3} - C_{2n-4} &= 3C_{2n-5} + (C_{2n-6} - C_{2n-5}), \\
 C_{2n-5} &= (-2)(C_{2n-6} - C_{2n-5}) + C(2n-7), \\
 &\vdots \\
 C_3 &= (-2)(-2) + 1 \text{ или } C_3 = 2 \cdot 3 + (-1), \\
 \pm 2 &= (\pm 2)(\pm 1) + 0
 \end{aligned}$$

(имеется точно $2n - 1$ делений). Можно задать алгоритм Евклида, требующий $2n - 1$ делений, из которых $n - 1$ нецентрированных (записанных в виде $a = qb + r$):

$$\begin{aligned}
 C_{2n} &= 3C_{2n-1} + C_{2n-2} && \text{(нецентрированное),} \\
 C_{2n-1} &= 2C_{2n-2} + (-C_{2n-3}) && \text{(центрированное),} \\
 C_{2n-2} &= (-3)(-C_{2n-3}) + C_{2n-4} && \text{(нецентрированное),} \\
 -C_{2n-4} &= (-2)C_{2n-4} + C_{2n-5} && \text{(центрированное),} \\
 C_{2n-4} &= 3C_{2n-5} + C_{2n-6} && \text{(нецентрированное),} \\
 &\vdots \\
 C_2 &= (\pm 3)(\pm C_1) + C_0 && C_0 = 0 \text{(центрированное,} \\
 &&& \text{остальное нулевое).}
 \end{aligned}$$

Используя теорему Лазара, легко проверить, что первая или вторая последовательности оптимальны по длине.

45. (Неэффективное) решето Эратосфена для многочленов

а. Пусть $Crible(2..n)$ — последовательность булевых переменных со значениями «истинно». Решето Эратосфена останавливается на очередном значении $iCribWe(i)$ истинным и вычеркивает все кратные $i * j$ числа $i(Crible(i * j) \leftarrow False)$, а затем повторяет эту процедуру для следующего i . Таким образом, простые числа от 2 до n в точности те i , для которых $Crible(i)$ истинно.

```

for(i = 2; i == n; i++){
  if (Cribble(i)){
    for(j = i; n == (n/i))
      ; j++){
      Cribble(i * j) ← False;
    }
  }
}

```

b. Пусть A — множество унитарных непостоянных многочленов над $\mathbb{Z}/p\mathbb{Z}$ и B — объединение $\bigcup_{k=1}^{\infty} \times [0, p^k]$. Построим биективное отображение A на B следующим образом: если $P(X) = X^k + a_{k-1}X^{k-1} + \dots + a_1X + a_0$ — многочлен из A ($k \geq 1$), то поставим ему в соответствие пару $(k, a_{k-1}p^{k-1} + \dots + a_1p + a_0)$ из B .

```

for(i = 0; i == Bn - 1; i++) {
  if(Crible(i)) {
    for(j = i; j == Bn - 1) {
      Crible( $\sigma^{-1}(\sigma(i) * \sigma(j))$ ) ← False;
    }
    exception if(Overflow \rightarrow null);
  }
}

```

Алгоритм 2.7: Решето Эратосфена

Отобразим теперь B на \mathbb{N} , сопоставляя паре $1 \times [0, p]$ отрезок $[0, p]$, паре $2 \times [0, p^2]$ отрезок $[p, p + p^2]$ и так далее $B_n = p + p^2 + \dots + p^n$, то построим следующую биекцию B на $\mathbb{N} = [B_0, B_1] \cup [B_1, B_2] \cup [B_2, B_3] \cup \dots$. Очевидно, B_n — число унитарных непостоянных многочленов степени $\leq n$. Пусть n — целое число, σ — взаимнооднозначное отображение интервала $[0, B_n]$ на множество унитарных непостоянных многочленов степени $\leq n$. Теперь получаем алгоритм 10, в котором $Q * R$, обозначает произведение многочленов Q и R (при помощи Overflow этот алгоритм игнорирует исключения, при которых $\deg(Q) + \deg(R) > n$). По окончании работы этого алгоритма определяются все неприводимые многочлены степени $\leq n$: это в точности те многочлены $\sigma(i)$, для которых Crible(i) истинно. Разумеется, этот алгоритм неэффективен по времени и объему! Кроме того, он позволяет получить список неприводимых многочленов по модулю 2 степени ≤ 10 или по модулю 3 степени ≤ 8 .

d. Ниже приведен краткий список результатов, которые должна давать программа, реализующая предшествующий алгоритм:

степень 2 :	$X^2 + X + 1,$	
степень 3 :	$X^3 + X + 1,$	$X^3 + X^2 + 1,$
степень 4 :	$X^4 + X + 1,$	$X^4 + X^3 + 1,$
	$X^4 + X^3 + X^2 + X + 1,$	

$$\begin{array}{ll}
\text{степень 5 : } & X^5 + X^2 + 1, & X^5 + X^3 + 1, \\
& X^5 + X^3 + X^2 + X + 1, & X^5 + X^4 + X^2 + X + 1, \\
& X^5 + X^4 + X^3 + X + 1, & X^5 + X^5 + X^4 + X^2 + 1, \\
\text{степень 6 : } & X^6 + X + 1, & X^6 + X^3 + 1, \\
& X^6 + X^4 + X^2 + X + 1, & \\
& X^6 + X^4 + X^3 + X + 1, & X^6 + X^5 + 1, \\
& X^6 + X^5 + X^2 + X + 1, & \\
& X^6 + X^5 + X^3 + X^2 + 1, & X^6 + X^5 + X^4 + X + 1, \\
& X^6 + X^5 + X^5 + X^2 + 1. &
\end{array}$$

46. Многочлены $X^n - 1$

а. Равенства

$$X^{un+vm} - 1 = \begin{cases} (X^{un} - 1) - X^{un+um}(X^{-vm-1}), & u \geq 0, v \leq 0, \\ (X^{un} - 1)X^{vm} + (X^{um} - 1), & y \geq 0, v \geq 0, \end{cases}$$

доказывают, что $X^d - 1$ — линейная комбинация $X^n - 1$ и $X^m - 1$. Для доказательства первой части упражнения заметим, что d положительно и меньше n и m , а поэтому $d = un + vm$ с, например, $u \geq 0$, $v \leq 0$. Наконец, d делит n и m , а потому $X^d - 1$ делит $X^n - 1$ и $X^m - 1$. Чтобы перейти к произвольному кольцу, достаточно повторить для него предыдущие рассуждения.

б. Конечно, нет: рассмотреть $f(X) = X$ и $g(X) = X + 2$.

с. $X^n - 1 \bmod X^m - 1 = X^{n \bmod m} - 1$.

47. Неприводимые многочлены над \mathbb{Z}

Для P_k осуществим замену переменных $X \rightarrow X + 1$ и получим $P_k(X + 1) = 2 + \sum_{i=1}^{2^k} \binom{2^k}{i}$. Теперь можно применить критерий Эйзенштейна. Заметим, что P_k это циклотомический многочлен $\Psi_{2^{k+1}}(X)$; в самом деле, для корня α многочлена P_k проверяется, что $\alpha^{2^k} = -1$, $\alpha^{2^{k+1}} = 1$. Значения $P_k(2)$ — это числа Ферма, числа, из которых только 5 считаются простыми.

Для Q_n можно также применить критерий Эйзенштейна или доказать неприводимость непосредственно (что сводится к изменению доказательства критерия в данном конкретном случае). Предположим, что

$$Q_n = (a_p X^p + \dots + a_0)(b_r X^r + \dots + b_0).$$

Тогда a_0 или b_0 равно 2; допустим, что это a_0 . Теперь без труда доказывается (используя то, что разложение нетривиально), что все a_i , четные,

а это противоречит тому, что коэффициент при старшем члене много члена Q_n есть 1. Замечание, аналогичное предыдущему: многочлены $X_n - 1$ не являются неприводимыми, хотя самые большие известные простые числа — числа вида $2^n - 1$ (числа Мерсенна).

48. Оценка числа неприводимых множителей

а. Запишем $X^n - a = (b_q X^q + \dots + b_0)(c_r X^r + \dots + c_0)q > 0$, $r > 0$ Нетрудно показать, что p делит все b_i или все c_i , а это противоречит тому, что коэффициент при старшем члене у произведения равен 1 (можно также применить критерий Эйзенштейна).

б. Пусть $(Q = b_q X^q + \dots + b_0)q < n$ — делитель P . $A/I[X]$ — область целостности (так как I — простой) и в нем

$$a_n X^n = (b_q x^q + \dots + b_0)(c_r X^r + \dots + c_k X^k),$$

с $k < n$ и $c_k \neq 0$. Однако из $a_k = 0 = b_0 c_k$ следует, что $b_0 = 0$ т.е. в A $b_0 \in I$.

б. Применим результаты предыдущего пункта с $I = (p)$: всякий делитель P имеет свободный член, кратный p , и, следовательно, если g — число неприводимых сомножителей P , то $p^g | a_0$

49. Неприводимый многочлен, приводимый по модулю всякого простого числа

а. Для $p = 2$ можно проверить, что $X^4 + 1 = (X + 1)^4$ в $\mathbb{Z}/2\mathbb{Z}[X]$

б. Если $p \equiv (\text{mod } 4)$ то -1 есть квадрат по модулю p и

$$X^4 + 1 = (X^2 + \sqrt{-1})(X^2 - \sqrt{-1}), \quad \text{в } \mathbb{Z}/p\mathbb{Z}[X]$$

$\sqrt{-1}$ обозначает квадратный корень из -1 по модулю p . Также, если 2 — квадрат (квадратичный вычет) по модулю p , то $X^4 + 1 = (X^2 + \sqrt{2}X + 1)(X^2 - \sqrt{2}X + 1)$ и, конечно, если -2 квадратичный вычет по модулю p , то $X^4 + 1 = (X^2 + \sqrt{2}X - 1)(X^2 - \sqrt{2}X - 1)$ Однако одно из трех чисел $-1, 2, -2$, наверняка, квадрат по модулю p : действительно, если -1 и 2 не-квадраты, то таковым является $-2 = -1 \times 2$. Которое из этих чисел квадрат, можно узнать с помощью следующих критериев:

$$-1 \text{ квадрат} \Leftrightarrow p \equiv 1(4), 2 \text{ квадрат} \Leftrightarrow p \equiv \pm 1(8),$$

$$-2 \text{ квадрат} \Leftrightarrow p \equiv \pm 1(4)$$

Приводим несколько примеров:

$$X^4 + 1 = (X^2 - 2)(X^2 + 2) \pmod{5},$$

$$X^4 + 1 = (X^2 + 5X + 1)(X^2 - 5X + 1) \pmod{23},$$

$$X^4 + 1 = (X^2 + 3X + 1)(X^2 - 3X + 1) \pmod{11},$$

d. Сравнение $-1 \equiv 7^2 \pmod{25}$ дает разложение:

$$X^4 + 1 = (X^2 + 7) \times (X^2 - 7) \pmod{25}.$$

Если рассмотреть разложение $X^4 + 1$ по модулю 15 в произведение двух унитарных непостоянных многочленов $X^4 + 1 = P(X)Q(X) \pmod{15}$, то легко проверить, что

$$X^4 + 1 = (X^2 + 2)(X^2 - 2)Z/5Z[X],$$

$$X^4 + 1 = (X^2 + X - 1)(X^2 - X - 1)Z/3Z[X],$$

последние — разложения на неприводимые элементы, а значит, имеем:

$$P(X) \equiv X^2 + 2 \pmod{5}, Q(X) \equiv X^2 - 2 \pmod{5}$$

и

$$P(X) \equiv X^2 + X + 1 \pmod{3}, Q(X) \equiv X^2 - X + 1 \pmod{3} \quad (14)$$

или

$$P(X) \equiv X^2 - X + 1 \pmod{3}, Q(X) \equiv X^2 + X + 1 \pmod{3} \quad (15)$$

Между тем, многочлены $P_0(X) = X^2 + 10X + 7Q_0(X) = X^2 + 5X + 13$ удовлетворяют (14), откуда в силу единственности $P(X) \equiv P_0(X) \pmod{15}$ и $Q(X) \equiv Q_0(X) \pmod{15}$. Аналогично, многочлены $P_1(X) = X^2 + 5X + 7$ и $Q_1(X) = X^2 + 10X + 13$ удовлетворяют (15) и, следовательно, $P(X) \equiv P_1(X) \pmod{15}$ и $Q(X) \equiv Q_1(X) \pmod{15}$. Однако, можно проверить, что ни P_0Q_0 ни P_1Q_1 не равны $X^4 + 1$ по модулю 15...

50. Нули многочленов с целыми коэффициентами

a. Запишем $0 = a_0 + \sum_{i=1}^{n-1} a_i(p/q)^i + (p/q)^n$. Освобождаясь от знаменателя, получаем $0 = p^n + \sum_{i=1}^{n-1} a_i p^i q^{n-i} + a_0 q^n$. Отсюда получаем сначала, что q делит p , что приводит к $q = 11$, а потом, что p делит a_0 . **b.** Как и ранее, приходим к выражению
$$0 = a_n p^n + \sum_{i=1}^{n-1} a_i p^i q^{n-i} + a_0 q^n$$
 что доказывает (p и q взаимно просты), что $p \mid a_0 q \mid a_n$. Это позволяет “находить” все рациональные корни многочлена, так как число делителей $a_0 a_n$ конечно. При необходимости это позволяет частично обосновать примитивность P .

51. Расширенный критерий Эйзенштейна

а. а. Рассмотрим разложение P в произведение неприводимых много членов. Так как $p|_0 p^2|0$, то он делит свободный член лишь одного из неприводимых факторов, скажем $Q = \sum b_i X^i$. $BA/2(p)[X]$ имеем

$$a_n X^n + \dots + a_k X^k = (b_q X^q + \dots + b_0)(c_r X^r + \dots + c_0),$$

и, по только что доказанному, $b_0 = 0, b_0 \neq 0$. Раскрывая скобки и сравнивая, без труда получим, что все коэффициенты Q нулевые, за исключением коэффициента при k -й степени X . Следовательно, степень Q не меньше k . Следует также отметить, что свободный член рассматриваемого неприводимого многочлена делится на p

Когда k равно n , степени многочлена P , получаем критерий Эйзенштейна

б. Если P разложим, то он может быть записан в одной из форм (так как ни 5, ни -5 не являются его корнями): $P = X^4 + 3X^3 + 3X^2 - 5 = (X^2 + aX \pm 5)(X^2 + bx \mp 1)$ Это приводит к системе уравнений $a + b = 3$ и $-5b = 0$, которая не имеет целых решений. Следовательно, P неприводим.

Что касается многочлена Q , то в зависимости от четности a возможны два случая. Если a четно, то сразу можно применить критерий Эйзенштейна и Q неприводим. Если a нечетно и Q разложим, то один из его неприводимых множителей имеет степень, не меньшую 2, и его свободный член делится на 2.

Если Q имеет целый корень, то, следовательно, это может быть только 1 или -1 ; действительно, при $= -3$ имеем $Q = (X - 1)(5X^3 - X^2 + 2X - 2)$, а при $= 17$ $Q = (bX^2 + cX \pm 1)(dX^2 + eX^2 \pm 1)$ с четными c и e , откуда достаточно быстро приходим к противоречию.

52. Неприводимость циклотомических многочленов

а. Запишем $X^n - 1 = PQ$ и предположим, что $P(\alpha^p) \neq 0$ Тогда $Q(\alpha^p) = 0$, и, следовательно, многочлен $Q(X^p)$ имеет корень α . Тогда α — корень НОД($Q(X^p)$). Но P неприводим, а значит $P|Q(X^p)$.

Перейдем к $Z/pZ[X]$. В этом кольце имеем формальное тождество $Q(X^p) = Q(X)^p$. Если π — неприводимый делитель P , то π также не приводимый делитель Q , а значит, π^2 делит $X^n - 1Z/pZ[X]$, что невозможно, так как $X^n - 1$ не имеет кратных множителей даже в $Z/pZ[X]$

(рассмотреть для доказательства производную $X^n - 1$ и учесть, что $p|n$). Полученное противоречие показывает, что α^p корень P .

б. Пусть P — неприводимый множитель φ_n . Тогда он обращается в нуль на некотором примитивном корне из 1 степени n , а значит, и на всех остальных тоже. Действительно, пусть α — примитивный корень, тогда все остальные примитивные корни имеют вид α^k , $c(k, n) = 1$. Следовательно, $P = \Phi_n$.

53. Идеалы в $K[[X]]$

Всякий формальный ряд, свободный член которого ненулевой, обратим и, следовательно, порождает $K[[X]]$. Свободные идеалы $K[[X]]$ состоят из формальных рядов с нулевым свободным членом.

Пусть I идеал в $K[[X]]$ и $f \in I$ многочлен с наименьшим среди многочленов I показателем n таким, что $f = X^n \sum a_i X^i c a_0 \neq 0$. Тогда $f \in (X^n)$ и, соответственно, $X^n \in I$. Следовательно, $I = (X^n)$. Итак, идеал из $K[[X]]$ исчерпываются идеалами $0, (X), (X^2), \dots, (X^n), \dots$ и $K[[X]]$.

54. О факториальных кольцах

а. а. Отображение φ котором идет речь, с одной стороны, биективно, а с другой стороны, является морфизмом (моноидов): $\varphi(\epsilon\epsilon', v + v') = \varphi(\epsilon, v)\varphi(\epsilon', v')$. В частности, любой элемент $a \in A \setminus 0$ единственным образом записывается в виде $\epsilon(a) \prod_{p \in P} p^{v_p(a)}$ и $a|b$ эквивалентно тому, что $v_p(a) \leq v_p(b)$ для всякого $p \in P$.

Любая теория делимости в A может быть сведена к вычислениям в \mathbb{N}^P . Например, проверим, что $d = \prod_{p \in P} p^{\min(v_p(a), v_p(b))}$ является НОД (a, b) . Действительно, если заданы $v, w \in \mathbb{N}^P$ то множество $U = \{ \min(v_p, w_p) \}_{p \in P}$ — наибольшее со свойством $u_p \leq v_p, u_p \leq w_p$ для любого $p \in P$. Поэтому и множество главных идеалов нётерово по включению, так как $a \subset b$ эквивалентно $v(a) \geq v(b)$, а \mathbb{N}^P нётерово по отношению к упорядочению \geq (отображение, которое каждому $v \in \mathbb{N}^P$ ставит в соответствие $\sum_p v_p p \in A$ строго возрастает).

Отметим, что отображение φ устанавливает изоморфизм группы $U(A) \times \mathbb{Z}^P K^* (K — \text{поле частных } A)$.

б. Данные свойства делимости в факториальных кольцах сводятся к вычислениям в \mathbb{N}^P . Например, $c(a, b) = (ca, cb)$ сводится к проверке равенства $\gamma + \min(\alpha, \beta) = \min(\gamma + \alpha, \gamma + \beta)$, которое очевидно. В кольцах главных идеалов эти свойства можно проверять, опираясь на соотношения Везу: например, если $Ad = +$, то $Acd = + \dots$

с. Единственная нетривиальная импликация заключается в доказательстве того, что если два элемента имеют НОД, то любой неприводимый элемент p прост: если a и b такие, что $p|ab$, то нужно доказать, что p делит a или b . Рассмотрим для этого НОД d элементов ap и ab (НОД существует по предположению). Элементы p и a делят ap и ab и, следовательно, $p|d$ и $a|d$. Поэтому элемент d/a определен и делит p (ибо $d|ap$). Так как p неприводим, то $1|d/a$, а тогда $d \Rightarrow \Rightarrow |$ или $p|d/a$ и в этом случае $d \Rightarrow |ab \Rightarrow |$.

55. Условия, при которых факториальное кольцо является кольцом главных идеалов

а. Пусть P — простой идеал A . Так как нужно доказать, что P главный, то можно считать, что $P \neq 0$ и $P \neq A$. Пусть $x \in -0, x$ необратим и записывается в виде произведения неприводимых $x = p_1 \dots p_k$. Идеал P простой, а поэтому содержит один из $p_i : (p_i) \subset P$. В силу максимальности идеала (p_i) получаем необходимое утверждение.

Докажем, $Ax/d + Ay/d = A$. Предположим, что $Ax/d + Ay/d \subsetneq A$, тогда идеал $Ax/d + Ay/d$ содержится в некотором максимальном идеале (теорема Крулля), а следовательно, в простом и главном, скажем P . Тогда элемент p делит как x/d , так и y/d , которые взаимно просты — противоречие! Значит $Ax/d + Ay/d = A$ и $ax + ay = Ad$.

Теперь ясно, что всякий конечно порожденный идеал — главный. Но так как A факториально, то множество главных идеалов A нётерово. Кольцо, в котором множество конечно порожденных идеалов нётерово (по включению), само нётерово (наличие неконечнопорожденного идеала позволяет построить строго возрастающую последовательность конечно порожденных идеалов). Так как всякий конечно порожденный идеал A главный, то A — кольцо главных идеалов.

б. Пусть x и y — два элемента A . Если $d = (x, y)$, то x/d и y/d взаимно просты, и следовательно, $1 \in Ax/d + Ay/d$, откуда $ax + ay = Ad$. Отсюда следует искомое утверждение (действительно, если выполняется пункт (b), то выполняется и (a)).

с. Для доказательства рассмотрим следующие умножения $y \mapsto yx$: эти отображения инъективны при $x \neq 0$ и биективны в силу конечности. Рассмотрим последний случай (самый сложный). Пусть $x \in R \setminus 0$, по предположению $K[x] \xrightarrow{\cdot x} K[x]$ инъективно, но это отображение K -линейно и $K[x]$ — конечномерно (так как x алгебраичен над K), а, стало быть, сюръективно, и существует такой элемент x' , что

$xx' = 1$ Аналогично, существует x'' с $x''x = 1$, но известно, что в таком случае $x' = x''$. Так как всякий элемент x обратим в R , то R — поле

d. Можно считать, что A не является полем. Идеал $I \cap \mathbb{Z}$ — простой идеал в \mathbb{Z} , и осталось доказать, что он ненулевой. I содержит ненулевой элемент x , и x является корнем унитарного многочлена $\sum a_i X^i$ с целыми коэффициентами, причем можно считать, что его степень не ниже n . Тогда $a_0 \neq 0$ (иначе после вынесения x получили бы унитарный многочлен с целыми коэффициентами степени $< n$), и поэтому $a_0 = -\sum_{i=1}^n a_i x^i \in I \cap \mathbb{Z}$. Значит, идеал $p\mathbb{Z}$ ненулевой и имеет вид $p\mathbb{Z}$. Инъективное отображение $\mathbb{Z} \rightarrow A/I$ определяет инъекцию $\mathbb{Z}/I \cap \mathbb{Z} \rightarrow A/I$ и алгебра A/I является алгеброй над полем $\mathbb{Z}/p\mathbb{Z}$, а так как она без делителей нуля и алгебраическая над $\mathbb{Z}/p\mathbb{Z}$, то является полем по предыдущему пункту. Тогда I — максимальный идеал.

56. Произведение упорядоченных нётеровых множеств

a. Достаточно это сделать для двух упорядоченных множеств, что не вызывает особых проблем.

b. Если $\chi_A = 0, 1$, то упорядоченное множество $E^{\mathbb{N}}$ изоморфно множеству всех подмножеств \mathbb{N} с отношением вложения (при помощи характеристической функции: χ_A, χ_B тогда и только тогда, когда $A \subset B$). Рассматриваемое подпространство S образовано конечными подмножествами или конечными дополнениями и, очевидно, не нётерово: $0 \subset 0 \subset 0, 1 \subset 0, 1, 2 \subset \dots$. В общем случае для сравнимых между собой $a, b \in E$ $a < b$, например, имеется строго возрастающая последовательность в подпространстве S :

$$(a, a, a, a, a, a, \dots) < (a, b, a, a, a, a, \dots) < (a, b, b, a, a, a, \dots) < (a, b, b, b, a, a, \dots) < \dots$$

которая доказывает, что S не нётерово, а $E^{\mathbb{N}}$ тем более.

c. Доказательство аналогично доказательству, касающемуся $A[X]$ (стр. 233).

d. Если M — упорядоченное по включению множество идеалов из M , то, как показано в курсе, имеется строго возрастающее отображение $M \times E_N \rightarrow E_{M/N}$. Впрочем, отображение, которое идеалу I кольца $A[X]$ ставит в соответствие $\text{dom}_n(I)_{n \in \mathbb{N}} \in E_A^{\mathbb{N}}$, — строго возрастающее отображение со значениями в множестве возрастающих последовательностей идеалов.

57. Конечные (коммутативные) поля

а. Для доказательства первой части использовать то, что факторкольцами \mathbb{Z} без делителей нуля являются лишь $\mathbb{Z}/p\mathbb{Z}$, где p — простое, и само \mathbb{Z} . Для доказательства второй части заметим, что включение $K \subset K'$ наделяет K' структурой векторного пространства над K (а значит, $|K'| = |K| \dim_K^{K'}$). Коммутативность не использовалась.

б. Отметим сначала, что корни многочлена $X^q - X$ простые, так как его производная -1 ! Для первой части, если K поле из q элементов, то K^* — мультипликативная группа, состоящая из $q-1$ элементов, откуда $x^{q-1} = 1$ для всякого $x \in K^*$, а стало быть, $x^q = x$ для любого $x \in K$. Иначе говоря, K содержится среди корней многочлена $X^q - X$ и, следовательно, совпадает с этим множеством из соображений порядка. Все корни многочлена $X^q - X$ лежат в K , а следовательно, и в Ω . Это доказывает единственность в Ω поля из q элементов.

Обратно, предположим, что корни $X^q - X$ содержатся в Ω . Так как они различны, то их точно q . Обозначим через $\sigma : \Omega \rightarrow \Omega$ отображение, задаваемое как $\sigma(x) = x^p$. Так как $(x+y)^p = x^p + y^p$, то σ — изоморфизм Ω на себя. Если $q = p^n$, то отображение $r : x \mapsto x^q$, равное σ^n , также изоморфизм. Отсюда следует, что \mathbb{F}_q , подполе ω с q элементами.

с. Если $K \subset H \subset K'$, $|H|$ — степень числа $|K|$, $|K'|$ — степень числа $|H|$, а значит, H — поле из q^d элементов с $d|n$. Обратно, если $d|n$, то все корни многочлена $X^{q^d} - X$ лежат в K' (поскольку он делит многочлен $X^{q^n} - X$, чьи корни образуют K') и, следовательно, $x \in K' | x^{q^d} = x$ — промежуточное поле из q^d элементов.

Отображение, которое делителю d числа n ставит в соответствие поле $\mathbb{F}_{q^d} = \{x \in K' | x^{q^d} = x\}$, является изоморфизмом упорядоченных множеств в том смысле, что $d_1 | d_2$ эквивалентно $\mathbb{F}_{q^{d_1}} \subset \mathbb{F}_{q^{d_2}}$.

Обозначим через τ изоморфизм $x \mapsto x^q$. Тогда:

$$\tau^s(x) = x\tau^r(x) = x \iff \tau^{(s,r)}(x) = x$$

(рассмотреть соотношение Безу между $(s, r), r, s$). Этого достаточно, чтобы ответить на последний вопрос.

д. Так как $|K'| = p^n$, то многочлен $X^{p^n} - X$ обнуляется на K' и можно рассмотреть многочлен $Q = (X^{p^n} - X)/P$ (по теореме 35 многочлен P целит: $X^{p^n} - X$). Так как $\deg Q < p^n$, то существует $y \in K'$ такой, что $Q(y) \neq 0$, и тогда y — корень P . Оставшаяся часть вопроса затруднений не представляет.

е. Ясно, что $r^n = \text{Id}_{K'}$ обратно, если $r^m = \text{Id}_{K'}$, то множество K' порядка q^n содержится в множестве корней многочлена $X^{q^m} - X$ и,

значит, $n \leq m$. Так как K' изоморфно полю $\mathbb{F}_p[X]/(Q)$, то существует $x \in K'$, который образует K' над \mathbb{F}_p , а тем более над K , т.е. $K' = K[x]$. Элементы $x, \tau(x), \dots, \tau^{n-1}(x)$ — это все различные корни минимального многочлена P элемента x над K . Автоморфизм σ , постоянный над K , не изменяет и коэффициенты P , а следовательно, переводит x , корень P , в другой корень $\tau^i(x)$. Так как морфизмы τ^i и σ совпадают на x , то они совпадают всюду.

Уравнение $x^{p^k} = x$ в локальном кольце характеристик и p

а. Существуют такие y и z , что $1 - xy \in I$, $1 - xz \in J$. Тогда $(1 - xy)(1 - xz) = 1 - x(y + z - xyz) \in IJ$, что доказывает обратимость x по модулю IJ .

б. Перепишем уравнение $x(x^{q-1} - 1) \equiv 0 \pmod{I^n}$. Если $x \in I$, то $x^{q-1} \in I$ и $x^{q-1} \notin I$ (в противном случае сумма x^{q-1} и $1 - x^{q-1}$, равная 1, принадлежала бы I). Так как I максимален, то $1 - x^{q-1}$ обратим по модулю I , а, следовательно, и по модулю I^n и $x \equiv 0 \pmod{I^n}$. Напротив, если $x \notin I$, то x обратим по модулю I и I^n , а значит, $x^{q-1} \equiv 1 \pmod{I^n}$.

с. Так как q — степень характеристики p , то отображение $x \rightarrow x^q$ есть морфизм алгебр и, следовательно, S_1 и S_n — подалгебры, очевидно удовлетворяющие $\pi(S_n) \subset S_1$. Осталось доказать, что если x является решением $X^q \equiv X \pmod{I^n}$, удовлетворяющим $x \equiv 0 \pmod{I}$, то $x \equiv 0 \pmod{I^n}$. Но это следует из пункта **б**.

д. Так как $a \equiv 0 \pmod{I}$, то существует такое i , что для $j > i$ $a^{q^j} \equiv 0 \pmod{I^n}$. Положим $s = a + a^q + a^{q^2} + \dots + a^{q^i}$. Ясно, что $s \in I$, а кроме того, $s^q = a^q + a^{q^2} + \dots + a^{q^{i+1}}$ откуда $s^q = s - a + a^{q^{i+1}} \equiv s - a \pmod{I^n}$.

Доказательство того, что $\pi(S_n) = S_1$ сводится к доказательству того, что решение $x^q \equiv x \pmod{I}$ “продолжается” до решения y по модулю I^n , т.е. существует $x \equiv y \pmod{I}$ и $y^q \equiv y \pmod{I^n}$. Приме ним предыдущее рассуждение при $a = x^q - x$ (которое лежит в I) и получим элемент $a \in I$ такой, что $s - s^q \equiv x^q - x \pmod{I^n}$, но это можно переписать в виде $x + s \equiv (x + s)^q \pmod{I^n}$ и взять $x = x + s$.

е. Если $\pi : A/I^n \rightarrow A/I$, то $\text{Ker } \pi = I/I^n$ — максимальный идеал в A/I^n если J — максимальный идеал в A , содержащий I^n , то он содержит I и, следовательно, $J = I$. Поэтому I/I^n — единственный максимальный идеал в A/I^n . Используя то, что A/I^n — локальное кольцо, легко доказать, что $x^q = x$ в A/I^n эквивалентно $x = 0$ или $x^{q-1} = 1$. Что бы получить конечный результат, нужно предположить, что некоторая степень M равна нулю (в случае, когда $M = I/I^n$, имеем $M = 0$).