

ОГЛАВЛЕНИЕ

ГЛАВА 1. ОСНОВНЫЕ ПОНЯТИЯ	27
1.1. АЛГОРИТМЫ	27
1.2. МАТЕМАТИЧЕСКОЕ ВВЕДЕНИЕ	37
1.2.1. Математическая индукция	38
1.2.2. Числа, степени и логарифмы	49
1.2.3. Суммы и произведения	56
1.2.4. Целочисленные функции и элементарная теория чисел	68
1.2.5. Перестановки и факториалы	75
1.2.6. Биномиальные коэффициенты	82
1.2.7. Гармонические числа	105
1.2.8. Числа Фибоначчи	109
1.2.9. Производящие функции	118
1.2.10. Анализ алгоритма	127
*1.2.11. Асимптотические представления	138
*1.2.11.1. Символ O	138
*1.2.11.2. Формула суммирования Эйлера	143
*1.2.11.3. Применение асимптотических формул	148
1.3. MIX	156
1.3.1. Описание MIX	156
1.3.2. Язык ассемблера компьютера MIX	178
1.3.3. Применение к перестановкам	198
1.4. НЕКОТОРЫЕ ФУНДАМЕНТАЛЬНЫЕ МЕТОДЫ ПРОГРАММИРОВАНИЯ	221
1.4.1. Подпрограммы	221
1.4.2. Сопрограммы	229
1.4.3. Программы-интерпретаторы	237
1.4.3.1. Имитатор MIX	239
*1.4.3.2. Программы трассировки	248
1.4.4. Ввод и вывод	251
1.4.5. История и библиография	266
ГЛАВА 2. ИНФОРМАЦИОННЫЕ СТРУКТУРЫ	271
2.1. ВВЕДЕНИЕ	271
2.2. ЛИНЕЙНЫЕ СПИСКИ	277
2.2.1. Стеки, очереди и деки	277
2.2.2. Последовательное распределение	283
2.2.3. Связанное распределение	295

2.2.4. Циклические списки	315
2.2.5. Дважды связанные списки	322
2.2.6. Массивы и ортогональные списки	341
2.3. ДЕРЕВЬЯ	352
2.3.1. Обход бинарных деревьев	362
2.3.2. Представление деревьев в виде бинарных деревьев	380
2.3.3. Другие представления деревьев	395
2.3.4. Основные математические свойства деревьев	410
2.3.4.1. Свободные деревья	410
2.3.4.2. Ориентированные деревья	420
*2.3.4.3. Лемма о бесконечном дереве	431
*2.3.4.4. Перечисление деревьев	435
2.3.4.5. Длина пути	449
*2.3.4.6. История и библиография	456
2.3.5. Списки и “сборка мусора”	459
2.4. МНОГОСВЯЗНЫЕ СТРУКТУРЫ	476
2.5. ДИНАМИЧЕСКОЕ ВЫДЕЛЕНИЕ ПАМЯТИ	488
2.6. ИСТОРИЯ И БИБЛИОГРАФИЯ	512
ОТВЕТЫ К УПРАЖНЕНИЯМ	521
ПРИЛОЖЕНИЕ А. ТАБЛИЦЫ ЗНАЧЕНИЙ НЕКОТОРЫХ КОНСТАНТ	683
A.1. Основные константы (десятичные)	683
A.2. Основные константы (восьмеричные)	684
A.3. Значения гармонических чисел, чисел Бернулли и чисел Фибоначчи	685
ПРИЛОЖЕНИЕ Б. ОСНОВНЫЕ ОБОЗНАЧЕНИЯ	687
ПРЕДМЕТНО-ИМЕННОЙ УКАЗАТЕЛЬ	692

От издателей русского перевода

На мировом рынке компьютерной литературы существует множество книг, предназначенных для обучения основным алгоритмам и используемых при программировании. Их довольно много, и они в значительной степени конкурируют между собой. Однако среди них есть особая книга. Это трехтомник “Искусство программирования” Д. Э. Кнута, который стоит вне всякой конкуренции, входит в золотой фонд мировой литературы по информатике и является настольной книгой практически для всех, кто связан с программированием.

Мы как издатели видим ценность книги в том, что она предназначена не столько для обучения технике программирования, сколько для обучения, если это возможно, “искусству” программирования, предлагает массу рецептов усовершенствования программ и, что самое главное, учит самостоятельно находить эти рецепты.

Ни для кого не секрет, что наши программисты являются одними из наиболее высококвалифицированных специалистов в мире. Они достойно представляют за рубежом отечественную школу программирования и информатики, которая внесла значительный вклад в формирование фундаментальных основ компьютерных наук. Для сохранения такого уровня и продвижения вперед необходимо своевременное издание на русском языке книг, отражающих основные мировые достижения в этой области. Трехтомник “Искусство программирования” Д. Э. Кнута — одна из таких книг.

Мы гордимся тем, что библиотеки программистов, преподавателей, студентов, старшеклассников и многих других пополнятся этой классической книгой и что тем самым мы внесем свой вклад в формирование более глубокого понимания основ компьютерных наук. Мы глубоко убеждены, что книга “Искусство программирования” Д. Э. Кнута способна приблизить человека к совершенству. Надеемся, наше издание на русском языке этой замечательной книги еще раз подтвердит, что истинные ценности с годами не устаревают.

— Виктор Штонда, Геннадий Петриковец, Алексей Орлович,
издатели

О книге “Искусство программирования”

У каждой книги своя судьба. Одни появляются незаметно и так же незаметно исчезают в потоке времени, покрываясь пылью на полках библиотек. Другие в определенный период пользуются спросом у узкого круга специалистов, пока им на смену не приходят новые справочники. Третьи, поднимаясь над временем, оказывают мощное влияние на технологическое развитие общества. Книг, относящихся к последней категории, не так уж и много. Их выход в свет — всегда праздник. Проходят годы, изменяются технологии, но новые поколения с постоянным интересом перечитывают их страницы. Именно к таким книгам относится предлагаемый читателю многотомный труд известного американского ученого Дональда Эрвина Кнута “Искусство программирования”.

Прошло почти 30 лет со времени первого издания в 1972 году в США этой книги. Она была переведена на большинство языков мира, в том числе и на русский. К настоящему времени на территории стран СНГ трехтомник Д. Э. Кнута стал библиографической редкостью. В 1998 году в США вышло третье издание “Искусства программирования”. В нем сохранена последовательность изложения материала прежних версий, но значительно расширен список ссылок, в который включены свежие и наиболее важные результаты, добавлены новые упражнения и комментарии, устранены неточности. Учитывая популярность во всем мире “Искусства программирования”, давно следовало ожидать появления нового переводного издания на русском языке, которое вы и держите в руках.

В чем же успех “Искусства программирования” Д. Э. Кнута?

Во-первых, эта книга — великолепное учебное пособие по составлению и анализу компьютерных алгоритмов. Ее разделы могут быть включены во многие университетские курсы по технологиям программирования, теории алгоритмов, дискретной математике. Книгу могут изучать и школьники старших классов, знакомые с основами программирования. В качестве основного языка записи алгоритмов автор выбрал язык машинных команд гипотетического универсального компьютера MIX. Это позволяет строить оптимальные программы с учетом особенностей вычислительных машин. Перенести MIX-программы на реальные ЭВМ или переписать их на языках высокого уровня не составляет особого труда. Логика работы программ почти всегда поясняется простыми блок-схемами.

Во-вторых, тщательно подобранный материал, вошедший в книгу, включает в себя основные фундаментальные классы алгоритмов, которые в том или ином виде наиболее часто встречаются в практике программирования.

В-третьих, немаловажным фактором успеха книги Д. Э. Кнута является энциклопедичность изложения. Профессор Кнут отличается уникальной способностью отслеживать проблему от исторических предпосылок ее зарождения до современного состояния. Многочисленные ссылки на работы старых мастеров (вплоть до времен античности), заключенные в современный контекст, создают у читателя особое чувство причастности к историческому развитию научных идей и методов.

В-четвертых, следует отметить мастерство изложения. Книга рассчитана на широкий круг читателей — от начинающих студентов до программистов-профессионалов. Каждому будет интересно изучать компьютерные алгоритмы на своем уровне. Материал

самодостаточен. Для понимания сути методов не требуется знания особых разделов математики или специальных технологий программирования. Прослеживается определенная “музыкальная” композиция сюжетного построения (дома у Д. Э. Кнута есть небольшой орган, на котором он играет).

Список составляющих успеха “Искусства программирования” можно легко продолжить.

Автор этих строк прослушал курс “Искусство программирования” в изложении профессора Кнута в 1976–1977 годах во время стажировки в Станфордском университете. Тогда формировалась алгоритмическая основа технологий программирования, у истоков которой стоял Д. Э. Кнут. Было много обсуждений, семинаров, творческих замыслов.

Значительные книги всегда связаны с судьбой автора. Дональд Эрвин Кнут начал работу над “Искусством программирования” в 1962 году. Продолжает ее и сейчас. У него много планов. Впереди новые тома “Искусства программирования”, которых с нетерпением ждут читатели.

— Профессор Анатолий Анисимов

От редактора перевода

Со времени первого издания книги “Искусство программирования” Д. Э. Кнута прошло около 25 лет. Тем не менее книга не только не устарела, но по-прежнему остается основным руководством по искусству программирования, книгой, по которой учатся понимать суть и особенности этого искусства.

За эти годы на английском языке вышло уже третье издание 1-го и 2-го томов, а также второе издание 3-го тома. Автор внес в них значительные изменения и существенные дополнения. Достаточно сказать, что число упражнений практически удвоилось, а многие упражнения, включенные в предыдущие издания (особенно ответы к ним), модифицированы. Существенно дополнены и переделаны многие главы и разделы, исправлены неточности и опечатки, добавлены многочисленные новые ссылки на литературу, использованы теоретические результаты последних лет.

Значительно преобразилась глава 3, особенно разделы 3.5 и 3.6, а также разделы 4.5.2, 4.7, 5.1.4, 5.3, 5.4.9, 6.2.2, 6.4, 6.5 и др.

Естественно, возникла необходимость в новом издании книги.

Перевод выполнен по третьему изданию 1-го и 2-го томов и второму изданию 3-го тома. Кроме того, учтены дополнения и исправления, любезно предоставленные автором.

При переводе мы старались сохранить стиль автора, обозначения и манеру изложения материала. В большинстве случаев использовались термины, принятые в научной литературе на русском языке. При необходимости приводились английские эквиваленты. По многим причинам, в частности из-за сложности некоторых разделов, читать книгу “Искусство программирования” далеко непросто. Одной из причин, которые затрудняют понимание книги, является манера изложения автора; привыкнув к ней, можно существенно облегчить чтение.

Из-за обилия материала (часто мало связанного между собой) невозможно построить книгу так, чтобы различные понятия и определения вводились сразу же при первом упоминании о них. Поэтому в главе 1 могут обсуждаться без ссылок понятия, строгие определения которых приводятся в 3-м томе. Именно поэтому так велика роль предметного указателя, без которого понимание книги было бы существенно затруднено. Надеемся, что читатель не будет удивлен, найдя ссылки на главы 7, 8 и последующие не вошедшие в предлагаемые три тома главы. Мы вместе с автором надеемся, что очень скоро они будут опубликованы и, безусловно, сразу же появятся в русском переводе в качестве продолжения этого издания.

Следует также обратить внимание на далеко не всегда стандартные обозначения, которыми пользуется автор. Так же, как и определения, эти обозначения могут появиться в 1-м томе, а вводиться во 2-м. Поэтому без указателя обозначений пользоваться книгой было бы чрезвычайно трудно. Хочу также обратить внимание на запись [A], где A — некоторое высказывание. Эта запись встречается в формулах, а иногда и в тексте, и обозначает величину, равную индикатору A.

— Профессор Ю. В. Козаченко

ПРЕДИСЛОВИЕ

Уважаемые читатели!
Вы держите в руках книгу,
издать которую Вы просили нас в тысячах писем.
Нам пришлось потратить годы на то,
чтобы самым тщательным образом проверить
и перепроверить бесконечное множество рецептов
и отобрать для вас самые лучшие, самые интересные,
самые совершенные.
Теперь без тени сомнения мы можем сказать,
что если вы будете следовать инструкциям,
то каждое блюдо будет получаться у вас таким же, как и у нас,
даже если раньше вы никогда не занимались
приготовлением пищи.
— Поваренная книга Мак-Колла (1963)

ПРОЦЕСС подготовки программ для цифрового компьютера — это очень увлекательное занятие. И дело не только в том, что оно оправдывает себя с экономической и научной точек зрения; оно может вызвать также эстетические переживания, подобные тем, которые испытывают творческие личности при написании музыки или стихов. Вы держите в руках первый том многотомного издания, цель которого — дать читателю разнообразные знания и умения, из которых и состоит ремесло программиста.

Последующие главы *не* являются введением в компьютерное программирование; предполагается, что вы уже имеете некоторый опыт в этой области. На самом деле предъявляемые к читателю требования очень просты; тем не менее начинающему программисту потребуются время и практика, чтобы понять, что собой представляет цифровой компьютер. Итак, читатель должен иметь:

- а) некоторое представление о том, как работает цифровой компьютер с хранимой программой; при этом необязательно разбираться в электронике, главное — понимать, каким образом команды можно сохранять в памяти компьютера, и затем последовательно их выполнять;
- б) способность ставить задачу с помощью четких и определенных терминов, понятных компьютеру (у компьютеров нет разума, присущего человеку, поэтому они делают в точности то, что им приказывают, не больше и не меньше; именно этот факт обычно труднее всего уяснить начинающим пользователям);
- с) знание самых простых компьютерных методов, таких как организация циклов (повторное выполнение некоторого набора команд), а также использование подпрограмм и переменных с индексами;

- d) знание распространенных компьютерных терминов, например “память”, “регистры”, “биты”, “плавающая точка”, “переполнение”, “программное обеспечение”; большинство терминов, которые не определены в тексте, поясняются в алфавитном указателе в конце каждого тома.

Эти четыре условия, вероятно, можно объединить в одном требовании: читатель должен иметь опыт написания и отладки по меньшей мере четырех программ хотя бы для одного компьютера.

Я старался писать эти книги так, чтобы они могли служить нескольким различным целям. Во-первых, они представляют собой справочное пособие, в котором сосредоточены знания из нескольких важных областей науки. Во-вторых, они могут использоваться в качестве пособий для самообразования и учебников по программированию или информатике для университетов. В связи с этим я включил в текст большое количество упражнений и предоставил ответы на большинство из них. Кроме того, я попытался сосредоточить внимание на фактах, вместо того чтобы “лить воду” и заниматься общими рассуждениями.

Этот трехтомник предназначен для всех, кто серьезно интересуется компьютерами, а не только для профессионалов. В сущности, одна из моих главных целей состояла в том, чтобы сделать методы программирования более доступными для специалистов из других областей. Как правило, эти специалисты получают большие преимущества, используя компьютеры, но не могут позволить себе тратить время на поиски необходимой информации, крупницы которой разбросаны по множеству технических журналов.

Тему этих книг можно сформулировать следующим образом: “Нечисленный анализ”. Компьютеры обычно ассоциируются с решением численных задач, таких как нахождение корней уравнения, численное интерполирование, интегрирование и т. д. Но в этом трехтомнике подобные темы не рассматриваются (за исключением случаев, когда это необходимо сделать по ходу изложения). Численное компьютерное программирование — необычайно интересная и бурно развивающаяся область; на эту тему написано очень много хороших книг. Но с 60-х годов компьютеры все чаще и чаще применяются для решения проблем, в которых числа играют второстепенную роль. Теперь на первый план выходит способность компьютера принимать решения, а не просто выполнять арифметические операции. При решении нечисленных задач иногда требуется выполнять операции сложения и вычитания, но потребность в умножении и делении возникает довольно редко. Но, конечно, даже тот, кто в основном занимается численным компьютерным программированием, только выиграет от изучения нечисленных методов, так как они лежат и в основе числовых программ.

Результаты исследований в области нечисленного анализа разбросаны по многим техническим журналам. Моя цель состояла в том, чтобы извлечь из этого огромного объема информации только фундаментальные методы, которые можно применять в разнотипных ситуациях программирования. Я попытался обобщить выбранную информацию, чтобы получить то, что в большей или меньшей степени можно назвать “теорией”, а также показать, как применять эту теорию при решении различных практических задач.

Конечно, “нечисленный анализ” — крайне неудачное название для данной области науки. Оно неудачно прежде всего потому, что содержит только отрицание

другого понятия; гораздо лучше было бы выбрать более содержательный термин, не имеющий приставки “не”. Название “обработка информации” охватывает более широкую область, чем рассматриваемый здесь материал, а “методы программирования” — более узкую. Я считаю, что для темы, освещаемой в данных книгах, самым подходящим является название *анализ алгоритмов*, которое можно расшифровать как “теория свойств некоторых компьютерных алгоритмов”.

Полный набор книг, озаглавленный как *Искусство программирования*, имеет следующую основную структуру.

Том 1. Основные алгоритмы

Глава 1. Основные понятия

Глава 2. Информационные структуры

Том 2. Получисленные алгоритмы

Глава 3. Случайные числа

Глава 4. Арифметика

Том 3. Сортировка и поиск

Глава 5. Сортировка

Глава 6. Поиск

Том 4. Комбинаторные алгоритмы

Глава 7. Комбинаторный поиск

Глава 8. Рекурсия

Том 5. Синтаксические алгоритмы

Глава 9. Лексикографический поиск

Глава 10. Синтаксический анализ

В томе 4 рассматривается очень большая тема, поэтому на самом деле он состоит из трех отдельных книг (томов 4А, 4В и 4С). Планируется также выпуск двух дополнительных томов по более специализированным темам: том 6, *Теория языков* (глава 11), и том 7, *Компиляторы* (глава 12).

Я приступил к этой работе в 1962 году с намерением написать единственную книгу, содержащую все перечисленные главы, но вскоре понял, что необходимо глубоко рассматривать выбранные темы, а не просто “скользить по поверхности”. В результате получился текст такого объема, что материала каждой главы оказалось более чем достаточно для изучения в течение одного университетского семестра. И стало ясно, что необходимо разбить материал на несколько отдельных томов. Я знаю, что книга, содержащая только одну-две главы, выглядит довольно странно, но решил сохранить первоначальную нумерацию глав, чтобы упростить перекрестные ссылки. Планируется выпуск сокращенного варианта томов 1–5, который будет служить более общим справочником и/или учебником для студентов; в нем будет содержаться основная часть материала данных томов, а более специальная информация будет опущена. В сокращенном издании будет сохранена такая же нумерация глав, как и в полном.

Том 1 можно рассматривать как “пересечение” полного набора глав, в том смысле, что он содержит основные сведения, которые используются во всех остальных

книгах. С другой стороны, тома 2–5 можно читать независимо один от другого. Том 1 — это не только справочник, который необходимо использовать как пособие при чтении других томов; он может служить также университетским учебником либо пособием для самообразования по теме *структуры данных* (основное внимание следует уделить главе 2) или *дискретная математика* (основное внимание следует уделить разделам 1.1, 1.2, 1.3.3 и 2.3.4), или *программирование на языке машинных команд* (основное внимание следует уделить разделам 1.3 и 1.4).

Эти главы написаны с другой точки зрения, чем та, которая используется в самых современных книгах по программированию, т. е. я не пытался научить читателя пользоваться чужим программным обеспечением. Вместо этого я стремился научить читателя писать собственные программы более высокого качества.

Моя первоначальная цель заключалась в том, чтобы познакомить читателей с передовыми научными исследованиями в каждой из рассматриваемых областей знания. Но очень сложно постоянно быть в курсе дел отрасли, которая является экономически выгодной; бурный рост компьютерной науки сделал невозможным осуществление моей мечты. Образно говоря, я очутился на берегу безбрежного океана, содержащего десятки тысяч маленьких результатов, которые были получены десятками тысяч талантливых людей по всему миру. Поэтому мне пришлось поставить перед собой новую цель — сосредоточиться на “классических” методах, которые останутся актуальными в течение многих десятилетий, и описать их как можно лучше по мере моих возможностей. В частности, я попытался проследить историю каждой темы и заложить прочный фундамент ее дальнейшего развития. Я старался использовать точную терминологию, согласованную с той, которая применяется в современных публикациях, и попытался сообщить обо всех известных идеях последовательного программирования, выделяющихся простотой и изяществом формулировок.

Теперь несколько слов о математическом содержании данного многотомного издания. Материал излагается так, что он вполне доступен даже лицам со средним образованием; более сложные фрагменты они смогут просмотреть или просто опустить. В то же время те, кто имеют склонность к математике, смогут изучить интересные математические методы, связанные с дискретной математикой. Подобная двойственность представления информации была достигнута, с одной стороны, за счет присвоения рейтинга каждому упражнению (чтобы читатель смог отличать сложные с математической точки зрения упражнения от простых), а с другой стороны, благодаря такой организации разделов, при которой главные математические результаты сформулированы *перед* доказательствами. Доказательства предлагаются либо провести самостоятельно в качестве упражнений (ответы на которые приводятся в отдельном разделе), либо найти в конце раздела.

Читатель, который, главным образом, интересуется программированием, а не математикой, может прекратить чтение раздела, как только математический материал станет слишком сложным для восприятия. С другой стороны, читатель-математик найдет для себя очень много интереснейших фактов. Многие математические публикации на тему программирования были ошибочными, поэтому одна из целей данной книги — предоставить читателю правильное математическое обоснование предмета изложения. И так как я считаю себя математиком, моя прямая

обязанность — правильно (насколько смогу) изложить материал с математической точки зрения.

Для чтения большей части математического материала вполне достаточно знания элементарной математики, так как почти вся остальная теория разрабатывается здесь же. Но иногда у меня возникает необходимость в более глубоких теоремах теории комплексного переменного, теории вероятностей, теории чисел и т. д. В подобных случаях я ссылаюсь на книги, содержащие подробное изложение данных тем.

Самое сложное решение, которое мне пришлось принять при подготовке этих книг, касалось способа представления различных методов. Преимущества блок-схем и пошаговых описаний алгоритмов хорошо известны; эти вопросы обсуждаются в статье “Computer-Drawn Flowcharts” в журнале *ACM Communications*, Vol. 6 (September 1963), pages 555–563. Для описания любого компьютерного алгоритма необходим также формальный и точный язык. Поэтому мне нужно было решить, какой язык использовать: алгебраический, такой как ALGOL или FORTRAN, либо машинно-ориентированный. Вероятно, многие сегодняшние компьютерные специалисты не согласятся с моим решением использовать машинно-ориентированный язык, но я убедился, что это был правильный выбор. На то существуют следующие причины.

- a) На программиста большое влияние оказывает язык, на котором написаны программы. В настоящее время превалирует тенденция к выбору самых простых, а не самых оптимальных для компьютера конструкций языка. А программист, который знает машинно-ориентированный язык, стремится использовать более эффективные методы и таким образом создает более совершенные программы.
- b) Все нужные нам программы, написанные на машинно-ориентированном языке, за редким исключением будут иметь небольшой размер. А это значит, что при наличии компьютера, обладающего минимальной вычислительной мощностью, проблем с использованием таких программ у нас не возникнет.
- c) Языки высокого уровня не подходят для обсуждения важных деталей, имеющих отношение к низкому уровню, таких как связь сопрограмм, генерирование случайных чисел, арифметика высокой точности и многие другие проблемы, связанные с эффективным использованием памяти.
- d) Тот, кто серьезно интересуется компьютерами, должен хорошо знать машинный язык, так как он лежит в основе работы компьютера.
- e) Некоторое знание машинного языка необходимо в любом случае, чтобы разобратся в выходных данных программ, приведенных во многих примерах.
- f) Новые алгебраические языки входят и выходят из моды приблизительно каждые пять лет, в то время как я пытаюсь говорить о “вечных истинах”.

С другой стороны, я признаю, что писать программы на языках высокого уровня и отлаживать эти программы значительно проще. В сущности, начиная с 1970 года, я сам редко использовал машинный язык низкого уровня для собственных программ, так как современные компьютеры обладают большим объемом памяти и высоким быстродействием. Но для решения многих проблем, рассматриваемых в данной

книге, наибольшее значение имеет искусство программирования. Например, некоторые комбинаторные вычисления нужно повторять триллионы раз, и мы сэкономим приблизительно 11,6 дней работы за счет того, что сократим время вычислений во внутреннем цикле всего на одну микросекунду. Аналогично имеет смысл приложить дополнительные усилия для написания программы, которая будет использоваться много раз в течение каждого дня на множестве компьютеров, тем более что написать эту программу нужно только один раз.

А если принять решение использовать машинно-ориентированный язык, то какому языку следует отдать предпочтение? Я мог бы выбрать язык для конкретной машины X , но тогда те, кто используют другой компьютер, подумают, что данная книга написана только в расчете на обладателей компьютера X . Более того, машина X , вероятно, имеет много характерных особенностей, для которых совершенно неприменим материал данной книги, но все же его необходимо изложить. И наконец, через два года фирма — производитель машины X выпустит машину $X + 1$ или $10X$, и компьютер X больше никого не будет интересовать.

Чтобы решить эту проблему, я попытался разработать “идеальный” компьютер с очень простыми правилами работы (изучить которые можно, скажем, всего за час) и очень похожий на реальные машины. Для студента нет никакой причины избегать изучения характеристик различных компьютеров; после изучения одного языка все остальные будут усваиваться гораздо легче. Кроме того, серьезный программист должен быть готов к тому, что в ходе работы ему придется сталкиваться с различными машинными языками. Поэтому остается только один недостаток использования вымышленной машины — сложность запуска написанных для нее программ. К счастью, на самом деле это не проблема, так как много добровольцев предложили свои услуги по написанию имитаторов гипотетической машины. Такие имитаторы идеальны для учебных целей, и работать с ними даже проще, чем с реальным компьютером.

Я старался ссылаться на самые лучшие старые статьи по каждой теме, а также упоминать новые работы. Ссылаясь на литературные источники, я использовал стандартные сокращения для названий периодических изданий, за исключением наиболее часто цитируемых журналов, для которых применялись следующие сокращения.

CACM — Communications of the Association for Computing Machinery

JACM — Journal of the Association for Computing Machinery

Comp. J. — The Computer Journal (British Computer Society)

Math. Comp. — Mathematics of Computation

AMM — American Mathematical Monthly

SICOMP — SIAM Journal on Computing

FOCS — IEEE Symposium on Foundations of Computer Science

SODA — ACM-SIAM Symposium on Discrete Algorithms

STOC — ACM Symposium on Theory of Computing

Crelle — Journal für die reine und angewandte Mathematik

Например, “*SACM* 6 (1963), 555–563” означает ссылку на журнал, упомянутый в одном из предыдущих абзацев этого предисловия. Сокращение “*CMath*” я использовал также для обозначения книги *Конкретная математика*, на которую есть ссылка во введении к разделу 1.2.

Большая часть технического материала этих книг приходится на упражнения. Если идея нетривиального упражнения принадлежала не мне, то я старался упомянуть ее автора. Ссылки на литературу обычно даются в тексте раздела либо в ответе к упражнению. Но во многих случаях в основе упражнений лежат неопубликованные материалы, на которые нельзя дать ссылки.

В течение долгих лет работы над этими книгами мне помогали многие люди, которым я благодарен от всей души. Прежде всего, я хочу выразить благодарность моей жене Джилл (Jill) за ее бесконечное терпение, за подготовку некоторых иллюстраций и за постоянную помощь во всем. Я признателен также Роберту В. Флойду (Floyd Robert W.) за то, что в 60-х годах он посвятил столько времени работе над улучшением и углублением данного материала. Тысячи других людей также оказали мне неоценимую помощь. Чтобы просто перечислить их имена, понадобилась бы еще одна такая книга! Многие из них любезно разрешили мне использовать их старые неопубликованные работы. Мои исследования в Калифорнийском технологическом институте и Станфордском университете щедро финансировались Национальным научным фондом (National Science Foundation) и департаментом морских исследований (Office of Naval Research). Издательство Addison-Wesley всегда оказывало мне всемерную помощь и поддержку с того самого времени, когда в 1962 году я приступил к работе над проектом. Мне кажется, что для всех этих людей лучшая благодарность — данная публикация. Она показывает, что их вклад привел к появлению книг, в которых, я надеюсь, мне удалось написать то, чего они ожидали.

Предисловие к третьему изданию

Потратив десять лет на разработку систем компьютерного набора METAFONT и T_EX, я теперь могу осуществить свою мечту — применить эти системы для набора книги *Искусство программирования*. Наконец-то мне удалось внести полный текст этой книги в персональный компьютер и таким образом получить ее электронную версию, что позволит в дальнейшем вносить любые изменения в технологию печати и отображения на экране. Такой способ работы дал мне возможность сделать буквально тысячи улучшений; я добился того, о чем так долго мечтал.

В этом новом издании я смог проверить каждое слово в тексте, стараясь сохранить юношеский задор моих первоначальных исследований и в то же время внести большую зрелость суждений. Были добавлены десятки новых упражнений, а на десятки старых даны новые или улучшенные ответы.

 Таким образом, работа над книгой *Искусство программирования* продолжается. Именно поэтому некоторые части данной книги начинаются пиктограммой “В процессе построения” (это своеобразное извинение за то, что приведены не самые новые данные). Мои папки переполнены важными материалами, которые я планирую включить в окончательное, славное четвертое издание тома 1; оно выйдет, вероятно, через 15 лет. Но сначала я должен закончить тома 4 и 5. Я хочу, чтобы они были опубликованы сразу же, как только будут готовы к печати.

Большая часть тяжелой работы по подготовке этого нового издания была выполнена Филлис Винклер (Phyllis Winkler) и Сильвио Леви (Silvio Levy), которые профессионально набирали и редактировали текст второго издания, а также Джеффри Олдхэмом (Jeffrey Oldham), который конвертировал почти все оригинальные иллюстрации в формат METAPOST. Я исправил все ошибки, которые бдительные читатели (Бэрри) обнаружили во втором издании (а также ошибки, которые, увы, не заметил никто), и постарался избежать появления новых ошибок в новом материале. Тем не менее я допускаю, что некоторые огрехи все же остались, и хотел бы исправить их как можно скорее. Поэтому за каждую опечатку*, а также ошибку, относящуюся к сути излагаемого материала или к приведенным историческим сведениям, я охотно заплачу \$2,56 тому, кто первым ее найдет. На Web-странице, адрес которой приведен на обороте титульной страницы, содержится текущий список всех ошибок, о которых мне сообщили**.

*Станфорд, Калифорния
Апрель 1997*

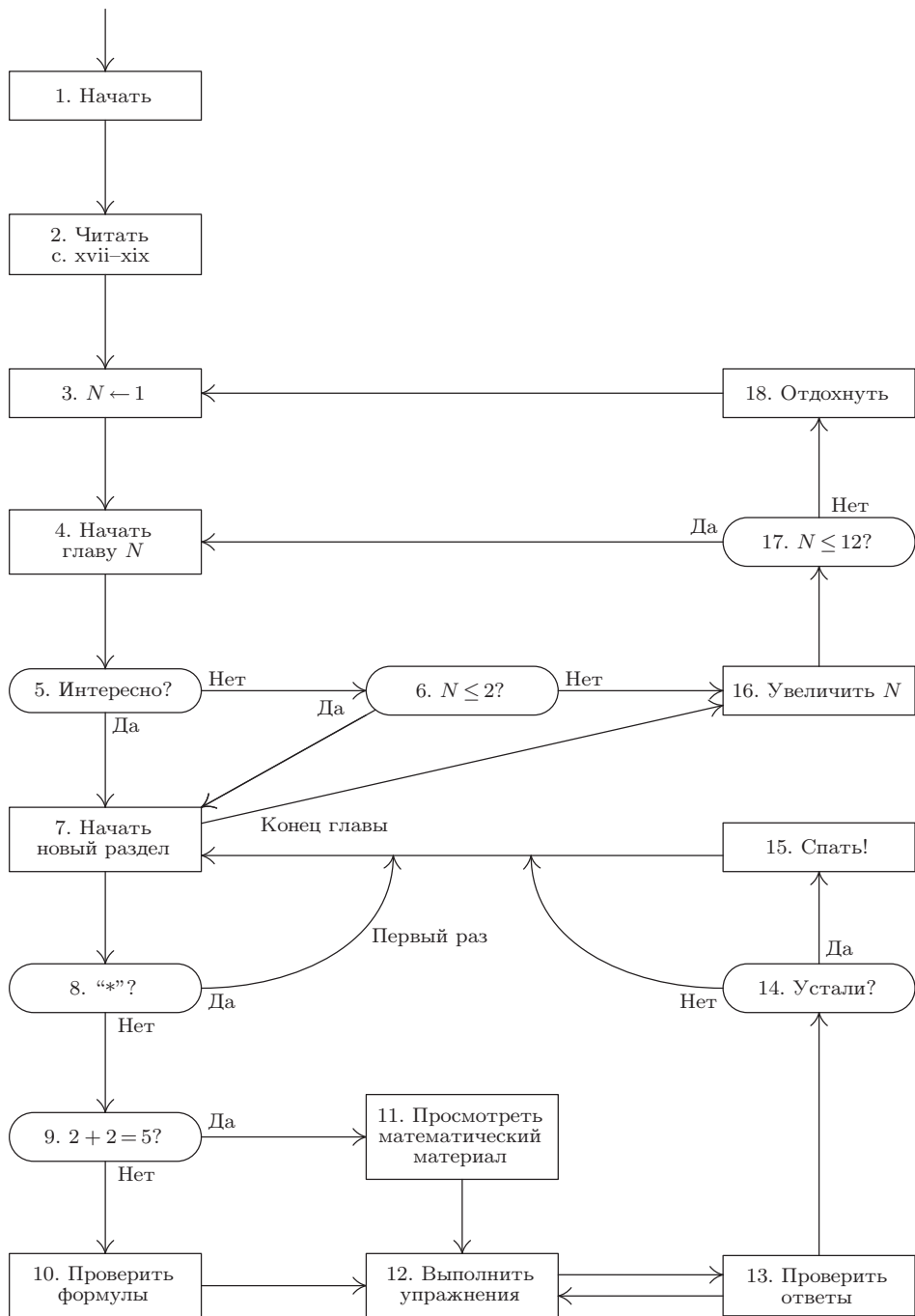
D. E. K.

За последние двадцать лет мир изменился.

— Билл Гейтс (Bill Gates) (1995)

* Имеется в виду оригинал настоящего издания. — *Прим. ред.*

** Ошибки, известные на момент подготовки русского издания, были исправлены. — *Прим. ред.*



Блок-схема процедуры чтения книг этой серии.

Процедура чтения книг этой серии

1. Начните читать настоящую процедуру, если вы этого еще не сделали. *Продолжайте в точном соответствии с указанными шагами.* (Общая форма этой процедуры и сопровождающей ее блок-схемы будет использоваться на протяжении всей книги.)
2. Прочтите примечания к упражнениям (с. 23–25).
3. Установите N равным 1.
4. Начните чтение главы N . *Не* читайте эпиграфы, помещенные в ее начале.
5. Вам интересен предмет этой главы? Если да, перейдите к шагу 7; если нет, перейдите к шагу 6.
6. $N \leq 2$? Если нет, перейдите к шагу 16; если да, то все-таки просмотрите главу. (В главах 1 и 2 содержится важный вводный материал, а также обзор основных методов программирования. Эти главы следует хотя бы просмотреть, чтобы ознакомиться с условными обозначениями и компьютером MIX.)
7. Начните чтение следующего раздела главы; но, если вы уже дошли до конца главы, перейдите к шагу 16.
8. Отмечен ли номер раздела символом “*”? Если да, то при первом чтении этот раздел можно пропустить (в нем рассматривается специальный вопрос, который интересен, но не имеет первостепенного значения). Вернитесь к шагу 7.
9. Есть ли у вас наклонности к математике? Если математика для вас — китайская грамота, то перейдите к шагу 11; в противном случае перейдите к шагу 10.
10. Проверьте математические выкладки, выполненные в этом разделе (и сообщите автору о замеченных ошибках). Перейдите к шагу 12.
11. Если в текущем разделе содержится много математических выкладок, то лучше не читайте их. Тем не менее вам следует ознакомиться с основными результатами раздела; они, как правило, либо приводятся в самом начале раздела, либо выделены *курсивом* в самом конце сложной части текста.
12. Выполните рекомендуемые упражнения к разделу в соответствии с указаниями, приведенными в примечаниях к упражнениям (которые вы прочитали на шаге 2).
13. Поработав над упражнениями в свое удовольствие, сравните полученные ответы с теми, которые приведены в соответствующем разделе в конце книги (если для

этих задач даны ответы). Прочтите также ответы к упражнениям, над которыми у вас не было времени поработать. (*Замечание.* В большинстве случаев имеет смысл сначала прочесть ответ к упражнению n , а затем приступить к упражнению $n + 1$, поэтому шаги 12 и 13 обычно выполняются одновременно.)

14. Вы устали? Если нет, вернитесь к шагу 7.
15. Отправляйтесь спать. А когда проснетесь, вернитесь к шагу 7.
16. Увеличьте N на 1. Если $N = 3, 5, 7, 9, 11$ или 12, то возьмите следующий том этой серии книг.
17. Если N меньше или равно 12, то вернитесь к шагу 4.
18. Поздравляю! Теперь постарайтесь убедить друзей в том, что необходимо приобрести экземпляр тома 1 и начать его читать. Сами же возвращайтесь к шагу 3.

Горе тому, кто читает только одну книгу.

— ДЖОРДЖ ГЕРБЕРТ (GEORGE HERBERT), *Jacula Prudentum*, 1144 (1640)

*Единственный недостаток всех
литературных произведений в том,
что они слишком длинны.*

— ВОБЕНАРГ (VAUVENARGUES), *Réflexions*, 628 (1746)

Книги банальны. Гениальна только жизнь.

— ТОМАС КАРЛЕЙЛЬ (THOMAS CARLYLE), *Journal* (1839)

ПРИМЕЧАНИЯ К УПРАЖНЕНИЯМ

УПРАЖНЕНИЯ, приведенные в этой серии книг, предназначены как для самостоятельной проработки, так и для семинарских занятий. Очень трудно и, наверное, просто невозможно выучить предмет, только читая теорию и не применяя ее для решения конкретных задач, которые заставляют задуматься о прочитанном. Более того, мы лучше всего заучиваем то, до чего дошли самостоятельно, своим умом. Поэтому упражнения занимают важное место в данном издании. Я приложил немало усилий, чтобы сделать их как можно более информативными, а также отобрать задачи, которые были бы не только поучительны, но и позволяли читателю получить удовольствие от их решения.

Во многих книгах простые упражнения даются вперемишку с исключительно сложными. Это не всегда удобно, так как читателю хочется знать заранее, сколько времени ему придется затратить на решение задач (иначе в лучшем случае он их только просмотрит). В качестве классического примера подобной ситуации можно привести книгу Ричарда Беллмана (Richard Bellman) *Динамическое программирование* (М.: Изд-во иностр. лит., 1960). Это очень важная, новаторская работа, но у нее есть один недостаток: в конце некоторых глав в разделе “Упражнения и научные проблемы” среди серьезных, еще нерешенных проблем, попадаются простейшие вопросы. Говорят, что кто-то однажды спросил д-ра Беллмана, как отличить упражнения от научных проблем, и он ответил: “Если вы можете решить задачу, значит, это упражнение; в противном случае это научная проблема”.

Совершенно очевидно, что в книге, подобной этой, должны быть приведены и сложные научные проблемы, и простейшие упражнения. Поэтому, чтобы читатель не ломал голову, пытаясь отличить одно от другого, были введены рейтинги, которые определяют степень сложности каждого упражнения. Эти рейтинги имеют следующее значение.

Рейтинг Объяснение

- | | |
|----|--|
| 00 | Чрезвычайно простое упражнение, на которое можно ответить сразу же, если прочитанный материал понят. Упражнения подобного типа почти всегда можно решить “в уме”. |
| 10 | Простая задача, которая заставляет задуматься над прочитанным, но не представляет особых трудностей. На ее решение вы затратите не больше минуты; в процессе решения могут понадобиться карандаш и бумага. |
| 20 | Средняя задача, которая позволяет проверить, понял ли читатель основные положения изложенного материала. Чтобы получить исчерпывающий ответ, может понадобиться примерно 15–20 минут. |
| 30 | Задача умеренной сложности. Для ее решения может понадобиться более двух часов (а если одновременно вы смотрите телевизор, то еще больше). |

- 40 Достаточно сложная или трудоемкая задача, которую вполне можно включить в план семинарских занятий. Предполагается, что студент должен справиться с ней, затратив не слишком много времени, и решение будет нетривиальным.
- 50 Научная проблема, которая (насколько известно автору в момент написания книги) пока еще не получила удовлетворительного решения, хотя найти его пытались очень многие. Если вы нашли решение подобной проблемы, то опубликуйте его; более того, автор данной книги будет очень признателен, если ему сообщат решение как можно скорее (при условии, что оно правильно).

Интерполируя по этой “логарифмической” шкале, можно понять, что означает любой промежуточный рейтинг. Например, рейтинг 17 говорит о том, что упражнение немного проще, чем задача средней сложности. Если задача с рейтингом 50 будет впоследствии решена каким-либо читателем, то в следующих изданиях данной книги и в списке ошибок, опубликованных в Internet, она может иметь рейтинг 45 (адрес Web-страницы приводится на обороте титульной страницы).

Остаток от деления рейтинга на 5 показывает, какой объем рутинной работы потребуется для решения данной задачи. Таким образом, для решения упражнения с рейтингом 24 может потребоваться больше времени, чем для упражнения с рейтингом 25, но для последнего необходим более творческий подход.

Автор очень старался правильно присвоить рейтинги упражнениям, но тому, кто составляет задачи, трудно предвидеть, насколько сложными они окажутся для кого-то другого. К тому же одному человеку некая задача может показаться простой, а другому — сложной. Таким образом, определение рейтингов — дело достаточно субъективное и относительное. Я надеюсь, что рейтинги помогут вам получить правильное представление о степени трудности задач, но их следует воспринимать в качестве ориентира, а не в качестве абсолюта.

Эта книга написана для читателей с различным уровнем математической подготовки и научного кругозора, поэтому некоторые упражнения рассчитаны исключительно на тех, кто серьезно интересуется математикой или занимается ею профессионально. Если рейтингу предшествует буква *M*, значит, математические понятия и обоснования используются в упражнении в большей степени, чем это необходимо тому, кто интересуется в основном программированием алгоритмов. Если же упражнение отмечено буквами *HM*, то для его решения необходимо знание высшей математики в большем объеме, чем дается в настоящей книге. Но пометка *HM* совсем необязательно означает, что упражнение трудное.

Перед некоторыми упражнениями стоит стрелка “►”, которая означает, что они особенно поучительны и их очень рекомендуется выполнять. Само собой разумеется, никто не ожидает, что читатель (или студент) будет решать *все* задачи, поэтому наиболее важные из них и были выделены. Но это ни в коем случае не означает, что другие упражнения выполнять не стоит! Каждый читатель должен хотя бы попытаться решить все задачи, рейтинг которых меньше или равен 10. Стрелки помогут выбрать задачи с более высокими рейтингами, которые следует решать в первую очередь.

К большинству упражнений приведены ответы, помещенные в отдельном разделе в конце книги. Пожалуйста, пользуйтесь ими разумно: ответ смотрите только

после того, как приложите все усилия, чтобы решить задачу самостоятельно, либо если у вас совершенно нет времени на ее решение. Ответ будет поучителен и полезен для вас только в том случае, если вы ознакомитесь с ним *после* того, как найдете свое решение или изрядно потрудитесь над задачей. Ответы к задачам излагаются очень кратко и схематично, так как предполагается, что читатель честно пытался решить задачу собственными силами. Иногда в приведенном решении дается меньше информации, чем спрашивалось, но чаще бывает наоборот. Вполне возможно, что полученный вами ответ окажется лучше того, который помещен в книге, или вы найдете ошибку в ответе. В таком случае автор был бы очень признателен, если бы вы как можно скорее подробно сообщили ему об этом; тогда в последующих изданиях книги будет опубликовано более удачное решение, а также имя его автора.

Решая задачи, вы, как правило, можете пользоваться ответами к предыдущим упражнениям, за исключением случаев, когда это будет оговорено особо. Рейтинги упражнениям присваивались в расчете именно на это, и вполне возможно, что рейтинг упражнения $n + 1$ ниже рейтинга упражнения n , даже если результат упражнения n является его частным случаем.

Условные обозначения

► Рекомендуется

M С математическим уклоном

HM Требуется знания высшей математики

00 Простейшее (ответ дать немедленно)

10 Простое (на одну минуту)

20 Средней трудности (на четверть часа)

30 Повышенной трудности

40 Высокой трудности

50 Научная проблема

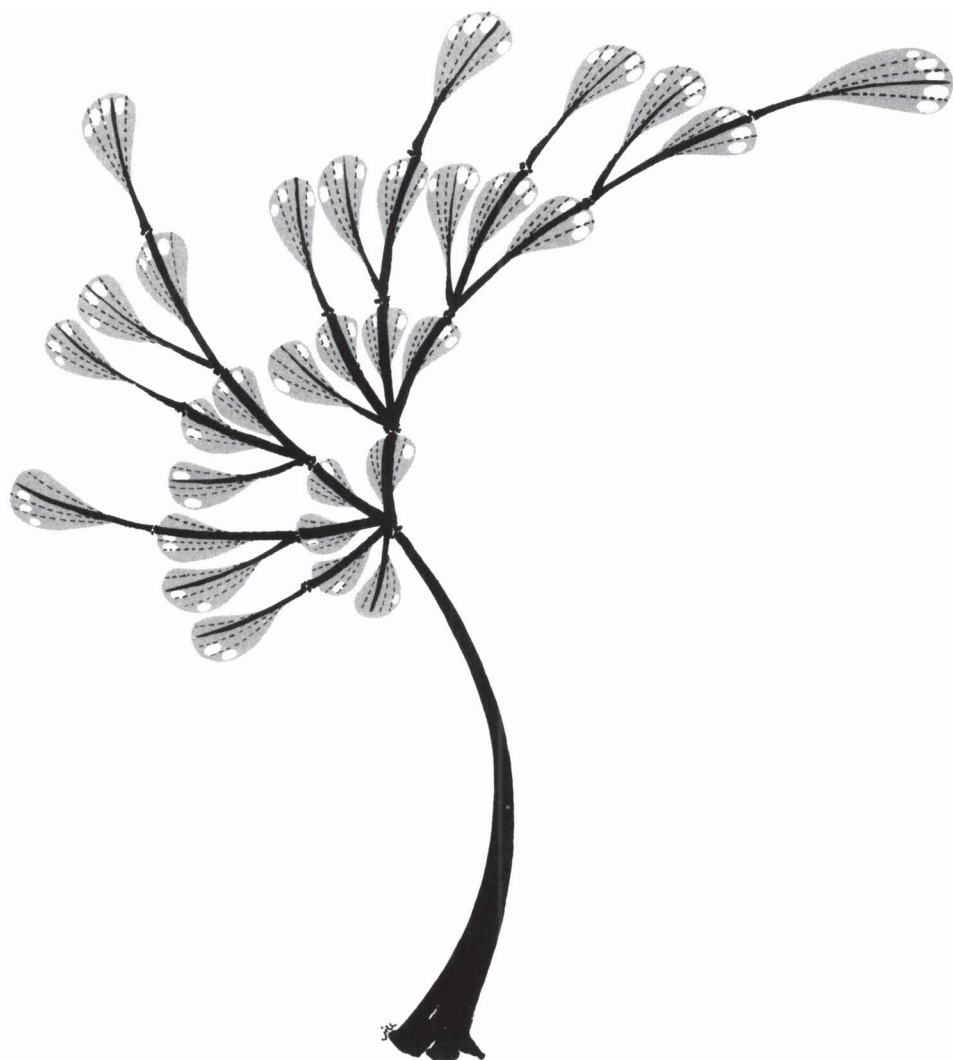
УПРАЖНЕНИЯ

- 1. [00] Что означает рейтинг *M20*?
- 2. [10] Какое значение для читателя имеют упражнения, которые приводятся в учебниках?
- 3. [14] Докажите, что $13^3 = 2197$. Обобщите ответ. (Скучных задач, подобных этой, автор старался избегать.)
- 4. [HM45] Докажите, что если n — целое число, $n > 2$, то уравнение $x^n + y^n = z^n$ неразрешимо в целых положительных числах x, y, z .

Мы обсудили этот вопрос со всех сторон.

Перед нами факты, изложенные систематично и по порядку.

— ЭРКЮЛЬ ПУАРО (HERCULE POIROT),
Убийство в восточном экспрессе (1934)



ГЛАВА 1

ОСНОВНЫЕ ПОНЯТИЯ

*Многие не сведущие
в математике люди
думают, что поскольку назначение
аналитической машины Бэббиджа (Babbage) —
выдавать результаты в численном виде,
то природа происходящих в ней процессов
должна быть арифметической и численной,
а не алгебраической и аналитической.
Но они ошибаются. Машина может упорядочивать
и комбинировать числовые значения
так же, как и буквы или любые другие символы
общего характера. В сущности, при выполнении
соответствующих условий она могла бы
выдавать результаты и в алгебраическом виде.*

— АВГУСТА АДА (AUGUSTA ADA), графиня Лавлейс* (Lovelace) (1844)

*Прошу вас, ради всего святого, сначала научитесь простому
и только потом переходите к сложному.*

— ЭПИКТЕТ (EPICTETUS), *Беседы* IV.i

1.1. АЛГОРИТМЫ

Понятие *алгоритм* является основным для всей области компьютерного программирования, поэтому начать мы должны с тщательного анализа этого термина. Слово “алгоритм” (algorithm) (иногда используется устаревшее слово “алгорифм”. — *Прим. перев.*) уже само по себе представляет большой интерес. На первый взгляд может показаться, будто кто-то собирался написать слово “логарифм” (logarithm), но случайно переставил первые четыре буквы. Этого слова еще не было в издании словаря *Webster’s New World Dictionary*, вышедшем в 1957 году. Мы находим там только устаревшую форму “algorism” — старинное слово, которое означает “выполнение арифметических действий с помощью арабских цифр”. В средние века абакисты считали на абаках (счетных досках), а алгоритмики использовали “algorism”. В эпоху Возрождения происхождение этого слова оказалось забытым. Одни лингвисты того времени пытались объяснить его значение путем сочетания

* Дочь великого английского поэта Дж. Г. Байрона, которую считают основоположницей программирования. Большинство идей и принципов программирования для аналитической машины Бэббиджа было рассмотрено в ее книге “Комментарии”. — *Прим. перев.*

слов *algiros* [больной] и *arithmos* [число], другие не соглашались с таким толкованием и утверждали, что это слово происходит от “King Algor of Castile”. Наконец историки математики обнаружили истинное происхождение слова “algorism”: оно берет начало от имени автора знаменитого персидского учебника по математике, Abū ‘Abd Allāh Muḥammad ibn Mūsā al-Khwārizmī (Абу Абд Аллах Мухаммед ибн Муса аль-Хорезми) (ок. 825 г.), означающего буквально “Отец Абдуллы, Мухаммед, сын Мусы, уроженец Хорезма”*. Аральское море в Центральной Азии когда-то называлось озером Хорезм, и район Хорезма (Khwārizm) расположен в бассейне реки Амударья южнее этого моря. Аль-Хорезми написал знаменитую книгу *Kitāb al-jabr wa’l-muqābala* (Китаб аль-джебр валь-мукабала — “Книга о восстановлении и противопоставлении”). От названия этой книги, которая была посвящена решению линейных и квадратных уравнений, произошло еще одно слово — “алгебра”. [О жизни и научной деятельности аль-Хорезми речь идет в работе Н. Zemanek, *Lecture Notes in Computer Science* **122** (1981), 1–81.]

Постепенно форма и значение слова *algorism* исказились; как объясняется в словаре *Oxford English Dictionary*, это слово “претерпело множество псевдоэтимологических искажений, включая последний вариант *algorithm*, где произошла путаница” с корнем слова греческого происхождения *arithmetic*. Этот переход от “algorism” к “algorithm” кажется вполне закономерным ввиду того, что происхождение рассматриваемого слова было полностью забыто. В старинном немецком математическом словаре *Vollständiges mathematisches Lexicon* (Leipzig, 1747) дается следующее определение слова *algorithmus*: “Этот термин включает в себя понятие о четырех типах арифметических операций, а именно: о сложении, умножении, вычитании и делении”. Латинское выражение *algorithmus infinitesimalis* в то время использовалось для определения “способов выполнения действий с бесконечно малыми величинами, открытых Лейбницем (Leibniz)”.

К 1950 году слово “алгоритм” чаще всего ассоциировалось с алгоритмом Евклида, который представляет собой процесс нахождения наибольшего общего делителя двух чисел. Этот алгоритм приведен в книге Евклида (Euclid) *Начала* (книга 7, предложения 1 и 2). Думаю, имеет смысл привести здесь описание этого алгоритма.

Алгоритм Е (*Алгоритм Евклида*). Даны два целых положительных числа m и n . Требуется найти их *наибольший общий делитель*, т. е. наибольшее целое положительное число, которое нацело делит оба числа m и n .

Е1. [Нахождение остатка.] Разделим m на n , и пусть остаток от деления будет равен r (где $0 \leq r < n$).

Е2. [Сравнение с нулем.] Если $r = 0$, то выполнение алгоритма прекращается; n — искомое значение.

Е3. [Замещение.] Присвоить $m \leftarrow n$, $n \leftarrow r$ и вернуться к шагу Е1. ■

Разумеется, у Евклида этот алгоритм сформулирован не совсем так. Приведенная выше формулировка иллюстрирует стиль, в котором алгоритмы будут представлены на протяжении всей этой книги.

Каждому рассматриваемому алгоритму присваивается идентифицирующая буква (в предыдущем примере использовалась буква Е), а шагам алгоритма — эта же

* В русскоязычных источниках это историческое лицо обычно упоминается под именем “аль-Хорезми”. — *Прим. перев.*

буква в сочетании с числом (E1, E2, E3). Главы книги подразделяются на пронумерованные разделы; внутри раздела алгоритмы обозначаются только буквами. Но когда на эти алгоритмы делают ссылки из других разделов, то к букве присоединяется номер соответствующего раздела. Например, сейчас мы находимся в разделе 1.1; внутри этого раздела алгоритм Евклида называется “Алгоритм E”, но сослаться на него в последующих разделах мы будем как на алгоритм 1.1E.

Каждый шаг любого алгоритма, например E1 в вышеприведенном алгоритме, начинается заключенной в квадратные скобки фразой, которая как можно более кратко выражает содержание данного шага. Обычно эта фраза отражается также в сопровождающей алгоритм *блок-схеме*, такой как на рис. 1, чтобы читатель мог легко представить себе описанный алгоритм.

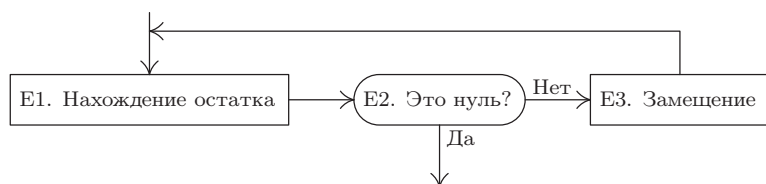


Рис. 1. Блок-схема алгоритма E.

За краткой фразой следует формулировка (выраженная с помощью слов и символов) *действия*, которое нужно выполнить, или решения, которое нужно принять. Могут присутствовать также заключенные в круглые скобки *комментарии* (например, второе предложение шага E1). Комментарии играют роль пояснений к шагу; с их помощью часто указываются некоторые постоянные характеристики переменных или текущие цели данного этапа. В комментариях не определяются действия, которые являются составной частью алгоритма; они служат только для удобства читателя, чтобы по возможности помочь ему разобраться в алгоритме.

Стрелка “ $\leftarrow$ ”, используемая на шаге E3, обозначает важнейшую операцию *замещения*, которую иногда называют *присвоением* или *подстановкой*: запись $m \leftarrow n$ указывает, что значение переменной m замещается текущим значением переменной n . В начале работы алгоритма E m и n — это заданные первоначальные значения, но по окончании его работы эти переменные будут иметь, вообще говоря, совершенно другие значения. Стрелка используется для того, чтобы отличать операцию замещения от отношения равенства. Мы не будем говорить: “Установим $m = n$ ”, а, вероятно, спросим: “Действительно ли $m = n$?”. Знак “ $=$ ” обозначает условие, которое можно проверить, а знак “ $\leftarrow$ ” — действие, которое можно выполнить. Операция *увеличение n на единицу* обозначается через $n \leftarrow n + 1$ (и читается так: “ n замещается значением $n + 1$ ” или “ n принимает значение $n + 1$ ”). Вообще говоря, запись “переменная $\leftarrow$ формула” означает, что формула будет вычислена на основании текущих значений всех входящих в нее переменных, а полученный результат будет присвоен переменной, стоящей слева от стрелки (таким образом, вычисленный по формуле результат заменит собой предыдущее значение переменной слева). Лица, не имеющие достаточного опыта программирования, иногда говорят, что “ n переходит в $n + 1$ ” и для обозначения операции увеличения n на единицу

используют запись $n \rightarrow n + 1$. Такая система обозначений и формулировок противоречит стандартным соглашениям и может привести только к путанице, поэтому ее следует избегать.

Обратите внимание, что на шаге ЕЗ очень важен порядок действий. Действительно, записи “присвоить $m \leftarrow n, n \leftarrow r$ ” и “присвоить $n \leftarrow r, m \leftarrow n$ ” — это совсем не одно и то же. Из второй записи следует, что предыдущее значение n будет потеряно до того, как его смогут присвоить m . На самом деле эквивалентом второй записи будет “присвоить $n \leftarrow r, m \leftarrow r$ ”. Когда нескольким переменным присваивается одно и то же значение, в одном выражении можно использовать несколько стрелок. Так, например, операцию “ $n \leftarrow r, m \leftarrow r$ ” можно записать как “ $n \leftarrow m \leftarrow r$ ”. Операцию взаимного обмена значениями двух переменных можно записать так: “Обмен $m \leftrightarrow n$ ”. Ее можно записать и с помощью новой переменной t следующим образом: “Присвоить $t \leftarrow m, m \leftarrow n, n \leftarrow t$ ”.

Выполнение алгоритма начинается с шага, имеющего наименьший номер (обычно это шаг 1). Затем последовательно выполняются следующие шаги, если нет каких-либо указаний, нарушающих естественный порядок выполнения. На шаге ЕЗ указание “Вернуться к шагу Е1” явным образом определяет порядок вычислений. На шаге Е2 действию предшествует условие “Если $r = 0$ ” и если $r \neq 0$, то оставшаяся часть предложения не применяется и нет указаний на выполнение в этом случае каких-либо действий. Конечно, мы могли бы добавить дополнительное предложение “Если $r \neq 0$, то перейти к шагу ЕЗ”, но это совершенно излишне.

Жирная вертикальная черточка “**┃**”, помещенная в конце шага ЕЗ, обозначает окончание алгоритма и продолжение текста.

Итак, мы обсудили практически все соглашения об обозначениях, которые используются в алгоритмах, приведенных в книге. Осталось выяснить только, как обозначать величины с индексами (или “подстрочными” индексами), которые являются элементами упорядоченного массива. Предположим, у нас есть n величин: $v_1, v_2, \dots, v_n$. Для обозначения j -го элемента вместо записи v_j часто используется запись $v[j]$. Аналогично массив иногда предпочитают обозначать как $a[i, j]$, вместо того чтобы использовать два подстрочных индекса, как в записи a_{ij} . Иногда для обозначения переменных используются имена, состоящие из нескольких букв, обычно прописных. Например, TEMP может быть именем переменной, использующейся для временного хранения вычисленного значения, а PRIME[K] может обозначать K-е простое число, и т. д.

До сих пор мы говорили о *форме записи* алгоритмов, а теперь давайте попробуем *выполнить* один из них. Хочу сразу заметить, что читателю *не* следует рассчитывать на то, что алгоритмы можно читать, как роман. Такое чтение приведет к тому, что вам будет трудно понять, что же на самом деле происходит при выполнении алгоритма. Чтобы проверить алгоритм, в нем нужно разобраться, и лучший способ понять, как он работает, — испытать его. Поэтому я предлагаю вам взять карандаш и бумагу и прорабатывать от начала до конца каждый алгоритм сразу же, как только он встретится в тексте. Обычно к примеру алгоритма прилагается схема, в противном случае читатель легко сможет представить ее. Это самый простой и доступный способ разобраться в алгоритме, в то время как все остальные подходы оказываются неэффективными.

Итак, давайте в качестве примера разберем алгоритм Е. Предположим, что $m = 119$ и $n = 544$; начнем с шага Е1. (Сейчас можете просто следить за изложением, так как мы разберем алгоритм и подробно все выпишем.) Деление m на n в этом случае выполняется просто, даже очень просто, так как частное равно нулю, а остаток — 119. Таким образом, $r \leftarrow 119$. Переходим к шагу Е2. Поскольку $r \neq 0$, на этом шаге никакие действия не выполняются. На шаге Е3 присваиваем $m \leftarrow 544$, $n \leftarrow 119$. Очевидно, что если первоначально $m < n$, то частное на шаге Е1 всегда оказывается равным нулю и в ходе выполнения алгоритма всегда происходит взаимный обмен значений переменных m и n , хотя и таким громоздким способом. Поэтому можно добавить дополнительный шаг.

Е0. [Гарантировать, что $m \geq n$.] Если $m < n$, то выполнить взаимный обмен $m \leftrightarrow n$.

В результате алгоритм изменится незначительно (разве что увеличится на один шаг), но зато время его выполнения сократится примерно в половине случаев.

Вернемся к шагу Е1. Находим, что $\frac{544}{119} = 4\frac{68}{119}$, поэтому $r \leftarrow 68$. В результате на шаге Е2 снова не выполняются никакие действия, а на шаге Е3 присваиваем $m \leftarrow 119$, $n \leftarrow 68$. В следующих циклах сначала получаем $r \leftarrow 51$ и $m \leftarrow 68$, $n \leftarrow 51$, а затем — $r \leftarrow 17$ и $m \leftarrow 51$, $n \leftarrow 17$. Наконец, в результате деления 51 на 17 получаем $r \leftarrow 0$. Таким образом, на шаге Е2 выполнение алгоритма прекращается. Наибольший общий делитель 119 и 544 равен 17.

Вот что такое алгоритм. Современное значение слова “алгоритм” во многом аналогично таким понятиям, как *рецепт*, *процесс*, *метод*, *способ*, *процедура*, *программа*, но все-таки слово “algorithm” имеет дополнительный смысловой оттенок. Алгоритм — это не просто набор конечного числа правил, задающих последовательность выполнения операций для решения задачи определенного типа. Помимо этого, он имеет пять важных особенностей.

1) *Конечность*. Алгоритм всегда должен заканчиваться после выполнения конечного числа шагов. Алгоритм Е удовлетворяет этому условию, потому что после шага Е1 значение r меньше, чем n . Поэтому если $r \neq 0$, то в следующем цикле на шаге Е1 значение n уменьшается. Убывающая последовательность положительных целых чисел имеет конечное число членов, поэтому шаг Е1 может выполняться только конечное число раз для любого первоначально заданного значения n . Но имейте в виду, что количество шагов может быть сколь угодно большим; выбор слишком больших значений m и n приведет к тому, что шаг Е1 будет выполняться более миллиона раз.

Процедура, обладающая всеми характеристиками алгоритма, за исключением, возможно, конечности, называется *методом вычислений*. Евклид (Euclid) предложил не только алгоритм нахождения наибольшего общего делителя, но и аналогичное ему геометрическое построение “наибольшей общей меры” длин двух отрезков прямой; это уже метод вычислений, выполнение которого не заканчивается, если заданные длины оказываются несоизмеримыми.

2) *Определенность*. Каждый шаг алгоритма должен быть точно определен. Действия, которые нужно выполнить, должны быть строго и недвусмысленно определены для каждого возможного случая. Я надеюсь, что алгоритмы, приведенные в данной книге, удовлетворяют этому критерию. Но дело в том, что они описываются обычным языком, и поэтому существует возможность неточного понимания

читателем мысли автора. Чтобы преодолеть это затруднение, для описания алгоритмов были разработаны формально определенные *языки программирования*, или *машинные языки*, в которых каждый оператор имеет строго определенное значение. Многие алгоритмы в этой книге даются как на обычном языке, так и на языке программирования. Метод вычислений, выраженный на языке программирования, называется *программой*.

Рассмотрим в качестве примера алгоритм Е. Применительно к шагу Е1 критерий определенности означает, что читатель обязан точно понимать, что значит разделить m на n и что такое остаток. Но в действительности нет никакого общепринятого соглашения по поводу того, что это означает, если m и n не являются целыми положительными числами. Каким будет остаток от деления -8 на $-\pi$? Что понимать под остатком от деления $59/13$ на нуль? Поэтому в данном случае критерий определенности означает следующее: мы должны быть уверены, что в каждом случае выполнения шага Е1 значениями m и n всегда будут целые положительные числа. Если сначала по предположению это верно, то после шага Е1 r — это целое неотрицательное число; при условии перехода к шагу Е3 оно является также ненулевым. Таким образом, поставленное требование выполнено и m и n — это действительно целые положительные числа.

3) *Ввод*. Алгоритм имеет некоторое (возможно, равное нулю) число *входных данных*, т. е. величин, которые задаются до начала его работы или определяются динамически во время его работы. Эти входные данные берутся из определенного набора объектов. Например, в алгоритме Е есть два входных значения, а именно — m и n , которые принадлежат множеству *целых положительных чисел*.

4) *Вывод*. У алгоритма есть одно или несколько *выходных данных*, т. е. величин, имеющих вполне определенную связь с входными данными. У алгоритма Е имеется только одно выходное значение, а именно — n , получаемое на шаге Е2. Это наибольший общий делитель двух входных значений.

(Можно легко *доказать*, что это число действительно является наибольшим общим делителем. После шага Е1 имеем

$$m = qn + r,$$

где q — некоторое целое число. Если $r = 0$, то m кратно n и, очевидно, в этом случае n — наибольший общий делитель m и n . Если $r \neq 0$, то любой делитель обоих чисел m и n должен быть также делителем $m - qn = r$ и любой делитель n и r — также делителем $qn + r = m$. Таким образом, множество делителей чисел $\{m, n\}$ совпадает с множеством делителей чисел $\{n, r\}$. Следовательно, пары чисел $\{m, n\}$ и $\{n, r\}$ имеют один и тот же *наибольший* общий делитель. Таким образом, шаг Е3 не изменяет ответа исходной задачи.)

5) *Эффективность*. Алгоритм обычно считается *эффективным*, если все его операторы достаточно просты для того, чтобы их можно было точно выполнить в течение конечного промежутка времени с помощью карандаша и бумаги. В алгоритме Е используются только следующие операции: деление одного целого положительного числа на другое, сравнение с нулем и присвоение одной переменной значения другой. Эти операции являются эффективными, так как целые числа можно представить на бумаге с помощью конечного числа знаков и так как существует по меньшей

мере один способ (“алгоритм деления”) деления одного целого числа на другое. Но те же самые операции были бы *неэффективными*, если бы они выполнялись над действительными числами, представляющими собой бесконечные десятичные дроби, либо над величинами, выражающими длины физических отрезков прямой, которые нельзя измерить абсолютно точно. Приведем еще один пример неэффективного шага: “Если 4 — это наибольшее целое n , при котором существует решение уравнения $w^n + x^n + y^n = z^n$ для целых положительных чисел w , x , y и z , то перейти к шагу E4”. Подобная операция не может считаться эффективной до тех пор, пока кто-либо не разработает алгоритм, позволяющий определить, действительно ли 4 является наибольшим целым числом с требуемым свойством.

Давайте попробуем сравнить понятие “алгоритм” с рецептом из кулинарной книги. Предполагается, что рецепт обладает свойством конечности (хотя и говорят, что “кто над чайником стоит, у того он не кипит”), имеет входные данные (такие, например, как яйца, мука и т. д.) и выходные данные (обед “на скорую руку” и т. п.), но хорошо известно, что ему не хватает определенности. Инструкции из кулинарных рецептов очень часто бывают неопределенными, например: “Добавьте щепотку соли”. “Щепотка” определяется как количество, “меньшее $1/8$ чайной ложки”, и что такое соль, вероятно, тоже известно всем. Но куда именно нужно добавить соль — сверху? сбоку? Инструкции “Слегка потрясите, пока смесь не станет рассыпчатой” и “Подогрейте коньяк в маленькой кастрюльке” будут вполне понятны опытному повару, но они не годятся для алгоритма. Алгоритм должен быть определен настолько четко, чтобы его указаниям мог следовать даже компьютер. Тем не менее программист может многому научиться, прочитав хорошую поваренную книгу. (Честно говоря, автор едва устоял перед искушением назвать настоящий том “Поваренная книга программиста”. Но, может, когда-нибудь он попытается написать книгу под названием “Алгоритмы для кухни”).

Следует отметить, что для практических целей ограничение, состоящее в конечности алгоритма, в сущности, является недостаточно жестким. Используемый на практике алгоритм должен иметь не просто конечное, а *достаточно ограниченное*, разумное число шагов. Например, существует алгоритм определения того, может ли игра в шахматы всегда быть выиграна белыми при условии, что не было сделано ни одной ошибки (см. упр. 2.2.3–28). Этот алгоритм позволил бы решить проблему, представляющую огромный интерес для тысяч людей, но можно биться об заклад, что окончательный ответ на данный вопрос мы не узнаем никогда. Все дело в том, что для выполнения указанного алгоритма требуется невероятно большой промежуток времени, хотя сам алгоритм и является конечным. В главе 8 будут обсуждаться конечные числа, которые велики настолько, что, в сущности, находятся за пределами нашего понимания*.

На практике нам нужны не просто алгоритмы, а *хорошие* алгоритмы в широком смысле этого слова. Одним из критериев качества алгоритма является время, необходимое для его выполнения; данную характеристику можно оценить по тому, сколько раз выполняется каждый шаг. Другими критериями являются адаптируемость алгоритма к различным компьютерам, его простота, изящество и т. д.

Часто решить одну и ту же проблему можно с помощью нескольких алгоритмов и нужно выбрать наилучший из них. Таким образом, мы попадаем в чрезвычайно

* Глава 8 не входит в данное трехтомное издание. — *Прим. ред.*

интересную и крайне важную область *анализа алгоритмов*. Предмет этой области состоит в том, чтобы для заданного алгоритма определить рабочие характеристики.

В качестве примера давайте исследуем с этой точки зрения алгоритм Евклида. Предположим, нам нужно решить следующую задачу: “Пусть задано значение n , а m может быть любым целым положительным числом. Тогда чему равно *среднее* число T_n выполнений шага E1 алгоритма E?”. Прежде всего необходимо убедиться в том, что задача имеет смысл, поскольку нам предстоит найти среднее при бесконечно большом количестве значений m . Но совершенно очевидно, что после первого выполнения шага E1 значение будет иметь только остаток от деления m на n . Поэтому все, что мы должны сделать для нахождения значения T_n , — это испытать алгоритм для $m = 1, m = 2, \dots, m = n$, подсчитать суммарное число выполнений шага E1 и разделить его на n .

А теперь рассмотрим еще один важный вопрос, касающийся *поведения* T_n как функции от n : можно ли ее аппроксимировать, например, функцией $\frac{1}{3}n$ или $\sqrt{n}$? На самом деле это чрезвычайно сложная и интересная математическая проблема, которая еще не решена окончательно; более подробно она будет рассмотрена в разделе 4.5.3. Можно доказать, что при больших значениях n T_n ведет себя, как функция $(12(\ln 2)/\pi^2) \ln n$, т. е. она пропорциональна *натуральному логарифму* n . Заметим, что коэффициент пропорциональности нельзя просто взять и угадать; чтобы определить его, нужно затратить определенные усилия. Более подробно об алгоритме Евклида, а также о других способах вычисления наибольшего общего делителя будет говориться в разделе 4.5.2.

Для обозначения области подобных исследований автор использует термин *анализ алгоритмов*. Основная идея заключается в том, чтобы взять конкретный алгоритм и определить его количественные характеристики. Время от времени мы будем также выяснять, является ли алгоритм оптимальным в некотором смысле. *Теория алгоритмов* — это совершенно другая область, в которой, в первую очередь, рассматриваются вопросы существования или не существования эффективных алгоритмов вычисления определенных величин.

До сих пор наше обсуждение алгоритмов носило достаточно общий характер, и, вероятно, “математически настроенный” читатель утвердился в мысли, что все предыдущие комментарии представляют собой очень шаткий фундамент для построения какой-либо теории алгоритмов. Поэтому давайте подведем итог данного раздела, кратко описав метод, с помощью которого понятие алгоритма можно строго обосновать в терминах математической теории множеств. Формально определим *метод вычислений* как четверку (Q, I, Ω, f) , где Q — это множество, содержащее подмножества I и Ω , а f — функция, переводящая множество Q в себя. Кроме того, f оставляет неподвижными точки множества Ω , т. е. $f(q)$ равно q для всех элементов q из множества Ω . Эти четыре элемента, Q, I, Ω, f , представляют соответственно состояния вычисления, ввод, вывод и правило вычислений. Каждое входное значение x из множества I определяет *вычисляемую последовательность* $x_0, x_1, x_2, \dots$ следующим образом:

$$x_0 = x \quad \text{и} \quad x_{k+1} = f(x_k) \quad \text{для} \quad k \geq 0. \quad (1)$$

Говорят, что вычисляемая последовательность *заканчивается через k шагов*, если k — это наименьшее целое число, для которого x_k принадлежит Ω , и что она дает

выходное значение x_k для заданного x . (Заметим, что если x_k принадлежит Ω , то и x_{k+1} принадлежит Ω , так как в этом случае $x_{k+1} = x_k$.) Некоторые вычисляемые последовательности могут никогда не заканчиваться, но *алгоритм* — это метод вычислений, который заканчивается через конечное число шагов для всех x из I .

Например, алгоритм Е в этих терминах можно формализовать следующим образом. Пусть элементами множества Q будут все величины (n) , все упорядоченные пары (m, n) и все упорядоченные четверки $(m, n, r, 1)$, $(m, n, r, 2)$ и $(m, n, p, 3)$, где m , n и p — это целые положительные числа, а r — неотрицательное целое число. Пусть I — это подмножество всех пар (m, n) , а Ω — подмножество всех величин (n) . Определим функцию f следующим образом:

$$\begin{aligned} f((m, n)) &= (m, n, 0, 1); & f((n)) &= (n); \\ f((m, n, r, 1)) &= (m, n, \text{остаток от деления } m \text{ на } n, 2); \\ f((m, n, r, 2)) &= (n), & \text{если } r = 0, & \quad (m, n, r, 3) \text{ в противном случае;} \\ f((m, n, p, 3)) &= (n, p, p, 1). \end{aligned} \tag{2}$$

Соответствие между данной записью и алгоритмом Е очевидно.

В этой формулировке понятия “алгоритм” не содержится ограничение, касающееся эффективности, о котором упоминалось ранее. Например, Q может быть множеством бесконечных последовательностей, которые нельзя вычислить с помощью карандаша и бумаги, а f может включать операции, которые простой смертный сможет выполнить не всегда. Если мы хотим ограничить понятие “алгоритм” таким образом, чтобы в нем могли содержаться только элементарные операции, то введем ограничения на элементы Q , I , Ω и f , например, следующим образом. Пусть A — это ограниченное множество букв, а A^* — множество всех строк, определенных на множестве A (т. е. множество всех упорядоченных последовательностей $x_1 x_2 \dots x_n$, где $n \geq 0$ и x_j принадлежит A для $1 \leq j \leq n$). Идея заключается в следующем: закодировать состояния вычисления таким образом, чтобы они были представлены строками из множества A^* . Теперь пусть N — целое неотрицательное число, а Q — множество всех пар (σ, j) , где σ принадлежит A^* , а j — целое число, $0 \leq j \leq N$. Пусть I — подмножество пар из Q , для которых $j = 0$, а Ω — подмножество пар из Q , для которых $j = N$. Если θ и σ — строки из A^* , то мы будем говорить, что θ входит в σ , если σ имеет вид $\alpha\theta\omega$, где α и ω — некоторые строки. И в завершение определим функцию f с помощью строк θ_j , ϕ_j и целых чисел a_j , b_j , $0 \leq j < N$ следующим образом:

$$\begin{aligned} f(\sigma, j) &= (\sigma, a_j), & \text{если } \theta_j \text{ не входит в } \sigma; \\ f(\sigma, j) &= (\alpha\phi_j\omega, b_j), & \text{если } \alpha \text{ является самой короткой строкой,} \\ & & \text{для которой } \sigma = \alpha\theta_j\omega; \\ f(\sigma, N) &= (\sigma, N). \end{aligned} \tag{3}$$

Метод вычислений, удовлетворяющий этому определению, безусловно, является эффективным. Кроме того, опыт показывает, что в таком виде можно представить любую задачу, которая решается с помощью карандаша и бумаги. Существует также много других по сути эквивалентных способов формулировки понятия эффективного метода вычислений (например, с помощью машины Тьюринга). Приведенная выше формулировка практически совпадает с той, которую дал А. А. Марков

(Markov) в своей книге *Теория алгоритмов* [Труды АН СССР, Ин-т математики. — 1954. — 42. — 376 с.], а впоследствии исправил и расширил Н. М. Нагорный (Nagorny) (М.: Наука, 1984).

УПРАЖНЕНИЯ

1. [10] В тексте показано, как взаимно заменить значения переменных m и n с помощью символа замены, а именно — полагая $t \leftarrow m$, $m \leftarrow n$, $n \leftarrow t$. Покажите, как в результате ряда замен можно преобразовать *четверку* переменных (a, b, c, d) в (b, c, d, a) . Другими словами, новое значение переменной a должно стать равным первоначальному значению b и т. д. Постарайтесь выполнить преобразование с помощью минимального числа замен.
2. [15] Докажите, что в начале выполнения шага E1 m всегда больше n , за исключением, возможно, только первого случая выполнения этого шага.
3. [20] Измените алгоритм E (из соображений эффективности) таким образом, чтобы исключить из него все тривиальные операции замены типа “ $m \leftarrow n$ ”. Запишите этот новый алгоритм в стиле алгоритма E и назовите его алгоритмом F.
4. [16] Чему равен наибольший общий делитель чисел 2 166 и 6 099?
- 5. [12] Покажите, что для процедуры чтения книг этой серии, приведенной в предисловии, не хватает трех из пяти условий для того, чтобы она стала настоящим алгоритмом! Укажите также некоторые различия в форме записи этой процедуры и алгоритма E.
6. [20] Чему равно T_5 (среднее число случаев выполнения шага E1 при $n = 5$)?
- 7. [M21] Пусть m известно, а n — любое целое положительное число. Пусть U_m — среднее число случаев выполнения шага E1 из алгоритма E. Покажите, что U_m четко определено. Существует ли какая-либо связь между U_m и T_m ?
8. [M25] Придумайте эффективный формальный алгоритм вычисления наибольшего общего делителя целых положительных чисел m и n , определив соответствующим образом θ_j , ϕ_j , a_j , b_j (как в формулах (3)). Пусть входные данные представлены строкой $a^m b^n$, т. е. за a , взятым m раз, следует b , взятое n раз. Постарайтесь найти самое простое решение, насколько это возможно. [Указание. Воспользуйтесь алгоритмом E, но вместо деления на шаге E1 присвойте $r \leftarrow |m - n|$, $n \leftarrow \min(m, n)$.]
- 9. [M30] Предположим, что $C_1 = (Q_1, I_1, \Omega_1, f_1)$ и $C_2 = (Q_2, I_2, \Omega_2, f_2)$ — методы вычислений. Например, C_1 может обозначать алгоритм E (см. формулу (2)) при условии, что m и n ограничены по величине, а C_2 — компьютерную программу, реализующую алгоритм E. (Тогда можно считать, что Q_2 — это набор всех состояний машины, т. е. всех возможных конфигураций ее памяти и регистров, f_2 определяет элементарную машинную операцию, а I_2 — начальное состояние, которое включает программу определения наибольшего общего делителя, а также значения m и n .)
Сформулируйте теоретико-множественное определение понятия “ C_2 является представлением C_1 ” или “ C_2 имитирует C_1 ”. Интуитивно это означает, что любая вычисляемая последовательность C_1 имитируется C_2 , за исключением того, что у C_2 может быть больше шагов, на которых выполняются вычисления, и можно получить больше информации из состояний. (Таким образом, мы получим точную интерпретацию утверждения “Программа X является реализацией алгоритма Y ”.)

1.2. МАТЕМАТИЧЕСКОЕ ВВЕДЕНИЕ

В ЭТОМ РАЗДЕЛЕ мы рассмотрим математические обозначения, используемые в книге *Искусство программирования*, и выведем основные формулы, которые будут часто применяться. Даже читатель, которого не интересуют сложные математические выкладки, должен понять *смысл* формул, чтобы иметь возможность пользоваться готовыми результатами.

Математические обозначения используются в этой книге для двух основных целей: для описания частей алгоритма и для анализа его рабочих характеристик. В предыдущем разделе были приведены обозначения, используемые в описаниях алгоритмов; как вы уже знаете, они достаточно просты. Но для анализа алгоритмов нам нужны другие, специальные обозначения.

Большинство рассматриваемых нами алгоритмов будет сопровождаться математическими подсчетами, определяющими ожидаемую скорость выполнения алгоритма. В этих вычислениях будут использоваться знания практически из всех разделов математики (для изложения всех используемых математических понятий потребовалась бы отдельная книга). Тем не менее для выполнения большинства вычислений используются знания математики на уровне школьного курса алгебры, поэтому читатель, знакомый с элементарными вычислениями, сможет разобраться почти во всех математических выкладках. В случаях, когда нам понадобятся более глубокие результаты из теории комплексного переменного, теории групп, теории чисел, теории вероятностей и т. д., материал будет излагаться как можно проще либо будет дана ссылка на другие источники информации.

Математические методы, используемые при анализе алгоритмов, имеют свои отличительные особенности. Например, нам довольно часто придется выполнять суммирование конечного числа рациональных чисел или решать рекуррентные уравнения. Подобные темы обычно очень поверхностно освещаются при чтении математических дисциплин, поэтому назначение следующих разделов — не только потренироваться в использовании обозначений, но и проиллюстрировать типы и методы вычислений, которые будут нам особенно необходимы.

Важное замечание. Хотя в следующих разделах содержатся обширные сведения из различных областей математики, которые совершенно необходимы для изучения компьютерных алгоритмов, большинство читателей сначала не увидят особой связи между этим материалом и программированием (за исключением раздела 1.2.1, в котором такая связь очевидна). Читатель, конечно, может сразу приступить к внимательному изучению следующих разделов, приняв на веру слова автора о том, что данные темы действительно очень важны. Но я считаю, что главной побудительной силой является интерес. Поэтому, наверное, лучше поступить иначе: *сначала просто просмотреть этот раздел, а затем* (после знакомства с применением указанных методов в следующих главах) *снова вернуться к нему для более глубокого изучения.* Ведь если во время первого чтения книги потратить слишком много времени на изучение данного математического материала, то можно никогда не дойти до вопросов программирования! Тем не менее каждый читатель должен ознакомиться хотя бы с общим содержанием этих разделов и даже во время первого чтения попытаться выполнить несколько упражнений. Разделу 1.2.10 следует уделить особое внимание, так как это отправная точка для большей части

теоретического материала, излагаемого впоследствии. В разделе 1.3 происходит резкий переход от “чистой математики” к “чистому программированию.”

Более подробно последующий материал излагается в книге R. Graham, D. Knuth, O. Patashnik, *Concrete Mathematics*, second edition (Reading, Mass.: Addison-Wesley, 1994) (Грэхем Р., Кнут Д., Паташник О. *Конкретная математика*. — М.: Мир, 1998). Впоследствии, ссылаясь на эту книгу, мы будем называть ее просто *CMath*.

1.2.1. Математическая индукция

Пусть $P(n)$ — некоторое утверждение, касающееся целого числа n , например, “ n умножить на $(n + 3)$ — четное число” или “если $n \geq 10$, то $2^n > n^3$ ”. Предположим, нам нужно доказать, что утверждение $P(n)$ верно для всех положительных целых чисел n . Существует важный метод доказательства этого факта, который состоит в следующем.

- а) Доказать, что $P(1)$ верно.
- б) Доказать, что “если $P(1), P(2), \dots, P(n)$ справедливы, то $P(n + 1)$ также справедливо”; это доказательство должно иметь силу для любого целого положительного n .

В качестве примера рассмотрим следующие известные с древних времен равенства, которые многие исследователи открывали независимо друг от друга:

$$\begin{aligned} 1 &= 1^2, \\ 1 + 3 &= 2^2, \\ 1 + 3 + 5 &= 3^2, \\ 1 + 3 + 5 + 7 &= 4^2, \\ 1 + 3 + 5 + 7 + 9 &= 5^2. \end{aligned} \tag{1}$$

В общем виде эти равенства можно записать следующим образом:

$$1 + 3 + \dots + (2n - 1) = n^2. \tag{2}$$

Давайте назовем это утверждение $P(n)$ и докажем, что оно верно для любого положительного n . Согласно методу, описанному выше, имеем следующее.

- а) “ $P(1)$ верно, так как $1 = 1^2$.”
- б) “Если все утверждения $P(1), \dots, P(n)$ справедливы, то, в частности, верно и $P(n)$; следовательно, выполняется соотношение (2). Добавляя к обеим частям этого уравнения $2n + 1$, получаем

$$1 + 3 + \dots + (2n - 1) + (2n + 1) = n^2 + 2n + 1 = (n + 1)^2.$$

Таким образом, утверждение $P(n + 1)$ также справедливо.”

Этот метод можно считать алгоритмической процедурой доказательства. В самом деле, следующий алгоритм дает доказательство утверждения $P(n)$ для любого целого положительного n в предположении, что пп. (а) и (б) уже выполнены.

Алгоритм I (*Построить доказательство*). Для заданного целого положительного числа n этот алгоритм (рис. 2) выдаст доказательство того, что утверждение $P(n)$ верно.

- I1.** [Доказать $P(1)$.] Присвоить $k \leftarrow 1$ и в соответствии с п. (а) выдать доказательство утверждения $P(1)$.
- I2.** [$k = n$?] Если $k = n$, закончить выполнение алгоритма; требуемое доказательство выдано.
- I3.** [Доказать $P(k + 1)$.] Согласно п. (b) выдать доказательство того, что “Если все утверждения $P(1), \dots, P(k)$ справедливы, то $P(k + 1)$ также справедливо”. Вывести фразу “Мы уже доказали, что если утверждения $P(1), \dots, P(k)$ верны, то верно и $P(k + 1)$ ”.
- I4.** [Увеличить k .] Увеличить k на 1 и перейти к шагу I2. ■

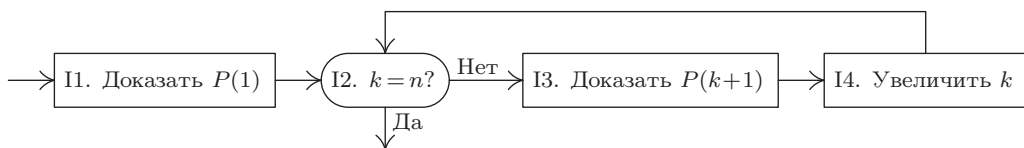


Рис. 2. Алгоритм I: математическая индукция.

Поскольку этот алгоритм выдает доказательство утверждения $P(n)$ для любого заданного n , метод доказательства, сформулированный в пп. (а) и (b), логически обоснован. Он называется *доказательством методом математической индукции*.

Понятие математической индукции следует отличать от того, что в научной практике обычно называют индуктивным методом. Данный метод заключается в том, что ученый делает некоторые наблюдения и создает “по индукции” общую теорию или выдвигает гипотезу, объясняющую эти факты. Например, на основании пяти соотношений (1), приведенных выше, мы могли бы сформулировать соотношение (2). В этом смысле индукция — не более чем догадка или попытка объяснить конкретную ситуацию; математики называют это эмпирическим результатом или предположением.

Для того чтобы прояснить суть дела, рассмотрим еще один поучительный пример. Пусть $p(n)$ обозначает количество *разбиений* числа n , т. е. количество различных способов записи числа n в виде суммы целых положительных чисел (порядок слагаемых значения не имеет). Так как для числа 5 существует ровно семь таких способов записи, т. е.

$$1 + 1 + 1 + 1 + 1 = 2 + 1 + 1 + 1 = 2 + 2 + 1 = 3 + 1 + 1 = 3 + 2 = 4 + 1 = 5,$$

то $p(5) = 7$. На самом деле установить первые пять значений функции $p(n)$ довольно легко:

$$p(1) = 1, \quad p(2) = 2, \quad p(3) = 3, \quad p(4) = 5, \quad p(5) = 7.$$

На этом основании мы могли бы предварительно сформулировать по индукции предположение о том, что последовательность $p(2), p(3), \dots$ пробегает множество

простых чисел. Для проверки данной гипотезы продолжаем вычисления и находим $p(6)$. Ура! $p(6) = 11$, что подтверждает наше предположение.

[Но, к сожалению, оказывается, что $p(7)$ равно 15. Увы, все идет насмарку, и приходится начинать сначала. Известно, что значения $p(n)$ отличаются довольно сложным поведением, хотя С. Рамануджану (S. Ramanujan) удалось угадать и доказать много замечательных фактов, касающихся этих чисел. Более подробную информацию можно найти в книге G. H. Hardy, *Ramanujan* (London: Cambridge University Press, 1940), гл. 6 и 8.]

Математическая индукция не имеет ничего общего с тем индуктивным методом, который мы только что описали. Это не догадка, а неопровержимое доказательство утверждения; скажу больше: это доказательство бесконечного числа утверждений, по одному для каждого n . “Индукцией” этот метод назван только потому, что сначала нужно выдвинуть предположение о том, *что* нужно доказать, а *затем* уже применять метод математической индукции. Начиная с этого момента, слово “индукция” в книге будет использоваться только для обозначения доказательства методом математической индукции.

Существует геометрический способ доказательства соотношения (2). На рис. 3 для $n = 6$ показано, что n^2 клеток разбиты на группы $1 + 3 + \dots + (2n - 1)$ клеток. Но в конечном счете этот рисунок можно считать “доказательством” только в случае, если мы покажем, что данное построение можно выполнить для любого n . А это, в сущности, и будет доказательством по индукции.

В нашем доказательстве соотношения (2) был использован только частный случай (b); мы просто показали, что из справедливости $P(n)$ следует справедливость $P(n + 1)$. Это очень важный случай, который встречается довольно часто, но в следующем примере будут проиллюстрированы более широкие возможности метода математической индукции. Определим *последовательность Фибоначчи* $F_0, F_1, F_2, \dots$ с помощью такого правила: $F_0 = 0, F_1 = 1$, а каждый последующий член равен сумме двух предыдущих. Таким образом, первые члены этой последовательности выглядят так: 0, 1, 1, 2, 3, 5, 8, 13, ... (более подробно мы изучим эту последовательность в разделе 1.2.8). А теперь докажем, что если через ϕ обозначить число $(1 + \sqrt{5})/2$, то имеем

$$F_n \leq \phi^{n-1} \quad (3)$$

для всех целых положительных n . Назовем эту формулу утверждением $P(n)$.

Если $n = 1$, то $F_1 = 1 = \phi^0 = \phi^{n-1}$, поэтому п. (a) выполнен. Переходя к п. (b), заметим сначала, что $P(2)$ также справедливо, поскольку $F_2 = 1 < 1.6 < \phi^1 = \phi^{2-1}$. А теперь, если все $P(1), P(2), \dots, P(n)$ справедливы и $n > 1$, то мы знаем, в частности, что справедливы $P(n - 1)$ и $P(n)$. Поэтому $F_{n-1} \leq \phi^{n-2}$ и $F_n \leq \phi^{n-1}$. Складывая данные неравенства, получаем

$$F_{n+1} = F_{n-1} + F_n \leq \phi^{n-2} + \phi^{n-1} = \phi^{n-2}(1 + \phi). \quad (4)$$

					11
				9	
			7		
		5			
	3				
1					

Рис. 3. Нечетные числа в сумме дают квадрат.

Важное свойство числа ϕ , которое и является причиной первоочередного выбора этого числа для данной задачи, состоит в том, что

$$1 + \phi = \phi^2. \quad (5)$$

Подставив (5) в (4), получим $F_{n+1} \leq \phi^n$, а это и есть утверждение $P(n+1)$. Таким образом, п. (b) выполнен и формула (3) доказана методом математической индукции. Обратите внимание, что п. (b) мы выполняли двумя различными способами: непосредственно доказали $P(n+1)$ при $n=1$ и использовали индуктивный метод при $n>1$. Это было необходимо, так как при $n=1$ ссылка на $P(n-1) = P(0)$ была бы незаконной.

Математическую индукцию можно использовать также для доказательства фактов, касающихся *алгоритмов*. Давайте рассмотрим следующее обобщение алгоритма Евклида.

Алгоритм Е (*Обобщенный алгоритм Евклида*). Даны два целых положительных числа m и n . Требуется найти их наибольший общий делитель* d и два целых числа a и b , таких, что $am + bn = d$.

Е1. [Инициализация.] Присвоить $a' \leftarrow b \leftarrow 1$, $a \leftarrow b' \leftarrow 0$, $c \leftarrow m$, $d \leftarrow n$.

Е2. [Деление.] Пусть q и r — это частное и остаток от деления c на d соответственно. (Тогда $c = qd + r$, где $0 \leq r < d$.)

Е3. [Остаток — нуль?] Если $r = 0$, то выполнение алгоритма прекращается; в этом случае имеем $am + bn = d$, как и требовалось.

Е4. [Повторение цикла.] Присвоить $c \leftarrow d$, $d \leftarrow r$, $t \leftarrow a'$, $a' \leftarrow a$, $a \leftarrow t - qa$, $t \leftarrow b'$, $b' \leftarrow b$, $b \leftarrow t - qb$ и вернуться к шагу Е2. ■

Если изъять из алгоритма переменные a , b , a' и b' и использовать m и n в качестве вспомогательных переменных c и d , то получим старый алгоритм 1.1Е. В новой версии алгоритма выполняется немного больше вычислений, так как необходимо определить коэффициенты a и b . Предположим, что $m = 1\,769$ и $n = 551$. Тогда последовательно (после шага Е2) имеем:

a'	a	b'	b	c	d	q	r
1	0	0	1	1769	551	3	116
0	1	1	-3	551	116	4	87
1	-4	-3	13	116	87	1	29
-4	5	13	-16	87	29	3	0

Проверяя полученные результаты, убеждаемся в том, что все правильно, так как $5 \times 1769 - 16 \times 551 = 8845 - 8816 = 29$, т. е. мы получили наибольший общий делитель чисел 1 769 и 551.

Теперь нужно *доказать*, что рассматриваемый алгоритм работает правильно при любых m и n . Для этого попробуем применить метод математической индукции к следующему утверждению $P(n)$: “Алгоритм Е дает правильное решение для заданного n и всех целых m ”. Но провести подобное доказательство не так-то просто, поэтому нужно доказать сначала несколько дополнительных фактов. После

* Сокращенно — gcd (greatest common divisor). — Прим. перев.

некоторого изучения проблемы выясняется, что нужно доказать какой-то факт, связанный с коэффициентами a , b , a' и b' . Этот факт заключается в том, что равенства

$$a'm + b'n = c, \quad am + bn = d \quad (6)$$

верны в каждом случае выполнения шага Е2. Данные равенства можно доказать непосредственно, заметив, что они безусловно справедливы после первого выполнения шага Е2 и что шаг Е4 не меняет это положение вещей (см. упр. 6).

Теперь мы готовы индукцией по n доказать, что алгоритм Е работает правильно. Если m кратно n , то очевидно, что алгоритм работает правильно, поскольку его работа заканчивается на шаге Е3 в первом же цикле и мы получаем верный результат. Это происходит всегда, когда $n = 1$. Поэтому остается провести доказательство для случая, когда $n > 1$ и m не является кратным n . В такой ситуации в первом цикле осуществляется переход к шагу Е4 и выполняются операции присвоения $c \leftarrow n$, $d \leftarrow r$. И так как $r < n$, *по индукции можно предположить*, что окончательное значение d — это наибольший общий делитель чисел n и r . Из доказательства, приведенного в разделе 1.1, следует, что пары $\{m, n\}$ и $\{n, r\}$ имеют одинаковые наибольшие общие делители и, в частности, один и тот же наибольший общий делитель. Значит, d — это наибольший общий делитель чисел m и n и согласно (6) $am + bn = d$.

Фраза, которая в приведенном выше доказательстве выделена курсивом, является иллюстрацией того общепринятого условного языка, который так часто используется в доказательствах методом индукции. Например, выполняя п. (b), вместо того чтобы сказать “Теперь предположим, что утверждения $P(1), P(2), \dots, P(n)$ справедливы, и на этом основании докажем справедливость утверждения $P(n+1)$ ”, мы будем говорить просто “Теперь докажем утверждение $P(n)$; *по индукции мы можем предположить*, что $P(k)$ верно для любого $1 \leq k < n$ ”.

Если хорошо вдуматься и посмотреть на все вышесказанное с несколько иной точки зрения, то перед нами предстанет общий метод доказательства корректности *любого* алгоритма. Идея состоит в том, чтобы взять блок-схему некоторого алгоритма и к каждой стрелке добавить примечание о текущем состоянии дел в тот момент, который соответствует стрелке. На рис. 4 эти примечания (мы их будем называть также утверждениями) обозначены через $A1, A2, \dots, A6$. (Все они сделаны при дополнительном условии, что переменные принимают только целые значения; это условие опущено с целью экономии места.) В утверждении $A1$ даются первоначальные предположения о входных данных алгоритма, а в $A4$ формулируется положение о том, что мы хотим доказать по поводу выходных значений a , b и d .

Общий метод заключается в том, чтобы для каждого блока на блок-схеме доказать следующее:

если любое утверждение, которое соответствует стрелке, ведущей к блоку, верно до выполнения операции из этого блока, то все утверждения, которые соответствуют стрелкам, ведущим от блока, также верны после выполнения операции. (7)

Согласно описанному методу для нашего примера мы должны доказать, что если до выполнения шага Е2 верно $A2$ либо $A6$, то после выполнения этого шага верно $A3$. (В данном случае утверждение $A2$ является более сильным, чем $A6$, т. е. из $A2$

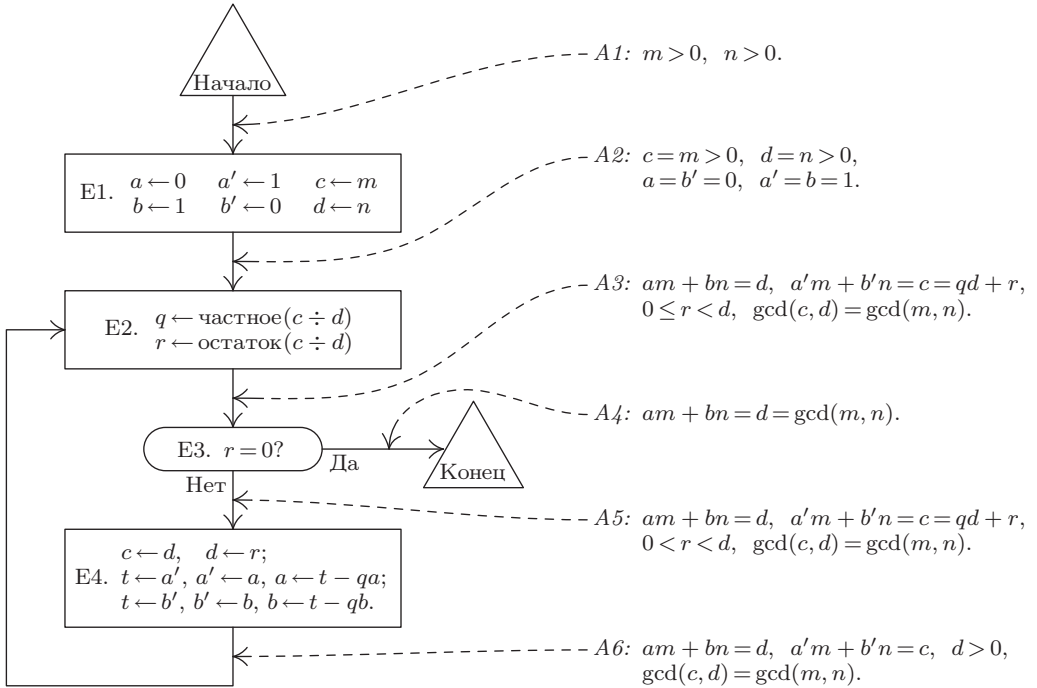


Рис. 4. Блок-схема алгоритма Е, дополненная примечаниями, которые доказывают правильность работы алгоритма.

следует А6. Поэтому нам достаточно доказать, что выполнение А6 до шага Е2 влечет за собой выполнение А3 после этого шага. Заметим, что условие $d > 0$ необходимо в А6 для того, чтобы операция Е2 имела смысл.) Нужно показать также, что из А3 (при условии, что $r = 0$) следует А4, из А3 (при условии, что $r \neq 0$) следует А5 и т. д. Все это доказывается очень просто.

Если доказать утверждение (7) для каждого блока, то все примечания к стрелкам будут верны в любом случае выполнения алгоритма. Теперь мы можем применить индукцию по числу шагов, т. е. по числу стрелок в блок-схеме. При прохождении первой стрелки (той, которая выходит из блока “Начало”) утверждение А1 верно, поскольку мы всегда исходим из предположения, что входные значения удовлетворяют заданным условиям. Таким образом, утверждение, которое соответствует первой стрелке, верно. Если утверждение, которое соответствует n -й стрелке, верно, то согласно (7) утверждение, которое соответствует $(n + 1)$ -й стрелке, тоже верно.

Исходя из этого общего метода доказательство правильности заданного алгоритма, очевидно, сводится к нахождению правильных утверждений, соответствующих стрелкам блок-схемы. Как только данное начальное препятствие будет преодолено, останется лишь рутинная работа, связанная с доказательством того, что каждое утверждение на входе в блок влечет за собой утверждение на выходе из блока. В действительности после того как вы придумаете самые трудные из этих утверждений, найти все остальные уже не составит труда. Скажем, если даны утверждения А1,

A_4 и A_6 , уже понятно, какими должны быть утверждения A_2 , A_3 и A_5 . В нашем примере самых больших творческих усилий потребует доказательство утверждения A_6 ; все остальное, в принципе, должно получиться автоматически. Поэтому я и не пытался давать для алгоритмов, приведенных в книге, формальные доказательства с той степенью детализации, которая отражена на рис. 4. Вполне достаточно сформулировать только главные утверждения. Обычно они приводятся либо в ходе обсуждения алгоритма, либо даются в скобках в тексте самого алгоритма.

Этот подход к доказательству корректности алгоритма имеет и другой, еще более важный аспект: *он отражает способ нашего понимания алгоритма*. Если помните, в разделе 1.1 я предупреждал о том, чтобы вы не читали алгоритмы, как роман. Я рекомендую проверять работу алгоритма на примере одного-двух наборов входных данных. И это не случайно, так как пробная “прогонка” алгоритма поможет вам мысленно сформулировать утверждения, соответствующие стрелкам на блок-схеме. Автор твердо убежден, что истинная уверенность в корректности алгоритма приходит только тогда, когда мысленно сформулированы все утверждения, приведенные на рис. 4. Отсюда следуют важные психологические выводы, касающиеся передачи алгоритма от одного лица к другому. Речь идет о том, что, объясняя алгоритм кому-либо другому, всегда следует явно формулировать основные утверждения, которые трудно получить автоматически. Например, в случае алгоритма Е нужно обязательно упомянуть утверждение A_6 .

Но бдительный читатель, конечно, заметил зияющую брешь в нашем последнем доказательстве алгоритма Е. Из него нигде не следует, что алгоритм обладает свойством конечности, т. е. рано или поздно его выполнение завершится. Мы доказали только, что *если* алгоритм конечен, то он дает правильный результат!

(Например, заметим, что алгоритм Е по-прежнему имеет смысл, если его переменные m , n , c , d и r принимают значения типа $u + v\sqrt{2}$, где u и v — целые числа*. Переменные q , a , b , a' , b' должны по-прежнему принимать целые значения. Если, например, на вход подать значения $m = 12 - 6\sqrt{2}$ и $n = 20 - 10\sqrt{2}$, то на выходе будет получен “наибольший общий делитель” $d = 4 - 2\sqrt{2}$ и коэффициенты $a = +2$, $b = -1$. Даже при таком расширении исходных предположений доказательства утверждений от A_1 до A_6 остаются в силе. Следовательно, на любом этапе выполнения этой процедуры все утверждения верны. Но если начать со значений $m = 1$ и $n = \sqrt{2}$, то вычисления никогда не закончатся (см. упр. 12). Следовательно, из доказательства утверждений A_1 – A_6 еще *не* следует, что алгоритм конечен.)

Доказательства конечности алгоритмов обычно проводят отдельно. Но в упр. 13 показано, что во многих важных случаях приведенный выше метод *можно* обобщить таким образом, чтобы включить доказательство конечности в виде промежуточного результата.

Итак, мы уже дважды доказали правильность алгоритма Е. Чтобы быть последовательными до конца, нам следовало бы попытаться доказать, что первый алгоритм в этом разделе, а именно — алгоритм I, также корректен. Ведь, в сущности, мы использовали алгоритм I, чтобы показать корректность любого доказательства по индукции. Но если мы попытаемся *доказать*, что алгоритм I работает правильно, то попадем в затруднительное положение: мы не сможем сделать это, не воспользовавшись снова индукцией! Итак, получается замкнутый круг.

* Определение деления с остатком в этом случае приведено в решении к упр. 12. — *Прим. ред.*

В последнее время *любое* свойство целых чисел принято доказывать с помощью индукции в ту или иную сторону. Ведь если мы обратимся к основным понятиям, то увидим, что целые числа, в сущности, *определяются* по индукции. Поэтому можно принять в качестве аксиомы утверждение о том, что любое целое положительное число n либо равно 1, либо может быть получено, если взять 1 за исходное значение и последовательно прибавлять по единице. Этого достаточно, чтобы доказать правильность алгоритма I. [Более подробно фундаментальные понятия, связанные с целыми числами, рассматриваются в статье Leon Henkin, On Mathematical Induction, АММ 67 (1960), 323–338.]

Таким образом, метод математической индукции глубоко связан с понятием числа. Первым европейцем, применившим в 1575 году метод математической индукции для получения строгих доказательств, был итальянский ученый Франческо Мауролико (Francesco Maurolico). В начале 17 века Пьер де Ферма (Pierre de Fermat) усовершенствовал этот метод; он называл его методом бесконечного спуска. Это понятие явно используется также в последних трудах Блеза Паскаля (Blaise Pascal) (1653). Термин “математическая индукция”, видимо, был придуман А. Де Морганом (A. De Morgan) в начале 19 века. [См. АММ 24 (1917), 199–207; 25 (1918), 197–201; Arch. Hist. Exact Sci. 9 (1972), 1–21.] Более подробно метод математической индукции рассматривается в книге Д. По́йа (G. Pólya) *Induction and Analogy in Mathematics* (Princeton, N. J.: Princeton University Press, 1954), Chapter 7 (“Математика и правдоподобные рассуждения”; т. 1, “Индукция и аналогия в математике” (М.: Изд-во иностр. лит., 1957), гл. 7).

Описанный выше метод доказательства алгоритмов с помощью утверждений, соответствующих стрелкам, и индукции, по существу, принадлежит Р. В. Флойду (R. W. Floyd). Он показал, что смысловое определение каждой операции в языке программирования можно сформулировать в виде логического правила. Это правило точно определяет, какие утверждения могут быть верны после выполнения операции, если известно, какие утверждения верны до ее выполнения. [См. “Assigning Meanings to Programs”, Proc. Symp. Appl. Math., Amer. Math. Soc., 19 (1967), 19–32.] Аналогичные идеи были независимо высказаны Питером Науром (Peter Naur), BIT 6 (1966), 310–316, который называл утверждения, соответствующие стрелкам на блок-схемах, общими снимками (general snapshots). Необходимо уточнить, что понятие “инвариант” было введено Ч. Э. Р. Хоаром (C. A. R. Hoare) (например, см. CACM 14 (1971), 39–45). В более поздних публикациях считалось, что выгоднее изменить направление, заданное Флойдом, на противоположное, т. е. что нужно исходить из утверждения, которое должно выполняться *после* операции, и доказывать, что “самое слабое предусловие”, которое должно иметь место *до* выполнения этой операции, на самом деле имеет место. Подобный подход позволяет разрабатывать новые алгоритмы, выбирая в качестве отправной точки характеристики желаемых выходных данных и двигаясь в обратном направлении (т. е. вверх по блок-схеме). При выполнении этого условия полученные алгоритмы обязательно будут корректными. [См. E. W. Dijkstra, CACM 18 (1975), 453–457; A Discipline of Programming (Prentice-Hall, 1976).]

На самом деле зачатки идеи индуктивных утверждений появились в 1946 году, в то время, когда Г. Г. Голдстейн (H. H. Goldstine) и Дж. фон Нейман (J. von Neumann) изобрели блок-схемы. Их первоначальные блок-схемы включали в себя “блоки