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 *arithmas* [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 Амударьи 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 H. 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 E (*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.

E1. [Нахождение остатка.] Разделим m на n, и пусть остаток от деления будет равен r (где $0 \le r < n$).

E2. [Сравнение с нулем.] Если $r = 0$, то выполнение алгоритма прекращается; n — искомое значение.

E3. [Замещение.] Присвоить $m \leftarrow n$, $n \leftarrow r$ и вернуться к шагу E1. ▮

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 E), а шагам алгоритма — эта

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

же 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 делаются 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 ссылаться 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.

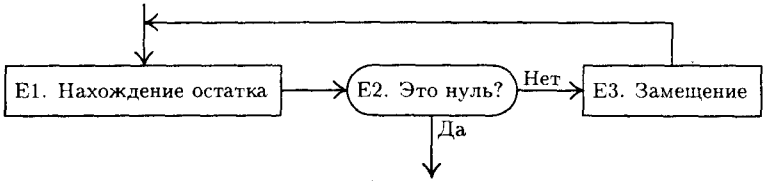

Рис. 1. Блок-схема алгоритма E.

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

Обратите внимание, что на шаге E3 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 E3 указание "Вернуться к шагу E1" явным образом определяет порядок вычислений. На шаге E2 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 E3", но это совершенно излишне.

Жирная вертикальная черточка "∎", помещенная в конце шага E3, 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: $v_1, v_2, \ldots, v_n$. 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 Е4". 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 $m = 1$, $m = 2$, ..., $m = n$, 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 $\left(12(\ln 2)/\pi^2\right)\ln n$, 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* $x_0, x_1, x_2, \ldots$ следующим образом:

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

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 E 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:

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

Соответствие между данной записью и алгоритмом E 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 $x_1x_2\ldots x_n$, 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 алгорифмов* [*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".)