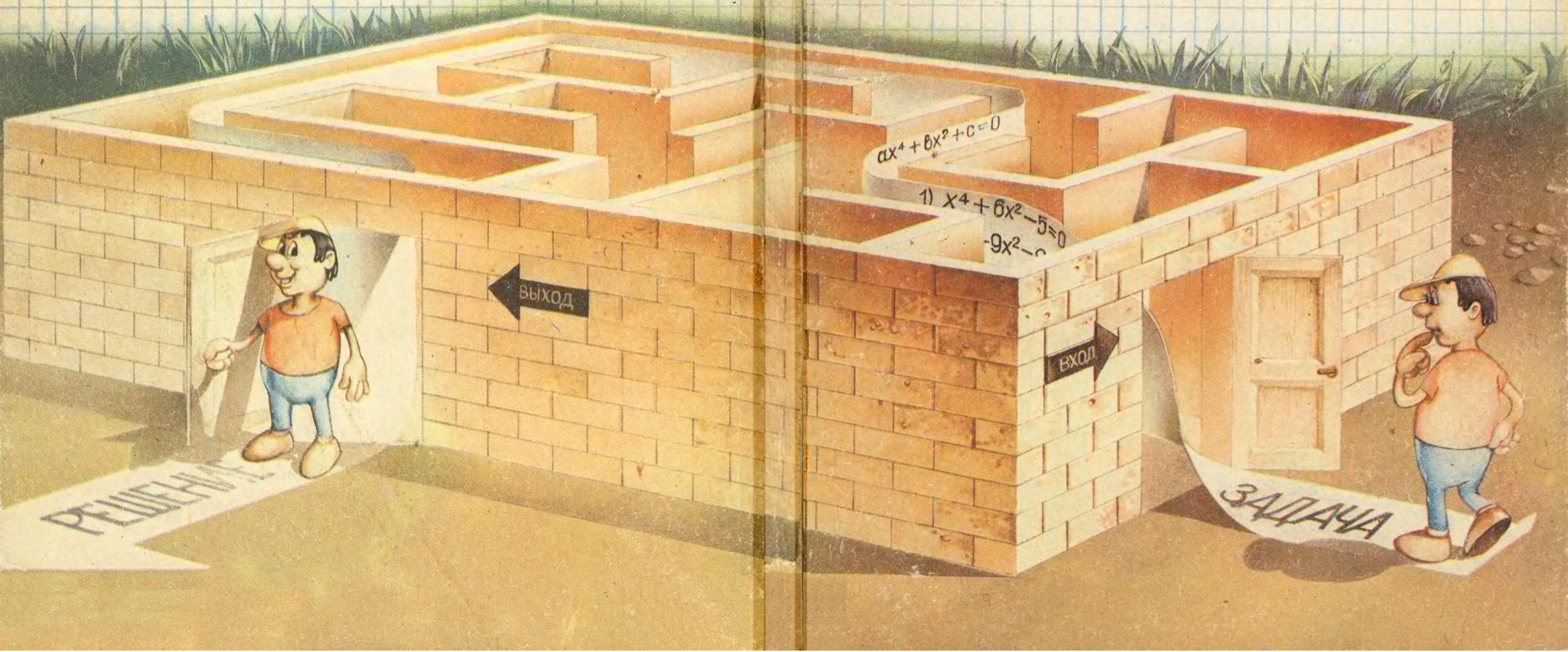
30 к.

Ю.А.Макаренков
А.А.Столяр

Что такое АЛГОРИТМ?

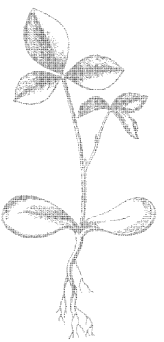


Ю.А.Макаренков
А.А.Столяр

Что такое АЛГОРИТМ?

Беседы со старшеклассником

МИНСК
«НАРОДНАЯ
АСВЕТА»
1989



ББК 22.12
М 15

Рецензент канд. физ.-мат. наук А. Ш. Блох

Макаренков Ю. А., Столяр А. А.

М 15 Что такое алгоритм?: Беседы со старшеклассником.— Мн.: Нар. асвета, 1989.— 127 с.: ил.

ISBN 5-341-00162-1.

В популярной форме рассказывается о важнейшем понятии современной математики и информатики — алгоритме. Формируется понятие алгоритма, рассматриваются приложения алгоритмов к решению игровых и прикладных задач и программированию. К упражнениям, помещенным в книге, даны указания и ответы.

4802020000—141
М————————162—88 **ББК 22.12**
М303(03)—89

ISBN 5-341-00162-1

Алгоритмы... кругом алгоритмы!

(Вместо предисловия)

Один из героев великого французского писателя Мольера, месье Журден, был страшно удивлен, узнав, что всю жизнь пользуется прозой. Мы с Вами, дорогой наш читатель, тоже можем удивляться, узнав, что всю жизнь мы исполняем огромное число всякого рода алгоритмов. Мы буквально окружены тесным кольцом алгоритмов, не сознавая, разумеется, этого и, по существу, не зная даже, что такое алгоритм.

В каждодневной жизни человеку приходится решать большое число разного рода задач, в широком смысле этого слова, не только математических или физических, которые требуют применения определенных алгоритмов.

Когда мы переходим улицу на регулируемом светофором перекрестке, мы выполняем определенный алгоритм, когда же переходим улицу в месте, не регулируемом светофором, выполняем другой алгоритм (эти алгоритмы заданы правилами уличного движения). Когда мы вытачиваем деталь на станке, то выполняем определенный алгоритм. Когда приготовляем чай, пользуемся определенным алгоритмом (иногда заданным инструкцией, напечатанной на упаковке). Когда медицинская сестра берет у нас кровь из пальца на анализ, она исполняет определенный алгоритм, и когда она делает нам инъекцию, то опять исполняет определенный алгоритм (этим алгоритмам ее обучали в медицинском училище). И когда мы берем книги в библиотеке, мы выполняем определенные правила пользования библиотечными книгами, т. е. тоже определенный алгоритм. И когда...

Разве возможно перечислить все задачи, при решении которых мы исполняем определенные алгоритмы? Одна школьная математика содержит огромное разнообразие алгоритмов, начиная от алгоритмов выполнения арифметических действий над натуральными числами, дробями и заканчивая алгоритмами решения всякого рода уравнений, неравенств, систем уравнений и т. д. Хотя, разумеется, мы очень редко применяем слово «алгоритм». В самой математике, где возникло понятие алгоритма, многие столетия разрабатывались алгоритмы для решения все новых и новых классов задач и при этом мате-

матики не задумывались над тем, что такое алгоритм.

Действительно, если найден общий метод решения любой задачи из нового класса задач, нам незачем задумываться над вопросом, является ли этот метод алгоритмом. Метод есть и мы его применяем к любой задаче данного класса задач, для которого он предназначен. Так формулу корней квадратного уравнения мы используем при решении любого квадратного уравнения. Интуитивно ясно, что данная формула задает общий метод (алгоритм) решения любого квадратного уравнения.

Почему же в течение многих веков в математике, где все строго определяется и доказывается, обходились лишь интуитивным понятием алгоритма?

Один из создателей математических теорий алгоритмов советский математик А. А. Марков (1903—1979) считал, что это было допустимо, пока термин «алгоритм» встречался в математике лишь в высказываниях следующего типа: «Для решения таких-то задач имеется алгоритм, и вот в чем он состоит».

По той же причине интуитивное понятие алгоритма достаточно и когда ставится вопрос о переводе уже имеющегося алгоритма в программу для решения задачи на ЭВМ. Программа — это лишь запись алгоритма на языке, понимаемом данной ЭВМ.

В математике часто возникают так называемые алгоритмические проблемы, когда требуется найти единый метод (алгоритм) для решения любой задачи из данного класса однотипных задач. Алгоритмические проблемы возникали и решались в различных областях математики на протяжении всей ее истории. Однако в первые десятилетия XX века накопилось много классов задач, для решения которых алгоритма найти не удавалось. Усилия, прилагавшиеся к нахождению алгоритмов решения этих классов задач, привели к мысли: а вдруг для того или иного класса просто невозможно найти алгоритм, т. е. искомый алгоритм не существует?

Если это действительно так, то необходимо прекратить поиск, ибо безнадежно искать то, чего нет. Но тогда возникает необходимость в доказательстве несуществования искомого алгоритма. А чтобы доказать несуществование алгоритма для решения некоторого класса задач, нужно уже знать точно, что такое алгоритм, т. е. надо перевести интуитивное понятие алгоритмов в точное, математическое понятие.

В рамках математической логики в 30-х годах

XX века было выработано точное определение понятия алгоритма, причем были построены различные варианты математического уточнения этого понятия, иными словами, различные математические теории алгоритмов. Благодаря этому удалось обнаружить неразрешимые алгоритмические проблемы, т. е. несуществование соответствующих алгоритмов. Теперь, когда не удается найти алгоритм для некоторого класса задач, можно попытаться доказать, что искомый алгоритм не существует.

Слово «алгоритм» все чаще употребляется все большим числом людей самых различных профессий, все чаще появляется на страницах печати, упоминается в связи с использованием ЭВМ, проникло и в школьное обучение. Этим словом обозначается одно из фундаментальных понятий современной математики и информатики.

Для изучения математики и информатики в рамках школьных программ, обязательных для всех учащихся, вполне достаточно интуитивного понятия алгоритма. Однако для тех учащихся VIII — XI классов, которые интересуются этими предметами, для любителей математики и информатики будет не только интересно, но и полезно более глубокое изучение понятия алгоритма, включающее его математическое уточнение.

Озаглавив эту книгу вопросом «Что такое алгоритм?», авторы ставят перед собой две цели: прежде всего закрепить и несколько обобщить уже имеющиеся у Вас интуитивные представления о том, что такое алгоритм, а затем ознакомить Вас с двумя математическими уточнениями этого понятия (в виде нормального алгоритма Маркова и машины Тьюринга).

В соответствии с поставленными целями содержание книги разделено на три части.

В первой части разъясняется на примерах из школьной математики, что обычно понимают под алгоритмом, какими свойствами обладает любой алгоритм, каковы основные виды алгоритмов. Рассматриваются три формы представления (задания) алгоритмов: с помощью словесного предписания, блок-схемы и алгоритмической записи, причем показывается эквивалентность этих форм переводом одной формы в другую.

Вторая часть книги посвящена двум упомянутым выше математическим уточнениям понятия алгоритма. Знакомство с точным, математическим понятием

алгоритма способствует и лучшему, правильному усвоению интуитивного понятия и его приложений. Но оно имеет и другое, не менее важное, назначение. Понятие алгоритма в виде идеализированной, допускающей точное математическое определение машины Тьюринга, предшествовавшей реальным ЭВМ, уже знакомит с принципами работы этих машин.

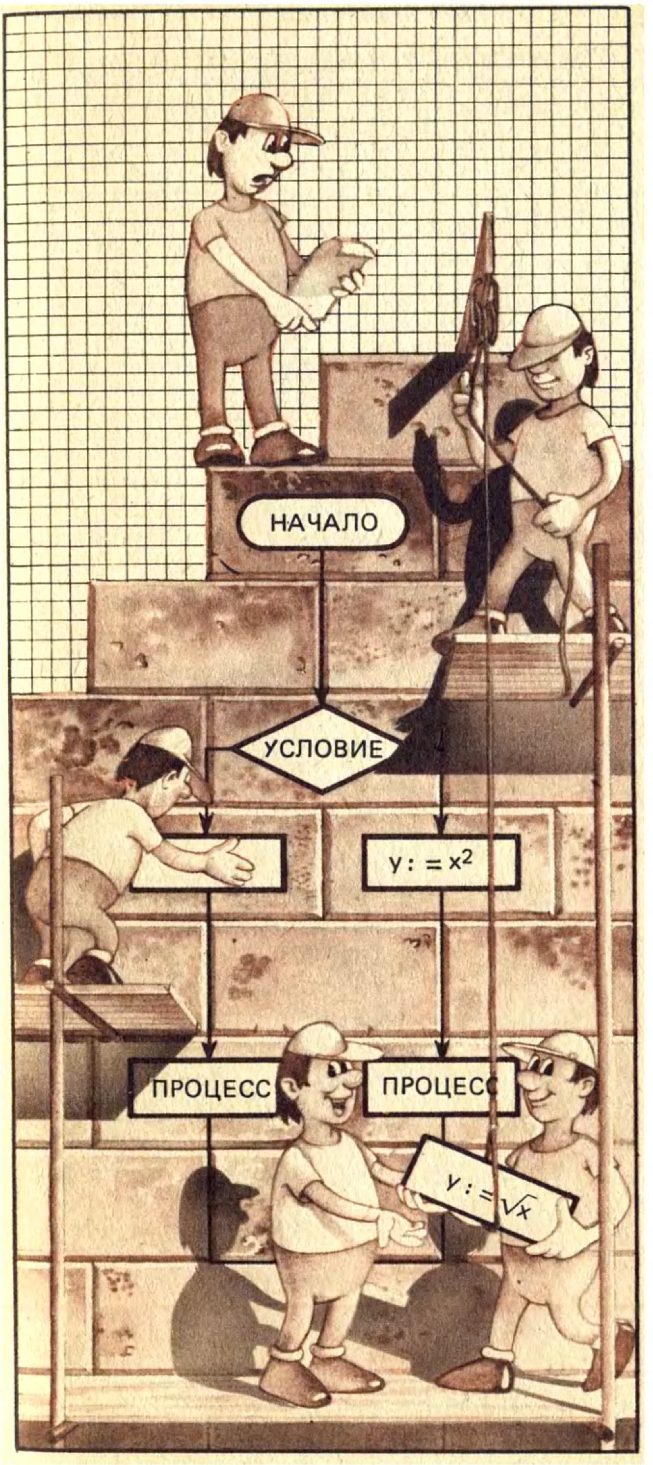
В заключительной части книги рассматриваются приложения алгоритмов к играм, к решению некоторых экономических задач, к программированию.

Уважаемый читатель! Авторы хотели бы особо подчеркнуть, что предлагаемая Вам книга не предназначена для легкого чтения. Ее нужно читать с бумагой и карандашом в руках, а точнее, ее нужно изучать, стараясь при этом самостоятельно решать все предлагаемые задачи.

Мы уверены в том, что труд, который Вы затратите при чтении этой книги, принесет пользу.

Желаем Вам успехов!

Авторы



Что такое алгоритм?

(Интуитивное понятие)

1. "Да, нет" и "Чему равно?"
2. Откуда взялось слово "алгоритм"?
3. Что же такое алгоритм?
4. Блок-схема алгоритма
5. Алгоритмическая запись
6. Какие бывают алгоритмы?

1. «Да, нет» и «Чему равно!»

Рассмотрим задачу: «Делится ли натуральное число a на натуральное число b?» (Или, в другой формулировке, «является ли число b делителем числа a?».)

По существу, мы имеем здесь не одну, а бесконечный класс однотипных задач (столько, сколько существует пар натуральных чисел $(a; b)$). Иногда класс однотипных задач называют иначе **общей задачей,** а получаемые путем подстановки вместо a и b конкретных натуральных чисел задачи этого класса называют **частными.** Так, например, частными задачами рассматриваемого класса задач (или рассматриваемой общей задачи) являются следующие: «делится ли 21 564 на 132?», «делится ли 1056 на 24?» и т. п.

Естественно возникает вопрос: существует ли общий метод, позволяющий для любой частной задачи этого класса в конечное число шагов дать требуемый ответ (в виде «да» или «нет»)?

Таким общим методом, как Вам известно из начальных классов, является процедура «деления углом»: делим a на b (для любой пары $(a; b)$ это выполнимо в конечное число шагов) и, если в остатке получится 0,— ответ будет «да», если же остаток отличен от нуля, ответ будет «нет».

Рассмотрим еще одну известную Вам задачу: «Имеет ли уравнение $ax^2 + bx + c = 0$, где $a, b, c \in Q$, вещественные корни?»

Здесь мы опять имеем общую задачу, или бесконечный класс однотипных частных задач (столько задач, сколько имеется троек $(a; b; c)$ рациональных чисел или сколько имеется квадратных уравнений с рациональными коэффициентами). И опять возникает тот же вопрос: существует ли общий метод, позволяющий для любой частной задачи этого класса в конечное число шагов дать требуемый ответ в виде «да» или «нет»?

Вам известен такой общий метод, и он состоит в следующем: определяем знак дискриминанта $D = b^2 - 4ac$ (это, очевидно, всегда выполнимо за конечное число шагов: вычисляем b^2, вычисляем $4ac$ и сравниваем два рациональных числа, b^2 и $4ac$); если $D \geqslant 0$, то ответ — «да», если же $D < 0$, то ответ — «нет».

Мы рассмотрели два класса задач, требующих ответа вида «да» или «нет». Вопрос, который мы дважды поставили, существует ли общий метод, позволяющий для любой задачи данного класса в конечное число шагов дать ответ «да» или «нет», называют **проблемой разрешения** для этого класса задач, а общий метод, если он существует, называется **разрешающей процедурой, разрешающим алгоритмом** или просто **алгоритмом.**

В математике встречаются, однако, не только такие классы задач, которые требуют ответа вида «да» или «нет». Имеется большое разнообразие классов задач, требующих в качестве ответа предъявления некоторого объекта (числа, множества чисел, функции и др.).

Формулировка таких задач начинается обычно словом «найти» или словами «чему равно?» и т. п.

Рассмотрим задачу: «Чему равна сумма двух натуральных чисел a и b?».

Здесь опять имеем бесконечный класс однотипных частных задач, требующих в качестве ответа предъявления определенного натурального числа (суммы данных двух натуральных чисел). И здесь возникает тот же вопрос: существует ли общий метод, позволяющий для любой частной задачи этого класса в конечное число шагов дать требуемый ответ, т. е. предъявить натуральное число, являющееся суммой двух данных натуральных чисел?

Такой общий метод существует. Мы надеемся, что Вы еще не забыли процедуру «сложения столбиком», которой Вас учили в начальных классах.

Рассмотрим другой пример: «Найти множество вещественных решений системы неравенств $\begin{cases} x > a \\ x < b \end{cases}$, где $a,\ b \in Z$».

Здесь опять имеем бесконечный класс однотипных задач (столько, сколько пар $(a; b)$ целых чисел) и возникает тот же вопрос: существует ли общий метод, позволяющий для любой частной задачи этого класса в конечное число шагов дать ответ в виде некоторого множества вещественных чисел (множества решений системы)?

Общий метод для этого класса задач Вам известен: сравниваем a и b (из двух целых чисел мы умеем определить в конечное число шагов, какое из них больше); если $a < b$, то ответ будет множество $E =]a;\ b[$, если же $a \geqslant b$, то $E = \varnothing$.

Мы рассмотрели два класса задач вида «чему равно» или «найти» (вместо «найти множество решений...», очевидно, можно сказать «чему равно множество решений...»). Поставленный для каждого из этих классов задач вопрос, «существует ли общий метод, позволяющий для каждой частной задачи данного класса в конечное число шагов дать требуемый ответ», называется **проблемой вычисления,** а искомый общий метод, если он существует, называется **вычислительной процедурой, вычислительным алгоритмом** или просто **алгоритмом.**

В школьной математике мы имеем дело с большим разнообразием разрешающих и вычислительных алгоритмов (и те и другие называются кратко **алгоритмами).**

* * *

Мы предполагаем, дорогой читатель, что у Вас возникли вопросы. Во-первых, откуда взялось само слово «алгоритм»? Во-вторых, пока алгоритмом назван общий метод решения любой частной задачи из некоторого класса однотипных задач, но это еще очень мало говорит о том, что такое алгоритм. А что такое «общий метод»? Не собираются ли авторы заменить одно непонятное слово

другим, не намного более ясным? (На этот вопрос мы сразу даем отрицательный ответ.) И какими свойствами должен обладать такой «общий метод», чтобы его можно было назвать алгоритмом? В-третьих, мы многократно повторяли «в конечное число шагов». А что такое «шаг»? И что означает вообще «выполнимо в конечное число шагов»?

Мы не уверены, что перечислили все вопросы, которые уже появились у Вас. С другой стороны, возможно, что некоторые из перечисленных вопросов у Вас не возникли. В таком случае мы их Вам подсказываем.

Ответы на эти и другие вопросы Вы найдете в следующих беседах.

Пока же предлагаем в качестве полезного упражнения решить задачи.

Задачи

1.1. Приведите примеры классов задач, требующих ответ вида «да» или «нет», и опишите (как умеете, пока) соответствующие разрешающие алгоритмы.

1.2. Приведите примеры классов задач вида «чему равно?» и опишите (как умеете, пока) соответствующие вычислительные алгоритмы.

2. Откуда взялось слово «алгоритм»!

Вы, очевидно, не подозреваете, что каждый раз, когда употребляете слово «алгоритм», произносите имя выдающегося средневекового ученого, 1200-летие со дня рождения которого отмечалось по решению ЮНЕСКО в 1983 году.

Мухамед ибн Муса ал-Хорезми (в переводе с арабского означает «Мухамед сын Мусы из Хорезма»), сокращенно Ал-Хорезми, уроженец Хивы. Его творческая деятельность протекала в IX веке главным образом в Багдаде, где в то время правил халиф Ал-Мамун, покровительствовавший математике и собравший в созданном им «Доме мудрости», своего рода академии наук, много крупных ученых того времени.

В одном из своих трудов Ал-Хорезми описал десятичную систему счисления и впервые сформулировал правила выполнения арифметических действий над целыми числами и простыми дробями.

Арабский оригинал этой книги Ал-Хорезми утерян, но имеется латинский перевод XII века, по которому Западная Европа ознакомилась с десятичной позиционной системой счисления и правилами выполнения в ней арифметических действий.

Ал-Хорезми стремился к тому, чтобы сформулированные им правила были понятными для всех грамотных людей. Достичь этого в IX веке, когда еще не была разработана математическая символика (знаки операций, скобки, буквенные обозначения и

т. п.), было чрезвычайно трудно. Однако Ал-Хорезми удалось выработать в своих трудах такой стиль четкого, строгого словесного предписания, который не давал читателю никакой возможности уклониться от предписанного или пропустить какие-нибудь действия.

В латинском переводе арифметического труда Ал-Хорезми правила начинались словами Dixit Algorizmi (Алгоризми сказал). В других латинских переводах автор именовался Algorithmus (Алгоритмус). Постепенно люди забыли, что Алгоризми — это автор правил, и стали сами эти правила называть алгоритмами. Так «Алгоризми сказал» постепенно преобразовалось в «алгоритм гласит».

Таким образом, слово «алгоритм» — латинизированное имя Ал-Хорезми. Как научный термин это слово первоначально обозначало лишь правила десятичной системы счисления. Затем, в течение столетий этот термин приобретает постепенно все более широкий смысл, обозначая уже не только правила десятичной системы счисления, но любые точные правила действий. В конце концов слово «алгоритм» стало научным термином, обозначающим одно из фундаментальных понятий современной математики и информатики, которое и является предметом наших с Вами бесед.

(Здесь, читатель, у Вас, очевидно, возникнет вопрос: что это такое «точное правило», чем оно отличается от «неточного»? И что это такое «правило действий»? О каких «действиях» идет речь? Так как мы сами предполагаем, что эти вопросы могут возникнуть у Вас, то мы в дальнейшем ответим на них.)

Появление слова «алгебра» тоже связано с Ал-Хорезми. Оно представляет собой часть названия его главного труда «Китаб ал-мухтасар фи хисаб ал-джабр в-ал-мукабала» (в дословном переводе с арабского означающего «краткая книга об исчислении восстановления и противопоставления»), что следует истолковывать как учение об уравнениях.

Эта, по существу, первая «алгебра» стала известной на западе Европы в латинском переводе, и слово «ал-джабр» (из арабского названия труда Ал-Хорезми) преобразовалось в слово «алгебра». Этим словом и стали обозначать всю научную область, которая до середины XIX века и представляла собой, главным образом, учение об уравнениях.

Вот эти два слова, «алгоритм» и «алгебра», ставшие научными терминами, навсегда сохранили в мировой науке память о трудах великого хорезмийца, сделали бессмертным имя Ал-Хорезми.

3. Что же такое алгоритм?

Мы понимаем, дорогой читатель, Ваше желание как можно быстрее узнать, что же такое алгоритм. У Вас, очевидно, создается впечатление, что авторы не очень уж торопятся давать опреде-

ление. Ну что же, это впечатление соответствует нашим намерениям. Причины? Во-первых, как уже было отмечено, мы имеем дело с понятием, которое веками применялось в математике без точного определения. И нам нужно подготовить Вас к такому определению, предварительно уточнить и прояснить, что интуитивно понимают под алгоритмом. Во-вторых, уточнение того, что мы обычно понимаем интуитивно, когда произносим слово «алгоритм», мы тоже предпочитаем начать не с общей декларации вида «под алгоритмом обычно понимают то-то и то-то», а с рассмотрения конкретных примеров, которые и помогут нам разъяснить для себя, что это такое алгоритм.

Итак, мы уже говорили в первой беседе о единых, общих методах решения определенных классов задач и называли их разрешающими и вычислительными алгоритмами или, короче, алгоритмами.

Как же можно представить эти «единые, общие методы»?

Обратимся еще раз к примерам.

1. Мы описали в первой беседе общий метод решения любой задачи из класса задач «Имеет ли уравнение $ax^2 + bx + c = 0$, где a, b, $c \in Q$, вещественные корни?» Но можно это описание представить в виде более четкого словесного предписания, разделенного на отдельные указания (пункты, команды, шаги):

1) Вычислить $D = b^2 - 4ac$. Перейти к ук. 2.
2) Если $D \geqslant 0$, то перейти к ук. 3, иначе — к ук. 4.
3) Уравнение имеет вещественные корни. Перейти к ук. 5.
4) Уравнение не имеет вещественных корней. Перейти к ук. 5.
5) Процесс решения задачи окончен.

Чем же отличается это предписание от обычного описания?

Оно однозначно определяет порядок выполнения действий, т. е. с чего начинать (каков первый шаг) и какой шаг непосредственно следует за каждым (кроме, разумеется, последнего, объявляющего процесс решения оконченным).

Обратите внимание на указание 2, сформулированное с помощью выделенных слов («если», «то», «иначе»). Оно выполняет важную роль, а именно **разветвляет** процесс решения. Если условие «$D \geqslant 0$», записанное между словами «если» и «то», выполняется (истинно), то следующим шагом будет тот, который предусмотрен указанием 3 (т. е. объявление результата «уравнение имеет вещественные корни» или, короче, «да»), если же оно не выполняется (ложно), т. е. $D < 0$, то указание 3 пропускается и следующим шагом является выполнение указания 4 (объявление результата «уравнение не имеет вещественных корней» или, короче, «нет»). Это можно представить в виде таблицы, не требующей комментариев:

$D \geqslant 0$	Последовательность шагов
И	1, 2, 3, 5
Л	1, 2, 4, 5

Конечно, можно было бы вместо условия «$D \geqslant 0$» поставить отрицание этого условия, т. е. «$D < 0$».

Предлагаем Вам решить эту задачу, т. е. составить предписание, аналогичное приведенному выше, но в котором вместо условия «$D \geqslant 0$» ставится условие «$D < 0$».

Мы рассмотрели пример разрешающего алгоритма. Приведем пример вычислительного алгоритма.

2. Рассмотрим класс задач «Решить уравнение $ax = b$ ($a, b \in Q$)». Если исходить из решения частных задач этого класса, например, $3x = 12$, $2x = -5$, $0{,}5x = 5$ и т. п. (мы опускаем здесь слова «решить уравнение»), то легко обнаружить, что применяем неизменно один и тот же метод: делим обе части уравнения на коэффициент при x. Но можно ли утверждать, что любое уравнение вида $ax = b$ решается этим методом, т. е. имеет единственный корень $x = \frac{b}{a}$?

Если, читатель, Вы внимательны, то заметите, что это будет так только при условии, что $a \neq 0$. А если $a = 0$, т. е. уравнение имеет вид $0 \cdot x = b$? Тогда уже делить на a нельзя, и все будет зависеть от b, т. е. если $b \neq 0$, уравнение не имеет решений, если же $b = 0$, то уравнение $0 \cdot x = 0$ удовлетворяется любым значением x, т. е. имеет бесконечное число корней.

Таким образом, общий метод решения любой задачи рассматриваемого класса можно представить в виде следующего словесного предписания:

1) Если $a \neq 0$, то перейти к ук. 2, иначе — к ук. 3.

2) Уравнение имеет единственный корень $x = \frac{b}{a}$. Перейти к ук. 6.

3) Если $b \neq 0$, то перейти к ук. 4, иначе — к ук. 5.

4) Уравнение не имеет корней. Перейти к ук. 6.

5) Любое число является корнем уравнения. Перейти к ук. 6.

6) Процесс решения окончен.

Как видно, в этом предписании уже имеются два указания (1 и 3), определяющие разветвления процесса. Соответствующая таблица разъясняет, как протекает процесс решения, однозначно определяемый этим предписанием, в каждом из возможных случаев выполнения (истинности) или невыполнения (ложности) двух выделенных условий ($a \neq 0$ и $b \neq 0$):

$a \neq 0$	$b \neq 0$	Последовательность шагов	Результат
И		1, 2, 6	$x = \frac{b}{a}$
Л	И	1, 3, 4, 6	Нет решений
Л	Л	1, 3, 5, 6	Бесконечное множество решений

У Вас может возникнуть вопрос: почему в таблице пропущен случай, когда $a \neq 0$ выполняется (И), а $b \neq 0$ не выполняется (Л)?

Дело в том, что когда условие $a \neq 0$ выполняется, то указание 3, предусматривающее проверку выполнения условия $b \neq 0$, пропускается. Здесь, независимо от того $b \neq 0$ или $b = 0$, уравнение имеет единственный корень $\frac{b}{a}$ (когда $b = 0$, этот корень равен нулю).

Как видно, и в этом примере нам удалось описать общий метод решения любой задачи из рассматриваемого класса однотипных задач (любого уравнения вида $ax = b$) в виде словесного предписания, однозначно определяющего порядок действий, т. е. каков первый шаг и какой шаг следует за каждым, не оставляя решающему задачу никакой свободы выбора очередного шага по своему усмотрению.

Общий метод решения задач данного класса можно, разумеется, описать с помощью предписаний, отличающихся от данного выделенными условиями.

Предлагаем Вам следующую задачу: составьте предписание, аналогичное приведенному выше, но в котором выделены условия: а) $a = 0$, $b = 0$; б) $a = 0$, $b \neq 0$; в) $a \neq 0$, $b = 0$, и постройте соответствующие таблицы.

Заметим, что встречаются различные названия для построенных нами описаний общих методов. То, что мы назвали «указанием, командой», иногда называют «предписанием», а то, что мы назвали «предписанием», называют в таком случае «системой предписаний».

3. В качестве третьего, несколько более сложного примера рассмотрим класс задач «Чему равен наибольший общий делитель двух неравных натуральных чисел a, b (НОД $(a, b) = ?$)»

Ясно, что здесь мы опять имеем бесконечный класс однотипных частных задач, столько, сколько имеется пар натуральных чисел. И есть общий метод решения любой задачи этого класса, известный под названием «алгоритм Евклида», даже два варианта этого метода.

Рассмотрим сначала вариант этого метода, использующий действие деления, известный поэтому и как метод последовательного деления. Он основан на свойстве: НОД (a, b) = НОД (b, r), где r — остаток от деления a на b.

Покажем на частной задаче, например, «НОД (280, 45) = = ?», в чем состоит этот метод. Опишем его с помощью словесного предписания, состоящего из последовательности указаний:

1. Делить большее число (280) на меньшее $(45) : 280 = 45 \times \times 6 + 10$.

2. Делить меньшее число (45) на остаток $(10) : 45 = = 10 \cdot 4 + 5$.

3. Делить первый остаток (10) на второй $(5) : 10 = = 2 \cdot 5 + 0$.

4. Так как в последнем делении получен остаток, равный нулю, то предпоследний остаток или последний,отличный от нуля остаток (5) и является наибольшим общим делителем данных двух чисел, т. е. НОД(280, 45) = 5.

Перейдем теперь к описанию этого метода в общем виде, т. е. для любой пары неравных натуральных чисел (a, b), где, допустим, $a > b$ (если $b > a$, можно первым шагом поменять их местами и большее число называть a, меньшее b): делим a на b; если остаток $r_1 = 0$, то НОД $(a, b) = b$, если $r_1 \neq 0$, то делим b на r_1, если остаток $r_2 = 0$, то НОД $(a, b) = r_1$, если же $r_2 \neq 0$, то делим r_1 на r_2, и так далее до получения остатка $r_n = 0$. Тогда НОД $(a, b) = r_{n-1}$, где r_{n-1} — последний отличный от нуля остаток.

Однако здесь возникает вопрос: всегда ли описанный процесс последовательного деления закончится, т. е. всегда ли в конечное число шагов получим нулевой остаток?

Легко доказать, что описанная процедура конечна. Действительно, любой остаток неотрицателен $(r_i \geqslant 0)$, а последовательность остатков убывает $(b > r_1 > r_2 > \ldots \geqslant 0)$. Следовательно, она конечна и последний остаток равен нулю.

Но описание, содержащее выражение «и так далее», не годится. Кроме того, нетрудно заметить, что здесь повторяется многократно одно и то же действие (деление большего числа на меньшее), меняются лишь числа, компоненты действия, причем меняются они определенным образом, закономерно. Возникает вопрос: нельзя ли построить такое предписание, чтобы действие деления содержалось в нем лишь один раз, но было бы точно определено, до каких пор надо повторить это действие и над какими числами оно выполняется в каждом повторении?

Оказывается, можно. Но при этом придется исходным переменным a и b на каждом шагу **присвоить** новые значения. Так, сначала мы делим a на b; затем, присвоив a значение b, а b — значение r_1 и разделив опять a на b, по существу, мы уже делим b на r_1; дальше, присвоив a значение r_1, а b — значение r_2 и разделив a на b, мы делим, по существу, r_1 на r_2.

Таким образом получаем следующее предписание:

1. Разделить a на b. Перейти к ук. 2.
2. Если остаток $r = 0$, то перейти к ук. 4, иначе — к ук. 3.
3. Присвоить a значение b, b — значение r. Перейти к ук. 1.
4. НОД $(a, b) = b$. Перейти к ук. 5.
5. Процесс окончен.

Как видите, мы получили такое же компактное, как в первых двух примерах, описание «алгоритма Евклида» в виде предписания, состоящего всего из 5 указаний, хотя шагов в процессе решения различных частных задач этого класса оказывается разное число, иногда и довольно большое.

Мы получили пример алгоритма, в котором какое-то действие (или какая-то последовательность действий) повторяется. Это — пример **циклического алгоритма.** Повторяющаяся часть такого ал-

горитма называется **циклом.** Переменная, от которой зависит повторение цикла, называется **параметром цикла.** В нашем примере параметром цикла является переменная r. Пока $r \neq 0$ цикл повторяется. Как только r принимает значение 0, процесс оканчивается.

Покажем на таблице выполнение предписания для двух частных задач:

задача I	шаги	a	b	r	$r=0$	НОД(280, 45)
	1	280	45	10	Л(нет)	
	2	45	10	5	Л(нет)	
	3	10	5	0	И(да)	5

задача II	шаги	a	b	r	$r=0$	НОД(1448, 124)
	1	1448	124	84	Л(нет)	
	2	124	84	40	Л(нет)	
	3	84	40	4	Л(нет)	
	4	40	4	0	И(да)	4

Как видно, в задаче I мы имеем три повторения цикла, а в задаче II — четыре.

В другом варианте алгоритма нахождения наибольшего общего делителя двух чисел, как мы уже говорили, вместо деления используется вычитание. Такую возможность легко обнаружить. Если проанализировать известный алгоритм «деления углом», то окажется, что он сводится к последовательности вычитаний. Так, например, для частной задачи I получаем следующий процесс решения по второму варианту алгоритма:

a	280	235	190	145	100	55	45	35	25	15	10	5
b	45	45	45	45	45	45	10	10	10	10	5	5

Приведенная таблица показывает применение для исходной пары чисел (280, 45) следующего предписания (другого варианта алгоритма Евклида):

1. Если $a = b$, то перейти к ук. 4, иначе — к ук. 2.
2. Определить большее из чисел. Обозначить его через a. Перейти к ук. 3.
3. Заменить a разностью $a - b$ (присвоить a значение $a - b$). Перейти к ук. 1.
4. НОД $(a, b) = b$. Перейти к ук. 5.
5. Процесс окончен.

Хотя в обоих вариантах предписания состоят из 5 указаний (команд), но при одних и тех же исходных данных (a, b) эти предписания определяют различное число шагов. Во втором варианте получаем намного большее число повторений цикла, чем в первом, но повторяется при этом более простое действие (в рас-

смотренной нами частной задаче I первый вариант содержал 3 повторения деления, второй — 12 повторений вычитания). Для исполнителя-человека более удобен первый вариант, для исполнителя-ЭВМ, по-видимому, второй.

* * *

Дорогой читатель, Вы, вероятно, хотите задать нам вопрос, можно ли уже сказать, пусть неточно, на таком, как Вы его называете, «интуитивном уровне», что же такое алгоритм?

Действительно, теперь мы уже можем следующим образом разъяснить, что такое алгоритм: алгоритм можно понимать, как **точное, понятное предписание о том, какие действия и в каком порядке необходимо выполнить, чтобы решить любую задачу из данного класса однотипных задач** (для которого и предназначен этот алгоритм).

Однако в этом разъяснении еще многое неясно. Например, что значит «точное предписание», «понятное предписание», «действие», «решить любую задачу», «класс однотипных задач».

Объясним смысл этих слов.

1) Что такое «точное предписание»? Это означает, что предписание, задающее алгоритм, должно быть составлено так, чтобы его исполнение было однозначно осуществимо и не требовало никаких свободно принимаемых (исполнителем) решений, чтобы были однозначно определены последовательность действий и результат. Это одно из свойств любого алгоритма, известное под названием **определенности** или **детерминированности** алгоритма.

2) Что означают слова «понятное предписание»? Это не только предписание, выраженное на понятном исполнителю языке. В конце концов, если оно написано на французском языке, его можно перевести на русский, английский и т. д. Главное состоит в том, чтобы предусмотренные предписанием действия были выполнимыми определенной категорией исполнителей, которым оно адресовано. Иначе говоря, чтобы эти действия принадлежали **системе действий** исполнителя. Под системой действий исполнителя понимают совокупность действий, которые он умеет выполнять. Так, приведенные выше предписания (в примерах 1, 2, 3) понятны Вам, дорогой читатель, так как предусмотренные этими предписаниями действия (арифметические) входят в Вашу систему действий. Но если бы мы построили предписание, включающее действие интегрирования, то оно было бы для Вас непонятным, если Вы еще не изучили начала математического анализа, так как такое действие пока не входит в Вашу систему действий. Или другой пример: предписание, содержащее действия токаря, вытачивающего определенную деталь у станка, будет непонятным авторам этой книги, которые никогда не изучали токарное дело и, естественно, эти действия не входят в их систему действий.

И еще одно обстоятельство, само собой разумеющееся: чтобы

предписание, задающее алгоритм, было понятным, оно должно быть конечным, т. е. выражено с помощью конечного текста. Бесконечно длинные предписания исключаются ввиду того, что они принципиально не допускают передачи исполнителю. Таким образом, говоря «понятное предписание», будем иметь в виду конечное предписание, предусматривающее действия, входящие в систему действий исполнителя. Это верно и для случая, когда исполнитель — человек, и для случая, когда исполнитель — машина (ЭВМ, робот и т. п.).

3) Что понимают под словом «действие»? Это, разумеется, далеко не только арифметические или, вообще, математические операции. Ведь алгоритмы встречаются отнюдь не только в математике. Это слово применяется в весьма широком (и пока неуточненном) смысле, охватывающем и действия токаря, изготовляющего деталь, и действия человека, переходящего улицу, и действия медсестры, берущей кровь на анализ или делающей инъекцию. И можно назвать еще тысячи разновидностей действий.

4) Что означает «решить любую задачу» из данного класса однотипных задач? Во-первых, это означает, что каждый алгоритм предназначен для решения не одной единственной задачи, а любой задачи из некоторого бесконечного класса однотипных задач. Алгоритм является единым методом, позволяющим по любому исходному объекту из определенного бесконечного множества исходных объектов получить искомый результат. В этом состоит свойство **массовости** алгоритма.

Во-вторых, «решить задачу» означает решить ее «за конечное число шагов». Получение результата за конечное число шагов составляет свойство **результативности** алгоритма.

В-третьих, так как предписание, задающее алгоритм, обеспечивает получение результата за конечное число шагов, то это означает также, что всякий алгоритм представляет собой упорядоченное конечное множество шагов. А всякое такое множество обладает свойством **дискретности.** Этим свойством обладает множество N натуральных чисел, хотя оно и бесконечное. Это свойство состоит в том, что для каждого натурального числа можно указать единственное непосредственно следующее за ним натуральное число, т. е. такое, что между ними нет других натуральных чисел. В любом алгоритме для каждого шага (кроме, разумеется, последнего) можно указать единственный (при данном выборе исходных объектов), непосредственно следующий за ним шаг, т. е. такой, что между ними нет других шагов. Поэтому и говорят, что алгоритм обладает свойством **дискретности** или **дискретной структурой.**

5) Что означает «класс однотипных задач»? Это мы уже разъясняли на примерах. Класс однотипных задач или общая задача обычно формулируется (в математике) с использованием некоторых переменных — параметров. Например, общая задача «Решить уравнение $ax^2 + bx + c = 0$ $(a, b, c \in Q)$» содержит три параметра

(a, b, c), при этом задается область значений параметров (Q). Подставляя вместо a, b, c в формулировке общей задачи любую тройку рациональных чисел, получаем частную задачу этого класса. Таким образом мы можем получить столько однотипных частных задач, сколько существует различных троек рациональных чисел, т. е. бесконечное множество. Исходным объектом для соответствующего алгоритма является любая тройка рациональных чисел.

6) И еще одно свойство, присущее любому алгоритму. Исходные объекты, промежуточные и окончательные результаты — **конструктивные объекты.** Эти объекты могут быть построены целиком или допускают кодирование посредством слов в некоторых конечных алфавитах (понятия «алфавит» и «слово в алфавите» получат точные определения во второй части книги). Например, натуральное число — конструктивный объект, так как любое натуральное число можно закодировать с помощью слова в алфавите $\{0, 1, 2, ..., 9\}$ десятичной системы счисления или с помощью слова в алфавите $\{0, 1\}$ двоичной системы счисления и т. п. Например, число сто семьдесят пять выражается с помощью слова 175 в алфавите десятичной системы счисления или с помощью слова 10 101 111 в алфавите двоичной системы.

Целые числа можно выразить словами в алфавите $\{0, 1, 2, ..., 9, -\}$, рациональные — словами в алфавите $\{0, 1, 2, ..., 9, -, /\}$.

Арифметические выражения, составленные из чисел, знаков арифметических действий и скобок, можно рассматривать как слова в алфавите $\{0, 1, 2, ..., 9, +, -, \cdot, /, (,)\}$.

Многочлены с рациональными коэффициентами и одной переменной x можно рассматривать как слова в алфавите $\{0, 1, 2, ..., 9, +, -, /, x\}$.

Не нужно, однако, думать, что всякие математические объекты конструктивны. Например, иррациональные числа, выражающиеся бесконечными десятичными непериодическими дробями, не являются конструктивными объектами, так как иррациональное число не удается ни построить целиком, ни закодировать с помощью слова в некотором конечном алфавите.

Итак, подведя некоторый итог сказанному, можно дать и следующее объяснение термину «алгоритм»: **под алгоритмом понимаем единый метод решения определенного класса однотипных задач, обладающий свойствами дискретности, массовости, определенности, результативности и оперирующий конструктивными объектами.**

Это новое объяснение согласуется с первым, согласно которому **под алгоритмом мы понимаем предписание, точное и понятное, определяющее, какие действия и в каком порядке необходимо выполнить для решения любой задачи из данного класса однотипных задач.**

Каждое из этих двух объяснений не является, однако, математическим определением понятия алгоритма, так как исполь-

зуемые в них слова не обозначают точных математических понятий.

Здесь, по-видимому, у Вас может возникнуть вопрос: «Является ли словесное предписание единственной формой представления алгоритма?»

Нет, разумеется. Есть более удобные, для определенных целей, формы представления алгоритмов, такие же точные и понятные. Таковыми являются, в частности, **блок-схема,** обладающая особой наглядностью, и **алгоритмическая запись,** приспособленная к переводу алгоритма в программу для ЭВМ.

Но перед тем как приступить к описанию этих форм представления алгоритмов, предложим Вам несколько задач.

Задачи

3.1. Постройте (по образцу приведенных выше) точное, понятное предписание для решения класса задач «Решить уравнение $ax^2 + bx + c = 0$ $(a, b, c \in Q)$».

3.2. Составьте словесное предписание, задающее алгоритм вычисления абсолютной величины числа.

3.3. Задайте с помощью словесного предписания алгоритм для решения уравнения (класса уравнений) вида $a(a-1)x = a^2 - 1$.

3.4. Составьте словесные предписания, задающие алгоритмы решения с помощью циркуля и линейки следующих задач:

а) разделить данный отрезок пополам;

б) построить биссектрису данного угла;

в) описать окружность около данного треугольника;

г) вписать окружность в данный треугольник.

3.5. Постройте словесное предписание, задающее алгоритм решения следующего класса задач:

а) решить неравенство $ax < 10$;

б) решить неравенство $ax < b$;

в) решить неравенство $ax > 5$;

г) решить неравенство $ax > b$.

3.6. Составьте словесное предписание, выражающее алгоритм решения системы уравнений

$$\begin{cases} ax + by = c, \\ a_1x + b_1y = c_1. \end{cases}$$

3.7. Составьте предписание, задающее алгоритм вычисления значения функции:

а) $y = \begin{cases} x + 1, \text{ если } x \leqslant 0, \\ x^2 + 1, \text{ если } x > 0; \end{cases}$

б) $y = \begin{cases} x - 1, \text{ если } x < 1, \\ x + 1, \text{ если } 1 \leqslant x < 5, \\ x^2 - 5, \text{ если } x \geqslant 5. \end{cases}$

Постройте графики этих функций.

3.8. Дано предписание:

1. Если $a > 5$, то перейти к ук. 3, иначе — к ук. 2.
2. Присвоить a значение $a + 5$. Перейти к ук. 4.
3. Присвоить a значение $a - 5$. Перейти к ук. 5.
4. Если $a < 15$, то перейти к ук. 2, иначе — к ук. 6.
5. Если $a > 15$, то перейти к ук. 3, иначе — к ук. 6.
6. Записать полученное значение a. Перейти к ук. 7.
7. Процесс окончен.

Выполнить алгоритм, заданный этим предписанием, для $a = -5$; 3; 7; 32.

4. Блок-схема алгоритма

Анализ приведенных в третьей беседе предписаний выявляет в их составе два основных вида команд: 1) команды, предусматривающие выполнение некоторого действия (в широком смысле), в результате которого образуется какой-то новый промежуточный (или конечный) результат из уже имеющихся к этому моменту промежуточных результатов или исходных данных, причем непосредственно следующий за этим шаг предписан алгоритмом однозначно, независимо от полученного промежуточного результата; 2) команды, предусматривающие проверку некоторого условия и решение вопроса о том, какой шаг будет следующим в зависимости от результатов проверки; выполнение команд этого вида (в отличие от первого) не приводит к новому промежуточному результату, оно лишь определяет дальнейший ход процесса решения задачи.

Так, например, в заданном выше предписании 3.8 к первому виду относятся команды 2, 3, 6, ко второму — 1, 4, 5.

Всякий алгоритм, заданный предписанием, может быть наглядно и точно представлен в виде **блок-схемы,** состоящей из блоков и стрелок следующим образом:

1) каждый шаг представляется с помощью **блока**;

2) блок, соответствующий команде первого вида, изображается прямоугольником, внутри которого записывается (словесно или символически) выполняемое действие (эти блоки называются также **арифметическими** или, в более общем виде, **перерабатывающими информацию,** так как далеко не всегда выполняемые действия являются арифметическими);

3) блок, соответствующий команде второго вида, изображается в виде ромба, внутри которого записывается проверяемое логическое условие (эти блоки называются также **логическими);**

4) если за шагом А непосредственно следует шаг В, то от блока А к блоку В проведена стрелка;

5) от каждого арифметического блока (АБ — прямоугольника) исходит только одна стрелка; от каждого логического блока (ЛБ — ромба) — две стрелки: одна с пометкой «да» (или «+»), идущая к блоку, следующему за ЛБ, если условие выполняется,

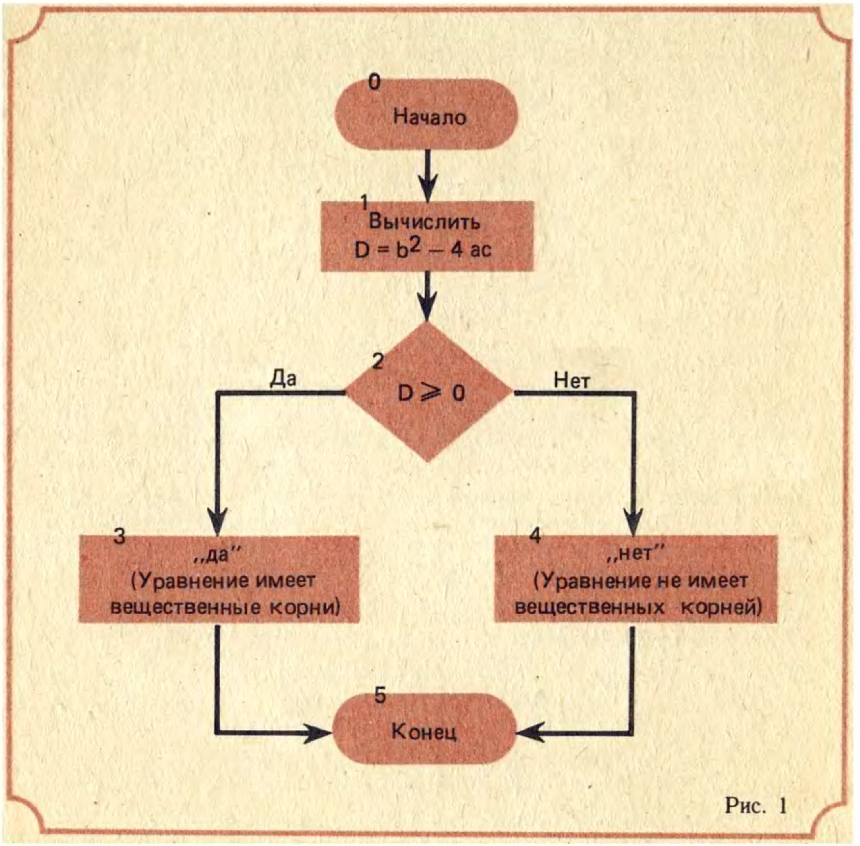


Рис. 1

другая — с пометкой «нет» (или «—»), идущая к блоку, следующему за ЛБ, если условие не выполняется (как видно, ЛБ осуществляет **ветвление** процесса);

6) начало и конец процесса выполнения алгоритма также изображаются блоками в виде фигур, ограниченных овалами, внутри которых записываются соответственно слова «начало», «конец»;

7) блоки «ввод» (исходных данных) и «вывод» (результатов) изображаются в виде параллелограммов (эти блоки обычно опускаются, когда алгоритмы адресованы исполнителю-человеку, но обязательно присутствуют в блок-схемах алгоритмов, предназначенных для ЭВМ, т. е. для перевода в программы).

Обратимся еще раз к примерам, приведенным в предыдущей беседе.

На рисунке 1 изображена блок-схема алгоритма для решения любой задачи из класса задач «Имеет ли уравнение $ax^2 + bx + c = 0$ $(a, b, c \in Q)$ вещественные корни?»

Блоки этой схемы занумерованы так, что команда предписания и соответствующий ей блок имеют один и тот же номер для наглядного показа перевода словесного предписания в блок-схему.

Как видно, блок-схема наглядно показывает, как разветвляется процесс решения в зависимости от выполнения или невыполнения условия ($D \geqslant 0$), иными словами, от его истинностного значения («да» или «нет», «И» или «Л»): если условие выполняется, то процесс продолжается по направлению, указанному левой стрелкой, помеченной словом «да», т. е. к блоку 3, иначе (если оно не выполняется) — по направлению, указанному правой стрелкой, помеченной словом «нет», т. е. к блоку 4.

На рисунке 2 изображена блок-схема алгоритма решения уравнения $ax = b$, т. е. любого уравнения такого типа. Здесь так же сохранена нумерация блоков, соответствующая нумерации команд предписания, переводимых в эти блоки.

Если Вы сравните схемы, изображенные на рисунках 1 и 2, сразу же заметите, что последняя представляет алгоритм более сложной структуры. В ней два логических блока (1 и 3), определяющих последовательность из двух ветвлений, а всего три направления продолжения процесса решения (1, 2; 1, 3, 4; 1, 3, 5) и соответственно три возможных результата.

На рисунке 3,*а* изображена блок-схема алгоритма Евклида для нахождения наибольшего общего делителя двух чисел (в первом варианте).

Здесь, по-видимому, необходимы некоторые разъяснения, так как в этой блок-схеме мы впервые отказались от словесных формулировок, заменив их символическими записями.

В словесном предписании первая команда была сформулирована так: «разделить a на b» (после того, как условились большее число обозначить через a). В блоке 1 мы это записали: «$r = \text{Ост}(a, b)$», что означает «r — остаток от деления a на b», так как именно остаток нас интересует.

В блоке 3 мы воспользовались и впредь будем пользоваться общепринятым символом «:=» для обозначения операции присваивания. Запись «$a := b$» означает «переменной a присвоить значение b» и, соответственно, запись «$b := r$» — «переменной b присвоить значение r». Таким образом, после первого выполнения действия, предписанного блоком 3, и возвращения к блоку 1 в записи «$r = \text{Ост}(a, b)$» переменная a уже будет иметь значение b, а b — значение r и, естественно, r будет обозначать новый (второй) остаток. Далее, если $r \neq 0$, т. е. условие (2) опять не выполняется, происходит второе выполнение содержания блока 3. После этого, применяя повторно операцию присваивания «$a := b$», так как b уже имеет значение r, то, по существу, это означает, что переменной a присваивается значение r, и т. д.

Но до каких пор «и т. д.»? Это четко определено на блок-схеме. До тех пор пока не выполняется условие «$r = 0$», т. е. пока

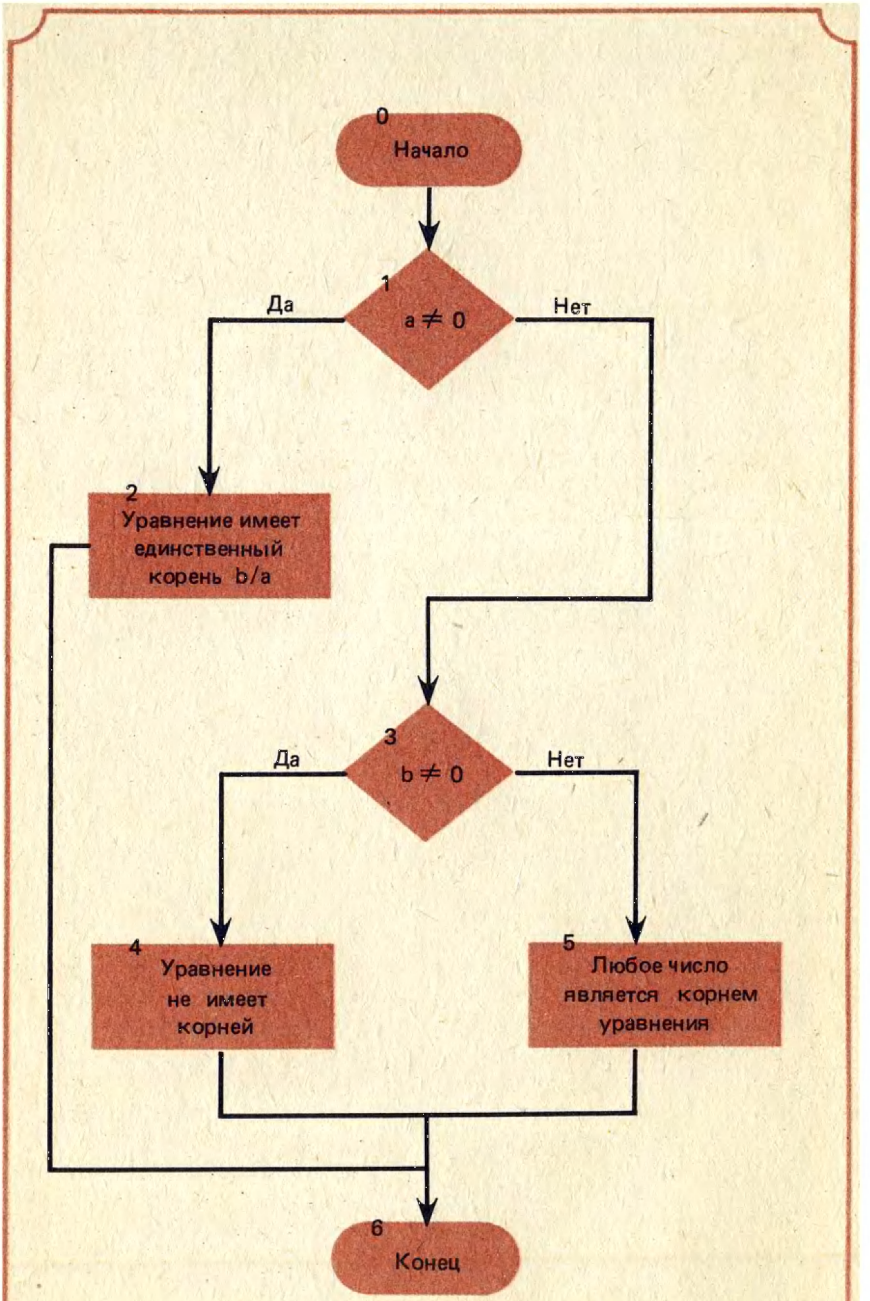

0
Начало
1
a ≠ 0
Да
Нет
2
Уравнение имеет единственный корень b/a
3
b ≠ 0
Да
Нет
4
Уравнение не имеет корней
5
Любое число является корнем уравнения
6
Конец

Рис. 2

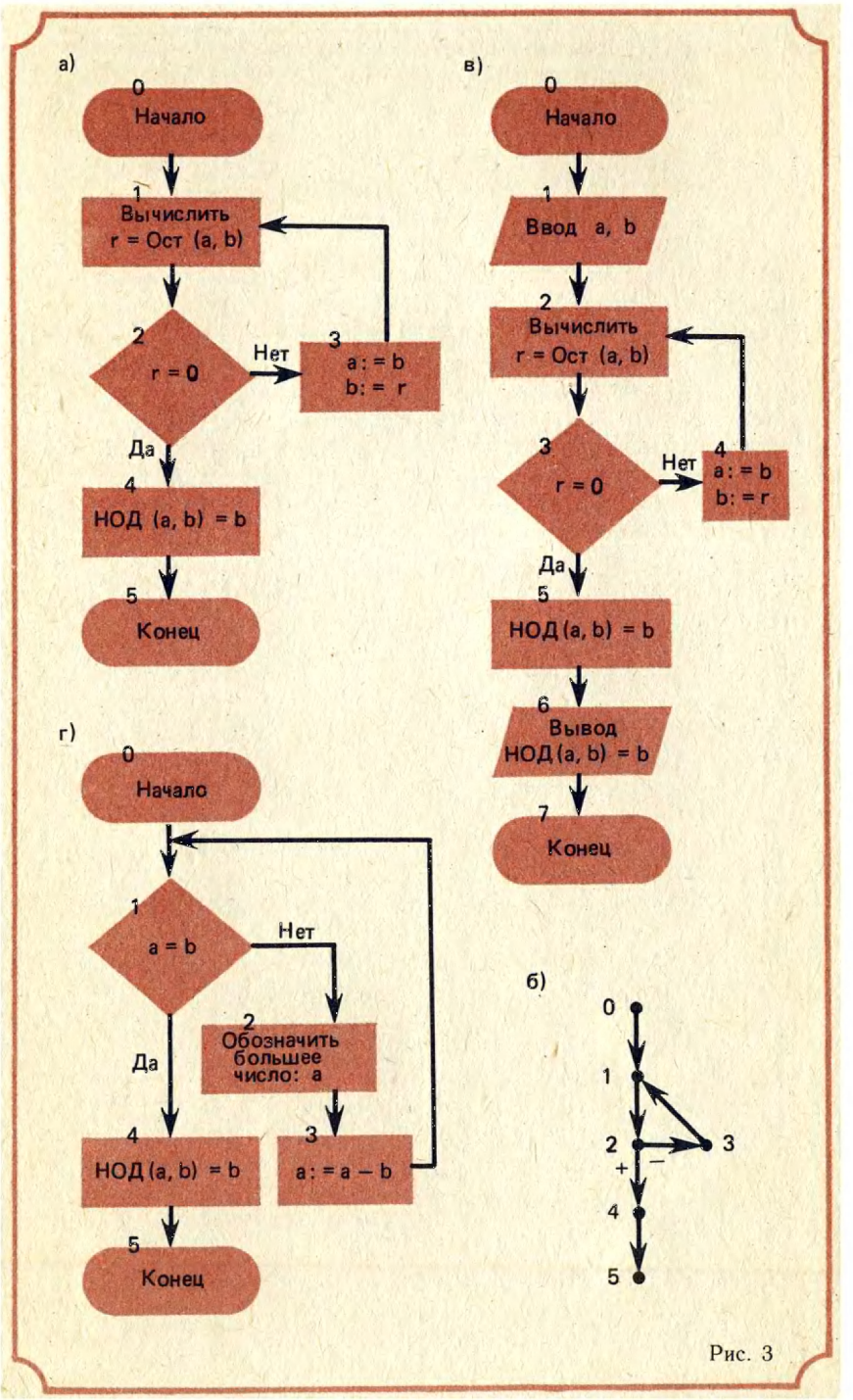

а)
0
Начало
1
Вычислить
r = Ост (a, b)
2
r = 0
Нет
3
a : = b
b : = r
Да
4
НОД (a, b) = b
5
Конец
в)
0
Начало
1
Ввод a, b
2
Вычислить
r = Ост (a, b)
3
r = 0
Нет
4
a : = b
b : = r
Да
5
НОД (a, b) = b
6
Вывод
НОД (a, b) = b
7
Конец
г)
0
Начало
1
a = b
Нет
2
Обозначить
большее
число: a
Да
4
НОД (a, b) = b
3
a : = a — b
5
Конец
б)
0
1
2
3
+
—
4
5

Рис. 3

выполняется «$r \neq 0$», цикл повторяется. Как только после очередного деления мы получили $r = 0$, блок-схема указывает на выход из цикла. Как видно, в этом случае логический блок выполняет иную функцию, чем в примерах 1, 2 (рис. 1, 2). Здесь условие $r = 0$ определяет окончание цикла. (Хотя и это можно рассматривать как вид ветвления: одно направление продолжения процесса, соответствующее стрелке с пометкой «нет», определяет цикл (2, 3, 1), другое, соответствующее стрелке с пометкой «да», определяет выход из цикла (2, 4).)

Блок-схема (рис. 3, *а*) наглядно изображает цикличность процесса (блоки 1, 2, 3, 1 образуют замкнутую линию). Еще нагляднее это видно на рисунке 3, *б*, на котором блоки изображены точками (хотя здесь имеется другое неудобство: не видно различия между блоками).

Фигура, изображенная на рисунке 3, *б*, состоящая из точек, определенным образом соединенных стрелками, называется **ориентированным графом.** Точки называются **вершинами** графа, а стрелки — его **ребрами.** Таким образом, блок-схема представляет собой ориентированный граф, при этом блоки являются вершинами графа.

Теория графов — сравнительно молодая ветвь математики, имеющая широкое поле приложения, одно из них — блок-схемы алгоритмов. Можно сказать, что блок-схема — представление алгоритма на языке графов.

Надеемся, что у Вас нет вопросов, касающихся построения блок-схем, и перечисленные ниже задачи не вызовут у Вас затруднений.

Задачи

4.1. Чем отличается блок-схема, изображенная на рисунке 3, *в*, от блок-схемы, изображенной на рисунке 3, *а*?

4.2. Какой алгоритм представлен блок-схемой, изображенной на рисунке 3, *г*?

4.3. Постройте блок-схемы алгоритмов, для которых Вы составили словесные предписания, решая задачи 3.1.—3.7.

4.4. Постройте блок-схему для предписания, заданного в задаче 3.8.

4.5. Используя правила уличного движения, составьте блок-схему алгоритма перехода улицы в месте: а) регулируемом светофором, б) нерегулируемом светофором.

5. Алгоритмическая запись

Мы уже говорили о том, что имеется и другая форма представления алгоритмов — «алгоритмическая запись», наиболее подходящая для перевода алгоритмов на некоторые языки программирования, т. е. в программу для ЭВМ.

Что же представляет собой алгоритмическая запись?

Это некоторая стандартная запись алгоритмов на определенном языке, именуемом **алгоритмическим языком** (АЯ).

Мы приступаем сейчас к изучению алгоритмического языка. Это означает, что мы должны описать следующие основные понятия: 1) алфавит алгоритмического языка, 2) способ конструирования выражений, 3) перечень используемых слов и 4) правила организации записей на алгоритмическом языке (синтаксические правила этого языка). Изучение языка включает, разумеется, и разъяснение смысла (семантики) конструкций из символов (слов и выражений) этого языка.

(Не думайте, дорогой читатель, что это — какой-то иностранный язык. Алгоритмический язык строится на базе русского языка и включает известный Вам математический язык.)

1. **Алфавит** — перечень символов (знаков), используемых в данном языке, называемых **буквами** алфавита этого языка. Алфавит алгоритмического языка строго не определен. В АЯ по необходимости используются строчные и прописные буквы русского, латинского, греческого алфавитов, математические и другие специальные знаки.

Используются цифры 0, 1, 2, 3, ..., 9 и записи чисел с помощью этих цифр в десятичной системе счисления, обычные знаки арифметических операций: $+$, $-$, $\times$, $/$ (только для деления используется знак / вместо :), а также знаки отношений: $<$, $\leqslant$, $>$, $\geqslant$, $=$, $\neq$. Все эти знаки имеют тот же смысл, что и в математическом языке.

Как Вам известно, буквы латинского алфавита a, b, c, ..., x, y, z или одна буква с индексом используются в математическом языке в качестве постоянных и переменных. **Постоянные** представляют собой имена чисел, векторов, прямых, плоскостей и т. д., **переменные** играют роль пустых мест в тексте, которые разрешают заполнить значениями этих переменных из заданной области их значений. Все это сохраняется и в АЯ. Только здесь нет ограничений для обозначений переменных. В качестве имен переменных величин, значения которых изменяются в процессе исполнения алгоритма, используются и слова русского языка, например, «площадь», «уравнение», или сочетания букв алфавита русского языка, например, «НОД» и т. д.

В зависимости от природы значений переменные делятся на числовые и нечисловые. В школьных курсах математики и физики чаще всего встречаются числовые переменные, значениями которых являются натуральные, целые или вещественные числа. В АЯ вводятся эти три типа числовых переменных (величин), а из нечисловых — переменные, значениями которых являются слова или тексты, называемые **литерными.**

2. **Выражения.** Выделяются **арифметические** и **логические** выражения.

С синтаксической точки зрения арифметические выражения в

АЯ строятся так же, как и в математическом языке, так как последний полностью включается в первый.

Логические выражения или **условия** конструируются по следующим правилам:

1) если два арифметических выражения соединить знаком какого-нибудь отношения ($<$, $>$, $=$), получим логическое выражение. Такие логические выражения назовем **простыми** или **элементарными условиями.** (Мы не включили здесь отношения $\leqslant$, $\geqslant$ и $\neq$, так как это уже сложные отношения, конструируемые определенным образом из перечисленных выше. Необходимо также отметить, что так как на АЯ записываются не только алгоритмы математического содержания, то простые условия могут образоваться и с помощью нематематических отношений. Например, «светофор показывает красный цвет», «вода в самоваре кипит», «станок работает» и т. д.);

2) если P — условие, то «**не** P» (или «**неверно, что** P») тоже условие, называемое **отрицанием** условия P (в математической логике отрицание P обозначается «$\neg P$» или «$\bar{P}$»). Например, условие $x+y\neq 3$ — отрицание условия $x+y=3$;

3) Если P и Q — условия, то «P **и** Q» — условие, называемое **конъюнкцией** двух условий P, Q (конъюнкция P **и** Q в математической логике обозначается «$P \wedge Q$»);

4) если P и Q — условия, то «P **или** Q» — условие, называемое **дизъюнкцией** двух условий P, Q (дизъюнкция P **или** Q в математической логике обозначается «$P \vee Q$»).

Например, из двух условий «$x<5$» и «$x>1$» можно составить новое условие (их конъюнкцию) «$x<5$ **и** $x>1$», которое обычно в математике записывается в виде двойного неравенства «$1<x<5$». Так как «$x=0$» и «$x>0$» — условия, то и «$x=0$ **или** $x>0$», т. е. дизъюнкция их, тоже условие, которое обычно записывается кратко с помощью нового, составного отношения «$x\geqslant 0$».

Между арифметическими и логическими выражениями имеется существенное различие и с семантической точки зрения, т. е. по тому, что они обозначают.

Арифметическое выражение обозначает число, т. е. является именем числа, если оно не содержит переменных (например, 5; $2+3$; $13\cdot(15-0{,}5)$), или именной формой числа, если содержит переменные. Именная форма числа обращается в имя числа, если вместо всех переменных подставить какие-нибудь их значения. Например, арифметическое выражение $(x+2)\cdot(y+3)$ — именная форма числа. Приведенная ниже таблица показывает, как эта именная форма обращается в имя числа при подстановке вместо x и y каких-нибудь их значений:

x	1	1	2	2	2	3 ...
y	1	2	1	2	3	2 ...
$(x+2)\cdot(y+3)$	12	15	16	20	24	25 ...

Логические выражения (или условия) всегда принимают одно из двух значений: «истина» (И) или «ложь» (Л), называемых также **истинностными значениями.**

Если условие не содержит переменных, то оно представляет собой высказывание или логическую постоянную (И или Л). Например, $3<3+2$ (И), $5+4=8$ (Л).

Наибольший интерес представляют условия, содержащие переменные (хотя бы одну). Именно такие условия встречаются в записях алгоритмов. Условие, содержащее переменные, обращается в истину или ложь (истинное или ложное высказывание) лишь при подстановке вместо переменных каких-нибудь их значений.

Для того чтобы уметь определять истинностное значение сложного условия по истинностным значениям составляющих его простых условий, необходимо определить отрицание, конъюнкцию и дизъюнкцию условий.

1) Отрицание условия P истинно, когда P ложно, и ложно, когда P истинно. Это можно записать в виде следующей **истинностной таблицы:**

P	$\neg P$
И	Л
Л	И

Например, пусть P — условие $x+2=5$, тогда его отрицание $x+2\neq 5$ и таблица

x	$x+2=5$	$x+2\neq 5$
1	Л	И
2	Л	И
3	И	Л
4	Л	И
...	...	...

показывают соотношение истинностных значений этих двух условий.

2) Конъюнкция «P **и** Q» истинна тогда и только тогда, когда истинны оба составляющие ее условия P и Q (это соответствует обычному смыслу союза «и»). Определение конъюнкции можно записать в виде следующей истинностной таблицы:

P	Q	P **и** Q
И	И	И
И	Л	Л
Л	И	Л
Л	Л	Л

Например:

x	$x>1$	$x<5$	$x>1$ **и** $x<5 (1<x<5)$
1	Л	И	Л
2	И	И	И
3	И	И	И
4	И	И	И
5	И	Л	Л
...	...	...	...

3) Дизъюнкция «P **или** Q» двух условий P и Q ложна тогда и только тогда, когда ложны оба составляющие условия (или истинна, когда истинно хотя бы одно из этих условий; это соответствует неразделительному смыслу союза «или» в обыденной речи):

P	Q	P **или** Q
И	И	И
И	Л	И
Л	И	И
Л	Л	Л

Например:

x	$x=3$	$x<3$	$x=3$ **или** $x<3 (x\leqslant 3)$
1	Л	И	И
2	Л	И	И
3	И	Л	И
4	Л	Л	Л
5	Л	Л	Л
...	...	...	...

В приведенных здесь примерах, иллюстрирующих отрицание, конъюнкцию и дизъюнкцию, мы предполагали, что область значений переменной x — множество N натуральных чисел. В таком случае соответствующие таблицы бесконечны, но видно, что для всех значений x, больших 5, конъюнкция $1<x<5$ и дизъюнкция $x\leqslant 3$ ложны.

3. **Служебные (ключевые) слова.** В АЯ выделяется некоторое число «слов», называемых **служебными** или **ключевыми** словами этого языка, смысл и способ применения которых в записях алгоритмов точно определяется. В качестве служебных слов АЯ используются сокращения некоторых русских слов или целиком некоторые русские слова. Служебные слова подчеркиваются и печатаются жирным шрифтом.

Прежде всего дадим словарь, переводящий служебные слова АЯ на русский язык. В этом словаре служебные слова расположены не в алфавитном порядке, а группами, так как они применяются для записи основных (базовых) структур алгоритмов (о чем будет идти речь дальше).

Словарь

	служебные слова АЯ	**их перевод на русский язык**
1	**алг**	алгоритм
	нач	начало
	кон	конец
2	**нат**	натуральная
	цел	целая
	вещ	вещественная

	лит	литерная
	таб	табличная
	арг	аргумент
	рез	результат
3	**если**	если
	то	то
	иначе	иначе
	все	все
4	**не**	не (неверно, что)
	и	и
	или	или
5	**пока**	пока
	нц	начало цикла
	кц	конец цикла

(В дальнейшем, для краткости, будем говорить «слова», понимая под этим «служебные слова» АЯ. Если же пойдет речь о словах другого, скажем русского, языка, мы будем называть их с указанием языка: «слова русского языка» или «русские слова».)

Слова можно читать так, как они записаны или в их переводе на русский язык.

Как видите, мы выделили в нашем словаре пять групп слов. Каждая группа слов имеет свое назначение. Слова первой группы применяются для записи заголовка, обозначения начала и конца записи алгоритма. Слова второй группы обозначают типы переменных (величин), исходные величины и результаты. Слова третьей группы используются для записи структур ветвления. Слова четвертой группы служат для конструирования сложных условий (мы их уже рассмотрели выше (п. 2)). Слова пятой группы предназначены для записи циклов.

4. **Правила организации алгоритмической записи.** Мы рассмотрим синтаксические правила записи алгоритмов на АЯ. Эти правила имеют совсем иной вид, чем, например, правила синтаксиса русского языка, представляя собой своеобразные «фигуры» расположения слов в записи алгоритмов.

Для представления этих правил в удобном, компактном виде введем еще термин «серия» для обозначения последовательности из некоторого числа, скажем n команд ($n \geqslant 1$), выполняемых одна за другой.

4.1. **Заголовок алгоритма.** Запись любого алгоритма на АЯ начинается с заголовка:

алг название алгоритма

Чаще всего, особенно в математике, физике и других областях знаний, встречаются алгоритмы, в которых используются перемен-

ные величины. В таком случае за названием алгоритма в скобках перечисляются все переменные, для каждой указывается тип и имя переменной (если используются несколько переменных одного типа, то название типа записывается один раз, а за ним перечисляются имена переменных этого типа).

Далее, в следующих строках за словом **арг** перечисляются имена переменных, обозначающих исходные данные, а за словом **рез** — имена переменных, обозначающих результаты.

Например, для алгоритма Евклида заголовок может быть записан на АЯ следующим образом:

алг ЕВКЛИДА (**нат** *a*, *b*, НОД)
  арг *a*, *b*
  рез НОД

В этой записи НОД — имя переменной, обозначающей результат.

4.2. **Общий вид записи алгоритма на АЯ:**

алг название алгоритма (типы и имена переменных)
  арг имена переменных
  рез имена переменных
нач
  серия
кон

Обратите внимание: слова **алг**, **нач** и **кон** расположены на одной вертикали (всегда на самой левой вертикали, и нарушение этого расположения слов означает нарушение соответствующего синтаксического правила). Слова **арг** и **рез** располагаются на другой вертикали, проходящей правее первых слов.

4.3. **Ветвление.** На рисунках 4,*а* и 4,*б* изображены два вида ветвлений: полное и неполное.

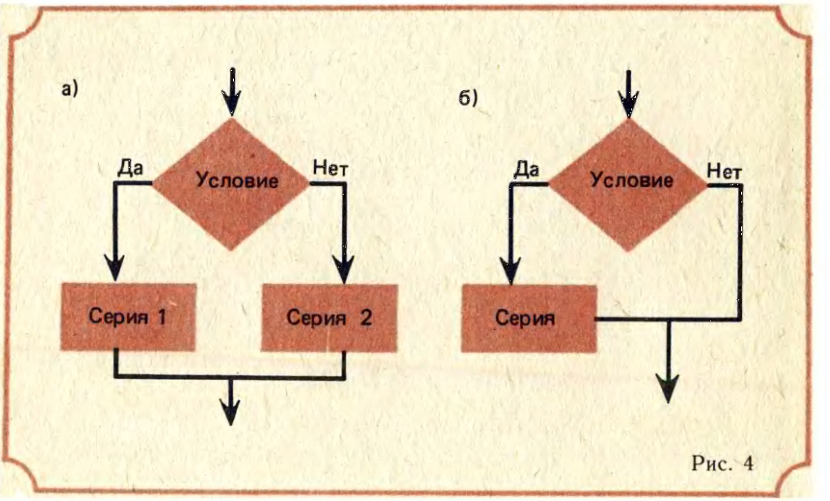


Рис. 4

На АЯ они записываются следующим образом:

а) **полное ветвление**
если условие
то серия 1
иначе серия 2
все

б) **неполное ветвление**
если условие
то серия 1
все

Обратите внимание на расположение слов: слова **если** и **все**, обозначающие начало и конец ветвления, расположены на одной вертикали, слова **то** и **иначе** — на другой, проходящей непосредственно правее первых слов. Таково синтаксическое правило этой конструкции. Какова же ее семантика? (По блок-схеме или алгоритмической записи семантика одна и та же.) Эта команда выполняется следующим образом: если условие выполняется (истинно), то выполняются команды серии 1 и пропускаются (не выполняются) команды серии 2, иначе, т. е. в противном случае, если условие не выполняется (ложно), то пропускаются команды серии 1 и выполняются команды серии 2.

Команда неполного ветвления выполняется следующим образом: если условие истинно, то выполняются команды из серии 1, затем команды, следующие после слова **все**; в противном случае, т. е. если условие ложно, команды серии 1 пропускаются и выполняются команды, следующие за словом **все**.

4.4. **Цикл (команда повторения).** На рисунке 5 изображена блок-схема цикла.
Она переводится на АЯ в следующую запись:

пока условие
нц
серия
кц

Слова **нц** и **кц** указывают на начало и конец серии повторяющихся команд (входящих в цикл).

Обратите внимание! Слова **пока, нц, кц** записываются на одной вертикали. Выполняется эта команда следующим образом: пока условие истинно, выполняются (возможно многократно) команды серии, в противном случае, т. е. если условие не выполняется, то команды серии пропускаются и выполняются команды, следующие за словом **кц**.

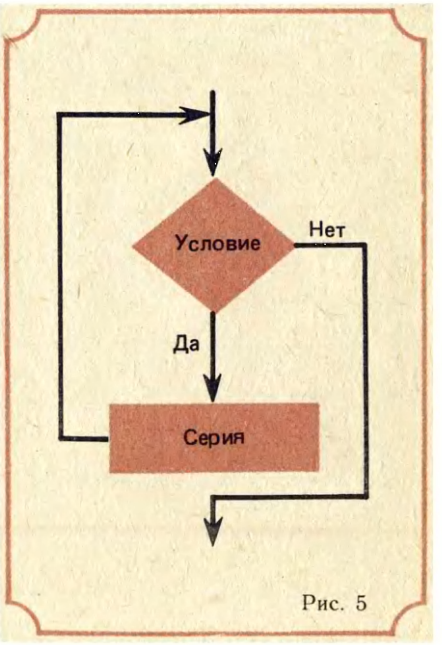


Рис. 5

5. Теперь мы уже можем показать, как записываются на АЯ алгоритмы, заданные с помощью словесных предписаний в беседе 3 и с помощью блок-схем в беседе 4.

5.1. Блок-схема, изображенная на рисунке 1, переводится в следующую алгоритмическую запись:

```
алг имеет ли уравнение вещ. корни (вещ a, b, c, лит y)
    арг a, b, c
    рез y
нач вещ D
    D := b^2 - 4ac
      если D ⩾ 0
        то y := «уравнение имеет вещественные корни» (или «да»)
        иначе y := «уравнение не имеет вещ. корней» (или «нет»)
      все
кон
```

5.2. Блок-схеме, изображенной на рисунке 2, соответствует следующая алгоритмическая запись:

```
алг решение уравнения ax = b (вещ a, b, x, лит y)
    арг a, b
    рез x, y
нач
    если a ≠ 0
         то x := b/a
         иначе если b ≠ 0
                   то y := «уравнение не имеет корней»
                   иначе y := «любое число — корень
                        уравнения»
                 все
    все
кон
```

5.3. Блок-схема, изображенная на рисунке 3,*а*, переводится в следующую алгоритмическую запись:

```
алг ЕВКЛИДА 1 (нат a, b, НОД)
    арг a, b
    рез НОД
нач нат r
    пока r ≠ 0
    нц
        r := Ост(a, b)
        a := b
        b := r
    кц
        НОД := b
кон
```

Второй вариант алгоритма Евклида (рис. 3,*г*) запишется так:

алг ЕВКЛИДА 2. (**нат** a, b, НОД)
  арг a, b
  рез НОД
нач нат x, y
  $x := a$; $y := b$
  пока $x \neq y$
  нц
    если $x > y$
      то $x := x - y$
      иначе $y := y - x$
    все
  кц
  НОД $:= x$
кон

Теперь Вам нужно внимательно изучить эти записи. Они сконструированы из частей, которые мы выше разъяснили.

Для тренировки в переводе блок-схем в алгоритмические записи предлагаем Вам решить несколько задач.

Задачи

5.1. Переведите на АЯ блок-схемы, которые Вы построили при решении задачи 4.3.

5.2. В прямоугольной системе координат задана точка $A(x, y)$, где $x \neq 0$, $y \neq 0$. Постройте блок-схему и составьте запись на АЯ алгоритма вычисления номера n координатной четверти, которой принадлежит точка A.

Указание: считать

$$n = \begin{cases} 1, \text{ если } x > 0,\ y > 0, \\ 2, \text{ если } x < 0,\ y > 0, \\ 3, \text{ если } x < 0,\ y < 0, \\ 4, \text{ если } x > 0,\ y < 0. \end{cases}$$

5.3. В прямоугольной системе координат задан центр $A(x_1, y_1)$ круга радиуса R и точка M (x_2, y_2). Составьте блок-схему и запись на АЯ алгоритма, определяющего, принадлежит ли точка M заданному кругу.

Указание: введите вспомогательные переменные $a = (x_1 - x_2)^2$, $b = (y_1 - y_2)^2$ и $d = \sqrt{a + b}$.

5.4. В прямоугольной системе координат заданы две точки $A_1(x_1, y_1)$ и $A_2(x_2, y_2)$. Составьте блок-схему и запись на АЯ алгоритма, определяющего, какая из данных точек расположена ближе к началу координат.

5.5. Составьте блок-схему и запись на АЯ алгоритма вычисления абсолютной величины числа.

Указание: воспользуйтесь определением абсолютной величины:

$$|x| = \begin{cases} x, \text{ если } x \geqslant 0, \\ -x, \text{ если } x < 0. \end{cases}$$

5.6. Составьте блок-схему и алгоритмическую запись алгоритма для вычисления значения функции:

$$y = \begin{cases} x + a, \text{ если } 1 < x \leqslant 5, \\ x + 2b, \text{ если } 5 < x < 20, \\ x - a, \text{ если } x \geqslant 20. \end{cases}$$

6. Какие бывают алгоритмы?

Этот вопрос, конечно, у Вас возникает.

Мы уже рассмотрели довольно много примеров алгоритмов (если считать и те, которые Вы построили, решая предложенные задачи), и у каждого из этих алгоритмов имеется либо ветвление, либо цикл, или то и другое.

Алгоритм, содержащий ветвление (хотя бы одно), а также процесс решения задачи в соответствии с таким алгоритмом, принято называть **разветвляющимся.**

Алгоритм, содержащий цикл (хотя бы один), а также процесс решения задачи в соответствии с таким алгоритмом, принято называть **циклическим.**

Ну а если алгоритм содержит и ветвление, и цикл? Тогда он и разветвляющийся, и циклический.

У Вас, естественно, может возникнуть вопрос: существуют ли алгоритмы, в которых нет ни ветвлений, ни циклов? Разумеется, существуют и такие алгоритмы. Они встречаются как в математике, так и вне ее, в частности в быту (например, различные рецепты приготовления пищи и др.). Это — наиболее простые по своей структуре алгоритмы, называемые **линейными.**

Приведем два примера.

1) **Вычисление по формуле.** Задача. Вычислить значение переменной y, заданной формулой

$$y = \frac{3x + 5}{x^2 + 1}$$

по данному значению x.

Сама эта формула еще не определяет однозначно алгоритм вычисления значения y по заданному значению x. Вычисления можно начать с числителя или со знаменателя.

Один из возможных алгоритмов вычисления по заданной формуле можно представить в виде следующего предписания:

1. $a := 3 \cdot x$
2. $a := a + 5$
3. $b := x \cdot x$

4. $b := b + 1$
5. $y := a/b$
6. Конец

Мы опустили здесь указания, к какой команде переходить, так как в этом предписании команды выполняются одна за другой строго в том порядке, в каком они записаны (и занумерованы).

В этом случае совершенно очевиден перевод предписания в блок-схему (рис. 6), состоящую из одних прямоугольных блоков (за исключением блоков начала и конца), или в запись на АЯ:

```
алг ВЫЧИСЛЕНИЕ y (вещ x, y)
    арг x
    рез y
нач вещ a, b
    a := 3 · x
    a := a + 5
    b := x · x
    b := b + 1
    y := a/b
кон
```

Чаще всего, особенно в математике, линейные алгоритмы входят в качестве составных частей (подалгоритмов) в более сложные алгоритмы (разветвляющиеся или циклические). Так, если необходимо вычислить значение переменной y по заданной выше формуле не для одного, а для нескольких значений переменной x, например, для $x = 0, 1, 2, \ldots, 10$, то алгоритм для решения этой задачи можно представить в виде циклического алгоритма, включающего приведенный выше алгоритм. Блок-схема этого циклического алгоритма изображена на рисунке 7. В ней через A мы обозначили блок-схему, изображенную на рисунке 6 (разумеется, без блоков начала и конца).

2) **Таджикский плов.** У вас, наверное, возникает вопрос: а причем тут таджикский плов?

Одному из авторов приходилось бывать в Таджикистане, где плов — национальное блюдо. Причем, мужчины готовят это вкусное блюдо не хуже женщин. Естественно, не желая уронить свое мужское достоинство, автор тоже научился этому и может поделиться с Вами соответствующим алгоритмом (кстати, одним из нескольких сот рецептов приготовления плова).

Итак, ПЛОВ ТАДЖИКСКИЙ (на 4 порции)

1. Очищенную луковицу обжарить в сильно разогретом жире (бараньем или растительном масле — 400 г), затем вынуть ее.
2. Положить в жир нарезанную баранину (800 г), нарезанный репчатый лук (500 г) и жарить до образования румяной корочки.
3. Добавить шинкованную морковь (800 г), специи и обжарить еще 8—10 мин.

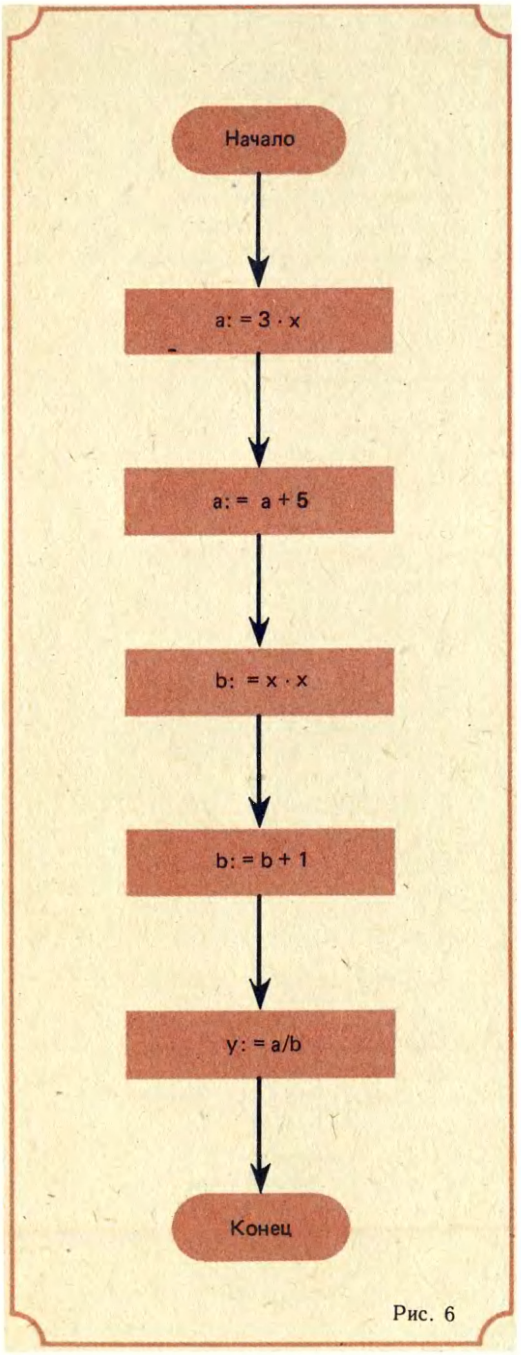


Рис. 6

4. Залить водой, посолить, положить промытый, предварительно замоченный рис (1 кг) и довести до готовности (воды влить примерно 1,5 л).

5. Готовый плов уложить горкой, украсить зеленью и кусочками мяса.

6. Конец (приготовления плова, начало еды).

Как видите, это предписание отвечает требованиям точности и понятности (разумеется для тех, кто умеет выполнять предусмотренные в нем действия), указывает последовательность действий, которые нужно выполнить для решения задачи приготовления таджикского плова.

Цель, с которой мы привели этот пример, разумеется, не только и не столько в том, чтобы научить Вас готовить плов, сколько в том, чтобы показать, как широк объем понятия «действие», охватывающий и арифметические действия, и действия «обжаривания луковицы», «шинкования моркови» и т. д.

* * *

Дорогой читатель! Все уточнения и разъяснения понятия алгоритма, некоторых форм его представления, приведенные выше, не перевели однако это понятие из разряда интуитивных в разряд математических.

Почему? Потому, что

такие слова, как «предписание», «действие», не получили математического определения. Кроме того, даже те слова, которые мы тщательно разъяснили, как, например, «точное», «понятное», тоже не стали после этого математическими, так как в этих разъяснениях мы пользовались другими словами, тоже обозначающими интуитивные понятия, но, возможно, более ясные, чем разъясняемые.

Таким образом, на вопрос «Что такое алгоритм?», вынесенный в название этой книги, мы пока ответили лишь на интуитивном уровне. И хотя понятие алгоритма на этом уровне обеспечивает, в общем, важнейшее приложение — перевод в программу для компьютера (ЭВМ), у Вас, любителя математики, неизбежно возникнет вопрос, а как все же перевести интуитивное понятие алгоритма в точное математическое. Этот вопрос, как и многие другие, аналогичные вопросы, возникающие в математике, на первый взгляд кажется чисто теоретическим. В действительности же он имеет и важное прикладное значение.

Во второй части книги мы еще раз ставим вопрос, что такое алгоритм, но на этот раз уже на другом, математическом уровне.

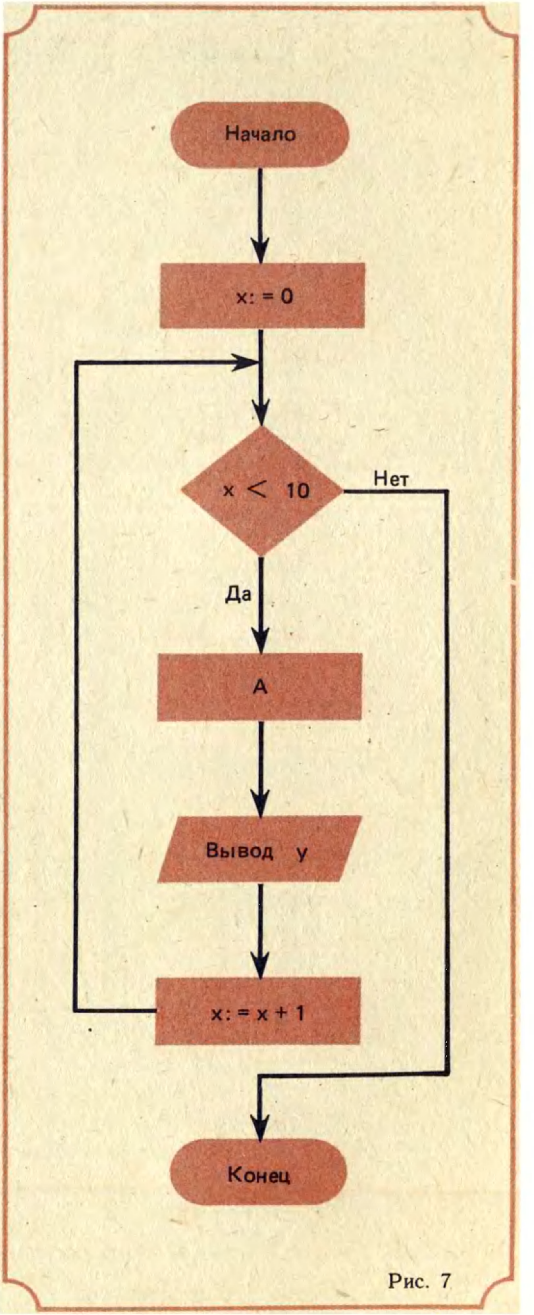


Рис. 7

Еще раз о том, что такое алгоритм

'Математическое понятие)

7. Алгоритмы над словами
8. Проблема слов
9. Нормальный алгоритм Маркова
10. Воображаемая машина
11. Машина Тьюринга — математическое понятие алгоритма

7. Алгоритмы над словами

В первой части Вы узнали, что одно из свойств алгоритмов состоит в том, что они оперируют с конструктивными объектами. Причем, такими объектами могут быть натуральные, целые, рациональные числа, их пары, тройки, многочлены с целыми или рациональными коэффициентами, линейные уравнения или системы таких уравнений с рациональными коэффициентами и другие. Эти объекты мы можем изобразить в виде слов в некотором конечном алфавите или, как иначе говорят, закодировать словами в конечном алфавите.

Например, натуральные числа можно записать в виде слов в алфавите {0, 1, 2, ..., 9}, как мы это делаем обычно, пользуясь десятичной системой счисления. Действительно, запись «3645» представляет собой слово в этом алфавите (конечную последовательность цифр — букв этого алфавита) и обозначает число — результат выполнения операций в выражении $3 \cdot 10^3 + 6 \cdot 10^2 + 4 \cdot 10 + 5$.

Если же взять алфавит, состоящий из единственной буквы — черточки { | }, то «слова» этого алфавита (мы взяли слова в кавычки, так как они очень странно выглядят): |, ||, |||, ||||, |||||, ... можно рассматривать как натуральные числа 1, 2, 3, 4, ... соответственно, закодированные словами этого однобуквенного алфавита.

Целые числа можно записать с помощью слов $-||$, $-|$, 0, |, || в трехбуквенном алфавите {0, |, —}. Добавляя к этому алфавиту еще одну букву «/» (косую палочку), мы уже можем с помощью слов этого четырехбуквенного алфавита записать рациональные числа: |/|||, $-$|||/||||, $-$||/|, ...

Таблицу $\begin{pmatrix} -3 & 0 \\ -2 & 1 \end{pmatrix}$, составленную из целых чисел, можно закодировать словом $-|||*0\square-||*|$ в алфавите {0,1, —, $*\square$}*), систему линейных уравнений $\begin{cases} x-4y=3 \\ 2x+y=-12 \end{cases}$ — словом $x-4y=3*2x+y=-12$, записанным в алфавите {0, 1, 2, ..., 9, +, —, =, *, x, y}; многочлен $x^3-2x^2+1/3\cdot x-6$ — словом $x\cdot x\cdot x-2\cdot x\cdot x+1+1/3\cdot x-6$, записанным в алфавите {0, 1, 2, ..., 9, +, —, ·, /, x}.

Положение о том, что исходные данные, промежуточные и окончательные результаты алгоритмов — конструктивные объекты, которые можно представить в виде слов в конечном алфавите, приводит к мысли рассматривать алгоритмы над словами, отвле-

---

) Буква «» служит для отделения чисел одной строки, «□» — для отделения строк.

каясь от того, записью каких известных нам объектов эти слова являются.

Предварительно рассмотрим основные понятия. К их числу относятся понятия алфавита и слова. Мы ими уже пользовались, предполагая, что интуитивно ясно, о чем идет речь. Теперь мы их определим.

Под **алфавитом** будем понимать конечное (непустое) множество. Элементы этого множества называются **буквами алфавита** (заметьте, это же самое понимают и под алфавитом любого естественного языка, например, русского, английского). Всякая конечная последовательность букв алфавита называется **словом** в этом алфавите. (Здесь уже имеется существенное различие с обычным пониманием слова в естественном языке: не всякая конечная последовательность букв алфавита русского языка воспринимается как слово в этом языке, например, «крнъю» не является словом русского языка, ничего не обозначает в нем.)

Рассмотренные нами слова имели определенный смысл: они обозначали вполне конкретные объекты — натуральные, целые, рациональные числа, таблицу (числовую), систему уравнений, многочлен. Сейчас мы займемся изучением абстрактных букв и слов — букв и построенных из них слов, которые сами по себе ничего не обозначают.

Введем еще понятие длины слова. Число вхождений букв в слове (каждая буква считается столько раз, сколько раз она входит в данное слово) называется его **длиной.**

Пусть наш алфавит состоит всего из двух букв: $\{a, b\}$. Тогда в этом алфавите существуют 4 слова длины 2: aa, ab, ba, bb; 8 слов длины 3 и т. д. Рассматриваются и «однобуквенные» слова — слова длины 1, их всего два: a и b. Удобно будет считать, что существует даже такое странное слово, которое вовсе не содержит букв,— **пустое** слово (аналог пустого множества), обозначаемое Λ. Длину пустого слова принимают равной числу 0.

И еще одно важное понятие — отношение вхождения одного слова в другое. Будем говорить, что слово P **входит** в слово Q, если слово Q имеет вид: P_1PP_2. При этом слова P_1 или P_2 (и оба) могут быть пустыми словами. В последнем случае слово Q есть слово P и можно считать, что каждое из них входит в другое.

Примеры. 1) В слове $babbbaaab$ имеются два вхождения слова ab: одно — начиная со второй буквы, а другое — с восьмой.

2) В слове $6ab - 2aabbb + ab - 2aabbb$ имеется четыре вхождения слова ab, два вхождения слова $-2a$.

Если слово Q имеет вид P_1PP_2 и слово P_1 имеет наименьшую возможную длину, то будем говорить о **первом вхождении** слова P в слово Q.

Мы будем рассматривать **преобразование** одних слов в другие посредством **подстановок** двух видов: $P \leftrightarrow Q$ или $P \rightarrow Q$, где P и Q —

слова в одном и том же алфавите (P или Q может быть пустым словом).

Применение подстановки $P \leftrightarrow Q$ к некоторому слову R допускает как замену вхождения левой части P правой частью Q, так и замену правой части Q левой частью P.

Применение же подстановки $P \rightarrow Q$, называемой **ориентированной,** к слову R допускает лишь замену любого вхождения левой части правой (разумеется, если имеется хотя бы одно вхождение левой части P в слове R).

Примеры. 1) Подстановка $ab \leftrightarrow ba$ применима четырьмя способами к слову *abaabba*. Замена каждого из двух вхождений левой части ab дает слова: *baaabba*, *abababa*, а замена каждого из двух вхождений ba дает слова: *aababba*, *abaabab*. К слову *bbb* эта подстановка неприменима.

2) Ориентированная подстановка $ab \rightarrow ba$ применима лишь двумя способами к слову *abaabba*, в результате чего получаем слова: *baaabba* и *abababa* соответственно.

3) Нахождение числового значения выражения $((10+2) \times (8-3)):(15-3)$ можно рассматривать как последовательность применения ориентированных подстановок: $(10+2) \rightarrow 12$, $(8-3) \rightarrow 5$, $(15-3) \rightarrow 12$, $(12 \cdot 5) \rightarrow 60$, $60:12 \rightarrow 5$ к слову $((10+2) \times (8-3)):(15-3)$, записанному в алфавите $\{0, 1, 2, \ldots, 9, +, -, \cdot, :, (,)\}$.

4) Нахождение решения системы линейных уравнений с целыми коэффициентами $\begin{cases} x-4y=3 \\ 2x+y=-12 \end{cases}$ можно рассматривать как последовательность применения ориентированных подстановок:

$$\begin{gathered} x \rightarrow 3+4y, \\ 2x+y=-12 \rightarrow 2(3+4y)+y=-12, \\ 2(3+4y)+y=-12 \rightarrow 6+9y=-12, \\ 6+9y=-12 \rightarrow 9y=-18, \\ 9y=-18 \rightarrow y=-2, \\ y=-2 \rightarrow x=-5 \end{gathered}$$

к слову $x-4y=3 * 2x+y=-12$, записанному в алфавите $\{0, 1, 2, \ldots, 9, +, -, =, *, x, y, (,)\}$.

Подстановка вида $P \leftrightarrow \Lambda$ означает, во-первых, что в любом слове вместо P можно «подставить пустое слово», т. е. попросту выбросить вхождение слова P. Если, например, подстановку $bb \leftrightarrow \Lambda$ применить таким образом ко второму вхождению слова bb в слово *abababbababb*, получится слово *abababbaba*.

Подстановка $P \leftrightarrow \Lambda$ означает, во-вторых, что в любом слове между любыми двумя буквами можно «вставить» слово P (естественно считать, что пустое слово входит между любыми буквами). Кроме того, применяя подстановку $P \leftrightarrow \Lambda$ «справа нале-

во», можно «приписать» слово P как впереди, так и позади любого слова.

Примеры алгоритмов над словами. Для изображения натуральных чисел будем пользоваться однобуквенным алфавитом $A=\{|\}$ ($1-|$, $2-||$, $3-|||$, ...).

Пример 1. Построим алгоритм, который по любому слову $\underbrace{|||...|}_{n}$ в алфавите A, содержащему n черточек, дает слово $\underbrace{|||...|}_{n+1}$ в этом же алфавите A, содержащее $n+1$ черточек.

Этот алгоритм может быть сформулирован в виде следующего словесного предписания: «Припиши к данному слову справа вертикальную черточку и остановись. Полученное слово и есть результат».

Рассмотренный алгоритм — алгоритм перехода от произвольного натурального числа к числу, на единицу большему, или алгоритм, перерабатывающий слово, изображающее число n, в слово, изображающее число $n+1$.

Действие приписывания черточки можно задать с помощью ориентированной подстановки: $|\rightarrow||$. Но алгоритм, заданный одной этой подстановкой, будет бесконечно прибавлять по одной черточке к исходному слову, никогда не останавливаясь (хорошая иллюстрация незавершимости процесса образования натурального ряда). Для указания остановки пользуются **заключительной подстановкой:** $|\rightarrow\cdot||$ (точка за стрелкой указывает на то, что эта подстановка применяется только один раз, после чего наступает окончание процесса).

Итак, наш алгоритм может быть задан одной заключительной подстановкой $|\rightarrow\cdot||$.

Пример 2. Добавим к алфавиту $A=\{|\}$ звездочку в качестве новой буквы. Получим новый алфавит $B=\{|, *\}$, пригодный для записи систем натуральных чисел. Так, если слова $||$, $||||$, $|||$, $|$ представляют натуральные числа 2, 5, 3, 1 соответственно, то слово $||*|||||*|||*|$ в алфавите B представляет уже систему этих чисел.

Построим алгоритм, позволяющий по любому слову $\underbrace{||...|}_{n_1}*$ $*\underbrace{||...|}_{n_2}*...*\underbrace{||...|}_{n_k}$ в алфавите B получать слово $\underbrace{||...|}_{n_1+n_2+\ldots+n_k}$ в этом же алфавите.

Требуемый алгоритм можно задать предписанием: «Сотри все звездочки и остановись. Полученное слово и есть результат».

Этот алгоритм является алгоритмом сложения системы натуральных чисел, перерабатывающим слово, обозначающее систему k натуральных чисел, в слово, обозначающее сумму этих чисел.

Этот алгоритм можно задать и с помощью ориентированной подстановки $|*|\rightarrow||$, так как каждое применение этой подстановки стирает одну звездочку до тех пор, пока в получаемых словах еще имеются звездочки. Когда же исчезнут все звездочки, т. е. не

будет больше вхождений левой части подстановки ($|*|$) в рассматриваемом слове, работа алгоритма заканчивается, и полученное слово (без звездочек) является результатом переработки исходного слова (содержащего звездочки) данным алгоритмом.

Этот алгоритм можно представить и с помощью блок-схемы, изображенной на рисунке 8.

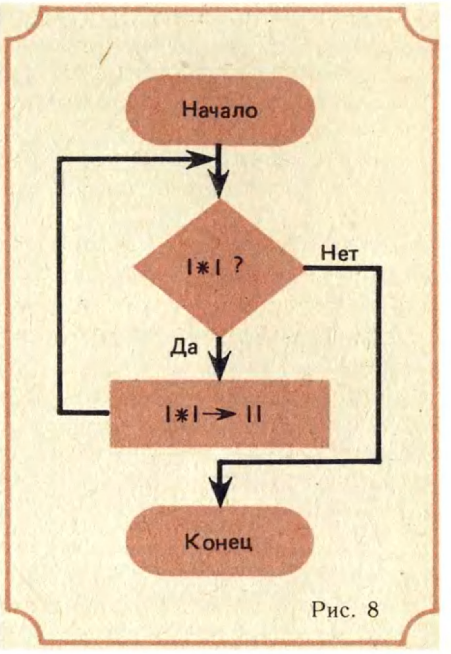


Рис. 8

На этой блок-схеме под записью «$|*|$?» надо понимать условие (или вопрос): «входит ли $|*|$ в рассматриваемое слово?» Здесь наглядно видно, что процесс применения подстановки, вообще говоря, является циклическим (разумеется, за исключением случая, когда исходное слово содержит не более одной звездочки).

Пример 3. Рассмотрим алгоритм, позволяющий по любому слову $\underbrace{||...|}_{n} * \underbrace{||...|}_{m}$ в алфавите $B = \{|, *\}$, изображающему пару натуральных чисел n и m, получить слово $\underbrace{||...|}_{|n-m|}$, изображающее модуль разности этих чисел.

Зададим алгоритм следующим предписанием:

1. Подставь вместо первого вхождения слова $|*|$ в данное слово звездочку, переходи к ук. 2.
2. Если во вновь полученном слове есть вхождение слова $|*|$, то переходи к ук. 1, иначе — к ук. 3.
3. Сотри звездочку, переходи к ук. 4.
4. Процесс вычисления остановить. Полученное слово является результатом.

Работа этого алгоритма над словом $||*|||$ приводит к следующей последовательности слов: $||*|||$, $|*||$, $*|$, $|$. Получаем: $|2-3| = 1$.

Содержащиеся в данном предписании указания о подстановке вместо первого вхождения слова $|*|$ в данное слово звездочки и стирания звездочки мы можем записать короче, трактуя эти указания как применение ориентированных подстановок соответственно: $|*| \to *$, $* \to \Lambda$. Поэтому рассмотренный нами алгоритм можно задать упорядоченной системой подстановок:

$$\begin{cases} |*| \to * \\ * \to \Lambda. \end{cases}$$

Условимся считать, что этот алгоритм в применении к произвольному слову P в алфавите $\{|, *\}$ работает так: берется первая по порядку подстановка, которая применима к слову P и при наличии нескольких вхождений ее левой части в слове P подстановку применяют к первому вхождению; для полученного таким образом слова P' опять берется первая в этом списке подстановка, которая к нему применима, и она применяется к первому вхождению своей левой части и т. д. Если после конечного числа таких шагов получим слово Q, к которому ни одна из подстановок уже неприменима, то будем говорить, что алгоритм **перерабатывает** слово P в слово Q.

Применим описанный алгоритм, например к слову $|||*||||$. Первая подстановка $|*| \to *$ применима к этому слову. Подчеркнем в данном слове первое вхождение левой части подстановки: $|||\underline{*|}|||$. В результате применения подстановки получим слово: $||\underline{*||}|$, к которому опять применима первая подстановка. Применяя ее, получим $\underline{|*|}|$. Это слово еще содержит вхождение левой части подстановки, в результате применения ее получим слово: $*|$, к которому первая подстановка уже неприменима, но применима вторая подстановка: $* \to \Lambda$. Результат ее применения — слово $|$. К слову $|$ неприменима ни первая, ни вторая подстановка. Следовательно, наш алгоритм перерабатывает слово $|||*||||$ в слово $|$.

Пример 4. Рассмотрим алгоритм, распознающий слова в алфавите $A = \{a, b\}$, содержащие равное число букв a и b.

Будем считать, что этот алгоритм перерабатывает слова, содержащие равное число букв a и b (и только их) в пустое слово.

В таком случае искомый алгоритм можно представить в виде следующего предписания:

1. Подставь в данное слово вместо первого вхождения слова ba слово ab. Переходи к ук. 2.

2. Если во вновь получившемся слове есть вхождение слова ba, то переходи к ук. 1, иначе — к ук. 3.

3. Выброси первое вхождение слова ab, переходи к ук. 4.

4. Если в получившемся слове есть вхождение слова ab, то переходи к ук. 3, иначе — к ук. 5.

5. Сравни полученное слово с пустым словом. Если оно совпадает с ним, то переходи к ук. 6, иначе — к ук. 7.

6. Исходное слово содержит равное число букв a и b. Переходи к ук. 8.

7. Исходное слово содержит неравные числа вхождений букв a и b. Переходи к ук. 8.

8. Процесс вычисления остановить.

В основе этого алгоритма заложен следующий принцип: в перерабатываемом слове P в алфавите $\{a, b\}$ делается перестановка букв b и a, пока слово не примет вид: $aa...abb...b$ (все буквы a стоят левее всех букв b). В получившемся слове затем последова-

тельно уничтожается равное число букв a и b (каждое слово ab заменяется пустым словом).

В этом примере алгоритма множество исходных объектов — множество всевозможных слов в алфавите $\{a, b\}$. В отличие от примеров 1—3 эти слова (пока) «ничего не обозначают», т. е. они не являются записью каких-либо известных нам объектов (чисел, многочленов и т. д.). Промежуточные и окончательные результаты этого алгоритма — слова в алфавите $\{a, b\}$.

Этот алгоритм мы можем задать с помощью следующей упорядоченной системы ориентированных подстановок:

$$\begin{cases} ba \to ab \\ ab \to \Lambda \end{cases}$$

или блок-схемы, изображенной на рисунке 9.

Работа этого алгоритма, например, над словом $ababba$ приводит к следующей последовательности слов: $a\underline{ba}bba$, $aab\underline{ba}$, $aa\underline{bab}$, $aa\underline{ab}bb$, $a\underline{ab}$, Λ. Поскольку слово $ababba$ переработано алгоритмом в Λ, получаем, что оно содержит равное число букв a и b.

Рассмотрим еще один пример.

Пример 5. Укажем алгоритм, позволяющий по любому слову в алфавите $\{a, b\}$ определить, имеет ли оно четную длину.

Условимся считать, что этот алгоритм должен перерабатывать слова в алфавите $\{a, b\}$, имеющие четную длину (и только их), в пустое слово. В таком случае этот алгоритм может быть задан следующей системой ориентированных подстановок:

$$\begin{cases} a \to | \\ b \to | \\ || \to \Lambda \end{cases}$$

Этот же алгоритм, разумеется, может быть оформлен и в виде предписания:

1. Вместо каждой буквы из алфавита $\{a, b\}$ выпиши |, переходи к ук. 2.
2. Выясни, содержит ли полученное слово вхождение слова ||. Если да, переходи к ук. 3, иначе — к ук. 4.
3. Выброси каждое вхождение слова ||, переходи к ук. 4.
4. Сравни полученное слово с пустым. Если они совпадают, то переходи к ук. 5, иначе — к ук. 6.
5. Слово имеет четную длину. Переходи к ук. 7.
6. Слово имеет нечетную длину. Переходи к ук. 7.
7. Процесс вычисления остановить.

Этот же алгоритм можно представить и с помощью блок-схемы, изображенной на рисунке 10.

В работе рассмотренного алгоритма над исходным словом в алфавите $\{a, b\}$ можно выделить два этапа: а) нахождение длины исходного слова; б) распознавание четности длины слова.

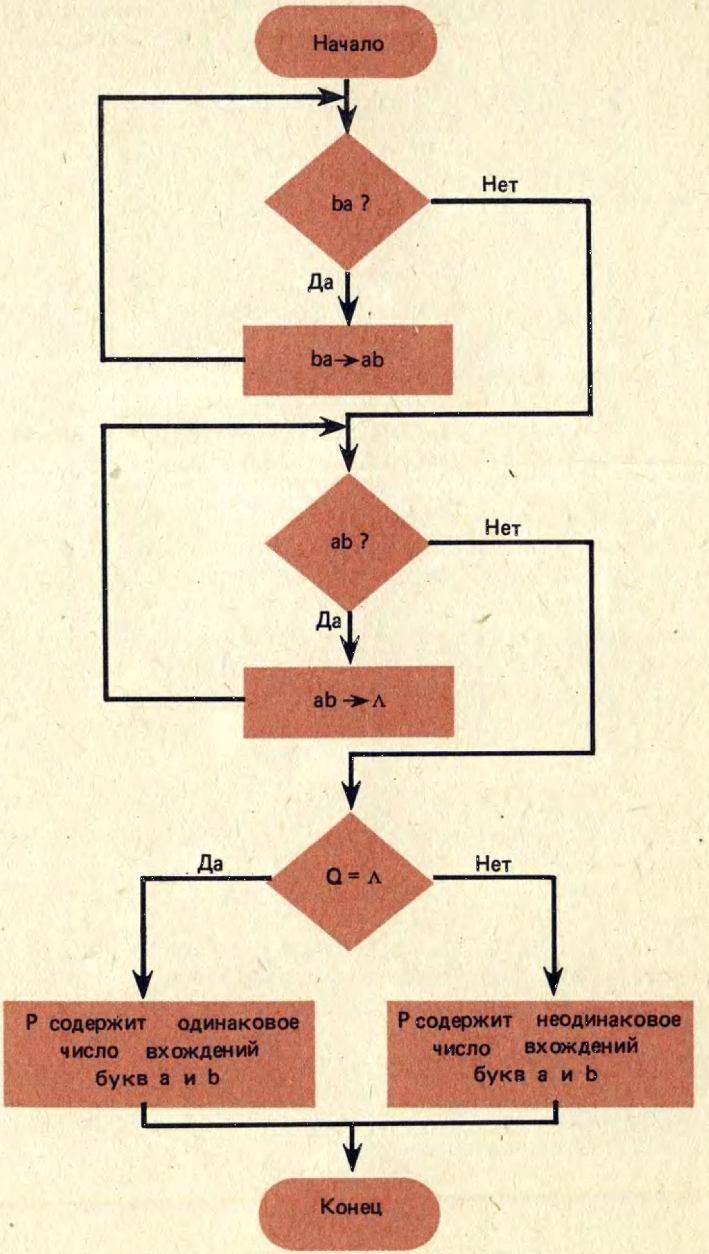

Начало
ba ?
Нет
Да
ba→ab
ab ?
Нет
Да
ab→Λ
Да
Q = Λ
Нет
P содержит одинаковое число вхождений букв a и b
P содержит неодинаковое число вхождений букв a и b
Конец

Рис. 9

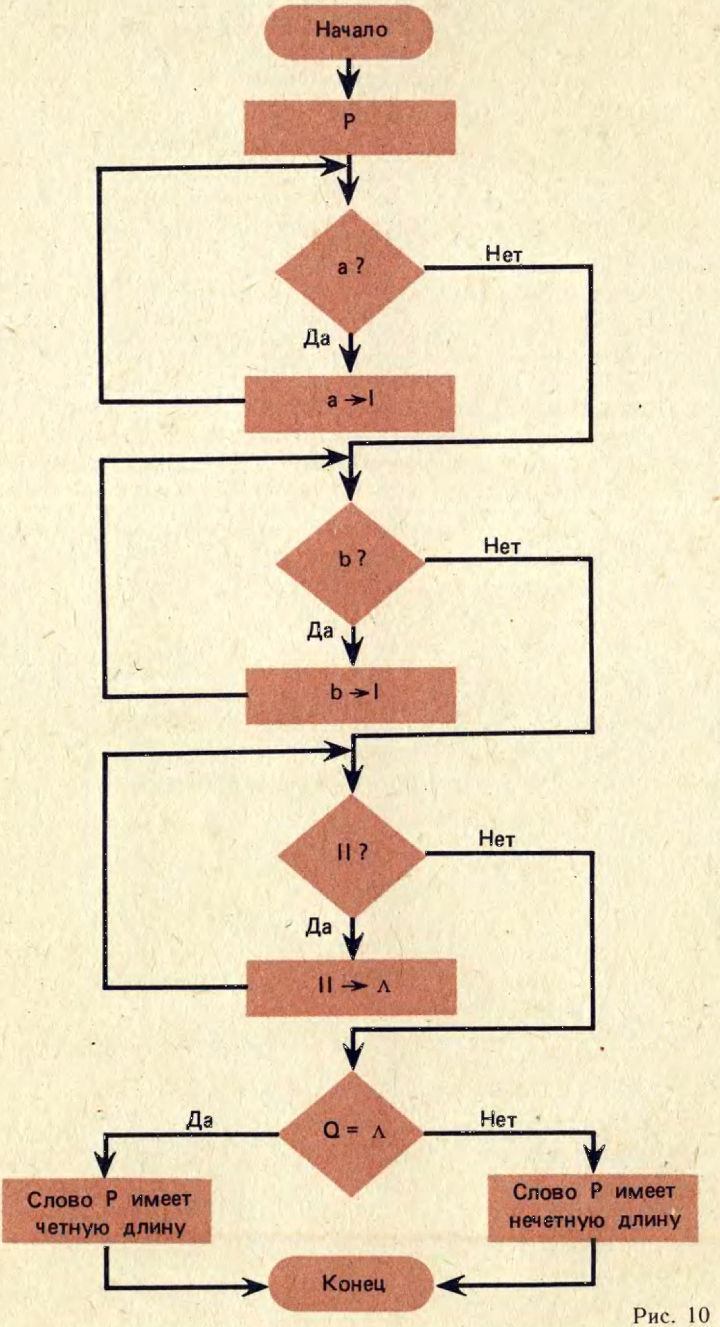

Начало
P
a ?
Нет
Да
a → l
b ?
Нет
Да
b → l
l l ?
Нет
Да
l l → Λ
Q = Λ
Да
Нет
Слово P имеет четную длину
Слово P имеет нечетную длину
Конец

Рис. 10

Рассмотрим теперь два алгоритма:

A_1 — перерабатывающий каждое слово в алфавите $\{a, b\}$ в его длину. Алгоритм A_1 задается системой ориентированных подстановок:

$$\begin{cases} a \to | \\ b \to | \end{cases}$$

A_2 — алгоритм, распознающий по записи любого натурального числа в алфавите $\{|\}$, является ли оно четным. Алгоритм A_2 задается ориентированной подстановкой $|| \to \Lambda$.

Работа алгоритма из примера 5 состоит в последовательном применении двух алгоритмов A_1 и A_2 таким образом, что исходным словом P для этого алгоритма является исходное слово P алгоритма A_1, а результат работы алгоритма A_1 над этим словом — слово P_1 — является исходным словом для алгоритма A_2. Результат работы алгоритма A_2 над словом P_1 — слово Q — есть результат работы рассматриваемого нами алгоритма над словом P.

Последовательное применение алгоритмов A_1 и A_2 (в заданном порядке) называется **композицией** этих алгоритмов и обозначается $A_1 \circ A_2$. Таким образом, алгоритм из примера 5 является композицией алгоритмов A_1 и A_2, называемых также **подалгоритмами** этого алгоритма.

* * *

Мы обещали Вам, дорогой читатель, задавать себе вопросы, которые могут у Вас возникнуть при чтении книги. Однако в этой беседе мы почти не задавали себе вопросы, так как нам представляется, что ее содержание чрезвычайно просто. Один вопрос все же поставим: зачем нужны алгоритмы над словами, ничего не обозначающими?

Отвечаем: во-первых, мы уже говорили, что эти слова ничего не обозначают лишь пока мы им не даем никакой интерпретации (никакого истолкования), а во-вторых, и это главное, рассмотрение таких абстрактных слов и алгоритмов над такими словами необходимо для подготовки к одному из существующих математических уточнений понятия алгоритма, к которому мы, хотя и медленно, но приближаемся.

А пока предлагаем Вам несколько несложных задач.

Задачи

7.1. Примените алгоритм из примера 2 к переработке слова $|||||*||||*|||*||$ в алфавите $B = \{|, *\}$. Что обозначают исходное слово и результат?

7.2. Примените алгоритм из примера 3 к переработке слова $|||||||*||||$ в алфавите $B = \{|, *\}$. Что обозначают исходное слово и результат?

7.3. Имитируйте работу алгоритма из примера 4 над словом *abbaaabab* в алфавите $\{a, b\}$.

7.4. Постройте с помощью системы ориентированных подстановок и блок-схемы алгоритм, перерабатывающий любое слово в алфавите $\{a, b, c\}$ в слово, в котором все буквы a (если они имеются) стоят левее всех остальных, а все буквы b — левее всех букв c.

7.5. Постройте алгоритм, преобразующий каждое слово в алфавите $\{a, b, c\}$ в пустое слово.

7.6. Постройте алгоритм, преобразующий каждое слово P в алфавите $\{a, b, c\}$ в слово $abcP$.

7.7. Используя алгоритмы из задач 7.5 и 7.6, постройте алгоритм, преобразующий каждое слово в алфавите $\{a, b, c\}$ в слово abc.

7.8. Алгоритм над словами в алфавите $\{a, b\}$ задан следующей системой ориентированных подстановок:

$$\begin{cases} ba \to ab \\ aa \to \Lambda \\ bb \to \Lambda \end{cases}$$

Примените этот алгоритм к следующим словам: 1) $bbaaababa$; 2) $abbababab$; 3) $ababab$; 4) $baabbaba$.

7.9. Докажите, что алгоритм из задачи 7.8 перерабатывает любое слово в алфавите $\{a, b\}$ в одно из следующих слов: a, b, ab, Λ.

8. Проблема слов

«Что еще за «проблема слов»?» — удивляетесь Вы. Потерпите немного, мы Вам все объясним. Но предварительно предложим Вам задачу.

Задача 8.1. При подаче некоторых команд на кораблях традиционно применяют флажки. При этом сигнальщиком используются только два положения флажков, соответствующие буквам, например a и b. В целях зашифровки одна и та же команда может быть передана по-разному. Пусть две команды означают одно и то же, если соответствующие этим командам слова получаются одно из другого при помощи некоторого количества операций: замена сочетания букв ba сочетанием ab и обратно, пропуск идущих подряд букв aa или bb и добавление в любое место сочетания этих букв.

Приведите примеры: 1) одинаковых команд; 2) различных команд; 3) другие виды команды, соответствующей слову ab; 4) докажите, что команды, соответствующие словам aba и $abbbaaa$, одинаковы; 5) докажите, что команды, соответствующие словам $aabba$ и $bbaab$, различны; 6) докажите, что слова ba, $baaa$, $bbba$, $aaab$, $bbab$ обозначают ту же команду, что и слово ab (что всем этим словам соответствуют одинаковые команды).

Эта задача, мы уверены, не вызывает у Вас затруднений. (Если же возникнут затруднения, взгляните в ответы. Это уже на крайний случай.)

Обратим внимание на то, что команды в рассматриваемой задаче соответствуют словам в алфавите $\{a, b\}$. Но различных (по своему начертанию) слов в алфавите $\{a, b\}$ существует бесконечное множество. Однако различные слова могут означать одно и то же (например, команды, соответствующие различным словам ab и ba, одинаковы). В этом случае слова будем называть **эквивалентными** и обозначать это знаком «$\sim$». Кроме обычных слов в алфавите, мы условились рассматривать и пустое слово Λ, не содержащее ни одной буквы (сигнальщик не подает команды).

Вот примеры эквивалентных слов: $aa \sim \Lambda$, $bb \sim \Lambda$, $ab \sim ba$, $aba \sim abbbaaa$.

Приведите примеры эквивалентных слов, возникших при решении задачи 8.1.

Сформулируем теперь более общую **задачу 8.2**: для любых двух команд, подаваемых сигнальщиком, требуется узнать, одинаковы они или нет.

В поставленной задаче речь идет о нахождении общего, единого метода, т. е. алгоритма, позволяющего по любым двум командам определить, одинаковы они или нет.

Придадим задаче 8.2 точную математическую формулировку, для чего введем некоторые понятия.

Ранее нами было рассмотрено понятие преобразования одних слов в другие посредством подстановок. Используя это понятие, мы можем сказать, что в условии задачи 8.1 рассматриваются преобразования слов (команд) посредством подстановок: $ba \leftrightarrow ab$, $aa \leftrightarrow \Lambda$, $bb \leftrightarrow \Lambda$.

Определение 1. Множество всех слов в некотором алфавите с какой-нибудь конечной системой подстановок называется **ассоциативным исчислением.**

Пример. В условии задачи 8.2 рассматривается множество всех слов (команд) в алфавите $\{a, b\}$ с системой подстановок $ba \leftrightarrow ab$, $aa \leftrightarrow \Lambda$, $bb \leftrightarrow \Lambda$ (обратите внимание — неориентированных подстановок), т. е. рассматривается ассоциативное исчисление с алфавитом $\{a, b\}$ и указанной системой подстановок.

Определение 2. Слова P и Q называются **смежными** в ассоциативном исчислении, если одно может быть преобразовано в другое путем однократного применения подстановки.

Примеры. Пары слов (команд) в задаче 8.2: ba, ab; abb, a; bab, abb — смежны.

Определение 3. Слова P и Q называются **эквивалентными** в ассоциативном исчислении, если существует последовательность слов P, P_1, P_2, ..., P_n, Q таких, что P и P_1 — смежные, P_1 и P_2 — смежные, ..., P_n и Q — смежные.

Примеры. Рассмотрим ассоциативное исчисление с алфавитом $\{a, b\}$ и системой подстановок: $ba \leftrightarrow ab$, $aa \leftrightarrow \Lambda$, $bb \leftrightarrow \Lambda$.

1) $aabbba \sim abaabb$, что видно из следующей последовательности слов: $aabbba$, $ababba$, $ababab$, $abaabb$, где подчеркнуты те

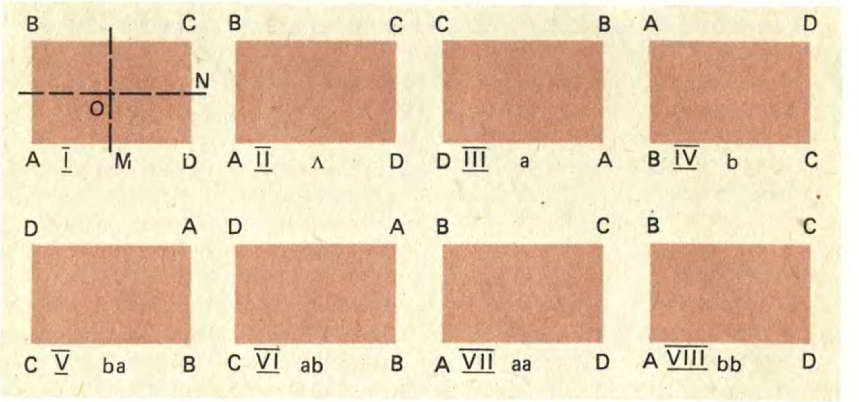


Рис. 11

подслова, которые в следующем слове заменены по соответствующей подстановке.

2) $aba \sim abbbaaa$: $a\underline{bbb}aaa$, $aba\underline{aa}$, aba.

Поскольку в условии задачи 8.2, как мы уже отмечали, рассматривается ассоциативное исчисление с алфавитом $\{a, b\}$ и системой подстановок: $ba \leftrightarrow ab$, $aa \leftrightarrow \Lambda$, $bb \leftrightarrow \Lambda$, то задаче 8.2 мы можем придать следующую **математическую формулировку:** указать алгоритм, позволяющий по любой паре слов данного ассоциативного исчисления определить, эквивалентны они или нет.

Приведем два решения этой задачи.

I. Воспользуемся геометрической иллюстрацией: возьмем какой-нибудь прямоугольник (рис. 11) и рассмотрим следующие его самосовмещения, т. е. движения, которые отображают прямоугольник в себя: 1) симметрия с осью OM, проходящей через центр симметрии прямоугольника O и параллельной двум его сторонам; 2) симметрия с осью ON, проходящей через центр симметрии прямоугольника O и параллельной двум другим его сторонам.

Эти самосовмещения назовем **элементарными** и обозначим буквами a, b соответственно. На рисунке 11 (III, IV) показано, как изменяется расположение вершин прямоугольника при каждом из элементарных самосовмещений.

Композиция двух или нескольких самосовмещений прямоугольника — также самосовмещение прямоугольника. Например, композиция ba есть результат симметрии относительно OM с последующей симметрией относительно ON — равносильно симметрии относительно центра прямоугольника (рис. 11,V), т. е. самосовмещение прямоугольника.

Композиция самосовмещений прямоугольника коммутативна (переместительна), расположение вершин прямоугольника, возникающее при самосовмещениях ab и ba,— одно и то же (рис. 11, V, VI), а также ассоциативно. Поэтому в композициях можно пренебрегать расстановкой скобок. Например, $(ab)(aa)$, $(((ab)a)a)$,

$abaa$ дают одно и то же самосовмещение прямоугольника — симметрию относительно центра O.

Рассмотрим теперь множество F, состоящее из элементарных самосовмещений a, b и всех самосовмещений прямоугольника, представимых в виде композиции конечного числа элементарных самосовмещений (т. е. в виде слов в алфавите $\{a, b\}$). Мы получим некоторое соответствие между множеством всех слов в алфавите $\{a, b\}$ и множеством самосовмещений. Это соответствие не является обратимым, так как различные слова в алфавите $\{a, b\}$ могут представлять одно и то же самосовмещение из F. Такие слова будем считать равными и обозначать обычным знаком равенства «$=$». Например, $ab = ba$, так как расположение вершин прямоугольника, возникающее в результате самосовмещений прямоугольника ab и ba, одно и то же (рис. 11, V, VI).

Каждое из слов aa, bb задает одно и то же самосовмещение прямоугольника, при котором все вершины остаются на прежних местах — **тождественное движение** (рис. 11, VII, VIII). Поскольку это движение ничего не меняет, будем изображать его пустым словом — Λ. Итак, имеют место равенства:

$$ba = ab, \tag{1}$$

$$aa = \Lambda, \tag{2}$$

$$bb = \Lambda. \tag{3}$$

Сравнивая равенства (1) — (3) с подстановками рассматриваемого ассоциативного исчисления, легко установить связь между этим исчислением и системой самосовмещений прямоугольника: две композиции элементарных самосовмещений прямоугольника задают одно и то же самосовмещение тогда и только тогда, когда изображающие их слова эквивалентны в соответствующем ассоциативном исчислении.

Докажем это. Действительно, из равенств (1)—(3) следует, что любые два смежные слова равны, так как при каждом применении любой допустимой подстановки к любому слову P оно переходит в равное слово. Тогда эквивалентность двух слов P и Q в исчислении влечет равенство их, так как в последовательности слов $P, P_1, P_2, \ldots, P_n, Q$, ведущей от P к Q, любые два смежные слова равны, следовательно, $P = Q$. Справедливо и обратное утверждение: если $P = Q$, то $P \sim Q$. Каждое слово рассматриваемого исчисления эквивалентно одному из слов: Λ, a, b, ab. Действительно, применяя к произвольному слову P подстановку $ba \leftrightarrow ab$, мы придем к эквивалентному слову, в котором все буквы a стоят левее всех букв b, т. е. к слову вида $\underbrace{aa\ldots a}_{l}\underbrace{bb\ldots b}_{m}$, где l и m — неотрицательные целые числа. Далее, применяя подстановки $aa \leftrightarrow \Lambda$, $bb \leftrightarrow \Lambda$ (т. е. удаляя пары идущих подряд букв aa, bb), получим, что каждое из слов $\underbrace{aa\ldots a}_{l}\underbrace{bb\ldots b}_{m}$ эквивалентно одному из слов: Λ, a, b, ab, а следовательно, и произвольное слово эквивалентно одному из этих слов. Из того, что $P = Q$ и что слова Λ, a, b, ab задают попарно различ-

ные самосовмещения, следует, что слова P и Q эквивалентны одному и тому же слову из указанного множества слов. Следовательно, слова P и Q эквивалентные одному и тому же слову, эквивалентны. Таким образом, мы доказали, что если $P = Q$, то $P \sim Q$.

Доказанное утверждение позволяет нам, исходя из рассмотренной геометрической иллюстрации, построить следующий алгоритм, распознающий эквивалентность слов в заданном ассоциативном исчислении, т. е. позволяющий по любой паре слов P и Q определять, эквивалентны они или нет:

1. Осуществить последовательно самосовмещения прямоугольника, соответствующие слову P. Перейти к ук. 2.
2. Осуществить последовательно самосовмещения прямоугольника, соответствующие слову Q. Перейти к ук. 3.
3. Сравнить расположения вершин прямоугольника. Если они одинаковы, то перейти к ук. 4, иначе — к ук. 5.
4. $P \sim Q$. Процесс окончен.
5. $P \not\sim Q$. Процесс окончен.

Теперь предложим Вам задачу.

Задача 8.3. Используя построенный алгоритм, установить, что 1) $aabb \not\sim b$; 2) $abab \sim \Lambda$.

II. Для рассматриваемого ассоциативного исчисления можно построить и другой алгоритм, распознающий эквивалентность любой пары слов в этом исчислении.

Присмотревшись к подстановкам $ab \leftrightarrow ba$, $aa \leftrightarrow \Lambda$, $bb \leftrightarrow \Lambda$ данного ассоциативного исчисления, можно заметить, что они сохраняют четность числа вхождений как буквы a, так и буквы b. Как мы уже знаем, любое слово эквивалентно в нашем исчислении одному из слов: Λ, a, b, ab (назовем их **эталонными**). Поскольку у любых двух эталонных слов либо четности числа вхождений буквы a, либо четности числа вхождений буквы b различны, как это видно из таблицы

	Λ	a	b	ab
a	0	1	0	1
b	0	0	1	1

эталонные слова попарно не эквивалентны.

Для построения алгоритма, распознающего эквивалентность слов в рассматриваемом исчислении, предварительно построим вспомогательный алгоритм — алгоритм приведения, который указывает для любого слова эквивалентное ему эталонное слово. Для этого рассмотрим систему ориентированных подстановок:

$$\begin{cases} ba \to ab \\ aa \to \Lambda \\ bb \to \Lambda. \end{cases}$$

Алгоритм приведения в применении к какому-либо слову P работает так: берется первая по порядку подстановка, которая при-

менима к слову P, и при наличии нескольких вхождений ее левой части в слове P подстановку применяют к первому вхождению; для полученного таким образом слова P' опять берется первая из подстановок, которая к нему применима, и она применяется к первому вхождению своей левой части и т. д. Если после конечного числа таких шагов получается слово Q, к которому ни одна из этих подстановок уже неприменима, то говорят, что алгоритм применим к слову P и перерабатывает его в слово Q.

Описанный алгоритм приведения применим к любому слову нашего исчисления и перерабатывает его в одно из следующих эталонных слов: Λ, a, b, ab. Действительно, если в слове P есть вхождения слова ba, то будет применяться первая подстановка, перемещающая букву a левее буквы b до тех пор, пока все a не окажутся левее всех b. Далее начнет работать вторая подстановка, вычеркивающая пары соседних a до тех пор, пока останется не более одного вхождения a. Затем начнется процесс вычеркивания пар соседних b (применяется третья подстановка) до тех пор, пока останется не более одного вхождения b. Таким образом, алгоритм приведения применим к любому слову P и перерабатывает его в одно из следующих эталонных слов Λ, a, b, ab. Так как каждое слово эквивалентно своему эталонному слову, то два слова эквивалентны в том и только том случае, когда им соответствует одно и то же эталонное слово или два эквивалентных эталонных слова. Но эталонные слова попарно не эквивалентны, поэтому два слова эквивалентны тогда и только тогда, когда им соответствует одно и то же эталонное слово. Отсюда разрешающий алгоритм для проблемы слов в рассматриваемом ассоциативном исчислении будет выглядеть так:

1. Найти эталонное слово для слова P (с помощью алгоритма приведения) — слово P'. Перейти к ук. 2.
2. Найти эталонное слово для слова Q — слово Q'. Перейти к ук. 3.
3. Сравнить эталонные слова P' и Q'. Если они совпадают, то перейти к ук. 4, иначе — к ук. 5.
4. $P \sim Q$. Процесс окончен.
5. $P \not\sim Q$. Процесс окончен.

Задача 8.4. Используя построенный алгоритм, определите, эквивалентны ли слова:

а) P: $aaaba$ и Q: $abab$;
б) P: $bbaaaab$ и Q: baa.

Рассмотрим другой пример ассоциативного исчисления и проблему слов в нем: «Можно ли указать алгоритм, позволяющий по любой паре слов ассоциативного исчисления с алфавитом $\{a, b\}$ и системой подстановок

$$\begin{cases} aaa \leftrightarrow \Lambda \\ bb \leftrightarrow \Lambda \\ ba \leftrightarrow aab \end{cases}$$

определить, эквивалентны они или нет?»

Рассмотрим множество F, состоящее из элементарных самосовмещений равностороннего треугольника ABC: a — поворот на 120° против часовой стрелки вокруг центра O $\triangle ABC$, b — симметрия с осью, проходящей через центр O треугольника и одну из его вершин, и всех самосовмещений равностороннего треугольника, представимых в виде композиции конечного числа элементарных совмещений (т. е. в виде слов в алфавите $\{a, b\}$). Подстановкам исчисления соответствуют равенства: $aaa = \Lambda$, $bb = \Lambda$, $ba = ab$ (слова aaa, bb задают тождественное отображение множества вершин треугольника на себя).

Покажем, что две композиции элементарных самосовмещений треугольника задают одно и то же самосовмещение тогда и только тогда, когда изображающие их слова эквивалентны в соответствующем исчислении.

Действительно, из равенства $aaa = \Lambda$, $bb = \Lambda$, $ba = aab$ следует, что любые два смежных слова равны, так как при каждом применении любой допустимой подстановки к любому слову P оно переходит в равное ему слово. Тогда эквивалентность двух слов P и Q в исчислении влечет равенство их, так как в последовательности слов $P, P_1, P_2, \ldots, P_n, Q$, ведущей от P и Q, любые два смежных слова равны, следовательно, $P = Q$. Справедливо и обратное утверждение: если $P = Q$, то $P \sim Q$. Покажем это. Каждое слово рассматриваемого исчисления эквивалентно одному из слов: Λ, a, b, aa, ab, aab. Действительно, применяя к произвольному слову P подстановку $ba \leftrightarrow aab$, мы придем к эквивалентному слову, в котором все буквы a стоят левее всех букв b, т. е. к слову вида $\underbrace{aa\ldots a}_{l}\ \underbrace{bb\ldots b}_{m}$, где l и m — неотрицательные целые числа. Далее, применяя подстановки $aaa \leftrightarrow \Lambda$, $bb \leftrightarrow \Lambda$ (т. е. удаляя идущие подряд буквы aaa, bb), получим, что каждое слово вида $\underbrace{aa\ldots a}_{l}\ \underbrace{bb\ldots b}_{m}$ эквивалентно одному из слов: Λ, a, b, aa, ab, aab, т. е. произвольное слово эквивалентно одному из этих (эталонных) слов. Из того, что $P = Q$ и что слова Λ, a, b, aa, ab, aab задают попарно различные самосовмещения треугольника (что устанавливается непосредственной проверкой), следует, что слова P и Q эквивалентны одному и тому же из указанного множества слов.

Следовательно, слова P и Q, эквивалентные одному и тому же слову, эквивалентны.

Искомый алгоритм распознавания эквивалентности любой пары слов P и Q будет выглядеть так:

1. Осуществить последовательно самосовмещения $\triangle ABC$, соответствующие словам P и Q. Перейти к ук. 2.

2. Сравнить расположения вершин треугольника. Если они одинаковы, то перейти к ук. 3, иначе — к ук. 4.

3. $P \sim Q$. Процесс окончен.

4. $P \not\sim Q$. Процесс окончен.

Предложим теперь Вам еще несколько задач.

Задачи

8.5. Постройте алгоритм, отличный от приведенного выше, распознающий эквивалентность слов в ассоциативном исчислении с алфавитом $\{a, b\}$ и системой подстановок:

$$\begin{cases} ba \leftrightarrow aab \\ aaa \leftrightarrow \Lambda \\ bb \leftrightarrow \Lambda. \end{cases}$$

8.6. Рассмотрим равносторонний треугольник ABC. Обозначим через R поворот плоскости на 120° против часовой стрелки вокруг центра O треугольника, через S — осевую симметрию плоскости с осью OB. Представьте в виде композиции преобразований R и S: а) поворот плоскости на 120° по часовой стрелке вокруг центра O треугольника; б) осевую симметрию с осью OA; в) осевую симметрию с осью OC.

8.7. Каким самосовмещением равностороннего треугольника ABC является композиция $X = RRSSRSRSSR$ преобразований R и S (см. задачу 8.5)?

8.8. Рассмотрим ассоциативное исчисление с алфавитом $\{a, b, c, d\}$ и системой подстановок: $ac \leftrightarrow ca$, $ad \leftrightarrow da$, $bc \leftrightarrow cb$.

а) Сколько смежных слов имеют слова: $abcd$, $aaabb$? Укажите их.

б) Эквивалентны ли слова $abcd$ и $cabd$?

8.9. Дано ассоциативное исчисление с алфавитом $\{a, b, c\}$ и системой подстановок: $ab \leftrightarrow ba$, $ac \leftrightarrow ca$, $bc \leftrightarrow cb$. Эквивалентны ли в этом исчислении пары слов: $abca$ и $cbaa$; $ccbaab$ и $bbacac$; $abcaa$ и $acba$; aca и bca?

8.10. Даны два алгоритма: A_1 — перерабатывающий каждое слово в алфавите $\{a, b\}$ в пустое слово и A_2 — преобразующий каждое слово Q в алфавите $\{a; b\}$ в слово aba. Какое слово является результатом преобразования слова P в алфавите $\{a, b\}$: а) алгоритмом $A_1 \circ A_2$; б) алгоритмом $A_2 \circ A_1$? Является ли операция композиции алгоритмов коммутативной?

8.11. Постройте алгоритм, позволяющий по любой паре слов ассоциативного исчисления с алфавитом $\{a\}$ и подстановкой $aa \leftrightarrow \Lambda$ определить, эквивалентны они или нет.

8.12. В языке некоторого племени всего два звука: «a» и «b». Два слова означают одно и то же, если одно получается из другого при помощи следующих операций: пропуска идущих подряд звуков $aaaa$, bb или $abab$ и добавления в любое место сочетаний этих звуков. Укажите алгоритм, позволяющий по любым двум словам определять, означают ли они одно и то же.

8.13. Постройте алгоритм, позволяющий по любой паре слов ассоциативного исчисления с алфавитом $\{a, b, c\}$ и системой подстановок: $aa \leftrightarrow \Lambda$, $bb \leftrightarrow \Lambda$, $cc \leftrightarrow \Lambda$, $ab \leftrightarrow c$, $ac \leftrightarrow b$, $ba \leftrightarrow c$, $cb \leftrightarrow a$, $ca \leftrightarrow b$ определять, эквивалентны они или нет.

8.14. Постройте алгоритм, позволяющий по любой паре слов ассоциативного исчисления с алфавитом $\{a, b\}$ и системой подстановок: $aaaa \leftrightarrow \Lambda$, $bb \leftrightarrow \Lambda$, $abababab \leftrightarrow \Lambda$ определять, эквивалентны они или нет. (Рассмотрите равнобедренный прямоугольный треугольник и элементарные движения плоскости: a — поворот на 90° против часовой стрелки вокруг вершины прямого угла, b — поворот на 180° вокруг середины гипотенузы.)

9. Нормальный алгоритм Маркова

Две предыдущие беседы (7 и 8) приблизили нас вплотную к одному из математических уточнений понятия алгоритма.

Среди рассмотренных нами примеров алгоритмов над словами нам встретились такие, которые задаются посредством упорядоченной системы ориентированных подстановок. Эти алгоритмы получили в математике название нормальных алгоритмов Маркова по имени советского математика и логика А. А. Маркова, создавшего теорию нормальных алгоритмов, которая принесла ему мировую известность. Предложенное им точное математическое понятие нормального алгоритма явилось крупным вкладом в мировую математическую науку.

Перейдем теперь к точному определению нормального алгоритма.

Пусть A — некоторый алфавит, не содержащий в качестве букв знаки «$\rightarrow$» и «$\cdot$». Обычной формулой подстановки в алфавите A называют слово $P \rightarrow Q$ (понятие «слово» мы тоже ранее определяли), где P и Q — слова в алфавите A. Заключительной формулой подстановки в алфавите A называют слово $P \rightarrow \cdot Q$, где P и Q — слова в алфавите A. P называется левой частью формулы подстановки, а Q — ее правой частью.

Для обыкновенной и заключительной подстановки можно ввести обобщенную запись «$P \rightarrow \gamma Q$, где γ есть $\cdot$ (точка) или пустое слово, т. е. $\gamma \in \{\cdot, \Lambda\}$.

Конечная непустая упорядоченная система подстановок

$$\begin{cases} P_1 \rightarrow \gamma_1 Q_1 \\ P_2 \rightarrow \gamma_2 Q_2 \\ \ldots\ldots\ldots\ldots \\ P_k \rightarrow \gamma_k Q_k \end{cases}$$

называется **нормальной схемой** в алфавите A, где $k \geqslant 1$, $\gamma_i \in \{\cdot, \Lambda\}$, P_i и Q_i — слова в алфавите A $(i = 1, 2, \ldots, k)$.

Нормальный алгоритм в алфавите A задается алфавитом A и нормальной схемой в этом алфавите. Он преобразует слова в алфавите A следующим образом.

Пусть дано слово P в алфавите A. Выясним, имеется ли среди формул подстановок нормальной схемы хотя бы одна с левой частью, входящей в слово P Если таких нет, то процесс приме-

нения алгоритма к слову P заканчивается и результатом применения алгоритма к слову P считается само слово P. Если же в нормальной схеме имеются формулы подстановок, левые части которых входят в слово P, берем самую верхнюю формулу из числа обладающих этим свойством (т. е. формулу подстановки с наименьшим номером) и применяем ее, заменив первое (самое левое) вхождение левой части формулы подстановки в слове P на правую часть этой формулы подстановки. Пусть P' — результат применения такой подстановки. Если использованная формула подстановки была заключительной, то процесс применения алгоритма к слову P заканчивается с результатом P', иначе применяем к слову P' тот же процесс поиска, который был только что применен к P и т. д. Но до каких пор «и т. д.»?

Если мы в конце концов получим такое слово Q, к которому уже не применима ни одна из формул подстановок нормальной схемы, то процесс применения алгоритма к слову P заканчивается и Q является результатом работы алгоритма над словом P. При этом возможно, что описанный процесс никогда не закончится,— в таком случае говорят, что алгоритм неприменим к слову P.

Например, нормальный алгоритм с алфавитом $A=\{|\}$ и нормальной схемой $\{\Lambda\to|$ неприменим ни к одному слову этого алфавита, так как процесс его применения к любому слову не заканчивается. Нормальный же алгоритм с тем же алфавитом $A=\{|\}$ и нормальной схемой $\{\Lambda\to\cdot|$ применим к любому слову $\underbrace{||...|}_{n}$ в алфавите $A=\{|\}$ с результатом $\underbrace{||...|}_{n+1}$.

Нормальный алгоритм, включая процедуру его применения, может быть представлен в виде блок-схемы, изображенной на рисунке 12.

(На этой блок-схеме запись «$P_i\to\gamma_iQ_i$» означает замену первого вхождения левой части P_i в рассматриваемом слове правой частью Q_i.)

У Вас, дорогой читатель, может возникнуть вопрос: а можно ли любой алгоритм представить в виде нормального? Вопрос вполне естественный, и ответ на него дается в виде предположения, известного под названием **тезиса Маркова: всякий алгоритм представим в виде нормального алгоритма.**

Но почему это только предположение (тезис)? Нельзя ли его доказать (или опровергнуть)? Дело в том, что формулировка тезиса не является математическим предложением, так как наряду с термином «нормальный алгоритм», обозначающим точное математическое понятие, в ней содержится и термин «алгоритм», обозначающий интуитивное понятие. Поэтому не может быть и речи о математическом доказательстве. Основания же для принятия этого тезиса имеются и пока нет противоречащего примера (т. е. алгоритма, который не удается преобразовать в нормальный) этот тезис сохраняет свою силу.

Предложим Вам теперь несколько несложных задач.

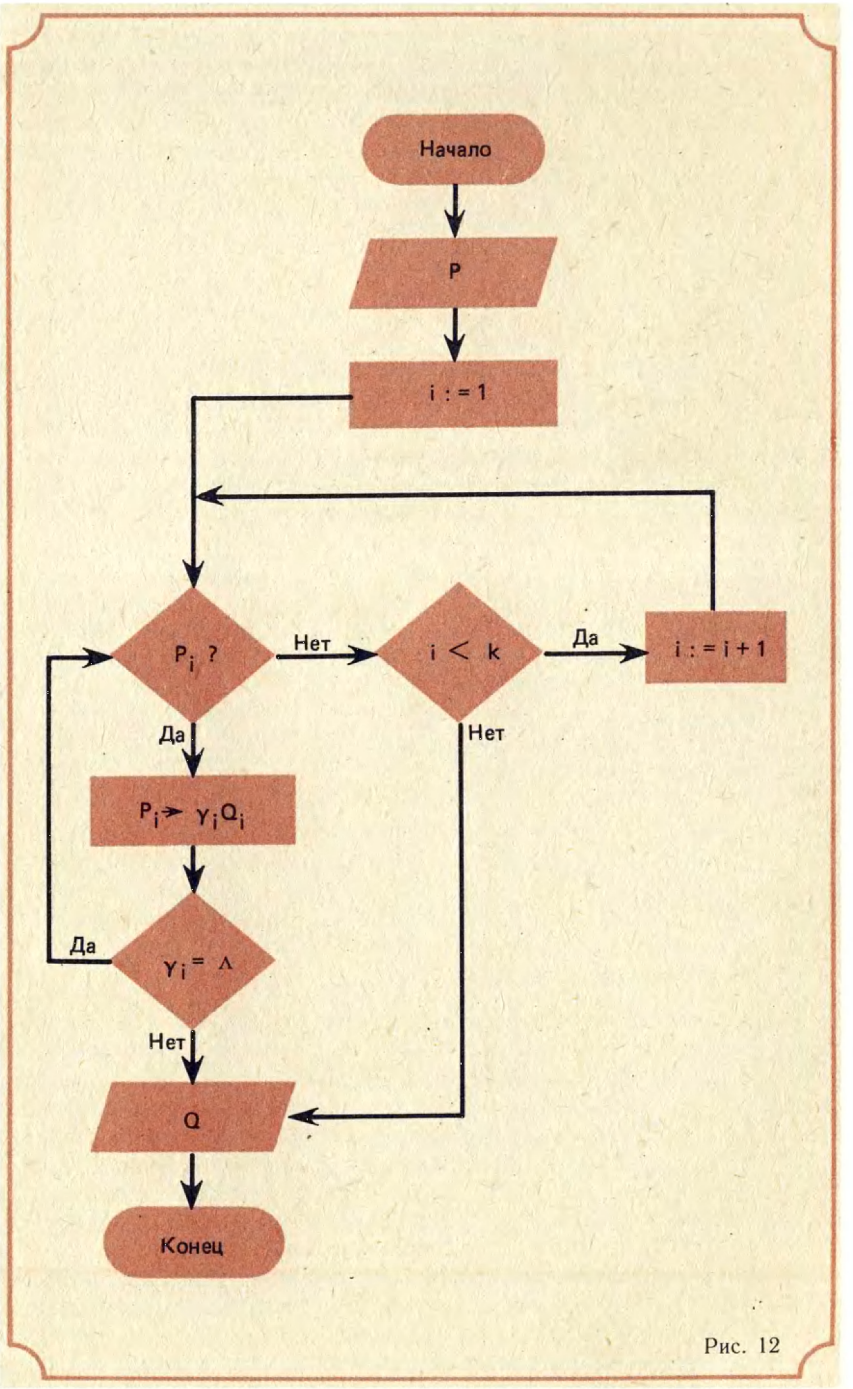

Начало
P
i : = 1
P_i ?
Нет
i < k
Да
i : = i + 1
Да
Нет
$P_i \to \gamma_i Q_i$
Да
$\gamma_i = \Lambda$
Нет
Q
Конец

Рис. 12

Задачи

9.1. В каких из задач 8.4.— 8.13 искомые алгоритмы представлены в виде нормальных алгоритмов? Укажите их алфавиты и нормальные схемы.

9.2. Нормальный алгоритм задан алфавитом $A=\{a, b\}$ и нормальной схемой

$$\begin{cases} ba\to ab \\ ab\to\Lambda. \end{cases}$$

Примените этот алгоритм к слову: 1) *bbaabab*; 2) *aabbaab*; 3) *abaabb*; 4) *aaa*; 5) *bbbb*.

9.3. Какое свойство слов в алфавите $A=\{a, b\}$ распознается с помощью алгоритма, заданного в задаче 9.2?

9.4. Укажите всевозможные результаты преобразования слов в алфавите $A=\{a, b\}$ алгоритмом из задачи 9.2.

9.5. Нормальный алгоритм задан алфавитом $A=\{a, b, |\}$ и нормальной схемой

$$\begin{cases} a\to| \\ b\to|. \end{cases}$$

Примените этот алгоритм к словам: 1) *abaabbb*; 2) *bababbaa*; 3) *aaa*; 4) *bbbb*; 5) *aabbb*||.

Что определяется для каждого исходного слова с помощью этого алгоритма?

9.6. Нормальный алгоритм задан алфавитом $A=\{a, b, |\}$ и нормальной схемой

$$\begin{cases} a\to| \\ b\to| \\ ||\to\Lambda. \end{cases}$$

Примените этот алгоритм к словам 1) — 5) из задачи 9.5.

10. Воображаемая машина

Характеризуя понятие алгоритма, мы уже отмечали, что предписание, задающее алгоритм, должно быть составлено таким образом, чтобы его исполнение было во всех деталях однозначно осуществимо и не требовало никаких свободно принимаемых решений. Другими словами, исполнитель алгоритма должен действовать согласно предписанию совершенно механически, «машинально», не вкладывая в свою работу никакой инициативы, изобретательности. Но что или кто может в наибольшей степени обладать таким свойством? Конечно же, машина.

Английский математик А. М. Тьюринг в 1937 году впервые предложил вариант уточнения понятия алгоритма в виде воображаемой вычислительной машины, известной теперь под названием **машина Тьюринга.**

Это воображаемая «машина» (в кавычках потому, что она вовсе не похожа на реальную машину с множеством деталей из металла, пластмассы или других материалов) — машина «на бумаге» или математическая модель машины.

Сначала дадим описание машины Тьюринга. Оно будет еще очень далеким от математического определения, но поможет нам вообразить себе эту «машину». Потом покажем, каким образом ее можно определить как математическое понятие.

Отметим прежде всего, что машина Тьюринга отличается от человека-вычислителя, выполняющего данные ему предписания, или от реально существующих вычислительных машин в двух отношениях.

Во-первых, машина Тьюринга не может ошибаться, т. е. она строго, без всяких отклонений выполняет программу, определяющую ее работу.

Во-вторых, машина Тьюринга снабжена потенциально бесконечной памятью. Вы спросите: а что это значит «потенциально бесконечная память»? Это значит, что хотя в каждый момент ее память хранит лишь конечное количество информации, однако для этого количества информации нет никакой верхней границы. Вообразим такую память машины Тьюринга в виде ленты, разделенной на клеточки — ячейки памяти,—которая может быть продолжена вправо сколько угодно (рис. 13).

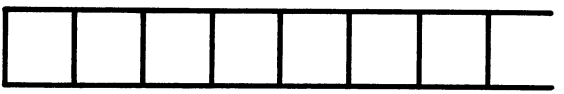


Рис. 13

В каждой ячейке может храниться только один знак из конечного множества знаков, называемого **алфавитом машины Тьюринга,** эти же знаки называются **буквами** алфавита. Ячейка может оказаться и пустой. Ради удобства договоримся: когда ячейка пустая, считать, что в ней хранится специальная буква алфавита, которую мы обозначим через s_0 и назовем **пустой буквой.** Таким образом, в каждой ячейке ленты в любой данный момент находится одна и только одна буква алфавита $A = \{s_0, s_1, s_2, \ldots, s_k\}$, причем только конечное число ячеек заполнено буквами из A, отличными от пустой буквы s_0.

Машина Тьюринга снабжена (в нашем воображении) **управляющей головкой,** которая в каждый момент обозревает (воспринимает) лишь одну ячейку памяти и может изменить информацию, находящуюся в ней. Представим себе это следующим образом: если в какой-то момент букву в обозреваемой ячейке нужно заменить другой, то старая буква стирается и в ячейку вписывается новая (так же, как на магнитофонной ленте при новой записи автоматически стирается старая запись). Если эту операцию записать в общем виде «$s_i \to s_j$», т. е. буква s_i заменена буквой s_j, то можно выделить следующие частные случаи:

а) $i = j$, т. е. буква в обозреваемой ячейке не изменилась;

б) $i \neq j$ — буква изменилась, при этом:

$б_1$) если $i = 0$, то в пустую ячейку вписана буква s_j и $б_2$) если $j = 0$, то стерта буква s_i в обозреваемой ячейке.

Управляющая головка за один такт работы машины может также передвинуться влево и воспринимать соседнюю слева ячейку (этот сдвиг головки будем обозначать через Л), управляющая головка может передвинуться вправо и воспринимать соседнюю справа ячейку (этот сдвиг мы обозначим через П), управляющая головка может остаться на месте и воспринимать ту же ячейку (этот «сдвиг» обозначим через Н). На рисунке 14 изображен фрагмент ленты, а также управляющая головка. В обозреваемой ячейке хранится буква s_i.

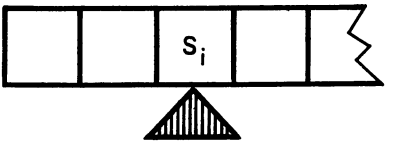


Рис. 14

Машина Тьюринга имеет конечное множество внутренних состояний: $Q = \{q_0, q_1, q_2, \ldots, q_m\}$. В каждый данный момент времени машина Тьюринга находится в одном из своих внутренних состояний. После выполнения очередного такта работы машина может изменить свое внутреннее состояние и воспринимать ячейку в следующий момент уже в новом состоянии. Разумеется, внутреннее состояние может остаться и прежним. Если машина в какой-то момент времени попадет в состояние q_0, то ее работа заканчивается (состояние q_0 называется **пассивным**). Из множества Q выделим еще одно состояние — q_1 — **начальное состояние.** В этом состоянии машина всегда начинает свою работу.

Итак, что же умеет наша воображаемая машина?

За один такт работы она может: 1) изменить содержимое обозреваемой ячейки памяти, т. е. заменить содержащуюся в ней букву алфавита другой; 2) совершить сдвиг влево или вправо на одну ячейку или остаться на месте и 3) изменить свое внутреннее состояние.

И больше машина Тьюринга ничего не умеет делать.

Вам может показаться, что это очень мало. В действительности, как Вы увидите дальше, это очень много.

Порядок работы машины Тьюринга с алфавитом $A = \{s_0, s_1, s_2, \ldots, s_k\}$ и множеством состояний $Q = \{q_0, q_1, q_2, \ldots, q_m\}$ полностью определяется **программой,** представляющей собой таблицу, содержащую $k + 1$ столбец и m строк (таблица 1). В клетке таблицы на пересечении i-го столбика ($i = 0, 1, 2, \ldots, k$) и j-й строки ($j = 1, 2, \ldots, m$) вписана **команда,** которую должна выполнить машина Тьюринга. Если машина в какой-то момент воспринимает управляющей головкой ячейку, в которой записана буква s_i, и

Таблица 1

	s_0	s_1	...	s_i	...	s_k
q_1						
q_2						
⋮						
q_j				s_hTq_l		
⋮						
q_m						

машина находится в состоянии q_j, команда будет записываться в виде трехбуквенного слова s_hTq_l, где $T \in \{Л, Н, П\}$, т. е. обозначает один из сдвигов управляющей головки.

Для выполнения команды машина проделывает следующую работу:

1) буква s_i, находящаяся в обозреваемой ячейке, меняется на s_h;

2) управляющая головка совершает сдвиг T, т. е. Л, Н или П;

3) машина переходит в состояние q_l.

Работа машины состоит из тактов, выполняемых в строгом соответствии с программой. Если в какой-то момент времени машина приходит в состояние q_0, то работа машины заканчивается. Программа полностью определяет работу машины, поэтому можно сказать, что машина Тьюринга задана, если задана ее программа.

Пример. Рассмотрим машину Тьюринга, программа которой задана таблицей 2.

Таблица 2

	s_0	0	1	2	3	4	5	6	7	8	9
q_1	$1Hq_0$	$1Hq_0$	$2Hq_0$	$3Hq_0$	$4Hq_0$	$5Hq_0$	$6Hq_0$	$7Hq_0$	$8Hq_0$	$9Hq_0$	$0Лq_1$

Как видно из программы, алфавит этой машины Тьюринга состоит из s_0 и десяти цифр: $A = \{s_0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, а множество Q — всего из двух состояний: $Q = \{q_0, q_1\}$.

Какую же задачу решает эта машина?

Допустим, что в памяти машины перед началом работы хранится число 385, т. е. лента имеет вид, показанный на таблице 3, *а*:

Таблица 3

а)

s_0	s_0	3	8	5	s_0	s_0	

б)

s_0	s_0	3	8	6	s_0	s_0	

Из таблицы 3, *а* видно, что машина в состоянии q_1 обозревает самый правый символ записи на ленте. Из программы (табл. 2) следует, что соответствующая паре (5, q_1) команда имеет вид: $6Hq_0$. Согласно этой команде машина стирает цифру 5 в обозреваемой ячейке и помещает в ней цифру 6, остается на месте и переходит в состояние q_0, т. е. останавливается, причем состояние памяти будет таким, как показано на таблице 3, *б*. Как видите, наша машина прибавила 1 к исходному числу.

Рассмотрим теперь другую начальную ситуацию, показанную на таблице 4, *а*, т. е. в памяти машины хранится число 4899.

Таблица 4

а)

	s_0	s_0	4	8	9	9	s_0	s_0
					q_1			

б)

	s_0	s_0	4	8	9	0	s_0	s_0
					q_1			

в)

	s_0	s_0	4	8	0	0	s_0	s_0
					q_1			

г)

	s_0	s_0	4	9	0	0	s_0	s_0
					q_0			

По соответствующей паре (9, q_1) команде $0Лq_1$ состояние ленты станет таким, как на таблице 4,*б*. Вновь в состоянии q_1 машина обозревает ячейку, в которой находится цифра 9, следовательно, опять выполняется команда $0Лq_1$, согласно которой в обозреваемой ячейке стирается цифра 9 и помещается цифра 0, машина сдвигается влево и остается в состоянии q_1 (табл. 4, *в*). Теперь уже программа требует выполнения соответствующей паре $(8, q_1)$ команды $9Hq_0$, согласно которой машина вписывает в обозреваемую ячейку цифру 9 (вместо 8), остается на месте и останавливается.

Таким образом, по окончании работы машины лента имеет вид, приведенный на таблице 4, *г*.

Как видите, и в этом случае, т. е. когда исходное число — 4899, машина прибавляет единицу и останавливается. Нетрудно заметить, что вообще машина, определяемая программой, заданной таблицей 2, воспринимая в начальном положении запись любого натурального числа в десятичной системе счисления, по окончании своей работы будет выдавать в качестве результата запись этого числа, увеличенного на 1.

В дальнейшем, как и в рассмотренном выше примере, будем считать, что машина всегда в начальном состоянии (т. е. в состоянии q_1) воспринимает самый правый символ записи на ленте

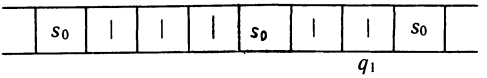
(разумеется, отличный от s_0), и называть это положение, в котором машина воспринимает запись на ленте, **стандартным.**

Рассмотрим задачу.

Задача 10.1. Какую работу выполнит машина Тьюринга, если она работает по программе:

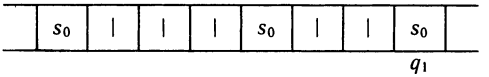
	s_0	|
q_1	$\|Hq_0$	$\|\Pi q_1$

Пусть перед началом работы машины ее память (лента) имеет вид

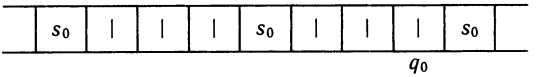


Согласно программе, обозревая ячейку, в которой хранится знак |, и находясь в состоянии q_1, машина производит следующее действие: не производя никаких изменений в воспринимаемой ячейке, управляющая головка движется вправо и машина остается в состоянии q_1.

Состояние ленты после первого такта работы машины будет таким:

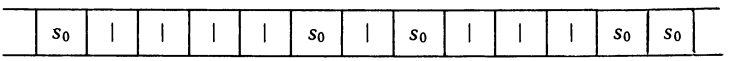


Дальнейшее действие приводит к остановке машины с состоянием ленты:



Таким образом, рассматриваемая машина отыскивает первую пустую ячейку справа от воспринимаемой, печатает на ней букву алфавита и останавливается, воспринимая эту ячейку.

Условимся представлять числа 0, 1, 2, 3, ... словами в алфавите {|}: |, ||, |||, ||||, ... соответственно. Для представления на ленте набора целых неотрицательных чисел x_1, x_2, ... x_n мы будем писать соответствующее число раз букву |, оставляя в точности одну пустую ячейку между каждыми двумя словами (конечными последовательностями букв |). Так, набор чисел 3, 0, 2 запишется на ленте следующим образом:



Учитывая это, работу рассмотренной машины (назовем ее машиной A) можно охарактеризовать следующим образом: машина A, воспринимая последнее число набора чисел (в стандартном

положении), увеличивает его на единицу и останавливается, воспринимая полученное число.

В дальнейшем мы будем иметь дело только с целыми неотрицательными числами (расширенным нулем множеством натуральных чисел) или наборами этих чисел и их записями в однобуквенном алфавите $\{|\}$.

Мы думаем, что все, о чем шла речь в этой беседе, настолько просто, что вряд ли у Вас возникли какие-нибудь вопросы.

Продолжим рассмотрение примеров машин Тьюринга. Для начала опишем несколько простых (элементарных) машин Тьюринга. Элементарные машины — это машины с алфавитом $\{s_0, |\}$, получаемым присоединением к однобуквенному алфавиту $\{|\}$ пустой буквы s_0. Результатом их применения к записи на ленте являются некоторые элементарные изменения (преобразования) этой записи.

Пример 1. Машина A, имеющая программу, данную в таблице 5, выше уже рассмотрена нами.

Таблица 5

A	s_0	|
q_1	$\|Hq_0$	$\|Пq_1$

Пример 2. Машина B воспринимает любое число из набора $x_1, x_2, \ldots, x_n$, уменьшает число палочек в его записи на одну и останавливается, воспринимая уменьшенное число. Так работает машина с программой, данной в таблице 6.

Таблица 6

B	s_0	|
q_1		$s_0Лq_0$

Предложим Вам простую задачу.

Задача 10.2. Изобразите на ленте в алфавите $\{s_0, |\}$ набор чисел 2, 3, 4 и пусть машина B воспринимает второе число в стандартном положении. Изобразите ленту после работы машины. Какой набор чисел будет записан на ней?

Пример 3. Машина C воспринимает набор чисел $x_1, x_2, \ldots, x_n$ в стандартном положении и через одну пустую ячейку справа от этого набора записывает число 0, после чего останавливается, воспринимая 0.

Программа машины C представлена таблицей 7.

Таблица 7

C	s_0	|
q_1	$s_0Пq_2$	$\|Пq_1$
q_2	$\|Hq_0$	

Программа машины C уже несколько сложнее программ машин A и B. Поэтому здесь уже может возникнуть вопрос: а как догадаться, что именно такой должна быть эта программа?

Можно рассуждать так: воспринимая набор чисел $x_1, x_2, \ldots, x_n$ в стандартном положении, машина воспринимает последнюю заполненную ячейку (записи числа x_n). Поэтому для выполнения описанной работы она должна двигаться вправо. Это действие приписываем начальному состоянию q_1. Но после того, как она совершает два таких сдвига, она должна записать в обозреваемой ячейке число 0 (т. е. однобуквенное слово |) и остановиться. Для совершения этого действия нужно еще одно состояние q_2.

Теперь, как определить содержание команд программы? Первая пара (содержимое воспринимаемой ячейки и состояние машины), которая возникает в начале работы — (|, q_1) — требует сдвига вправо: (|, q_1)→|Пq_1; вторая пара: (s_0, q_1)→s_0Пq_2; а третья — (s_0, q_2)→|Hq_0. Если заполнить этими командами таблицу, получим приведенную выше программу (табл. 7). Пустая клетка таблицы означает, что пара (|, q_2) не возникает в процессе работы этой машины (поэтому можно записать в этой клетке произвольную команду).

Задача 10.3. Изобразите на ленте в алфавите {s_0, |} набор чисел 1, 2, 3 и имитируйте работу машины C (изобразите ленту после каждого такта машины до ее остановки).

Пример 4. Машина D, отправляясь от воспринятого в стандартном положении числа, не самого правого на ленте, заполняет промежуток из пустых клеток (если имеется такой) между этим числом и ближайшим справа, оставляя между ними пустую клетку, и останавливается, воспринимая в стандартном положении получающееся число.

Так, машина D, примененная к ленте

	s_0	|	|	s_0	s_0	s_0	|	s_0	|	|	s_0	

в качестве результата выдает следующую запись на ленте:

	s_0	|	|	|	|	s_0	|	s_0	|	|	s_0	
					q_0							

Так работает машина с программой, данной в таблице 8.

Таблица 8

D	s_0	|
q_1		|Пq_2
q_2	|Пq_2	|Лq_3
q_3		s_0Лq_0

Дадим Вам пояснение к машине D (табл. 8): так как машина воспринимает число в стандартном положении, то в начальном состоянии q_1 машина совершает сдвиг вправо. Затем, в состоянии q_2 машина вписывает в пустую ячейку палочку и сдвигается вправо. Машина остается в этом состоянии, пока она воспринимает пустые ячейки. Как только машина заполнила все пустые ячейки между двумя числами, она должна совершать сдвиг влево и перейти в новое состояние q_3, в котором она стирает палочку в последней заполненной ячейке и останавливается.

Задача 10.4. Изобразите на ленте числа 3 и 2 в алфавите $\{s_0, |\}$, отделенные друг от друга четырьмя пустыми ячейками. Имитируйте работу машины D применительно к этой записи на ленте.

Пример 5. Машина r, примененная к произвольной записи на ленте, сдвигает воспринимаемую ячейку на одну ячейку вправо и затем останавливается, не изменяя записи на ленте.

Так работает машина с программой, данной в таблице 9.

Таблица 9

r	s_0	|
q_1	$s_0 П q_0$	$\| П q_0$

Пример 6. Машина l, примененная к произвольной записи на ленте, сдвигает воспринимаемую ячейку на одну ячейку влево и затем останавливается, не изменяя записи на ленте.

Задача 10.5. Составьте программу машины l.

Пример 7. Машина R, отправляясь от воспринятого в стандартном положении числа, не самого правого на ленте, идет вправо к стандартному положению ближайшего справа числа.

Программа машины R помещена в таблице 10.

Таблица 10

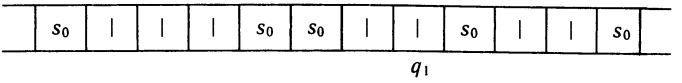
R	s_0	|
q_1	$s_0 П q_2$	$\| П q_1$
q_2	$s_0 П q_2$	$\| П q_3$
q_3	$s_0 Л q_0$	$\| П q_3$

Задача 10.6. Примените машину R к ленте

	s_0	|	|	|	s_0	s_0	|	|	s_0	|	|	s_0	.
				q_1									

Пример 8. Машина L, отправляясь от воспринятого в стандартном положении числа, не самого левого на ленте, идет влево к стандартному положению ближайшего слева числа.

Задача 10.7. Составьте программу машины L и проверьте ее работу на ленте



Пример 9. Рассмотрим машину Тьюринга, производящую следующую операцию: если на ленте дан набор чисел x_1, x_2, ..., x_n, воспринимаемый машиной в стандартном положении, то машина в заключительном состоянии имеет на ленте набор чисел x_1, x_2, ..., x_n, 3, воспринимаемый ею также в стандартном положении.

Так работает машина с программой, данной в таблице 11.

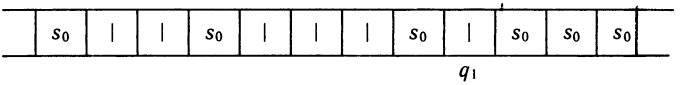
Таблица 11

	s_0	|
q_1	$s_0 П q_2$	$\| П q_1$
q_2	$\| Н q_3$	
q_3	$\| Н q_4$	$\| П q_3$
q_4	$\| Н q_5$	$\| П q_4$
q_5	$\| Н q_0$	$\| П q_5$

Поможем Вам разобраться с работой этой машины.

Порядок работы машины следующий: через одну пустую ячейку справа от набора x_1, x_2, ..., x_n машина пишет | (т. е. число 0) и переходит в состояние q_3, в котором находит следующую справа от записанной | пустую ячейку, записывает в нее | и переходит в состояние q_4, в котором также в соседней справа от пустой ячейке записывает | и в состоянии q_5 проделывает аналогичную работу: в пустой ячейке справа от | записывает | и останавливается.

Задача 10.8. Проимитируйте работу машины для записи на ленте



Пример 10. Рассмотрим машину Тьюринга, которая по записи любого числа на ленте распознает, оно больше нуля или равно нулю. В первом случае машина в качестве результата выдает число 1, записанное через одну пустую клетку справа от воспринимаемого числа, во втором — число 0, записанное также через одну пустую клетку справа от воспринимаемого числа.

Составим сначала схему работы этой машины (рис. 15).

В соответствии с этой схемой получим программу машины (табл. 12).

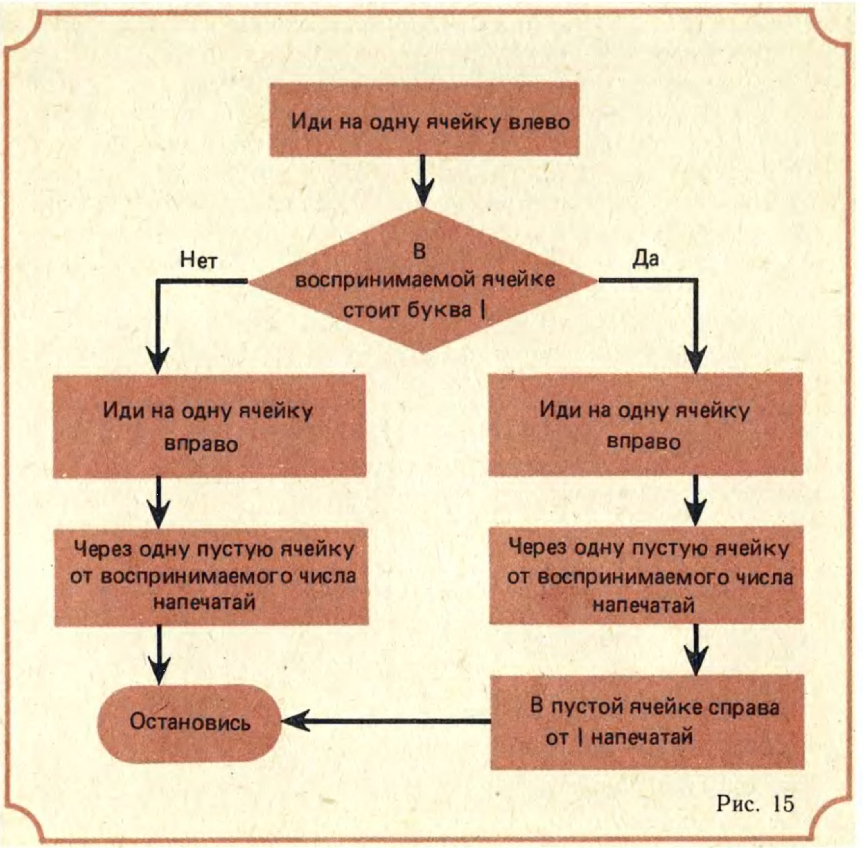


Рис. 15

Таблица 12

	s_0	|
q_1	$s_0Лq_2$	$\|Лq_2$
q_2	$s_0Нq_3$	$\|Нq_6$
q_3	$s_0Пq_4$	$\|Пq_4$
q_4	$s_0Пq_5$	$\|Пq_4$
q_5	$\|Нq_0$	
q_6	$s_0Пq_7$	$\|Пq_7$
q_7	$s_0Пq_8$	$\|Пq_7$
q_8	$\|Нq_9$	
q_9	$\|Нq_0$	$\|Пq_9$

Чтобы Вы убедились, что, работая по этой программе, машина действительно выполняет описанную выше работу, предложим Вам задачу.

Задача 10.9. Проимитируйте работу машины для записи на ленте:

а)

s_0	|	|	s_0	s_0

б)

s_0	|	s_0	s_0

Пример 11. Построим машину, которая, отправляясь от числа, воспринятого в стандартном положении, стирает все числа левее данного (если таковые имеются) до первого встречного промежутка из двух или более пустых ячеек и возвращается к стандартному положению первоначально воспринятого числа.

Для построения программы такой машины (по существу, «построить машину Тьюринга» означает построить ее программу) составим по аналогии с примером 10 схему работы этой машины (рис. 16).

Составим программу для этой машины (табл. 13).

Таблица 13

	s_0	|
q_1	$s_0Лq_2$	$\|Лq_1$
q_2	q_3	q_6
q_3	$s_0Пq_4$	
q_4	$s_0Пq_4$	$\|Пq_5$
q_5	$s_0Лq_0$	$\|Пq_5$
q_6	$s_0Нq_1$	$s_0Лq_6$

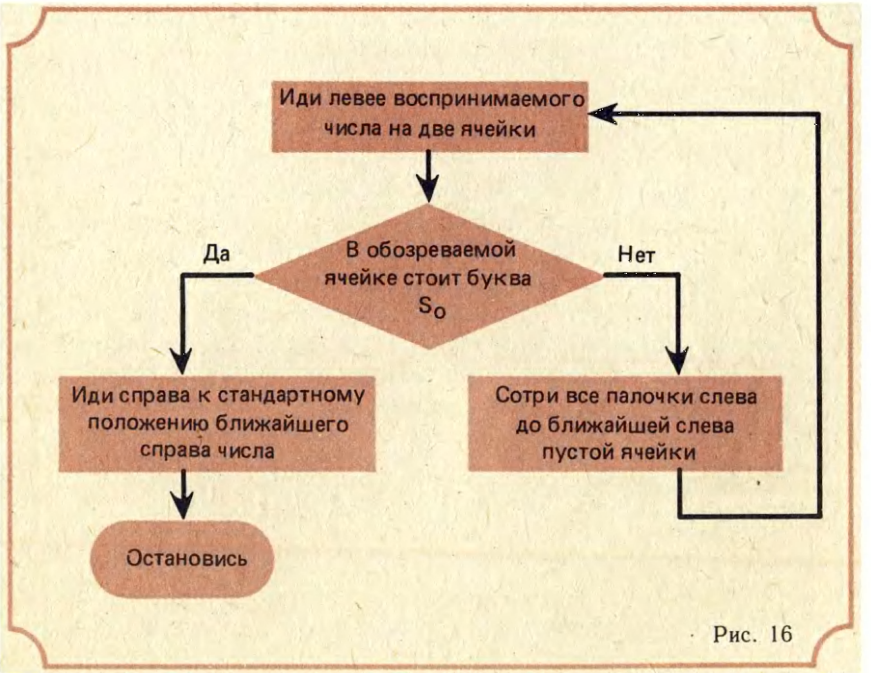
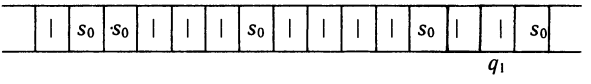


Рис. 16

Задача 10.10. Проимитируйте работу машины с такой программой (табл. 13) для исходной записи на ленте:

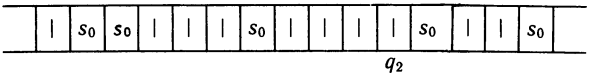


Из примеров 9, 10, 11 видно, что для решения даже простых задач нужны довольно сложные машины Тьюринга. Естественно возникает вопрос: нельзя ли «собрать», сконструировать машины Тьюринга путем соединения нескольких более простых, элементарных машин?

В примере 9 мы могли бы поместить одну за другой машину C и три машины A.

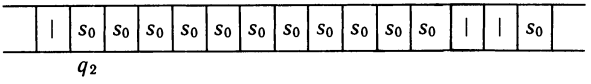
В примерах 10, 11 мы могли бы сначала сконструировать машины для частных задач и затем «собрать» из них нужную машину. Из этих примеров видно также, что наряду с простым соединением машин (когда машина M' должна работать независимо от того, какая буква стояла в последней обозреваемой ячейке перед остановкой M) была бы желательна возможность и их дифференцированного соединения, т. е. такого, что если в последней обозреваемой ячейке перед остановкой машины M стояла буква s_0, то дальше должна работать машина M_1, иначе, т. е. если там стояла буква |, — то машина M_2.

Пример 11 показывает, что если машина работает с записью на ленте, как в задаче 10.9, то, совершив три такта работы, она порождает следующее состояние на ленте, отличающееся от исходного обозреваемой ячейкой и состоянием



(это можете проверить по своему решению задачи).

Далее машина переходит в состояние q_6, в котором стирает все «палочки» до первой слева пустой ячейки и переходит в состояние q_1, затем повторяется та же самая операция, т. е. машина идет на одну клетку влево, обнаруживает палочку, переходит в состояние q_6, в котором стирает все палочки до первой слева пустой ячейки и переходит снова в состояние q_1. Такой же цикл шагов начинается снова, но оканчивается иначе, так как в состоянии q_2 запись на ленте будет такова:



Машина перейдет в состояние q_3 и, проходя по пустым клеткам, воспримет исходное число в стандартном положении и остановится.

Этими рассуждениями мы, по-видимому, немного разъяснили программу (табл. 13) и одновременно подготовились к более или менее естественному восприятию следующих операций над машинами.

Определение. Даны две машины Тьюринга M_1 и M_2, имеющие общий алфавит $A = \{s_0, s_1, s_2, \ldots, s_k\}$ и состояния $q_0, q_1, q_2, \ldots, q_l$ и $q'_0, q'_1, q'_2, \ldots, q'_m$ соответственно. Композицией машин M_1 и M_2 называется машина, обозначаемая $M_1 \circ M_2$, имеющая алфавит A и состояния $q_1, q_2, \ldots, q_l, q'_1, q'_2, \ldots, q'_m, q'_0$ (начальное состояние машины $M_1 \circ M_2$ — начальное состояние машины M_1, т. е. q_1, заключительное состояние — заключительное состояние машины M_2, т. е. q'_0). Программа этой машины строится из программ машин M_1 и M_2 (см. табл. 14).

Таблица 14

	s_0	s_1	s_2	...	s_k
q_1					
q_2					
⋮					
q_l					
q'_1					
q'_2					
⋮					
q'_m					

программа машины M_1, заключительное состояние которой заменено на q'_1 — начальное состояние машины M_2

программа машины M_2

В соответствии с этим определением машину из примера 9 мы могли бы записать так: $((C \circ A) \circ A) \circ A$.

Задача 10.11. а) Покажите, что операция композиции машин ассоциативна, т. е. для любых машин M_1, M_2, M_3

$$(M_1 \circ M_2) \circ M_3 = M_1 \circ (M_2 \circ M_3).$$

б) Является ли операция композиции машин коммутативной?

Окончательно машину из примера 9 мы можем записать: $C \circ A \circ A \circ A$ или $C \circ A^3$, если условиться считать $\underbrace{M \circ M \circ \ldots \circ M}_{k} = M^k$.

Определение. Даны 3 машины M_1, M_2, M_3, имеющие общий алфавит $A = \{s_0, s_1, s_2, \ldots, s_k\}$ и состояния $q_0, q_1, \ldots, q_l; q'_0, q'_1, \ldots, q'_m; q''_0, q''_1, \ldots, q''_n$ соответственно. **Ветвлением** машин M_1, M_2, M_3 называется машина, обозначаемая $M_1 \begin{cases} M_2 \\ M_3 \end{cases}$, имеющая алфавит A и

состояния $q_1, q_2, \ldots, q_l, q_{l+1}, q'_1, \ldots, q'_m q''_1, \ldots, q''_n, q_0$. Программа этой машины строится из программ машин M_1, M_2, M_3 так, как показано в таблице 15.

Таблица 15

	s_0	s_1	s_2	...	s_k	
q_1						программа машины M_1, заключительное состояние которой q_0 заменено на состояние q_{l+1}
q_2						
⋮						
q_l						
q_{l+1}						команда условного перехода
q'_1						программа машины M_2, заключительное состояние которой q'_0 заменено на q_{l+1}
⋮						
q'_m						
q''_1						программа машины M_3, заключительное состояние которой q''_0 заменено на q_0
⋮						
q''_n						

Проиллюстрируем теперь программу машины из примера 10 (табл. 16).

Таблица 16

		s_0	|	
M_1	q_1	$s_0 Л q_2$	$\| Л q_2$	программа машины l, заключительное состояние которой заменено на q_2
	q_2	$s_0 Н q_3$	$\| Н q_6$	команда условного перехода
M_2	q_3	$s_0 П q_4$	$\| П q_4$	программа машины r, заключительное состояние которой заменено на q_4
	q_4	$s_0 П q_5$	$\| П q_4$	программа машины C с начальным состоянием q_4 и заключительным q_0
	q_5	$\| Н q_0$	$\| П q_7$	
M_3	q_6	$s_0 П q_7$	$\| П q_7$	программа машины r
	q_7	$s_0 П q_8$	$\| П q_7$	программа машины C
	q_8	$\| Н q_9$		
	q_9	$\| Н q_0$	$\| П q_9$	программа машины A

Программа машины из примера 10 запишется, как видно, следующим образом: $M_1 \left\{ \begin{matrix} M_2 \\ M_3 \end{matrix} \right.$, где M_1, M_2, M_3 указаны слева от программы.

Дальнейший анализ программы машины из примера 10 показывает, что машина M_1— это машина l, машину M_2 можно представить в виде композиции машин r и C ($r \circ C$), а M_3— в виде $r \circ C \circ A$. Тогда машина из примера 10 окончательно запишется так:

$$l \left\{ \begin{matrix} r \circ C \\ r \circ C \circ A. \end{matrix} \right.$$

Как видите, из простых (элементарных) машин Тьюринга l, r, C, A с помощью операций композиции и ветвления мы сконструировали более сложную машину Тьюринга (пример 10).

Проанализируем и машину из примера 11. Но предварительно введем еще одну операцию, выполняемую над одной машиной. На целесообразность введения такой операции (зацикливание) мы обратили внимание при предварительном анализе примера 11.

Определение. Дана машина M, имеющая алфавит $A = \{s_0, s_1, s_2, \ldots, s_k\}$ и состояния $q_0, q_1, \ldots, q_l$. Программа машины M содержит по крайней мере две команды с заключительным состоянием q_0. Будем говорить, что машина M' получена из машины M с помощью операции **зацикливания,** если в одной из команд машины M, содержащих состояние q_0, это состояние заменено на одно из состояний $q_1, q_2, \ldots, q_l$.

Так, машина из примера 11 получена с помощью операции зацикливания из машины с программой, данной в таблице 17.

Таблица 17

	s_0	|
q_1	$s_0 Л q_2$	$\| Л q_1$
q_2	q_3	q_6
q_3	$s_0 П q_4$	
q_4	$s_0 П q_4$	$\| П q_5$
q_5	$s_0 Л q_0$	$\| П q_5$
q_6	$s_0 Н q_0$	$s_0 Л q_6$

Действительно, эта программа содержит две команды с заключительным состоянием q_0: $s_0 Л q_0$ и $s_0 Н q_0$. Заменив в команде $s_0 Н q_0$ состояние q_0 состоянием q_1, мы и получили машину из примера 11.

Проанализируем теперь программу этой машины, данную в таблице 18.

Таблица 18

	S_0	|	
q_1	$s_0 Л q_2$	$\|Л q_1$	машина P, идущая на две ячейки влево от числа
q_2	q_3	q_6	команда условного перехода
q_3	$s_0 П q_4$		машина R
q_4	$s_0 П q_4$	$\|П q_5$	
q_5	$s_0 Л q_0$	$\|П q_5$	машина Q, стирающая все единицы слева от ближайшей пустой ячейки
q_6	$s_0 Н q_1$	$s_0 Л q_6$	

Работа этой машины может быть следующим образом описана в терминах машин P, Q, R. Сначала используется машина P, затем в соответствии с тем, обозревает машина P в состоянии q_2 пустую ячейку или ячейку с буквой |, используется машина R или Q соответственно. В случае, если используется машина Q, ее результат подается в P. Записать это можно следующим образом: $\dot{P}\begin{Bmatrix} R \\ \dot{Q} \end{Bmatrix}$, где точки обозначают, что результат работы машины Q подается обратно в качестве входных данных для машины P.

Пример 12. Рассмотрим машину Тьюринга T_1, которая по записи любого числа, воспринятого в стандартном положении, по окончании работы через одну пустую ячейку от него выдает его копию, воспринимая ее в стандартном положении.

Машину T_1 построим, используя введенные ранее элементарные машины. Оказывается,

$$T_1 = C \circ \dot{L} \circ l \begin{Bmatrix} r \circ D \circ R \\ r \circ B \circ R \circ \dot{A} \end{Bmatrix}.$$

Возможно Вы затрудняетесь определить, как работает эта машина.

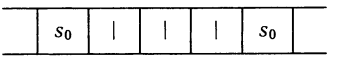
Порядок работы машины T_1 следующий: через одну пустую ячейку справа от записи числа x машина пишет | (т. е. число 0) (так работает машина C). Затем идет влево к стандартному положению ближайшего слева числа (так работает машина L), далее сдвигает воспринимаемую ячейку на одну ячейку влево (машина l). Если в этой ячейке стоит символ | (это означает, что исходное число не равно 0), то в результате работы машины r будет восприниматься исходное число в стандартном положении. Машина B уменьшит число палочек в его записи на одну и в результате работы машины R будет восприниматься ближайшее справа число, которое машина A увеличит на 1. После этого управ-

ление передается машине L и повторяется цикл до тех пор, пока машина l не «увидит» слева от воспринимаемого числа s_0. Затем управление передается машине r, которая будет воспринимать единственную оставшуюся палочку | от исходного числа. Далее машина D заполнит промежуток из пустых клеток, оставляя между восстановленным исходным числом и ближайшим справа (т. е. копией исходного числа) одну пустую клетку. Машина R идет вправо к стандартному положению ближайшего справа числа, т. е. в результате ее работы будет восприниматься в стандартном положении копия исходного числа.

Предложим теперь Вам следующую задачу.

Задача 10.12. а) Составьте программу машины T_1.

б) Имитируйте работу машины T_1 для записи на ленте:



* * *

Мы несколько нарушили наше обещание изложить содержание книги в виде кратких бесед. Десятая оказалась не такой уж краткой. Это объясняется нашим стремлением разъяснить, пока на интуитивном уровне, с помощью достаточного числа примеров и задач, что это за машина Тьюринга. Теперь мы сможем в следующей беседе показать, что машине Тьюринга можно дать математическое определение. Но перед этим предложим Вам еще несколько задач.

Задачи

10.13. Используя элементарные машины и операции над машинами, постройте:

а) машину Тьюринга, которая по записи набора чисел x_1, x_2, ..., x_n на ленте выдает в качестве результата набор чисел x_1, x_2, ..., x_n, 2, воспринимаемый в стандартном положении;

б) машину Тьюринга, которая по записи числа на ленте в качестве результата выдает число, увеличенное на единицу и записанное через одну пустую ячейку от данного (иными словами, машину, вычисляющую функцию прибавления единицы $f(a) = a + 1$).

(Построить машину Тьюринга означает построить программу машины, а затем проверить, ту ли машину построили, имитируя ее работу для конкретных записей на ленте. При составлении программ для реальных вычислительных машин эту проверку называют **отладкой** программы.)

10.14. Какие операции выполняют машины: а) T_1^2; б) R^2; в) R^k, $k \geqslant 1$; г) L^k, $k \geqslant 1$; д) $C \circ A^5$; е) $T_1 \circ A^4$?

10.15. Постройте машину Тьюринга T_2, которая по записи набора чисел x_1, x на ленте в качестве результата записывает справа от набора через одну пустую ячейку первое число этого набора.

10.16. Составьте программу машины: а) $C \circ A^3$; б) $T_1 \circ A^4$.

10.17. Составьте программу машины Тьюринга T_{x+y}, которая по записи набора чисел x, y выдает запись числа, являющегося их суммой, т. е. вычисляющей функцию $S(x, y) = x + y$.

(Указание: так работает машина Тьюринга

$$T_{x+y} = l \begin{cases} r \circ T_2 \circ C_m \\ r \circ T_2 \circ A \circ T_2 \circ B, \end{cases}$$

где через C_m обозначена машина из примера 11.)

11. Машина Тьюринга — математическое понятие алгоритма

Прежде всего покажем, что машина Тьюринга — математическое понятие, т. е. она может получить точное математическое определение. Затем мы покажем, что этот математический объект обладает всеми свойствами, характеризующими любой алгоритм, т. е. машина Тьюринга является математическим уточнением (одним из возможных) интуитивного понятия алгоритма.

11.1. Выясним, что мы имеем в каждой машине Тьюринга. Мы имеем три конечных множества A, Q и T и программу, полностью определяющую работу машины Тьюринга.

Что же представляет собой программа?

Каждой паре вида (s_i, q_i), где $s_i \in A$ и $q_i \in Q \backslash \{q_0\}$, соответствует тройка (s_j, t, q_j), где $s_j \in A$, $t \in T$ и $q_j \in Q$ (q_0 не участвует в парах (s_i, q_i), так как паре (s_i, q_0) уже ничего не соответствует, машина останавливается в заключительном состоянии q_0).

Множество всех пар вида (s_i, q_i), где $s_i \in A$ и $q_i \in Q \backslash \{q_0\}$, называется произведением множеств A и $Q \backslash \{q_0\}$ и обозначается $A \times Q \backslash \{q_0\}$. Аналогично, множество всех троек вида (s_j, t, q_j), где $s_j \in A$, $t \in T$ и $q_j \in Q$, называется произведением множеств A, T и Q и обозначается $A \times T \times Q$.

Таким образом, используя понятие произведения множеств, можно утверждать, что программа машины Тьюринга представляет собой функцию с областью определения $A \times Q \backslash \{q_0\}$, принимающую значения из множества $A \times T \times Q$, или отображение первого множества во второе: $A \times Q \backslash \{q_0\} \to A \times T \times Q$.

Теперь мы можем дать точное, математическое определение машине Тьюринга.

Определение. Машиной Тьюринга (МТ) называется система вида

$$(A, s_0, Q, q_1, q_0, T, \tau),$$

где A — конечное множество — **алфавит МТ,** $s_0 \in A$ и называется **пустой буквой** алфавита, Q — конечное множество, элементы

которого называются **состояниями** МТ (Q — множество состояний МТ), $q_1 \in Q$, q_1 — **начальное состояние МТ,** $q_0 \in Q$, q_0 — **пассивное** или **заключительное состояние МТ,** $T = \{Л, Н, П\}$ — **множество сдвигов МТ,** $\tau: A \times Q \setminus \{q_0\} \to A \times T \times Q$, τ — **программа МТ.**

Нетрудно убедиться в том, что в этом определении фигурируют только математические и логические термины (или символы): множество, конечное множество, элемент множества, отношение принадлежности ($\in$), функция ($\to$), произведение множеств, есть ($-$), равно ($=$). И никакие другие, нематематические или нелогические понятия в приведенной формулировке не используются.

К чему могли бы Вы придраться, дорогой читатель? Быть может, к слову «система»? Что же, заменим его словом «семерка», в смысле упорядоченного множества из семи элементов.

Теперь, наверное, ясно, почему мы так долго и тщательно развивали интуитивные представления о МТ. Если начать с математического определения, вряд ли что-нибудь было бы понятно.

11.2. Рассмотренные нами примеры машин Тьюринга показывают, что в МТ перерабатываются слова в алфавите машины согласно программе этой машины. Действительно, возьмем произвольное слово P в алфавите машины и запишем его слева направо на ленту так, чтобы управляющая головка машины воспринимала правый символ записи на ленте. Приведем машину в начальное состояние q_1. Машина начинает работать согласно программе. Слово Q на ленте, появившееся в момент прекращения работы машины (когда она приходит в заключительное состояние q_0), и является результатом переработки слова P.

Какую бы МТ, имеющую алфавит $A = \{s_0, s_1, ..., s_k\}$, состояния $q_0, q_1 ..., q_l$ и программу τ, мы ни взяли, можем считать, что имеется алгоритм, исходными объектами, промежуточными и окончательными результатами которого являются слова в алфавите A. Предписанием, задающим этот алгоритм, является программа τ. Другими словами, с математической точки зрения МТ — это алгоритм для переработки слов в алфавите этой машины (ради удобства мы отождествляем МТ с ее программой).

Проиллюстрируем теперь основные свойства алгоритмов на примере МТ.

1) **Массовость алгоритма.** Множество исходных данных для алгоритма — множество всевозможных слов в алфавите A машины. Это множество бесконечно, его элементы записываются на ленте машины.

2) **Результативность алгоритма.** Алгоритм по любому исходному данному позволяет в конечное число шагов получить результат. Программа МТ применяется единообразно ко всевозможным исходным данным и не меняется в процессе работы машины над исходным словом. Программа описывает переход от одного состояния к другому. Некоторое состояние опознается как заключительное. Появившееся при этом на ленте слово в алфавите A

является результатом переработки слова, записанного на ленте в начальном состоянии машины.

3) **Конструктивность объектов.** Исходные объекты, промежуточные и окончательные результаты для МТ — слова в алфавите A машины. Такие объекты, как мы видели ранее, являются конструктивными.

4) **Детерминированность (определенность) алгоритма.** Программа τ составлена таким образом, что ее исполнение однозначно осуществимо. Действительно, программа τ — это совокупность команд вида $s_i q_j \to s_m T q_l$, причем любые две различные команды не содержат одинаковых левых частей. При этом условии система команд не может требовать двух или более различных действий в одно и то же время. Свойство детерминированности означает также, что применение программы τ к одному и тому же слову в алфавите A приводит к одному и тому же результату с одной и той же последовательностью состояний ленты.

5) **Конечность предписания, задающего алгоритм.** Программа τ представляет собой конечное предписание, причем процесс вычислений протекает только согласно программе и исходным данным, ничего более не используется.

11.3. Во множестве всевозможных алгоритмов мы выделили в качестве подмножества класс алгоритмов специального вида — машин Тьюринга. Рассмотренные примеры МТ показывают, что алгоритмы, которые обычно задаются в виде словесного предписания В, мы задали посредством МТ, т. е. для каждого из этих алгоритмов нам удавалось построить МТ с некоторым алфавитом A (словами этого алфавита кодируются исходные данные, промежуточные и окончательные результаты алгоритма), состояниями $q_0, q_1, \ldots, q_l$ и программой τ, которая является записью предписания В в виде системы команд.

Эти примеры приводят к мысли: нельзя ли задавать посредством МТ и другие известные нам алгоритмы, задаваемые обычно с помощью предписаний. Другими словами, насколько «богат» класс МТ? Быть может он включает все алгоритмы?

Ответ на этот вопрос дает **тезис Тьюринга,** согласно которому **всякий алгоритм может быть задан посредством МТ.**

Это означает, что для всякого алгоритма, задающего отображение F множества M в множество K, можно построить МТ с алфавитом A, словами которого кодируются элементы множеств M и K, состояниями $q_0, q_1, \ldots, q_n$ и программой τ, которая работает следующим образом: если алгоритм перерабатывает элемент $a \in M$ в элемент $F(a) \in K$, то МТ, начиная работу со слова P, являющегося кодом $a \in M$, через конечное число шагов останавливается и дает в качестве результата на ленте слово R — код элемента $F(a) \in K$.

В тезисе Тьюринга речь идет, с одной стороны, о понятии алгоритма, которое не является точным математическим понятием; с другой стороны, о точном математическом понятии — МТ. Значение этого тезиса и заключается в том, что он уточняет поня-

тие алгоритма через математическое понятие — машину Тьюринга (так же, как тезис Маркова уточняет интуитивное понятие алгоритма с помощью математического понятия — нормальный алгоритм). Этот тезис не является теоремой, его нельзя доказать, поскольку он представляет собой утверждение о понятии алгоритма, которое не является точным математическим понятием. По существу, тезис Тьюринга — гипотеза, предположение. На чем же основана уверенность в справедливости этого предположения? Главным образом, на опыте. Все известные в математике алгоритмы могут быть заданы посредством МТ. Но содержание тезиса Тьюринга обращено не только в прошлое. Оно содержит и прогноз на будущее: всякий раз, когда в будущем какое-либо предписание будет признано алгоритмом, его можно будет также задать посредством МТ.

11.4. Итак, машину Тьюринга мы можем рассматривать как точное математическое понятие алгоритма. Это уточнение было выработано в науке лишь в тридцатые годы нашего века. До этого в математике обходились интуитивным понятием алгоритма, смысл которого иллюстрировался нами на многочисленных примерах. Почему же возникла необходимость в уточнении, математическом определении понятия алгоритма? В математике не было бы необходимости в определении понятия алгоритма, если была бы уверенность в том, что **все** поставленные математические задачи (и те, которые возникнут в будущем) могут быть алгоритмически решены. До тех пор пока математики занимались построением конкретных алгоритмов, и это им удавалось, они обходились интуитивным понятием алгоритма. Но в первые десятилетия XX века накопилось много классов задач, довольно простых по своей формулировке, для которых алгоритма найти не удавалось. И математикам пришла в голову мысль: а вдруг для того или иного класса задач просто невозможно найти алгоритм? А если его не существует, и они ищут то, чего нет?

Приведем примеры таких классов задач.

1) Мы располагаем алгоритмом, позволяющим по любому уравнению с целыми коэффициентами с одним неизвестным выяснить, имеет ли оно решение в целых числах или нет. А если рассмотреть уравнение с двумя или многими неизвестными? Существует ли алгоритм, позволяющий по произвольному уравнению с целыми коэффициентами выяснить, имеет оно целочисленное решение или нет? Эта задача была поставлена еще в 1901 году известным немецким математиком Д. Гильбертом (десятая проблема Гильберта) среди других проблем, которые, по его словам, XIX век передал в наследство XX. Вначале поиски математиков были направлены на нахождение алгоритма, но требуемый алгоритм так и не был найден. И лишь в 1970 году молодой советский математик Ю. В. Матиясевич доказал, что такого алгоритма не существует. Доказать это утверждение, пользуясь только интуитивным понятием алгоритма, было бы невозможно, ибо в таком

случае не ясно, несуществование чего мы собираемся доказывать. Имея же математическое уточнение понятия алгоритма, можно доказывать несуществование алгоритма. Несуществование алгоритма понимается, например, как несуществование машины Тьюринга, обладающей нужным свойством.

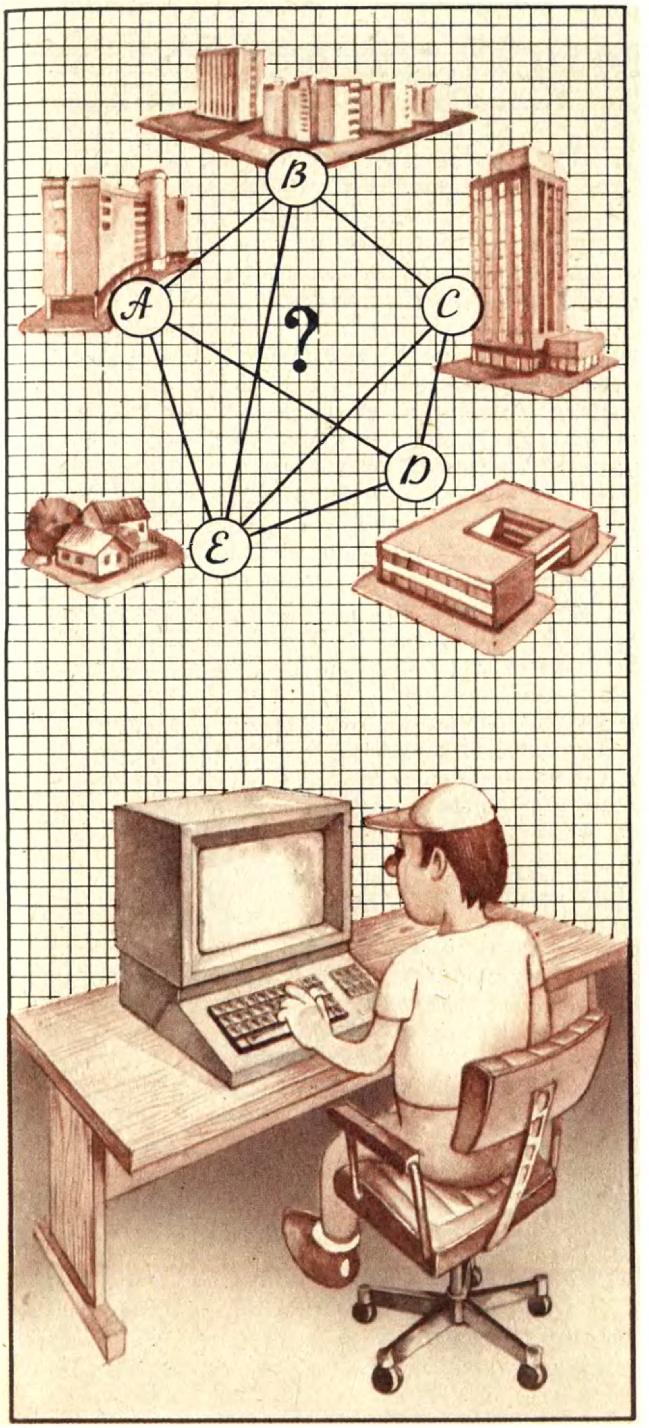
2) В качестве второго примера рассмотрим проблему слов в ассоциативном исчислении. Мы рассмотрели ряд конкретных ассоциативных исчислений, для которых существуют алгоритмы, распознающие эквивалентность любых двух слов. Естественно задать вопрос: существует ли алгоритм, позволяющий по любому ассоциативному исчислению выяснить, разрешима в нем проблема эквивалентности слов или нет?

А. А. Марковым было построено ассоциативное исчисление, для которого такого алгоритма распознавания эквивалентности любой пары слов не существует. При доказательстве несуществования алгоритма в построенном ассоциативном исчислении А. А. Марков использовал сформулированное им другое математическое уточнение понятия алгоритма в форме нормального алгоритма (впоследствии названного **алгоритмом Маркова),** задаваемого алфавитом и нормальной схемой в этом алфавите. Понятие нормального алгоритма было проиллюстрировано нами на примерах алгоритмов над словами.

Определение нормального алгоритма выделяет во множестве всевозможных алгоритмов подмножество алгоритмов специального вида — класс нормальных алгоритмов. Оказалось, что класс алгоритмов в форме машин Тьюринга и класс нормальных алгоритмов совпадают, эти алгоритмы **равносильны.** (Два алгоритма B и C, исходные объекты которых закодированы словами в алфавите A, называются **равносильными,** если для любого слова P в алфавите A результат работы этих алгоритмов над словом P есть одно и то же слово). Иными словами, для каждого алгоритма из класса машин Тьюринга существует равносильный ему алгоритм в классе нормальных алгоритмов, и наоборот.

В этом смысле две математические теории алгоритмов: теория нормальных алгоритмов и теория машин Тьюринга, считаются эквивалентными (равносильными).

Приложения алгоритмов



Мы уже говорили о том, что алгоритмы встречаются в различных областях деятельности человека. Особое значение имеют алгоритмы, накопленные в математике, потому что математика обслуживает другие науки и ее достижения являются достоянием всех наук.

Алгоритмы находят применение в народном хозяйстве при решении важного практического вопроса — построения планов. Проблема нахождения наилучшего, **оптимального** плана весьма актуальна, особенно в условиях реализации принятой XXVII съездом КПСС стратегии ускорения социально-экономического развития страны.

Нередко возникает потребность определить наилучший состав смесей, составленных из различных продуктов и обладающих определенными, заданными свойствами. Такие задачи возникают в сельскохозяйственной практике, в нефтеперерабатывающей, металлургической промышленности, в ряде других областей производства, а их решение тесно связано с применением алгоритмов.

Большое применение алгоритмы нашли при решении игровых, экономических задач. Наличие алгоритма в игровой задаче позволяет установить выигрышную стратегию, т. е. совокупность правил, определяющих однозначно выбор при каждом ходе в зависимости от конкретных ситуаций. В экономических задачах часто требуется определить наиболее выгодный (экономически) алгоритм решения.

Повышенный интерес к понятию алгоритма в наше время объясняется потребностями интенсивно развивающихся информатики и вычислительной техники.

Рассмотрим теперь конкретные примеры применения алгоритмов.

12. Программы для ЭВМ

12.1. В исполнении алгоритмов произошел переворот после появления в сороковые годы нашего столетия машин нового типа — электронных вычислительных машин (ЭВМ) или компьютеров. Оказалось, что человека — исполнителя алгоритма — можно заменить машиной, выполняющей тот же процесс. Программа для ЭВМ — это алгоритм, записанный на «понятном» ей языке.

Представим себе, дорогой читатель, что человек решает задачу, руководствуясь алгоритмом, заданным в виде предписания *В*. Для этого процесса характерны следующие моменты:

1) **Хранение информации.** Процесс решения предполагает наличие таких средств, которые обеспечивают хранение всех необходимых сведений (всей необходимой информации): исходных объектов для алгоритма, промежуточных и окончательных результатов, а также самого предписания *В*. Хранение информации обычно обеспечивается записью на листе бумаги. В действительности человек не записывает буквально все — кое-что он хранит

в своей памяти, а некоторые сведения берет из различных справочников и таблиц. Под «листом бумаги» будем понимать совокупность всех средств, обеспечивающих хранение информации (поэтому мы эти слова взяли в кавычки, чтобы подчеркнуть приписываемый им смысл обобщенной памяти).

2) **Обработка информации.** В ходе процесса решения человек выполняет определенные операции, предусмотренные алгоритмом. Каждая отдельная операция заключается в том, что человеком в соответствии с предписанием *В* извлекаются некоторые необходимые сведения из «листа бумаги», обрабатываются, а результат обработки помещается в определенное место «листа бумаги».

3) **Управление процессом.** Согласно предписанию *В* человек принимает решение об осуществлении той или иной операции на каждом этапе процесса.

В ЭВМ происходят те же процессы, какие были только что описаны, но уже без участия человека. В этих машинах имеются устройства, выполняющие функции хранения информации, обработки ее и управления всем процессом решения задачи.

1) **Запоминающее устройство (ЗУ) или память машины** играет роль «листа бумаги». Информация, которая поступает в машину, а также та, которая вырабатывается в ней, изображается в виде слов в двоичном алфавите {0, 1}. Выбор этого алфавита объясняется тем, что он проще всего осуществляется физически в электрических цепях в виде двух состояний: высокое напряжение (есть сигнал) и низкое напряжение (нет сигнала).

В ЗУ хранятся: исходные объекты, записанные в виде слов в алфавите {0, 1}, промежуточные и окончательные результаты, закодированные с помощью слов этого же алфавита, предписание *В*, записанное также в виде набора слов в этом алфавите. Другими словами, вся информация, которая поступает в машину, кодируется словами в алфавите {0, 1}, т. е. кодируется числами в двоичной системе счисления. Поэтому обработка всякой (не только числовой, но и текстовой) информации в ЭВМ происходит, по существу, как вычисление с двоичными числами. Причем эти вычисления осуществляются при выполнении не только арифметических, но и логических операций. Этим, в частности, объясняется название «вычислительная машина», хотя ЭВМ решают далеко не только вычислительные задачи. Они превращают решаемые ими задачи в вычислительные, и логические заключения тоже вычисляют.

ЗУ состоит из набора занумерованных **ячеек.** Эти номера называются **адресами** ячеек. Каждая ячейка хранит одно сообщение, закодированное в виде двоичного слова. (Мы не будем различать отдельных частей ЗУ, а будем исходить из представления о едином ЗУ.)

2) **Арифметическое устройство (АУ)** предназначено для выполнения операций (арифметических и логических, поэтому иногда его называют арифметико-логическим устройством —

АЛУ) над словами в двоичном алфавите. Переработка подаваемых в него данных в нужный результат происходит путем преобразования в электронном устройстве входных электрических сигналов, изображающих исходные данные, в электрические сигналы, изображающие окончательный результат. Входные сигналы поступают в АУ из ячеек памяти, где они хранились, а выходной сигнал отправляется из АУ в ту ячейку, в которой он будет храниться.

Рассмотрим, например, операцию, согласно которой содержимые ячеек с адресами 17 и 18 умножаются, а результат отправляется в ячейку с адресом 25. Чтобы такая операция осуществлялась в машине, необходимо, чтобы к началу этого такта работы были произведены соединения ячеек 17 и 18 с АУ и АУ с ячейкой 25; также необходимо, чтобы АУ было настроено на нужную операцию (в данном случае на умножение). Все это уже выполняет устройство управления.

3) **Устройство управления (УУ).** На каждом этапе работы машины УУ создает условия для выполнения очередной операции, предусмотренной введенной в машину **программой.** УУ, разумеется, условно называют «мозгом ЭВМ». УУ вместе с АУ называют также **процессором** (именно в этих устройствах происходит **процесс** решения задачи).

Кроме рассмотренных трех основных устройств ЭВМ (ЗУ, АУ и УУ), в ней имеются и другие, например, устройства для ввода исходных данных в машину, для вывода из нее результатов. Но эти устройства не имеют значения при рассмотрении принципов работы ЭВМ и поэтому мы их не будем рассматривать.

12.2. Для решения различных задач в ЭВМ предусматривается возможность выполнения определенного набора элементарных операций, кодируемых в виде отдельных команд. Систему команд машины или совокупность всех операций, которые она может выполнять, называют собственным языком машины (или языком машинных кодов). Программа вычислительной машины представляет собой запись алгоритма на языке этой машины. Однако составлять программу на языке машинных кодов — дело весьма трудоемкое. Кроме того, каждый тип ЭВМ имеет свой собственный язык, зависящий от технических данных машины. К тому же типы ЭВМ совершенствуются и видоизменяются, вместе с этим изменяется и их язык. Поэтому и возникла необходимость создания универсальных языков программирования, пригодных и удобных для записи любого алгоритма и не зависящих от конкретных машин, на которых предполагается реализовать этот алгоритм. Одновременно решалась и проблема снабжения ЭВМ так называемыми **трансляторами,** своего рода внутренними переводчиками с универсальных языков, называемых **алгоритмическими языками программирования,** на язык машинных кодов.

В принципе рассмотренные нами математические уточнения алгоритма — нормальный алгоритм Маркова и машина Тьюрин-

га — могут быть выбраны в качестве языков для алгоритмизации задач. Но эти языки оказались практически неприемлемыми для описания алгоритмов решения задач при реализации их на ЭВМ. Одна из причин — неэкономичность указанных языков. Как мы с вами видели, уже для описания даже простых алгоритмов соответствующий нормальный алгоритм или машина Тьюринга становятся слишком громоздкими. Поэтому решение практических задач с помощью ЭВМ вызвало появление большого числа работ, посвященных созданию алгоритмических языков программирования.

В настоящее время широкое распространение, особенно в практике обучения программированию, получил алгоритмический язык БЕЙСИК (*BASIC*), один из многочисленных языков программирования, близких к разговорному. Программа, записанная на этом языке, может быть реализована на любой машине, на которой имеется транслятор с языка БЕЙСИК на язык кодов данной конкретной машины. Запись программы на языке БЕЙСИК обладает большой наглядностью и близка к записи алгоритма в виде словесного предписания.

Мы не будем здесь изучать ни язык БЕЙСИК, ни какой-нибудь другой язык программирования. Это не входит в цели нашей книги, посвященной, как об этом свидетельствует само название, разъяснению и уточнению (математическому) понятия алгоритма. Однако, чтобы показать на конкретном примере, что программа для ЭВМ это не что иное, как запись алгоритма на языке программирования, мы должны предварительно ознакомиться с необходимыми элементами этого языка.

Мы уже имели возможность, в связи с изучением алгоритмического языка, обратить Ваше внимание на то, что изучение всякого языка, как правило, начинается с перечисления символов, входящих в алфавит этого языка. Такими символами, входящими в алфавит языка БЕЙСИК, являются, например, цифры от 0 до 9, прописные латинские буквы, знаки арифметических операций и др. Из знаков алфавита составляются слова, обозначающие числа, переменные, функции, арифметические выражения, операторы, представляющие составные части программы на БЕЙСИКе.

Прежде всего рассмотрим, как представляются на языке БЕЙСИК числа, переменные, функции и арифметические выражения.

1) В БЕЙСИКе допускается обычная десятичная запись чисел, с тем лишь изменением, что вместо запятой для отделения целой части от дробной используется точка. Например, числа 0,21 и —11,05 будут записываться 0.21 и —11.05.

2) Переменные обозначаются прописными буквами латинского алфавита или буквой и цифрой. Так, например, переменные x, y, a, b, c, x_1, x_2, a_3, ... на языке БЕЙСИК будут записываться так: X, Y, A, B, C, X1, X2, A3, ...

3) На языке БЕЙСИК имеются несколько стандартных функ-

ций, для которых введены специальные обозначения. Для примера приведем математические записи и записи на языке БЕЙСИК некоторых стандартных функций:

математическая запись	запись на языке БЕЙСИК
$\|a\|$	ABS(A)
$\sqrt{a}$	SQR(A)
$\sin a$	SIN(A)

В этих записях A — аргумент функции всегда заключается в скобки и представляет собой арифметическое выражение, в частности, переменную или число.

4) Арифметические выражения (в школе их называют алгебраическими) строятся из чисел и переменных с помощью знаков арифметических операций и скобок. На языке БЕЙСИК они записываются в строку, причем для арифметических операций приняты следующие символы:

$+$ — сложение,
$-$ — вычитание,
$\times$ — умножение,
$/$ — деление,
$\uparrow$ — возведение в степень.

Приведем несколько арифметических выражений в математической записи и на языке БЕЙСИК:

математическая запись	запись на языке БЕЙСИК
$\frac{a+b}{2}$	(A+B)/2
$\|x-y\|$	ABS(X-Y)
b^2-4ac	B↑2-4*A*C

5) **Операторы** — составные части программы, предписывающие выполнение определенных действий. Язык БЕЙСИК строится на базе английского языка, т. е. служебные слова, с которых начинаются операторы, английские, но их семантика (смысл, который им приписывается на языке БЕЙСИК) точно определена.

Мы рассмотрим лишь несколько операторов, с помощью которых уже можно составлять простейшие программы.

1) **Оператор присваивания.** С помощью этого оператора переменной присваивается значение некоторого арифметического выражения или изменяется прежнее (ранее присвоенное) значение переменной.

Общий вид оператора присваивания: LET X = A.

Здесь служебное слово LET означает «пусть» и служит названием оператора, X — переменная, A — арифметическое выражение (в частности, число). Этим общим видом задано синтаксическое правило записи оператора присваивания: за служебным словом LET записывается переменная, за ней знак = (равно), а за ним — арифметическое выражение.

Приведем несколько примеров операторов присваивания и пояснения к ним:

операторы	пояснения
LET X = 5	переменной x присвоить значение 5
LET Y = Y + 1	увеличить значение переменной y на 1
LET D = B↑2 − 4*A*C	переменной D присвоить значение выражения $b^2 - 4ac$

(В некоторых вариантах языка БЕЙСИК служебное слово LET в операторах присваивания опускается.)

2. **Оператор ввода.** С помощью оператора INPUT (ввод) в машину вводятся исходные данные. После служебного слова INPUT перечисляются переменные, значения которых вводятся во время выполнения программы.

Например, выполнение оператора

INPUT A, B, C

сводится к запросу трех чисел для переменных A, B, C во время выполнения программы (на экране появляется знак вопроса). После введения с клавиатуры этих чисел решение задачи продолжается.

3. **Оператор печати.** С помощью оператора печати PRINT (печать) производится печать результатов и пояснительных текстов.

Оператор печати состоит из служебного слова PRINT, за которым следует список выводимых элементов (одного или нескольких). Выводимыми элементами могут быть числа, переменные, выражения, тексты. Если выводится текст, то в операторе печати он заключается в кавычки. Выводимые элементы отделяются друг от друга запятой или точкой с запятой. В последнем случае результаты печатаются с пробелами между собой.

Приведем несколько примеров операторов PRINT и соответствующей печати выводимой информации.

оператор	печать
PRINT 1; 2; 3	1 2 3
PRINT «X1 = »; 5; «X2 = »; 3	$x_1 = 5$ $x_2 = 3$
PRINT «УРАВНЕНИЕ НЕ ИМЕЕТ КОРНЕЙ»	уравнение не имеет корней

4. **Оператор END (конец)** — последний оператор всякой программы, обозначает ее конец.

С помощью операторов LET, INPUT, PRINT, END мы уже можем записать на языке БЕЙСИК простейшие линейные алгоритмы. Необходимо лишь несколько разъяснений по поводу **оформления** программы. Операторы располагаются в отдельных строках и нумеруются. Номерами строк-операторов могут быть любые натуральные числа от 1 до 9999. Для того чтобы машина выполняла программу, используются **управляющие** команды, которые не нумеруются и не являются составными частями программы. Мы воспользуемся двумя такими командами: NEW (новый) и RUN (пуск). Первая означает начало работы с новой программой (ранее не записанной в памяти машины), вторая является

сигналом начала трансляции введенной в машину программы.

Теперь переведем на язык БЕЙСИК приведенный в беседе 6 линейный алгоритм вычисления значения переменной y, заданной формулой $y = \frac{3x+5}{x^2+1}$.

Этот алгоритм был представлен с помощью предписания, блок-схемы (см. рис. 6) и алгоритмической записи. Если осуществить «дословный» перевод на язык БЕЙСИК, мы получим следующую программу:

```
    NEW
10 INPUT X
20 LET A=3*X
30 LET A=A+5
40 LET B=X*X
50 LET B=B+1
60 LET Y=A/B
70 PRINT «Y=»; Y
80 END
```

Эту программу можно записать намного короче, используя лишь один оператор присваивания:

```
    NEW
10 INPUT X
20 LET Y=(3*X+5)/(X↑2+1)
30 PRINT «Y=»; Y
40 END
```

После введения программы по команде RUN машина начнет ее выполнять. Но, наткнувшись на оператор INPUT, она сразу же потребует, чтобы ввели значение x (на экране появится знак ?). После введения значения для x, тут же получите результат. Например, если Вы ввели значение 3, на экране или на бумажной ленте будет напечатано $y = 1.4$.

У Вас, вероятно, возникает вопрос: почему мы занумеровали строки-операторы числами 10, 20, 30, ..., а не 1, 2, 3, ...? Такая нумерация с интервалами дает возможность дописать пропущенный по ошибке оператор без изменения нумерации, приписав ему нужный номер. Причем, если, например, мы дописали в конце программы операторы с номерами 21, 22, 23, то машина будет их выполнять в нужной последовательности, т. е. 10, 20, 21, 22, 23, 30, ...

Возможно, у Вас возникает и такой вопрос: линейные алгоритмы — самые простые и не самые интересные, как же перевести на язык БЕЙСИК более сложные алгоритмы, разветвленные и циклические?

Для этого нам нужны еще несколько операторов.

5. **Оператор условного перехода (ветвления) IF.** С помощью этого оператора осуществляется ветвление процесса решения.

Общий вид этого оператора: IF P THEN N.

Здесь IF (если) — имя оператора, THEN (то) — вторая часть оператора («если ..., то»); P — условие, выражаемое обычно предложением, составленным из двух арифметических выражений, соединенных одним из знаков отношений $<$, $>$, $\leqslant$, $\geqslant$, $=$, $\neq$; N — номер оператора, на который произойдет переход в случае истинности предложения P, т. е. при выполнении условия.

Пример записи оператора условного перехода:

20 IF A $\neq$ 0 THEN 60

Как же машина работает, выполняя этот оператор?

Дойдя до оператора 20, машина проверяет, выполняется ли условие A $\neq$ 0. Если оно выполняется, машина переходит к выполнению оператора 60. Иначе, т. е. если условие A $\neq$ 0 не выполняется, машина выполняет оператор, следующий за оператором 20.

Как видите, название «условный переход» вполне оправдано. Но часто возникает потребность в безусловном переходе, в нарушении нормальной (в порядке увеличения номеров) последовательности операторов.

6. **Оператор перехода** состоит из служебного слова GO TO (перейти на) и номера оператора, на который надо перейти.

Так, оператор

50 GO TO 100

указывает машине, дошедшей до строки 50, перейти к строке (оператору) 100.

Теперь мы уже сможем перевести на язык БЕЙСИК какой-нибудь разветвленный алгоритм, например, алгоритм для решения уравнения $ax = b$, приведенного нами в беседе 3 с помощью предписания, затем (в беседе 4) переведенного в блок-схему (см. рис. 2) и в алгоритмическую запись (беседа 5).

Этот алгоритм переводится в следующую БЕЙСИК-программу:

```
   NEW
10 INPUT A, B
20 IF A ≠ 0 THEN 60
30 IF B ≠ 0 THEN 90
40 PRINT «ЛЮБОЕ ЧИСЛО — КОРЕНЬ УРАВНЕНИЯ»
50 GO TO 100
60 LET X = B/A
70 PRINT «X =»; X
80 GO TO 100
90 PRINT «УРАВНЕНИЕ НЕ ИМЕЕТ КОРНЕЙ»
100 END
```

Как же работает эта программа (или этот алгоритм)?

Мы составили таблицу (беседа 3), в которой указаны последовательности шагов процесса решения, однозначно определяемого словесным предписанием. Мы можем теперь перевести эти

последовательности шагов в последовательности операторов программы, однозначно определяющей процесс решения:

$A \neq 0$	$B \neq 0$	последовательность операторов	результат
И		10,20,60,70,80,100	$X = B/A$
Л	И	10,20,30,90,100	уравнение не имеет корней
Л	Л	10,20,30,40,50,100	любое число — корень уравнения

Отметим, что оператор IF может быть использован и для организации циклов (хотя на языке БЕЙСИК имеется, конечно, и специальный оператор цикла).

Покажем, как с помощью оператора IF можно перевести на язык БЕЙСИК циклический алгоритм вычисления наибольшего общего делителя двух чисел (второй вариант), представленный в виде блок-схемы (см. рис. 3, *г*) и алгоритмической записи (беседа 5).

Получаем следующую БЕЙСИК-программу:

```
   NEW
10 INPUT A, B
20 LET X = A
30 LET Y = B
40 IF X = Y THEN 100
50 IF X > Y THEN 80
60 LET Y = Y - X
70 GO TO 40
80 LET X = X - Y
90 GO TO 40
100 PRINT «НОД(A, B) = »; X
110 END
```

Посмотрим теперь, как работает эта программа, а точнее машина, выполняющая эту программу после введения в нее (в соответствии с оператором INPUT) пары чисел, например 30 и 12.

Последовательность операторов	ввод	X	Y	$X=Y$	$X>Y$	$X\neq Y$	вывод
10	30, 12						
20		30					
30			12				
40				нет			
50					да		
80		18	12				
40				нет			
50					да		
80		6	12				
40				нет			
50					нет		
60		6	6				
40				да			
100							НОД (30,12) = 6
110	конец						

И еще один пример. Мы рассмотрели в беседах 3, 4 и 5 в различных формах разрешающий алгоритм, отвечающий «да» или «нет» на вопрос: «имеет ли уравнение $ax^2+bx+c=0$ действительные корни?»

Если его немного продолжить, получим вычислительный алгоритм для решения квадратного уравнения $ax^2+bx+c=0 (a\neq 0)$.

Сначала представим его с помощью словесного предписания:

1. Вычислить $D=b^2-4ac$. Перейти к ук. 2.
2. Если $D\geqslant 0$, то перейти к ук. 4, иначе — к ук. 3.
3. Уравнение не имеет действительных корней. Перейти к ук. 6.
4. Уравнение имеет действительные корни. Перейти к ук. 5.
5. $x_1=\frac{-b+\sqrt{D}}{2a}$; $x_2=\frac{-b-\sqrt{D}}{2a}$. Перейти к ук. 6.
6. Процесс решения окончен.

Алгоритмическая запись этого алгоритма выглядит так:

алг КВУР* (**вещ** a, b, c, x_1, x_2, **лит** y)
  арг a, b, c
  рез x_1, x_2, y
нач **вещ** D
  $D:=b^2-4ac$
  если $D\geqslant 0$
    то $y:=$ «уравнение имеет действительные корни»
      $x_1=\frac{-b+\sqrt{D}}{2a}$
      $x_2=\frac{-b-\sqrt{D}}{2a}$
    иначе $y:=$ «уравнение не имеет действительных корней»
  все
кон

Любая из двух приведенных форм представления алгоритма легко переводится на язык БЕЙСИК.

Получаем следующую программу:

```
NEW
10 INPUT A, B, C
20 LET D = B ↑2 — 4 * A * C
30 IF D ⩾ 0 THEN 60
40 PRINT «УРАВНЕНИЕ НЕ ИМЕЕТ ДЕЙСТВИТЕЛЬНЫХ
   КОРНЕЙ»
50 GO TO 90
60 PRINT «УРАВНЕНИЕ ИМЕЕТ ДЕЙСТВИТЕЛЬНЫЕ
   КОРНИ»
70 PRINT «X1 = »; (— B + SQR(D)) / (2 * A)
80 PRINT «X2 = »; (— B — SQR(D)) / (2 * A)
90 END
```

*) КВУР — квадратное уравнение.

* * *

Приведенными примерами программ для ЭВМ, полученных переводом соответствующих алгоритмов на язык БЕЙСИК, мы хотели показать Вам, что программа для ЭВМ, записанная на каком-нибудь алгоритмическом языке программирования, представляет собой лишь новую, специальную форму записи алгоритма, предназначенную для машины, понимающей этот язык. А машина понимает тот или иной язык программирования, если она снабжена соответствующим транслятором, переводящим этот язык на язык кодов данной машины. Образно выражаясь, можно сказать, что в память машины вложен переводчик с соответствующего языка программирования. Одна и та же машина может понимать различные языки программирования, если один транслятор, записанный на диске или пленке, заменяется другим.

Мы надеемся, что теперь Вы и сами сумеете перевести на язык БЕЙСИК некоторые из алгоритмов, которые Вы построили при решении задач, предложенных в беседах 3, 4, 5.

Задачи

12.1. Переведите в БЕЙСИК-программы алгоритмы, которые Вы построили при решении задач: 3.2, 3.6, 3.7, 5.2, 5.3, 5.4.

13. Беспроигрышные алгоритмы

Рассматриваемые нами игры будут играми с предметами. В каждой игре участвуют два игрока, ходы которых чередуются. Для каждой партии возможен лишь один из двух исходов: выигрыш игрока A, начинающего игру, либо выигрыш его партнера — игрока B.

Пример 1. На столе n предметов (например, палочек или пуговиц), $n > 2$. Двое игроков поочередно берут предметы со стола, причем за один раз разрешается брать один или два предмета. Проигрывает тот, кому приходится взять последний предмет.

Рассматривая различные натуральные числа, больше двух, мы получим бесконечное множество конкретных игр: игру «6 предметов», игру «7 предметов», игру «45 предметов» и т. д. Возникает вопрос: есть ли алгоритм, позволяющий по любой конкретной игре определять, может ли игрок А, начинающий игру, выиграть в ней, т. е. есть ли **беспроигрышный алгоритм?**

Прежде чем построить требуемый алгоритм, рассмотрим конкретную игру «6 предметов» ($n = 6$). Ответим на вопрос: может ли игрок A, начинающий игру, поставить своего соперника перед необходимостью взять последний предмет?

Чтобы проанализировать полностью все возможности игрока A, изобразим графически игру посредством «дерева» (рис. 17).

Объясним, как нужно читать то, что изображено на этом де-

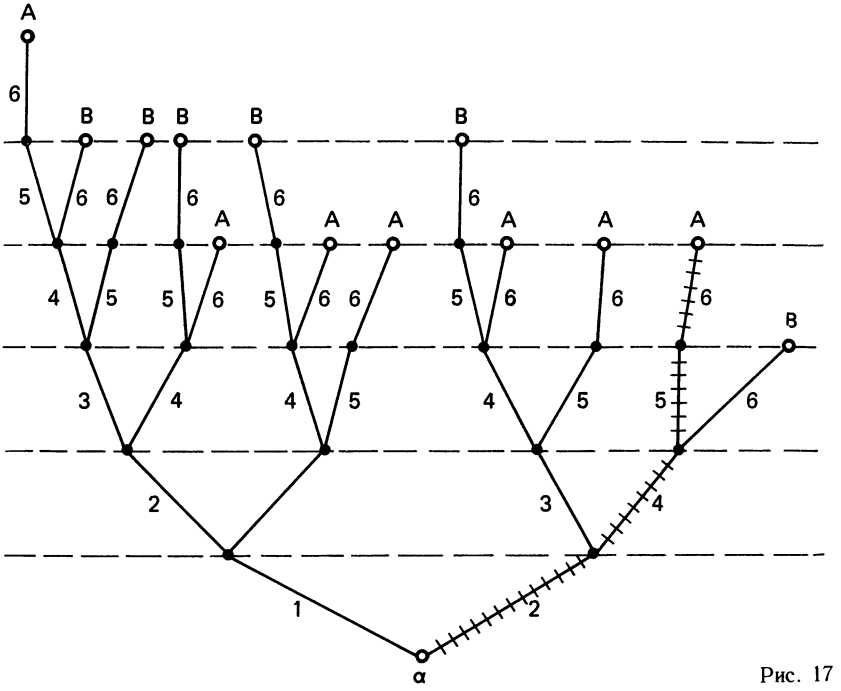


Рис. 17

реве. Вершины дерева представляют различные положения, которые могут возникнуть в партиях, разыгрываемых в соответствии с правилами игры. Поскольку при каждом ходу игрок имеет только две возможности, то из каждой вершины исходят две ветви. Для определенности примем, что левая ветвь, выходящая из любой вершины, соответствует изъятию одного предмета, а правая — двух. Партия изображается ломаной, которая начинается с самой нижней вершины α дерева (корень дерева), причем каждый ход изображается перемещением из данной вершины в какую-нибудь из примыкающих вершин более высокого уровня. Партия закончена тогда, когда достигнута какая-нибудь вершина, из которой не выходят ветви (концевая вершина). В зависимости от того, по какой ломаной развивается игра, игрок выигрывает или проигрывает. При каждой концевой вершине дерева проставлены буквы, соответствующие выигравшим (буква A в концевой вершине означает выигрыш игрока A, буква B — выигрыш игрока B). На каждой ветви указано общее число предметов, изъятых на данной стадии разыгрывания партии. Заштрихованная ломаная изображает партию: $\begin{matrix} A & B & A & B \\ 2 & 2 & 1 & 1 \end{matrix}$, в которой игрок A выигрывает.

С помощью дерева игры покажем, что у игрока A имеется возможность выиграть партию независимо от способа ведения игры

игроком B. Для этого игрок A должен вначале взять в первом ходу 2 предмета. В дереве игры это соответствует первому отрезку заштрихованной ломаной. В игре осталось только 4 предмета. Если B берет один предмет, то A должен во втором ходу взять 2 предмета и B проигрывает, так как он вынужден взять последний предмет. Если же B берет два предмета (вторая ветвь заштрихованной ломаной), тогда A берет только один предмет и B опять проигрывает.

Если же в первом ходу игрок A возьмет один предмет, то игрок B может вести игру так, чтобы наверняка ее выиграть. Для этого игрок B в первом ходу должен взять 1 предмет, во втором должен взять 2 предмета или 1 предмет в зависимости от того, взял игрок A в своем втором ходу соответственно 1 или 2 предмета.

Проведенный с помощью дерева анализ игры показал, что игрок A может вынудить игрока B взять последний предмет, если игрок A будет руководствоваться следующей системой указаний, представляющей собой беспроигрышный алгоритм:

1. Первый ход. A берет два предмета.
2. Очередной ход. Если B при своем очередном ходу взял l предметов ($1 \leqslant l \leqslant 2$), причем еще остались неотобранные предметы, то A берет $3 - l$ предметов.

Предложенная система указаний для A действительно является беспроигрышным алгоритмом, так как она обеспечивает игроку A выигрыш независимо от того, как будет играть B.

Теперь попробуйте решить следующую задачу.

Задача 13.1. Изобразите графически с помощью дерева игру «7 предметов» ($n = 7$). Может ли игрок A, начинающий игру, поставить своего соперника B перед необходимостью взять последний предмет?

Проанализируем задачу, используя построенное Вами дерево игры «7 предметов», и покажем, что ответ на вопрос, поставленный в задаче, отрицательный.

Игрок A в первом ходе может взять либо 1, либо 2 предмета. Если он берет 2 предмета, то игрок B в своем первом ходу возьмет 1 предмет, и когда игрок A во втором ходу возьмет 1 или 2 предмета, игрок B своим вторым ходом, взяв соответственно 2 или 1 предмет, вынудит игрока A в его третьем ходу взять последний предмет. Другими словами, игру выигрывает B, если он будет руководствоваться правилом: во время своего хода бери столько предметов, чтобы в сумме с числом предметов, взятых партнером, их было 3. В соответствии с этим игра будет развиваться следующим образом:

A	B	A	B	A		A	B	A	B	A
2	1	1	2	1	или	2	1	2	1	1

Если же игрок A в первом ходу берет один предмет, то игрок B, придерживаясь указанного правила, также выигрывает игру.

В этом случае возможны партии:

$$\begin{array}{ccccc} A & B & A & B & A \\ 1 & 2 & 1 & 2 & 1 \end{array} \quad \text{или} \quad \begin{array}{ccccc} A & B & A & B & A \\ 1 & 2 & 2 & 1 & 1 \end{array},$$

каждая из которых оканчивается проигрышем игрока A.

Построим теперь алгоритм игры, сформулированной в примере 1. С этой целью проведем расчет игры «от конца». Игрок A, сделав свой последний ход, должен оставить игроку B один предмет. Сколько же предметов он должен оставить, сделав предпоследний ход? Очевидно, 4. Действительно, если теперь B возьмет 1 или 2 предмета, то A может взять соответственно 2 или 1 предмет и во всех случаях на долю игрока B остается $4 - 3 = 1$ предмет. Аналогичным образом найдем, что еще раньше игрок A должен оставить 7 предметов. Если B берет 1 или 2 предмета, A может взять соответственно 2 или 1 предмет, и во всех случаях остается $7 - 3 = 4$ предмета. Рассуждая аналогично, найдем, что игрок A выиграет, если он сможет оставить следующие количества предметов, считая «от конца»: 1, 4, 7, 10, 13 и т. д. до числа, ближайшего к n, но меньшего, чем n. Обозначим его через N. Тогда, придерживаясь указанного правила, выигрывает игру первый игрок, если в первом ходу возьмет $n - N$ предметов. Если же $n - N = 0$, то выигрывает второй игрок. В соответствии с этим искомый алгоритм для игры «n предметов» выглядит так:

1. Среди чисел 1, 4, 7, 10, 13, ..., $1 + 3k$, ... найди число N, ближайшее к n, но меньшее или равное n. Переходи к ук. 2.
2. Найди разность $n - N$. Переходи к ук. 3.
3. Если $n - N \neq 0$, то переходи к ук. 4, иначе — к ук. 8.
4. Возьми в первом ходу $(n - N)$ предметов. Переходи к ук. 5.
5. Если твой партнер при своем очередном ходу взял l предметов $(1 \leqslant l \leqslant 2)$, то возьми $3 - l$ предметов. Переходи к ук. 6.
6. Проверь, есть ли неотобранные предметы. Если неотобранных предметов не осталось, то переходи к ук. 7, иначе — к ук. 5.
7. Прекрати игру, ты выиграл.
8. Выигрывает второй игрок.

Построенный алгоритм позволяет не только по любой конкретной игре определять, выигрывает ее начинающий игрок или нет, но и содержит, в случае положительного решения вопроса, систему указаний, обеспечивающую победу начинающему игроку, или, как говорят, строит выигрышную стратегию для начинающего игрока.

В случае $n - N = 0$ для игрока B также существует выигрышная стратегия, заключающаяся в следующем: если A при своем ходу взял l предметов $(1 \leqslant l \leqslant 2)$, причем остались еще неотобранные предметы, то B берет $3 - l$ предметов.

Рассмотрим теперь обобщение игры из примера 1.

Пример 2. На столе n предметов. Двое поочередно берут предметы со стола, причем за один раз разрешается брать от 1 до p предметов (p значительно меньше n). Проигрывает тот, кому приходится взять последний предмет. Построить алгоритм, позво-

ляющий по любой конкретной игре определять, может ли игрок, начинающий игру, выиграть.

Для построения алгоритма проведем (как и в первом примере) расчет игры от конца. Рассуждая аналогично, мы найдем, что выигрывает игру тот, кто сможет оставить своему противнику следующие количества предметов: $1, p+2, 2p+3, 3p+4$ и т. д. до числа, ближайшего к n, но меньшего или равного n. Обозначим его через N. Тогда, по тому же правилу, что и в примере 1, если $n-N \neq 0$, выигрывает начинающий игрок, если в первом ходу он возьмет $n-N$ предметов. Если же $n-N=0$, то выигрывает второй игрок.

В соответствии с этим нужный алгоритм выглядит так:

1. Среди чисел $1, p+2, 2p+3, 3p+4, \ldots, (k-1)p+k, \ldots$ найди число N, ближайшее к n, но меньшее или равное n. Переходи к ук. 2.
2. Найди разность $n-N$. Переходи к ук. 3.
3. Сравни разность $n-N$ с нулем. Если $n-N \neq 0$, переходи к ук. 4, иначе — к ук. 8.
4. Возьми в первом ходу $n-N$ предметов. Переходи к ук. 5.
5. Если твой партнер при своем очередном ходу взял l предметов ($l \leqslant p$), то возьми $(p+1)-l$ предметов. Переходи к ук. 6.
6. Проверь, есть ли неотобранные предметы. Если неотобранных предметов не осталось, переходи к ук. 7, иначе — к ук. 5.
7. Прекрати игру — ты выиграл.
8. Выигрывает второй игрок.

Описанную игру обычно называют игрой БАШЕ в честь французского математика, поэта и переводчика Клода Гаспара Баше де Мезириака (1581—1638), описавшего игру в книге «Занимательные и приятные числовые задачи», опубликованной в 1612 году.

Теперь несколько задач.

Задачи

13.2. Исходя из приведенного выше алгоритма для обобщенной игры БАШЕ (пример 2), составьте беспроигрышный алгоритм для случая $n=11$, $p=2$.

13.3. На столе n предметов ($n>2$). Двое игроков поочередно берут предметы со стола, причем за один раз разрешается брать 1 или 2 предмета. Выигрывает тот, кто берет последний предмет. Существует ли алгоритм, позволяющий по любой конкретной игре (по любому конкретному значению n) определять, может ли игрок, начинающий игру, выиграть в ней? (Описанная здесь игра называется **игрой-двойником** для игры из примера 1.)

13.4. Опишите игру-двойник для игры из примера 2. Постройте алгоритм, позволяющий по любой конкретной игре определять, может ли игрок, начинающий игру, выиграть в ней.

Пример 3. К началу игры имеется k групп предметов (например, камешков, палочек, орехов и т. д.). До первого хода

первая группа содержит n_1 предметов, вторая — n_2, в третьей — n_3 предметов и т. д., в k-й группе — n_k предметов. Играют двое, ходят поочередно. За очередной ход каждый из играющих может взять из одной группы любое количество предметов (даже все имеющиеся в этой группе предметы). Победителем считается тот, кто сумеет взять все оставшиеся предметы. Существует ли алгоритм, позволяющий по любой конкретной игре определять, может ли игрок, начинающий ее, выиграть?

(Эта игра была известна еще в древности. Современное название этой игры — «Ним», от английского слова NIM — «стянуть», «взять».)

Мы рассмотрим здесь лишь одну конкретную игру «Ним», когда $k=3$; n_1, n_2, n_3 соответственно равны 3, 4, 5.

Имеется хорошо разработанная теория игры «Ним» (Квант.— 1971.— № 2.— С. 22—23). Мы опишем ее кратко для указанной конкретной игры.

Прежде всего вводится понятие «проигрышной позиции». Позиция называется проигрышной, если игрок, делающий ход из этой позиции, проигрывает. Примером проигрышной позиции может быть позиция, когда перед очередным ходом осталось две группы предметов, содержащих по одному предмету: $n_1=1$, $n_2=1$. Игрок, делающий ход из этой позиции, проигрывает.

Предлагаем Вам з а д а ч у.

13.5. Докажите (перебором всех возможных вариантов хода), что позиции: а) $n_1=2$, $n_2=2$; б) $n_1=1$, $n_2=2$, $n_3=3$ проигрышные.

Практика привела к следующим двум предположениям (которые в теории доказываются): нельзя перейти одним ходом из одной проигрышной позиции к другой, и каждая проигрышная позиция после любого хода из нее перестает быть проигрышной.

Естественно возникает вопрос: как распознавать проигрышные позиции? Оказывается, что если числа, составляющие позицию, записаны в десятичной системе счисления, найти какую-нибудь закономерность, отличающую проигрышные позиции от выигрышных, чрезвычайно сложно. Но если эти числа записать в двоичной системе счисления, то можно обнаружить интересную закономерность. Вот еще одно интересное применение двоичной системы счисления!

Оказывается, что каждая проигрышная позиция записывается двоичными числами так, что в каждом разряде сумма единиц есть число четное. Так, например, решая задачу 13.5, Вы убедились в том, что позиция (1, 2, 3) проигрышная. Записав эту позицию в столбик (для наглядности) и переводя числа в двоичную систему счисления

1	1
2	10
3	11,

мы видим, что в каждом разряде двоичной записи число единиц — четное (равно 2).

Обратимся теперь к рассматриваемой нами конкретной игре. Исходная позиция:

3	11
4	100
5	101.

Эта позиция выигрышная, так как имеется столбец (второй) с нечетной суммой цифр.

Будем вести игру так, чтобы в результате каждого нашего очередного хода сумма цифр каждого разряда была четной, т. е. чтобы вынудить нашего соперника сделать свой очередной ход из проигрышной позиции. Как же это сделать? В нашем примере имеется единственный столбец с нечетной суммой цифр. Заменим единицу, которая нам мешает, нулем, а остальные цифры, расположенные правее нее, подберем так, чтобы ни в одном столбце сумма цифр не была нечетной. С этой целью достаточно выбрать в качестве первого двоичного числа единицу. Иначе говоря, мы должны взять два предмета из первой группы.

После этого останется позиция:

1	1
4	100
5	101,

в которой во всех столбцах (двоичной записи) четное число единиц (2), т. е. проигрышная позиция.

После того как партнер взял, например, 3 предмета из третьей группы, получаем позицию:

1	1
4	100
2	10

(впрочем, при любом ходу партнера мы получим хотя бы один столбец с нечетной суммой цифр).

Найдем самый левый столбец с нечетной суммой цифр. Опять заменим «мешающую» нам единицу нулем, а остальные цифры, расположенные правее нее, подберем так, чтобы ни в одном столбце сумма цифр не была нечетной. С этой целью мы должны взять один предмет со второй группы:

1	1
3	11
2	10

Допустим теперь, что партнер в своем очередном ходу взял 3 предмета из второй группы. После этого имеем:

1	1
0	0
2	10

Взяв один предмет из третьей группы, мы получим:

$$\begin{array}{|c} 1 \\ 0 \\ 1 \end{array} \quad \begin{array}{c} 1 \\ 0 \\ 1. \end{array}$$

Теперь у партнера есть две возможности: либо взять один предмет из первой группы, либо один предмет из третьей группы. Любая из них обеспечивает нам выигрыш.

Итак, выигрышными или безопасными для начинающего игру позициями являются такие, при которых сумма цифр хотя бы в одном разряде нечетна, проигрывает он в позициях (проигрышных или опасных), когда сумма цифр в каждом разряде четна. Действительно, если A находится в безопасном положении, он может взять несколько предметов так, чтобы B оказался в опасном положении. Для этого A должен выбрать такую группу, число предметов которой (в двоичной записи) имеет единицу в том столбце, где стоит первая нечетная цифра суммы, и из этой группы взять столько предметов, чтобы в каждом столбце сумма цифр стала четной. Тогда, какое бы число предметов ни взял игрок B, он изменит в одном из чисел хотя бы одну цифру, следовательно, изменится четность в соответствующем разряде. Таким образом, игрок B вернет игрока A снова в безопасное положение. Игра закончится выигрышем игрока A, если он приведет B в положение, когда все группы пусты (позиция (0, 0, 0) проигрышная).

В соответствии с этим сформулируем алгоритм в виде следующего предписания:

1. Представь количество предметов в каждой группе в двоичной системе счисления. Запиши эти числа «столбиком». Переходи к ук. 2.

2. Найди сумму цифр в каждом разряде. Если сумма цифр в каждом разряде четна, переходи к ук. 7, иначе — к ук. 3.

3. В каждый свой очередной ход найди самый левый столбец с нечетной суммой цифр. Переходи к ук. 4.

4. Возьми в очередном ходу столько предметов из той группы, которая дает нечетную сумму цифр в самом левом столбце, чтобы после этого сумма цифр в каждом разряде стала четной. Переходи к ук. 5.

5. Проверь, остались ли невзятые предметы. Если да, переходи к ук. 3, иначе — к ук. 6.

6. Прекрати игру — ты выиграл.

7. Выигрывает второй игрок.

14. Оптимальные планы

Среди часто встречающихся на практике экономических задач важное место занимает **транспортная задача.** Приведем алгоритмическую постановку этой задачи.

Имеется m пунктов производства: A_1, A_2, ..., A_m, в которых производится однородный продукт (например, сахар, соль и т. д.)

в количествах $a_1, a_2, \ldots, a_m$ единиц соответственно. Имеется n пунктов потребления: $B_1, B_2, \ldots, B_n$, которым данный продукт требуется в количествах $b_1, b_2, \ldots, b_n$ единиц соответственно, причем $a_1 + a_2 + \ldots + a_m = b_1 + b_2 + \ldots + b_n$, (т. е. весь производимый продукт распределяется, спрос каждого потребителя полностью удовлетворяется). Стоимость перевозки единицы продукции из каждого пункта производства в каждый пункт потребления известна: c_{ij} — стоимость перевозки единицы продукции из i-го пункта производства в j-й пункт потребления (c_{ij} — натуральные числа или 0).

Запишем условия транспортной задачи в виде таблицы 19.

Таблица 19

c_{11}	c_{21}	...	c_{m-11}	c_{m1}	b_1
c_{12}	c_{22}	...	c_{m-12}	c_{m2}	b_2
...	...	...	...	...	...
c_{1n-1}	c_{2n-1}	...	c_{m-1n-1}	c_{mn-1}	b_{n-1}
c_{1n}	c_{2n}	...	c_{m-1n}	c_{mn}	b_n
a_1	a_2	...	a_{m-1}	a_m	

Планом транспортной задачи (или планом перевозок) назовем таблицу 20 из $m \cdot n$ чисел x_{ij}, где x_{ij} обозначает количество единиц груза, который нужно доставить из i-го пункта производства в j-й пункт потребления ($i = 1, 2, \ldots, m$; $j = 1, 2, \ldots, n$).

Таблица 20

x_{11}	x_{12}	...	x_{1n}
x_{21}	x_{22}	...	x_{2n}
...	...	...	...
x_{m1}	x_{m_2}	...	x_{mn}

По смыслу задачи перевозки x_{ij} должны удовлетворять следующим условиям: 1) числа $x_{ij} \geqslant 0$ и 2) числа x_{ij} удовлетворяют системе (1) $m + n$ линейных уравнений с $m \cdot n$ неизвестными:

$$\begin{cases} x_{11} + x_{12} + \ldots + x_{1n} = a_1 \\ x_{21} + x_{22} + \ldots + x_{2n} = a_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_{m1} + x_{m2} + \ldots + x_{mn} = a_m \\ x_{11} + x_{21} + \ldots + x_{m1} = b_1 \\ x_{12} + x_{22} + \ldots + x_{m2} = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_{1n} + x_{2n} + \ldots + x_{mn} = b_n \end{cases} \qquad (1)$$

(первые m уравнений — условия вывоза всех грузов из пунктов отправления (производства), последние n уравнений — условия доставки всем потребителям необходимого количества грузов).

Определим два важных понятия.

Определение 1. План перевозок называется **допустимым,** если числа $x_{ij} \geqslant 0$ и удовлетворяют системе уравнений (1).

Оценим общую стоимость перевозок. Стоимость перевозки x_{ij} единиц груза из i-го пункта производства в j-й пункт потребления будет равна $c_{ij} \cdot x_{ij}$. Тогда общая стоимость перевозки всей продукции будет

$$S = c_{11}x_{11} + c_{12}x_{12} + \ldots + c_{1n}x_{1n} + \\ + c_{21}x_{21} + c_{22}x_{22} + \ldots + c_{2n}x_{2n} + \ldots + \\ + c_{m1}x_{m1} + c_{m2}x_{m2} + \ldots + c_{mn}x_{mn} \qquad (2)$$

Видим, что S — линейная функция от переменных x_{11}, x_{12}, ..., x_{1n}, ..., x_{21}, x_{22}, ..., x_{2n}, ..., x_{m1}, x_{m2}, ..., x_{mn}.

Определение 2. Допустимый план перевозок, обращающий в минимум линейную функцию S (транспортные расходы), называется **оптимальным.**

Поставим задачу: построить алгоритм, позволяющий по любой заданной таблице 19 находить оптимальный план перевозок, т. е. таблицу 20 из $m \cdot n$ неотрицательных чисел x_{ij}, удовлетворяющих системе уравнений (1) и обращающих в минимум линейную функцию (2).

Для построения требуемого алгоритма предварительно рассмотрим решение конкретной транспортной задачи ($m = 3$, $n = 4$), исходные данные которой помещены в таблице 21, и проанализируем ее решение.

Решение задачи начнем с составления таблицы, в которой указан какой-нибудь первоначальный план перевозок всей продукции. При построении первоначального плана будем заполнять последовательно клетки с минимальной во всей таблице стоимостью c_{ij} (просматриваем таблицу по строкам слева направо) перевозками, при этом величины x_{ij} полагаем равными минимуму из остатка груза в i-м пункте производства и недостающего груза в j-м пункте потребления. Процесс построения первоначального плана приведен в таблицах 22—27 и заключается в следующем.

Таблица 21

1	2	4	3	6
4	3	6	5	8
2	7	5	3	10
4	6	8	6	

Таблица 22

4 1	2	4	3	~~6~~ 2
4	3	6	5	8
2	7	5	3	10
~~4~~				

0

Таблица 23

4 1	2 2	4	3	~~6~~ ~~20~~
4	3	6	5	8
2	7	5	3	10
~~4~~	~~6~~	8	6	

0 4

Таблица 24

4 1	2 2	4	3	~~6~~ ~~20~~
4	4 3	6	5	~~8~~ 4
2	7	5	3	10
~~4~~ 1	~~6~~	8	6	

0 ~~4~~ 0

Таблица 25

4 1	2 2	4	3	~~6~~ ~~20~~
4	4 3	6	5	~~8~~ 4
2	7	5	6 3	~~10~~ 4
~~4~~	~~6~~	8	~~6~~	

0 ~~4~~ 0 0

Таблица 26

4	2			~~6~~ ~~20~~
	4			~~8~~ 4
		4	6	~~10~~ ~~4~~ 0
~~4~~	~~6~~	~~8~~	~~6~~	

0 ~~4~~ 0 4 0

Таблица 27

4 1	2 2	4	3	~~6~~ ~~20~~
4	4 3	4 6	5	~~8~~ ~~4~~ 0
2	7	4 5	6 3	~~10~~ ~~4~~ 0
~~4~~	~~6~~	~~8~~	~~6~~	

0 ~~4~~ ~~4~~ 0 0 0

Таблица 28

4		2		6
	6	2		8
		4	6	10
4	6	8	6	

;—

Таблица 29

4 1	2	2 4	3	6
4	6 3	2 6	5	8
-2	-7	-4 5	-6 3	-10
4	6	8	6	

Таблица 30

4	0	2	0
0	6	2	0
0	0	4	6

В таблице 21 выбираем клетку (1,1) с наименьшей стоимостью $c_{11} = 1$ и записываем в нее перевозку, равную $x_{11} = \min(6,4) = 4$. Пересчитываем записи продукции пунктов производства и потребности пунктов потребления: запас первого пункта производства теперь равен $6 - 4 = 2$, а потребность первого пункта равна $4 - 4 = 0$. Получили таблицу 22.

Поскольку потребность первого пункта полностью удовлетво-

рена, будем рассматривать далее таблицу без свободных клеток первого столбца и найдем в ней клетку (1,2) с наименьшей стоимостью $c_{12}=2$. Соответствующая перевозка равна $x_{12}=\min(2,6)=2$. Записываем это число в таблицу и пересчитываем запас продукции в первом пункте производства, $2-2=0$, и потребность во втором пункте, $6-2=4$. Получили таблицу 23, в которой исключаются из рассмотрения свободные клетки первой строки и первого столбца.

Далее клетки заполняем в следующем порядке: (2,2), (3,4), (3,3), (2,3).

Первоначальный план, записанный в таблице 27, является допустимым — при этом плане распределения перевозок вся продукция пунктов производства используется (сумма перевозок по каждой горизонтали равна количеству произведенного продукта, указанному справа), а потребности всех пунктов потребления удовлетворяются (сумма перевозок по каждой вертикали равна требуемому количеству продукта, указанному внизу).

Таблица 27 имеет еще одну особенность: ради удобства в ней совмещены две таблицы — таблица исходных данных и таблица перевозок. Величина стоимости c_{ij} записана в верхнем правом углу клетки, стоящей на пересечении i-й строки и j-го столбца ($i=1, 2, \ldots, m$; $j=1, 2, \ldots, n$) с той целью, чтобы оставить место для записи величины перевозки x_{ij}. В дальнейшем мы также будем совмещать в одну таблицы данных и перевозок.

При плане перевозок, записанном в таблице 27, общие транспортные расходы, выраженные в рублях, составят: $S_1=1\cdot 4+2\cdot 2+3\cdot 4+6\cdot 4+5\cdot 4+3\cdot 6=82$ (р.).

Следующим этапом решения задачи является улучшение полученного плана. В плане перевозок, полученном в таблице 27, не все клетки заполнены. Например, не заполнены клетки (1,3), (1, 4), (2, 1) и т. д. Это означает, что соответствующие им перевозки равны нулю, т. е. $x_{13}=0$, $x_{14}=0$, $x_{21}=0$ и т. д. Число заполненных клеток оказалось равным числу уравнений системы (1) (при $m=3$, $n=4$) без единицы, т. е. равно $m+n-1$. Транспортная задача, для которой число положительных перевозок в первоначальном допустимом плане и в последующих планах равно $m+n-1$, называется **невырожденной.** Мы рассматриваем только невырожденную транспортную задачу.

Посмотрим теперь, как повлияет на транспортные расходы любое возможное перераспределение перевозок. Изменить (увеличить или уменьшить) перевозки в клетках (1,1) и (3,4) нельзя, так как при этом нарушится условие равенства перевозок в этих столбцах соответствующим потребностям. Раз так, то нельзя менять перевозки в клетках (1,2) и (3,3), так как это нарушило бы условие равенства объемов производства по первой и третьей строке суммам перевозок по этим строкам. По тем же причинам нельзя менять перевозки в клетках (2,2) и (2,3). Значит, изменить план перевозок можно только, записав перевозку в какую-либо

свободную клетку. Отыщем первую свободную клетку в таблице 27. Это будет клетка (1,3). Посмотрим, что произойдет, если записать перевозку, равную 1, в эту клетку. Чтобы при этом не нарушить условий, задаваемых системой уравнений (1), нужно уменьшить на 1 перевозку в клетках (1,2) и (2,3). Уменьшив на 1 перевозку в (1,2), мы должны увеличить на 1 перевозку в (2,2), чтобы восстановить баланс по второму столбцу, нарушенный уменьшением поставки в (2,2). Перераспределяя перевозки, мы прошли по четырем клеткам. Путь нашего движения образовал так называемый цикл, показанный на рисунке 18.

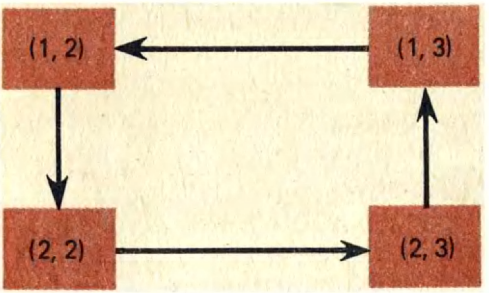


Рис. 18

Определение. Циклом называют наборы клеток вида (i_1, j_1), (i_1, j_2), (i_2, j_2), ..., $(i_k j_1)$ или (i_1, j_1), (i_2, j_1), (i_2, j_2), (i_3, j_2), ..., (i_1, j_k), такие, что каждая пара соседних клеток расположена либо в одном столбце таблицы, либо в одной строке, причем никакие три клетки цикла не лежат в одной строке (в одном столбце), а последняя клетка лежит в одной строке (в одном столбце) с первой.

Некоторые разновидности циклов показаны на рисунке 19.

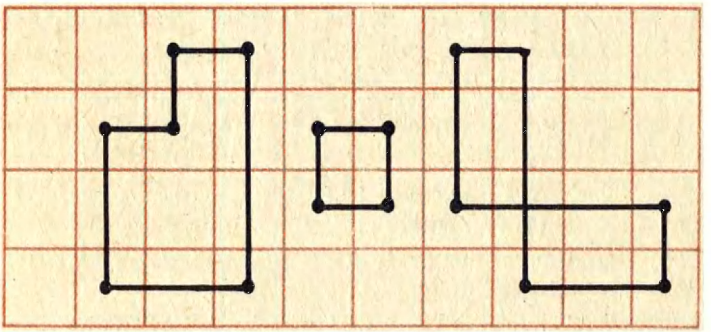


Рис. 19

Циклы представляют собой замкнутые многоугольники, одной из вершин которых является клетка со стоимостью, не обведенная квадратом (свободная клетка), а в остальных вершинах должны быть клетки со стоимостями, обведенные квадратами (заполненные клетки). В вершинах цикла записываются соответствующие стоимости, при этом стоимость, не обведенная квадратом, записывается со знаком плюс, а для остальных стоимостей знаки чередуются, как на рисунке 20.

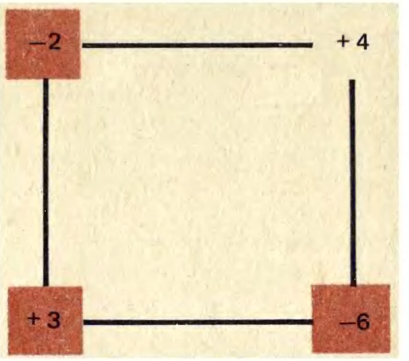


Рис. 20

Назовем величину, представляющую собой алгебраическую сумму стоимости c_{ij} в вершинах цикла **индексом свободной клетки.** В нашем примере индекс свободной клетки (1,3) $K_{13} = 4 - 2 + 3 - 6 = -1$. Индекс свободной клетки показывает, на сколько изменятся транспортные расходы при записи перевозки, равной 1, в эту клетку. Действительно, записав поставку, равную 1, в (1,3), мы увеличили транспортные расходы на 4 р. Уменьшив на 1 перевозку в (1,2), мы уменьшим транспортные расходы на 2 р., увеличив перевозку в (2.2), мы увеличили транспортные расходы на 3 р. и, наконец, уменьшив перевозку в (2,3), мы уменьшили транспортные расходы на 6 р. Таким образом, перераспределение единицы товара по циклу приведет к уменьшению транспортных расходов на 1 р. ($4 - 2 + 3 - 6 = -1 = K_{13}$). Если запись в (1,3) перевозки, равной 1, уменьшает транспортные расходы, следует записать сюда как можно большую поставку, соблюдая требования допустимости плана. Величина новой перевозки должна быть равна величине наименьшей перевозки в отрицательных вершинах цикла, так как если взять большую величину, то значение в том месте, где была наименьшая перевозка, после перераспределения станет отрицательным, что недопустимо. В нашем цикле есть две отрицательные вершины, соответствующие клеткам (1,2) и (2,3). Наименьшая по величине перевозка находится в (1,2) и равна 2. Прибавляем 2 к перевозкам x_{ij} в (1,3) и (2,2) (в положительных вершинах цикла) и вычитаем 2 из перевозок x_{ij} в (1,2) и (2,3). В результате получаем таблицу 28. План перевозок в таблице 28 допустимый, причем он лучше прежнего на $1 \cdot 2 = 2$ (р.), т. е. транспортные расходы по этому плану равны $S = 80$ (р.).

Выясним, нельзя ли улучшить план перевозок, приведенный в таблице 28. Для этого посмотрим, что дает перераспределение перевозок в свободные клетки. В таблице 28 находим первую свободную клетку — (1,2), строим для нее цикл (рис. 21) и определяем ее индекс: $K_{12} = 2 - 4 + 6 - 3 = 1$. Он показывает, что записывать в клетку (1,2) перевозку невыгодно: это приведет к увеличению транспортных расходов.

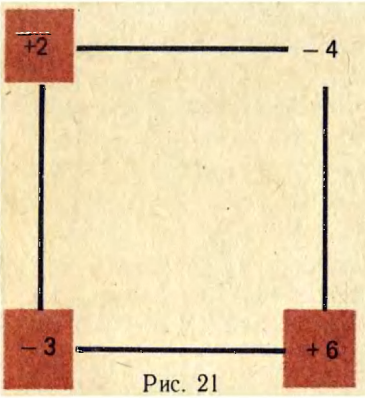


Рис. 21

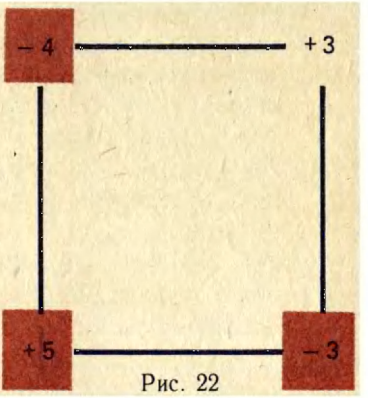


Рис. 22

Следующая свободная клетка — (1,4). Цикл для нее представлен на рисунке 22.

$K_{14} = -4+3-3+5=1$. Так как $K_{14} > 0$, то записывать перевозку в клетку (1.4) не будем.

Вы можете теперь самостоятельно проверить оставшиеся свободные клетки.

Задача 14.1. Составьте для оставшихся свободных клеток (2.1), (2,4), (3,1), (3,2) соответствующие им циклы и вычислите их индексы.

Отсутствие отрицательных индексов свидетельствует о том, что нельзя построить циклы, перемещение перевозок по которым уменьшит транспортные расходы. Значит, план перевозок, представленный таблицей 28, является оптимальным, и мы решили задачу.

Но у Вас может возникнуть вопрос: как построить цикл для свободной клетки? Для ответа на этот вопрос проанализируем, как мы получали циклы, изображенные на рисунках 21 и 22. Цикл, изображенный на рисунке 21, образует клетки подтаблицы таблицы 28, которая получается из нее удалением 1-го столбца (в нем содержится одна заполненная клетка), 4-го столбца (в нем также содержится одна заполненная клетка) и 3-й строки (в ней также содержится одна заполненная клетка, если заполненную клетку вычеркнутого 4-го столбца не принимать во внимание).

Аналогично получается цикл для клетки (1,4): проанализируем все строки таблицы 28 и вычеркнем те из них, которые содержат только одну заполненную клетку,— ни одну из строк вычеркнуть нельзя, так как все они содержат две или более заполненных клеток (клетка (1,4), для которой мы строим цикл, также считается заполненной). Затем просматриваем столбцы. Можно вычеркнуть первый и второй столбцы, так как они содержат по одной заполненной клетке. Затем опять просматриваем строки и вычеркиваем вторую строку, так как она уже содержит одну заполненную клетку. Процесс вычеркивания приводит, таким образом, к вычеркиванию первого и второго столбца, а также второй строки.

Оставшиеся заполненные клетки образуют цикл. В таблице 29 приведен процесс построения цикла для клетки (2,1).

Искомый оптимальный план перевозок представлен в таблице 30: он получается из таблицы 28 вычеркиванием четвертой строки — строки потребностей, 5-го столбца — столбца емкостей, вычеркиванием чисел, стоящих в верхних правых углах клеток таблицы (величин стоимостей) и записью в незаполненных клетках таблицы перевозок, равных 0.

При решении транспортной задачи мы усматриваем следующие этапы: 1) нахождение первоначального допустимого плана перевозок; 2) проверка найденного плана на оптимальность; 3) переход к лучшему допустимому плану. В соответствии с этим в алгоритме решения транспортной задачи выделим два этапа: I — построение первоначального допустимого плана перевозок и II — получение оптимального плана перевозок.

Прежде чем переходить к формулировке алгоритма, рассмотрим несколько задач.

Задачи

14.2. Являются ли планы перевозок, представленные в таблицах 31 и 32, допустимыми?

Таблица 31

7		8	15
1	10	9	20
8	9	18	

Таблица 32

11	14			25
	5	10		15
		5	5	10
11	19	15	5	

14.3. Найдите транспортные расходы для плана перевозок, представленного в таблице 32.

14.4. Составьте первоначальный допустимый план перевозок для транспортной задачи, условия которой записаны в таблице 33.

Таблица 33

				80
				100
				50
30	70	90	40	

Таблица 34

30	50			80
	20	80		100
		10	40	50
30	70	90	40	

14.5. Постройте циклы для свободных клеток (1,4) и (3,1) в плане перевозок, представленном в таблице 34. Найдите индексы K_{14} и K_{31}.

14.6. Проверьте, является ли план перевозок в таблице 34 оптимальным. Если нет, получите оптимальный план.

В соответствии с выделенными выше этапами решения транспортной задачи мы можем сформулировать алгоритм ее решения в виде следующего словесного предписания.

1. Исходя из любой заданной таблицы 19, отыскиваем первоначальный допустимый план. С этой целью просматриваем исходную таблицу по строкам слева направо и заполняем последовательно клетки перевозками с минимальной во всей таблице стоимостью c_{ij}, полагая x_{ij} равными минимуму из остатка груза в i-м пункте производства и недостающего груза в j-м пункте потребления.

Найденный план испытываем на оптимальность, для чего: а) строим цикл для первой свободной клетки полученной таблицы перевозок и вычисляем ее индекс; б) если этот индекс неотрицателен, то ищем следующую свободную клетку и проделываем для нее указанную в пункте а) операцию; в противном случае отыскиваем наименьшую перевозку в отрицательной полуцепи найденного цикла, делаем сдвиг по нему на это число и находим новый допустимый план.

2. Вновь полученный допустимый план испытываем на оптимальность, т. е. повторяем указанные в пунктах а) и б) операции. Так поступаем до тех пор, пока план не будет оптимальным.

Теперь обратим внимание на то, что встречающиеся в предписании указания: построить цикл для свободной клетки, вычислить индекс свободной клетки, сделать сдвиг по циклу на число — сами могут быть записаны в виде упорядоченной системы более простых указаний.

В связи с этим мы рассмотрим подробно два алгоритма: A_1 — алгоритм нахождения первоначального допустимого плана перевозок, A_2 — алгоритм приведения первоначального допустимого плана к оптимальному. Тогда предписание «исходя из произвольных данных, т. е. любой заданной таблицы 19, сначала применить алгоритм A_1, а затем к результату его работы — алгоритм A_2» составляет новый алгоритм — композицию двух данных алгоритмов $A_1 \circ A_2$, являющийся алгоритмом решения транспортной задачи.

Алгоритм A_1 задается предписанием:

1. Составь таблицу, содержащую $(m+1)$ строку и $(n+1)$ столбец, вписывая в верхний правый угол клетки (i, j), стоящей на пересечении i-й строки и j-го столбца, величину стоимости $c_{ij} (i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n)$. В клетки $(i, n+1)$ впиши числа $a_i (i = 1, 2, \ldots, m)$, в клетки $(m+1, j)$ впиши числа $b_j (j = 1, 2, \ldots, n)$. Переходи к ук. 2.

2. Среди клеток, не содержащих перевозок, найди клетку с наименьшей стоимостью, встретившуюся первой при просмотре таблицы по строкам — $c_{i_0 j_0}$. Переходи к ук. 3.

3. Найди min $(a_{i_0} b_{j_0})$. Переходи к ук. 4.

4. Сравни min (a_{i_0}, b_{j_0}) с нулем. Если min $(a_{i_0}, b_{j_0}) \neq 0$, то переходи к ук. 5, иначе клетка (i_0, j_0) считается заполненной — переходи к ук. 7.

5. Положи $x_{i_0 j_0} = \min (a_{i_0}, b_{j_0})$. Переходи к ук. 6.

6. Исправь таблицу, записав вместо a_{i_0} величину $a_{i_0} - \min (a_{i_0}, b_{j_0})$, а вместо b_{j_0} величину $b_{j_0} - \min (a_{i_0}, b_{j_0})$. Переходи к ук. 7.

7. Проверь, есть ли $a_i \neq 0$ или $b_j \neq 0$. Если да, то рассматривай таблицу без клеток, у которых $a_i = 0$ или $b_j = 0$. Переходи к ук. 2. Если таких клеток нет, то переходи к ук. 8.

8. Процесс вычисления остановить.

Первоначальный допустимый план найден.

Алгоритм A_2 задается предписанием:

1. Начни просмотр таблицы перевозок, двигаясь по строкам. Найди первую свободную клетку — (i_0, j_0). Переходи к ук. 2.

2. Построй цикл для клетки (i_0, j_0), для чего:

2.1. Проверь, в каждой ли строке и в каждом ли столбце таблицы содержится ровно две заполненные клетки (включая клетку (i_0, j_0)). Если да, то переходи к ук. 2.6, иначе — к ук. 2.2.

2.2. Просмотри все строки таблицы и вычеркни те из них, которые содержат только одну заполненную клетку. Переходи к ук. 2.3.

2.3. Рассматривай таблицу без вычеркнутых строк. Переходи к ук. 2.4.

2.4. Просмотри столбцы получившейся таблицы и вычеркни те из них, которые содержат только одну заполненную клетку. Переходи к ук. 2.5.

2.5. Рассматривай таблицу без вычеркнутых столбцов. Переходи к ук. 2.1.

2.6. Оставшиеся невычеркнутыми заполненные клетки таблицы образуют цикл для клетки (i_0, j_0). Переходи к ук. 3.

3. Клетки, входящие в цикл, пометь знаками + и — поочередно в направлении против часовой стрелки, начиная со свободной клетки. Переходи к ук. 4.

4. Посчитай сумму стоимостей l_1 для клеток цикла, помеченных знаком +. Переходи к ук. 5.

5. Посчитай сумму стоимостей l_2 для клеток цикла, помеченных знаком —. Переходи к ук. 6.

6. Найди $K_{i_0 j_0} = l_1 - l_2$. Сравни $K_{i_0 j_0}$ с нулем. Если $K_{i_0 j_0} < 0$, то переходи к ук. 8, иначе — к ук. 7.

7. Ищи следующую клетку, свободную и не просмотренную ранее. Если она есть, то переходи к ук. 2, иначе — к ук. 11.

8. Найди наименьшую перевозку θ в клетках цикла, помеченных знаком —. Переходи к ук. 9.

9. Положи $x_{ij} = \theta$. Переходи к ук. 10.

10. Из перевозок в клетках цикла, помеченных знаком — выч-

ти 0, а к перевозкам в клетках цикла, помеченных знаком $+$, прибавь 0. Клетку, перевозка в которой равна 0, считай свободной. Переходи к ук. 7.

11. Запиши в незаполненных клетках таблицы числа 0, сотри $(n+1)$-й столбец, $(m+1)$-ю строку, а также числа, стоящие в верхних правых углах клеток таблицы. Переходи к ук. 12.

12. Процесс решения остановить — оптимальный план найден.

При больших значениях m и n, разумеется, удобно использовать ЭВМ для решения транспортной задачи. Алгоритм $A_1 \circ A_2$ решения этой задачи может быть переведен в программу для ЭВМ.

Приведем формулировки еще двух важных экономических задач, решаемых аналогично транспортной.

Задача распределения кадров. Имеется m групп специалистов $A_1, A_2, \ldots, A_m$ в количествах $a_1, a_2, \ldots, a_m$ человек соответственно. Имеется n видов работ $B_1, B_2, \ldots, B_n$, проводимых на $b_1, b_2, \ldots, b_n$ рабочих местах, причем общее число специалистов равно общему числу рабочих мест, т. е. $a_1 + a_2 + \ldots + a_m = b_1 + b_2 + \ldots + b_n$. Каждая из групп $A_1, A_2, \ldots, A_m$ объединяет специалистов, обладающих одинаковой производительностью при выполнении работ $B_1, B_2, \ldots, B_n$. Производительность группы специалистов A_i при выполнении работы B_j известна; α_{ij} $(i = 1, 2, \ldots, m, j = 1, 2, \ldots, n)$. Найти оптимальное распределение специалистов по видам работ, т. е. такое распределение, при котором общий показатель производительности был бы максимальным.

Поставленная задача сводится к транспортной, состоящей в нахождении наименьшего значения линейной функции $\varphi = -(\alpha_{11}x_{11} + \alpha_{12}x_{12} + \ldots + \alpha_{1n}x_{1n} + \alpha_{21}x_{21} + \ldots + \alpha_{2n}x_{2n} + \ldots + \alpha_{m1}x_{m1} + \ldots + \alpha_{mn} + x_{mn})$.

Задача размещения сельскохозяйственных культур по хозяйствам. Имеется m видов сельскохозяйственных культур и n хозяйств, где их можно выращивать. Из-за различия условий (почв, расположения и т. д.) доход, получаемый с 1 га посева одной и той же культуры, в разных хозяйствах неодинаков. Обозначим его для i-й культуры и j-го хозяйства через c_{ij}. Общие площади посева культур a_i и площади пашни в хозяйствах b_j известны. Требуется составить такой план размещения сельскохозяйственных культур по хозяйствам, чтобы общий доход от их производства был максимальным.

И эта задача аналогичным образом сводится к транспортной.

Как видите, транспортная задача представляет класс однотипных задач.

* * *

Дорогой читатель!

Мы завершили наши беседы об алгоритмах и их приложениях, хотя, конечно, не исчерпали ими эту большую тему.

Надеемся, что прочитанная, а точнее, проработанная Вами книга повысила Ваш интерес к такому, на первый взгляд простому, а в действительности сложному и весьма важному в современной науке понятию, каким является алгоритм.

Вы ознакомились не только с интуитивным понятием алгоритма, но и с двумя его математическими уточнениями, приближающими Вас к пониманию возможности кодирования информации словами некоторого алфавита и ее преобразования в реальных ЭВМ. Этим самым книга расширяет Ваш математический кругозор и готовит Вас к усвоению основ информатики и вычислительной техники.

Успехов Вам!

Авторы

3.2. 1. Если $x \geqslant 0$, то перейти к ук. 2, иначе — к ук. 3.
2. $|x| = x$. Перейти к ук. 4.
3. $|x| = -x$. Перейти к ук. 4.
4. Процесс окончен.

(Примечание. Возможны и другие варианты предписаний, задающие этот же алгоритм.)

3.3. 1. Если $a \neq 0$ и $a \neq 1$, то перейти к ук. 2, иначе — к ук. 3.
2. $x = \dfrac{a+1}{a}$. Перейти к ук. 7.
3. Если $a = 0$, то перейти к ук. 4, иначе — к ук. 5.
4. Уравнение не имеет корней. Перейти к ук. 7.
5. $a = 1$. Перейти к ук. 6.
6. Любое число является корнем уравнения. Перейти к ук. 7.
7. Процесс окончен.

3.4.

а) 1. С помощью циркуля построить окружность с центром в конце A данного отрезка AB радиусом, большим половины отрезка AB. Перейти к ук. 2.
2. С помощью циркуля построить окружность с центром в конце B отрезка AB тем же радиусом. Перейти к ук. 3.
3. С помощью линейки соединить точки пересечения двух окружностей отрезком. Перейти к ук. 4.
4. Обозначить через O точку пересечения полученного отрезка с отрезком AB. Перейти к ук. 5.
5. O — середина отрезка AB. Процесс окончен.

(Примечание. Этот же алгоритм применяется и к решению задачи построения серединного перпендикуляра: отрезок, соединяющий точки пересечения двух окружностей,— серединный перпендикуляр к отрезку AB.)

в) Пусть дан ΔABC.
1. Построить серединный перпендикуляр к отрезку AB (с помощью алгоритма 3.4, а). Перейти к ук. 2.
2 Построить серединный перпендикуляр к отрезку BC (или AC). Перейти к ук. 3.
3. С помощью линейки найти точку пересечения O двух серединных перпендикуляров. Перейти к ук. 4.
4. С помощью циркуля построить окружность с центром O радиусом OA (или OB, или OC) Перейти к ук. 5.
5. Процесс окончен.

3.5.

а) 1. Если $a > 0$, то перейти к ук. 2, иначе — к ук. 3.
2. $x < \dfrac{10}{a}$. Перейти к ук. 6.
3 Если $a = 0$, то перейти к ук. 4, иначе — к ук. 6.
4. Любое число — решение неравенства. Перейти к ук. 6.
5. $x > \dfrac{10}{a}$. Перейти к ук. 6
6. Процесс окончен.

б) 1. Если $a > 0$, то перейти к ук. 2, иначе — к ук. 3.
2. $x < b/a$. Перейти к ук. 9.
3. Если $a = 0$, то перейти к ук. 4, иначе — к ук. 8.
4 Если $b = 0$, то перейти к ук. 5, иначе — к ук. 6.
5. Неравенство не имеет решений. Перейти к ук. 9.
6. Если $b > 0$, то перейти к ук. 7, иначе — к ук. 5.
7. Любое число — решение неравенства. Перейти к ук. 9.
8. $x > b/a$. Перейти к ук. 9.
9. Процесс окончен.

3.6. 1. Если $\dfrac{a}{a_1} \neq \dfrac{b}{b_1}$, то перейти к ук. 2, иначе — к ук. 3.

2. Единственное решение $(x, y)=\left(\frac{b_1c-bc_1}{ab_1-a_1b}, \frac{ac_1-a_1c}{ab_1-a_1b}\right)$. Перейти к ук. 6.

3. Если $\frac{b}{b_1}=\frac{c}{c_1}$, то перейти к ук. 4, иначе — к ук. 5.

4. Система имеет бесконечное множество решений (x, y): $\{(x, y)|ax+by=c\}$. Перейти к ук. 6.

5. Система не имеет решений. Перейти к ук. 6.

6. Процесс окончен.

3.7.

б) 1. Если $x<1$, то перейти к ук. 2, иначе — к ук. 3.

2. $y=x-1$. Перейти к ук. 6.

3. Если $1\leqslant x<5$, то перейти к ук. 4, иначе — к ук. 5.

4. $y=x+1$. Перейти к ук. 6.

5. $y=x^2-5$. Перейти к ук. 6.

6. Процесс окончен.

3.8.

a	результат
-5	15
3	18
7	2
32	12

4.1. В блок-схеме 3, *в*, в отличие от 3, *а*, имеются блоки ввода исходных данных и вывода результата.

4.2. Второй вариант алгоритма нахождения НОД (a, b).

4.3. (3.2) Рис. 23
(3.3) Рис. 24
(3.5, *б*) Рис. 25
(3.6) Рис. 26

4.5. б) Рис. 27

5.1.

(4.3, 3.2) **алг** Абсолютная величина (**вещ** $x, |x|$)
арг x
рез $|x|$
нач
если $x \geqslant 0$
то $|x| := x$
иначе $|x| := -x$
все
кон

(4.3, 3.3)

алг $a(a-1)$ $x=a^2-1$ (**вещ** a, x, **лит** y)
арг a
рез x, y
нач
если $a \neq 0$ и $a \neq 1$
то $x := (a+1)/a$
иначе **если** $a=0$
то y: = «нет решений»
иначе y: = «любое число — корень уравнения»
все

все
кон

(4.3, 3.5, б)
алг неравенство $ax<b$ (**вещ** a, b, x, **лит** y)
арг a, b
рез x, y

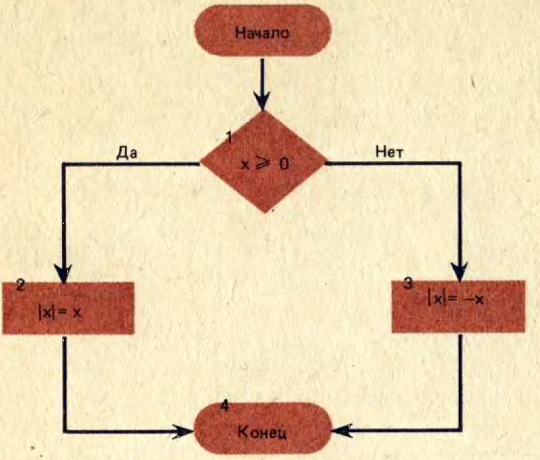

Начало
1
x ⩾ 0
Да
Нет
2
|x| = x
3
|x| = −x
4
Конец

Рис. 23

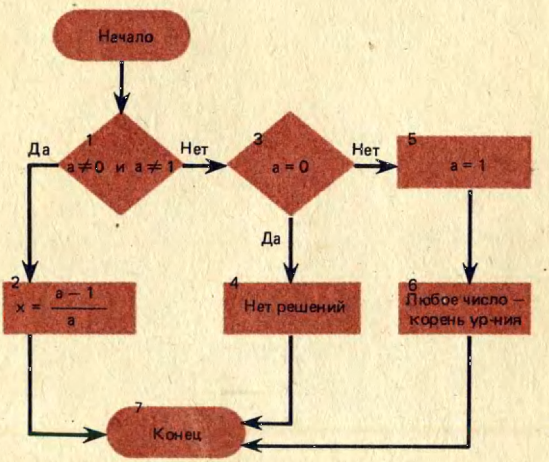

Начало
1
a ≠ 0 и a ≠ 1
Да
Нет
3
a = 0
Нет
5
a = 1
Да
2
x = (a − 1)/a
4
Нет решений
6
Любое число – корень ур-ния
7
Конец

Рис. 24

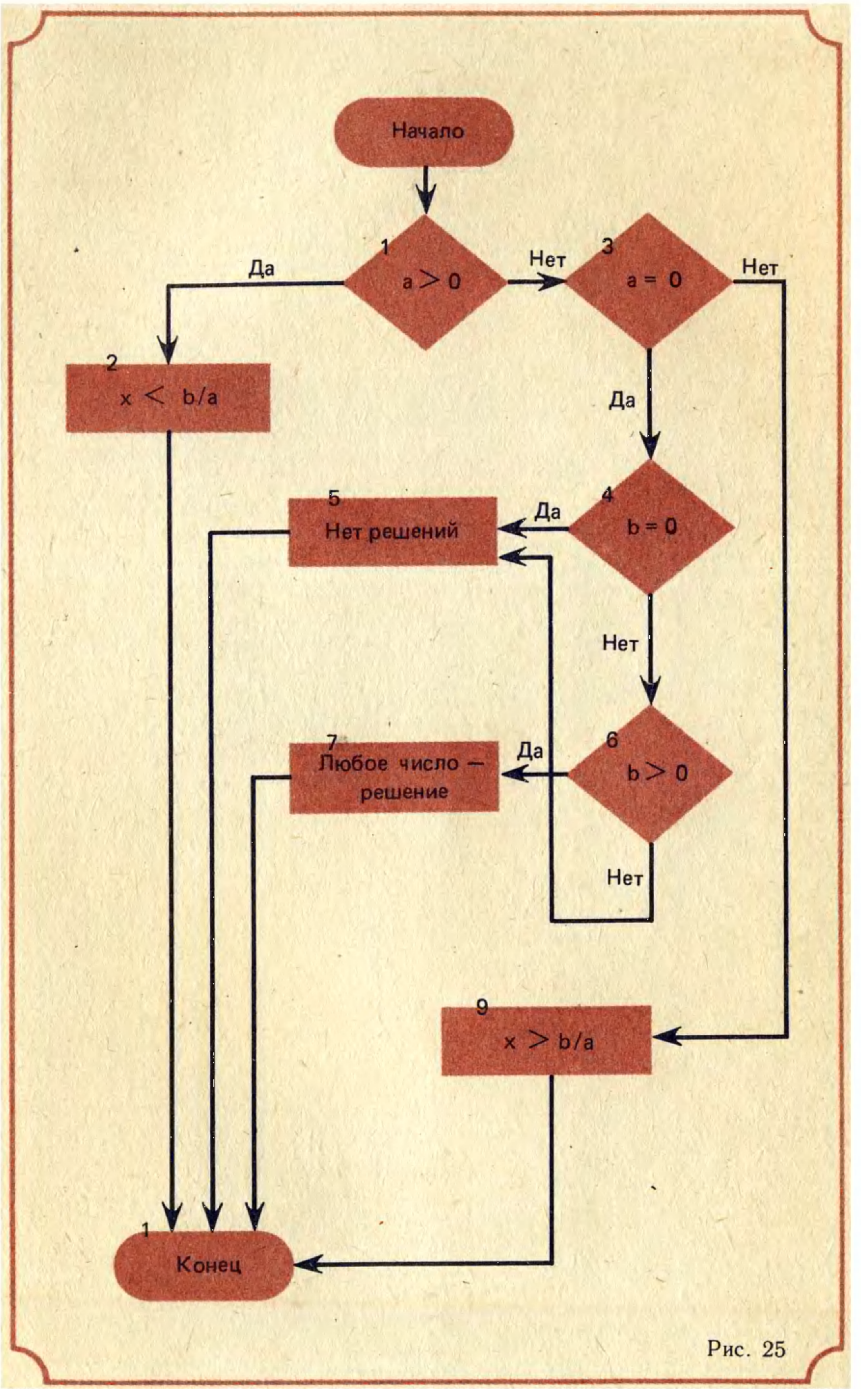

Начало
1
a > 0
Да
Нет
3
a = 0
Нет
2
x < b/a
Да
5
Нет решений
Да
4
b = 0
Нет
7
Любое число — решение
Да
6
b > 0
Нет
9
x > b/a
1
Конец

Рис. 25

```
нач
   если a > 0
      то x < b/a
      иначе если a = 0
                то если b = 0
                      то y: = «нет решений»
                      иначе если b > 0
                                то y: = «любое число — решение»
                                иначе y: = «нет решений»
                             все
                   все
                иначе x > b/a
             все
   все
кон
```

(4.3, 3.6)

```
алг решение системы (вещ a, a1, b, b1, c, c1, (x, y), лит T)
   арг a, a1, b, b1, c, c1
   рез (x, y), T
нач
   если a/a1 ≠ b/b1
      то (x, y): = ((b1c − bc1)/(ab1 − a1b), (ac1 − a1c)/(ab1 − a1b))
      иначе если b/b1 = c/c1
                то T: = «Бесконечное множество решений
                         {(x, y) | ax + by = c}»
                иначе T: = «Нет решений»
             все
все
кон
```

5.2.

```
алг Координатная четверть (вещ x, y, нат n)
   арг x, y
   рез n
нач
   если x < 0
      то если y < 0
            то n: = 3
            иначе n: = 2
         все
      иначе если y > 0
               то n: = 1
               иначе n: = 4
            все
   все
кон
```

5.3.

```
алг Принадлежность к кругу (вещ x1, y1, x2, y2, R, лит K)
   арг x1, y1, x2, y2, R
   рез K
нач вещ a, b, d
   a: = (x1 − x2)^2
   b: = (y1 − y2)^2
   d: = √(a + b)
   если d ⩽ R
      то K: = «точка M принадлежит кругу: M ∈ Кр (A, R)»
      иначе K: = «точка M не принадлежит
                  кругу: M ∉ Кр (A, R)»
   все
кон
```

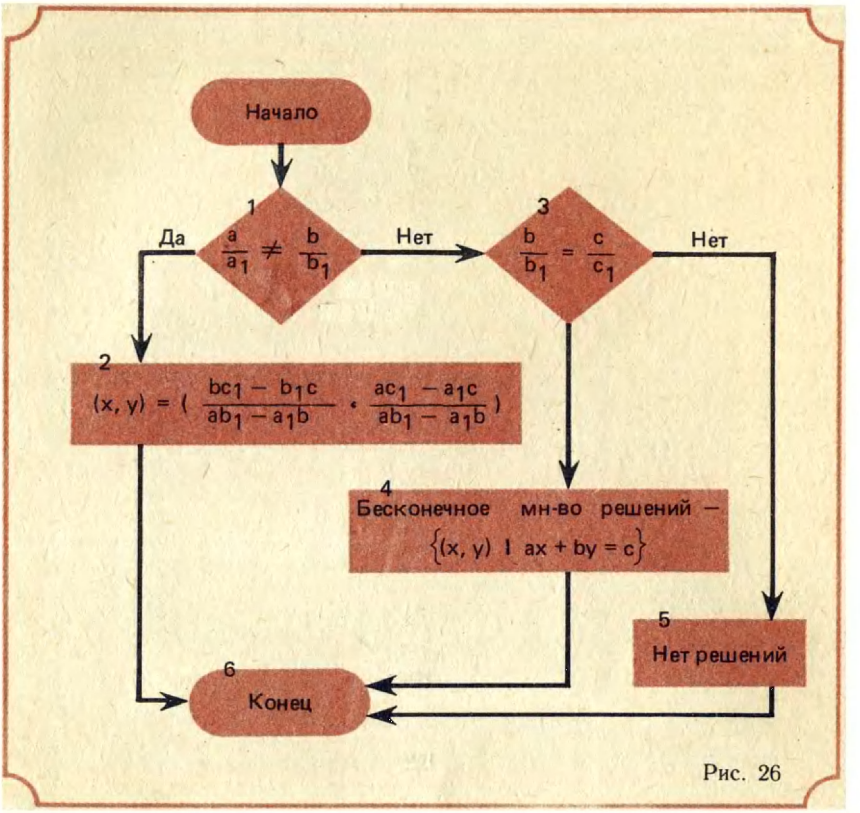


Рис. 26

5.4.

алг Ближе (**вещ** x_1, y_1, x_2, y_2, **лит** K)
  арг x_1, y_1, x_2, y_2
  рез K
нач **вещ** a_1, a_2
  $a_1 := x_1^2 + y_1^2$
  $a_2 := x^2 + y^2$
  если $a_1 < a_2$
    то $K :=$ «A_1 ближе к началу координат, чем A_2»
    иначе **если** $a_2 < a_1$
      то $K :=$ «A_2 ближе к началу координат, чем A_1».
      иначе $K :=$ «A_1 и A_2 одинаково удалены от начала координат»
    все
  все
кон

5.6.

алг значение функции (**вещ** x, a, b, y, **лит** c)
  арг x, a, b
  рез y, c

```
нач
  если 1 < x ⩽ 5
    то y: = x + a
    иначе если 5 < x < 20
      то y: = x + 2b
      иначе если x ⩾ 20
        то y: = x − a
        иначе c: = «функция
      все          не определена»
  все
кон
```

7.1. |||||||||*|||*||;
|||||||||||*||;
|||||||||||||.

Исходное слово обозначает систему чисел (5, 4, 3, 2), результат — число 14 (=5+4+3+2).

7.2. |||||||*||||;
||||||*|||;
|||||*||;
||||*|;
|||*;
|||.

Исходное слово обозначает пару чисел (7; 4), результат — число 3 (=|7—4|).

7.3. *abbaaabab*; *ababaabab*; *aabbaabab*; *aababababab*; *aaabbabab*; *aaababbab*; *aaaabbbab*; *aaaabbabb*; *aaaababbb*; *aaaaabbbb*; *aaaabbb*; *aaabb*; *aab*; *a*.

7.4. $\begin{cases} ba \to ab \\ ca \to ac \\ cb \to bc \end{cases}$

7.5. $A_1: \begin{cases} a \to \Lambda \\ b \to \Lambda \\ c \to \Lambda \end{cases}$

7.6. $A_2: \begin{cases} a \to \Lambda \\ b \to \Lambda \\ c \to \Lambda \\ \Lambda \to abcP \end{cases}$

7.7. $A_1 \circ A_2$.

7.8. 3) *ababab*; *aabbab*; *aababb*; *aaabbb*; *abbb*; *ab*.

4) *baabbaba*; *ababbaba*; *aabbbaba*; *aabbabba*; *aababbba*; *aaabbbba*; *aaabbbab*; *aaabbabb*; *aaababbb*; *aaaabbbb*; *aabbbb*; *bbbb*; *bb*; Λ.

7.9. Первая подстановка перерабатывает любое слово в алфавите {*a*, *b*} в слово, в котором все буквы *a* расположены левее всех букв *b*. Вторая подстановка удаляет все пары рядом стоящих букв *a*. Поэтому результирующее слово содержит не более одной буквы *a*. Третья подстановка удаляет все пары рядом стоящих букв *b*. Поэтому результирующее слово содержит не более одной буквы *b*. Следовательно, возможны лишь четыре результирующих слова: Λ, *a*, *b*, *ab*.

8.4. а) P: *aaaba*; Q: *abab*.
P': *b*; Q': Λ.
$P \not\sim Q$.

б) P: *bbaaaab*; Q: *baa*;
P': *b*; Q': *b*.
$P \sim Q$.

8.5. Эталонные слова: Λ, *a*, *b*, *aa*, *ab*, *aab*.

Вспомогательный алгоритм — алгоритм приведения — применимый к любому слову в алфавите {*a*, *b*} и перерабатывающий его в одно из следующих эталонных слов: Λ, *a*, *b*, *aa*, *ab*, *aab*.

$\begin{cases} ba \to aab, \\ aaa \to \Lambda, \\ bb \to \Lambda. \end{cases}$

Требуемый алгоритм выглядит так:

1. Найти эталонное слово для слова P — слово P', перейти к ук. 2.

2. Найти эталонное слово для слова Q — слово Q', перейти к ук. 3.

3. Сравни эталонные слова P' и Q'. Если они совпадают, то перейди к ук. 4, иначе — к ук. 5.

4. $P \sim Q$. Конец.

5. $P \not\sim Q$. Конец.

8.6. а) Поворот плоскости на 120° по часовой стрелке вокруг центра O треугольника совпадает с композицией $S \circ R \circ S$.

б) Осевая симметрия с осью OA совпадает с композицией $S \circ R$.

в) Осевая симметрия с осью OC совпадает с композицией $R \circ S$.

8.7. Так как $\begin{cases} S \circ R = R \circ R \circ S \\ R \circ R \circ R = E \\ S \circ S = E. \end{cases}$

то $X = RRSSRSRSSR =$

$= RR(SS)R(SR)(SS)R =$

$= RRER(RRS)ER = RRER(RRS)R =$

$= RRR(RS)R = ERR(SR) =$

$= RRRRS = ERS = RS.$

8.8. а) Слово *abcd* имеет одно смежное слово *cabd*. Слово *aaabb* не имеет смежных слов.

б) $abcd \sim cabd$.

8.9. 1) да; 2) да; 3) нет; 4) нет.

8.10. а) Результатом преобразования слова P в алфавите $\{a, b\}$ алгоритмом $A_1 \circ A_2$ является слово ab.

б) Результатом преобразования слова P в алфавите $\{a, b\}$ алгоритмом $A_2 \circ A_1$ является слово Λ.

8.11. Алгоритм приведения $\{aa \to \Lambda$ приводит к одному из двух эталонных слов: a, Λ.

9.1. 8,4; 8,10; 8,11; 8,12; 8.13.

9.2. 1) *bbabab*; *babbab*; *abbbab*; *abbabb*; *ababbb*; *aabbbb*; *abbb*; *bb*.

2) *aaa* (алгоритм неприменим к этому слову или оставляет его неизменным).

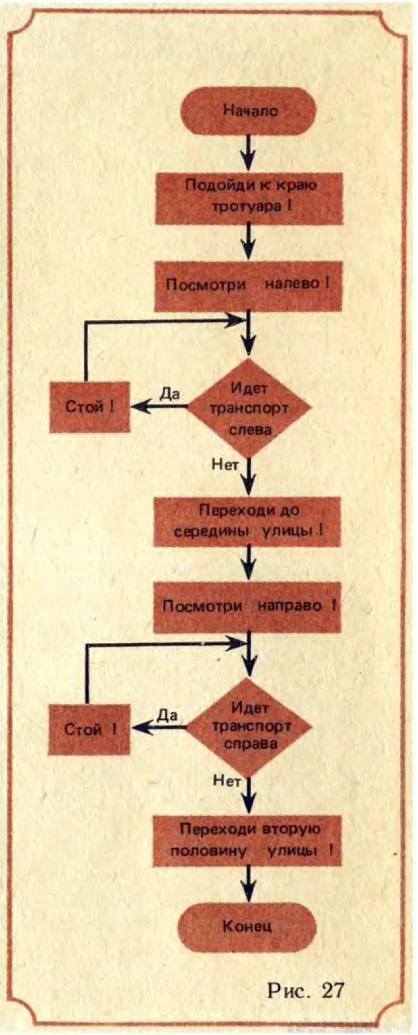


Рис. 27

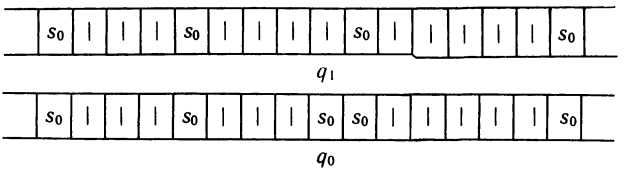
9.3. Одинаковое ли число букв (вхождений букв) a и b? Если результат преобразования слова — пустое слово, ответ «да», иначе — «нет».

9.4. Λ, $aa\ldots a$, $bb\ldots b$.

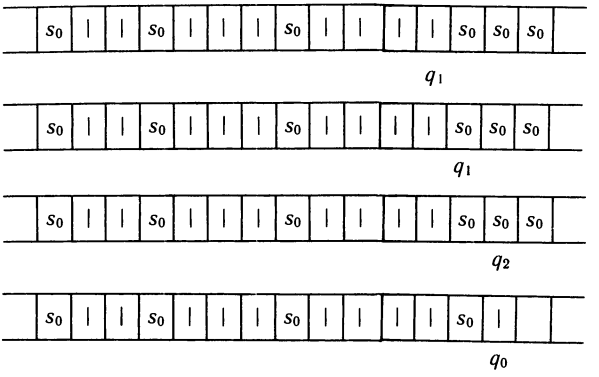
9.5. 1) *abaabbb*; |*baabbb*; ||*aabb*; |||*abbb*; ||||*bbb*; |||||*bb*; ||||||*b*; |||||||. Алгоритм определяет число вхождений букв (длину слова).

9.6. 1) Продолжение преобразования слова из 9.5, 1): |||||||; |||||; |||; |. Алгоритм определяет четность (нечетность) числа вхождений букв.

10.2.



10.3.



10.5.

l	s_0	|
q_1	$s_0 Л q_0$	$\| Л q_0$

10.7.

L	s_0	|
q_1	$s_0 П q_2$	$\| Л q_1$
q_2	$s_0 Л q_0$	$\| Н q_0$

10.11. а) Указание. Воспользоваться определением операции композиции машин Тьюринга.

б) Нет.

10.13. а) $C \circ A^2$; б) $T_1 \circ A$.

10.14. б) Машина R^2, отправляясь от воспринятого в стандартном положении числа, не самого правого на ленте, идет вправо и останавливается, воспринимая второе справа число от данного.

в) Машина R^k воспринимает k-е справа от данного число.

г) Машина L^k воспринимает k-е слева от данного число.

д) Машина $C \circ A^5$ по записи набора чисел x_1, x_2, ..., x_n на ленте выдает в качестве результата набор чисел x_1, x_2, ..., x_n, 5, воспринимаемый ею в стандартном положении.

е) Машина $T_1 \circ A^4$ по записи числа x на ленте выдает в качестве результата это число и через пустую клетку от него — число, увеличенное на 4.

12.1.

```
(3,2)10 INPUT X
    20 IF X>=0 THEN A=X\GO TO 40
    30 A=-X
    40 PRINT «ABS(X) =»;A
    50 END

(3,6)10 INPUT A,A1, B, B1, C, C1
    20 LET D=(A*B1)-(A1*B)
    30 LET D1=(B1*C)-(B*C1)
    40 LET D2=(A*C1)-(A1*C)
    50 IF D<>0 THEN 110
    60 IF D1=0 THEN 90
    70 PRINT «НЕТ РЕШЕНИЙ»
```

```
        80 GO TO 140
        90 PRINT «БЕСКОНЕЧНОЕ МНОЖЕСТВО РЕШЕНИЙ»
        100 GO TO 140
        110 LET X=D1/D
        120 LET Y=D2/D
        130 PRINT «(X, Y) = («; X; «,»; Y; »)»
        140 END
```

(3.7, *а*)
```
        10 INPUT X
        20 IF X<=0 THEN 50
        30 LET Y=X 2+1
        40 GO TO 60
        50 LET Y=X+1
        60 PRINT «Y=»; Y
        70 END
```

(3,7, *б*)
```
        10 INPUT X
        20 IF X<1 THEN 80
        30 IF X<5 THEN 60
        40 LET Y=X`2-5
        50 GO TO 90
        60 LET Y=X+1
        70 GO TO 90
        80 LET Y=X-1
        90 PRINT «Y=»; Y
        100 END
```

(5,2)
```
        10 INPUT X, Y
        20 IF X<0 THEN 60
        30 IF Y>0 THEN 110
        40 LET N=4
        50 GO TO 120
        60 IF Y<0 THEN 90
        70 LET N=2
        80 GO TO 120
        90 LET N=3
        100 GO TO 120
        110 LET N=1
        120 PRINT «N=»; N
        130 END
```

(5.3)
```
        10 INPUT X1, Y1, X2, Y2, R
        20 LET A=(X1-X2)^2
        30 LET B=(Y1-Y2)^2
        40 LET D=SQR (A+B)
        50 IF D<=R THEN 80
        60 PRINT «ТОЧКА НЕ ПРИНАДЛЕЖИТ КРУГУ»
        70 GO TO 90
        80 PRINT «ТОЧКА ПРИНАДЛЕЖИТ КРУГУ»
        90 END
```

(5.4)
```
        10 INPUT X1, Y1, X2, Y2
        20 LET A1=X1^2+Y1^2
        30 LET A2=X2^2+Y2^2
        40 IF A1<A2 THEN 100
        50 IF A2<A1 THEN 80
        60 PRINT «A1 И A2 ОДИНАКОВО УДАЛЕНЫ ОН НАЧАЛА КООРДИНАТ»
        70 GO TO 110
        80 PRINT «A2 БЛИЖЕ К НАЧАЛУ КООРДИНАТ»
        90 GO TO 110
        100 PRINT «A1 БЛИЖЕ К НАЧАЛУ КООРДИНАТ»
        110 END
```

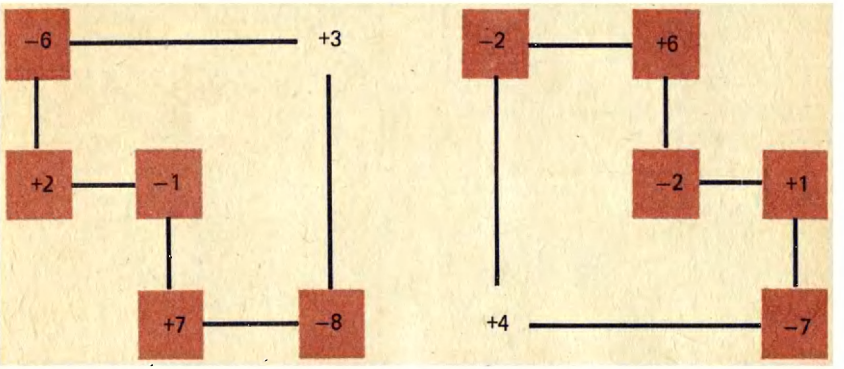
14.2. План перевозок, представленный в таблице 31, не является допустимым, а план, представленный в таблице 32,— допустимый.

14.3. Транспортные расходы $S = 3 \cdot 11 + 7 \cdot 14 + 3 \cdot 5 + 1 \cdot 10 + 3 \cdot 5 + 2 \cdot 5 = 33 + 98 + 15 + 10 + 15 + 10 = 186$ (р.).

14.4.

30^{2}	10^{6}	^{2}	40^{3}	80
^{5}	10^{2}	90^{1}	^{7}	100
^{4}	50^{5}	^{7}	^{8}	50
30	70	90	40	

14.5.



$K_{14} = 3 - 6 + 2 - 1 + 7 - 8 = -3$,
$K_{31} = 4 - 2 + 6 - 2 + 1 - 7 = 0$.

СОДЕРЖАНИЕ

Научно-популярное издание

МАКАРЕНКОВ Юрий Алексеевич
СТОЛЯР Абрам Аронович

ЧТО ТАКОЕ АЛГОРИТМ?

Зав. редакцией Б. А. Кимбар
Редактор К. М. Лукашевич
Мл. редактор Л. Н. Ясницкая
Художник А. А. Богуш
Художественный редактор Н. И. Евменова
Технические редакторы М. И. Чепловодская, Л. П. Сопот
Корректор С. А Янович

ИБ № 2641

Сдано в набор 31 07 87 Подписано в печать 16 09 88 АТ 14093 Формат 60×90¹/₁₆ Бумага офсетная № 2 Гарнитура литературная Офсетная печать Усл печ л 8 Усл кр-отт 16,5 Уч-изд л 7,34 Тираж 36 000 экз Заказ 691 Цена 30 к

Издательство «Народная асвета» Государственного комитета БССР по делам издательств, полиграфии и книжной торговли 220600, Минск, проспект Машерова, 11

Минский ордена Трудового Красного Знамени полиграфкомбинат МППО имени Я Коласа 220005, Минск, Красная, 23