Донецкий национальный технический университет

Факультет компьютерных наук и технологий
Кафедра «Прикладная математика»

Ю.Н. Добровольский

Машина Тьюринга, алгоритмы Маркова, рекурсивные функции. Решение задач

(Учебно-методическое пособие)

Для студентов специальности «Прикладная математика»

Донецк
2018

Государственное образовательное учреждение высшего профессионального образования
«Донецкий национальный технический университет»

Факультет компьютерных наук и технологий
Кафедра прикладной математики

Ю.Н. Добровольский

Машина Тьюринга, алгоритмы Маркова, рекурсивные функции. Решение задач

(Учебно-методическое пособие)

Для студентов специальности «Прикладная математика»

Рассмотрено
на заседании кафедры ПМ
протокол № _ от “__” _____ 2018 г.

Утверждено
учебно-издательским советом ДонНТУ
протокол №_ от “__ ” ____ 2018 г.

Донецк 2018

УДК 681.3.07

Павлыш В.Н, Добровольский Ю.Н. Машина Тьюринга, алгоритмы Маркова, рекурсивные функции. Решение задач. (Учебно-методическое пособие)-Д: ДонНТУ, 2018. – 34с.

Пособие посвящено решению задач по теме «Введение в теорию алгоритмов», изучаемой на втором курсе факультета КНТ ДонНТУ в рамках дисциплины «Алгоритмы и алгоритмические языки». Это задачи на составление алгоритмов в виде машины Тьюринга, нормальных алгоритмов Маркова, рекурсивных функций, а также задачи теоретического характера.

В пособии приводятся необходимые сведения по теории алгоритмов, подробно объясняются типичные приёмы решения задач и предлагается большой набор задач для самостоятельного решения.

Пособие рассчитано на студентов второго курса факультета КНТ ДонНТУ и преподавателей, ведущих семинарские занятия по программированию.

Автор: Ю.Н. Добровольский

Отв. за выпуск: В.Н. Павлыш, д.т.н., профессор

Оглавление

1. Машина Тьюринга

Идея создания машины Тьюринга, предложенная английским математиком А. Тьюригом в тридцатых годах XX века, связана с его попыткой дать точное математическое определение понятия алгоритма.

Машина Тьюринга является таким же математическим объектом, как функция, производная, интеграл, группа и т.д. Так же как и другие математические понятия, понятие машины Тьюринга отражает объективную реальность, моделирует некие реальные процессы.

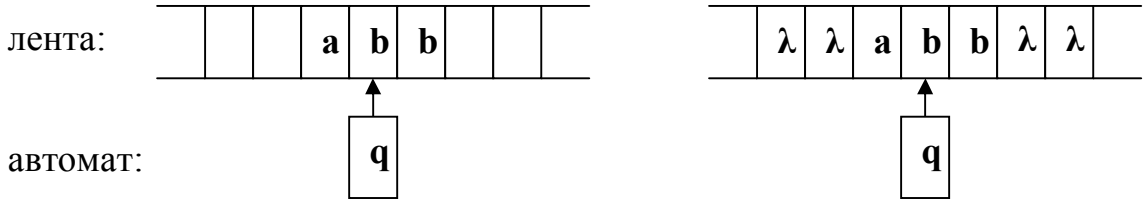
Этот математический аппарат был назван “машиной” по той причине, что по описанию его составляющих частей и функционированию он похож на вычислительную машину.

Принципиальное отличие машины Тьюринга от вычислительных машин состоит в том, что ее запоминающее устройство представляет собой бесконечную ленту: у реальных вычислительных машин запоминающее устройство может быть как угодно большим, но обязательно конечным. Машину Тьюринга нельзя реализовать именно из-за бесконечности ее ленты. В этом смысле она мощнее любой вычислительной машины.

1.1 Краткое описание машины Тьюринга

Структура машины Тьюринга.

Машина Тьюринга (МТ) состоит из двух частей – **ленты и автомата.**



Лента используется для хранения информации. Она бесконечна в обе стороны и разбита на клетки, которые никак не нумеруются и не именуются. В каждой клетке может быть записан один символ или ничего не записано. Содержимое клетки может меняться – в неё можно записать другой символ или стереть находящийся там символ.

Пустое содержимое клетки будем называть **символом «пусто»** и обозначать знаком **λ**. В связи с этим изображение ленты слева и справа одинаково.

Автомат – это активная часть МТ. В каждый момент он размещается под одной из клеток ленты и видит её содержимое; это **видимая клетка**, а находящийся в ней символ – **видимый символ**; содержимое соседних и других клеток автомат не видит. Кроме того, в каждый момент автомат находится в одном из **состояний**, которое будем обозначать буквой **q** с номерами $\mathbf{q_1}$, $\mathbf{q_2}$ и т.п. Находясь в некотором состоянии, автомат выполняет какую-то определённую операцию (например, перемещается направо по ленте, заменяя все символы **b** на **a**), находясь в другом состоянии – другую операцию.

Пару из видимого символа (a) и текущего состояния автомата (q) будем называть **конфигурацией** и обозначать **<a,q>**.

Автомат может выполнять три элементарных действия:

1) записывать в видимую клетку новый символ (менять содержимое других клеток автомат не может);

2) сдвигаться на одну клетку влево или вправо («перепрыгивать сразу через несколько клеток автомат не может;

3) переходить в новое состояние.

Такт работы машины Тьюринга.

МТ работает **тактами**, которые выполняются один за другим. На каждом такте автомат МТ выполняет три следующие действия, причем обязательно в указанном порядке:

1) записывает некоторый символ **a'** в видимую клетку (в частности, может быть записан тот же символ, что и был в ней, тогда содержимое этой клетки не меняется);

2) сдвигаться на одну клетку влево (обозначение – **L**), или на одну клетку вправо (обозначение – **R**), или остается неподвижным (обозначение – **N**);

3) переходит в некоторое состояние **q'** (в частности, может остаться в прежнем состоянии).

Формально действия одного такта будем записывать в виде тройки:

a' [L R N] q' – где конструкция с квадратными скобками означает возможность записи в этом месте любой из букв **L**, **R** или **N**. Например, такт ***Lq8** означает запись символа ***** в видимую клетку, сдвиг на одну клетку влево и переход в состояние $\mathbf{q_8}$.

Программа для машины Тьюринга.

Сама по себе МТ ничего не делает. Для того чтобы заставить её работать, надо написать для неё **программу**. Эта программа записывается в виде следующей таблицы:

	$\mathbf{a_1}$	$\mathbf{a_2}$	…	$\mathbf{a_i}$	…	$\mathbf{a_n}$	**λ**
$\mathbf{q_1}$							
…							
$\mathbf{q_i}$				**a' [L R N] q'**			
…							
$\mathbf{q_m}$							

Слева перечисляются все состояния, в которых может находиться автомат, сверху – все символы (в том числе и **λ**), которые автомат может видеть на ленте. На пересечениях же строки и столбца указываются те такты, которые должен выполнить автомат, когда он находится в соответствующем состоянии и видит на ленте соответствующий символ.

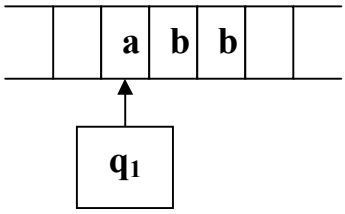
В целом таблица определяет действия МТ при всех возможных конфигурациях и тем самым полностью задаёт поведение МТ. Описать алгоритм в виде МТ – значит предъявить такую таблицу.

Правила выполнения программы.

До выполнения программы нужно проделать следующие предварительные действия:

1) надо записать на ленту **входное слово**, к которому будет применена программа, (внутри входного слова пустых клеток быть не должно, а слева и справа от него должны быть только пустые клетки). Пустое входное слово означает, что все клетки ленты пусты;

2) надо установить автомат в состояние $\mathbf{q_1}$ (указанное в таблице первым) и разместить его под первым символом входного слова:



После этих предварительных действий начинается выполнение программы. В таблице отыскивается ячейка на пересечении первой строки (т.к. автомат находится в состоянии $\mathbf{q_1}$) и того столбца, который соответствует первому символу входного слова (это необязательно левый столбец таблицы), и выполняется такт, указанный в этой ячейке. В результате автомат окажется в новой кофигурации. Теперь такие же действия повторяются, но уже для новой конфигурации: в таблице отыскивается ячейка, соответствующая состоянию и символу этой конфигурации, и выполняется такт из этой ячейки. И так далее.

Когда завершается выполнение программы? Введем понятие **такта останова.** Это такт, который ничего не меняет: автомат записывает в видимую клетку тот же символ, что и был в ней раньше, не сдвигается и остается в прежнем состоянии, т.е. это такт **a,N,q** для конфигурации **<a,q>**. Попав на такт останова, МТ, по определению, останавливается, завершая свою работу.

В целом возможны два исхода работы МТ над входным словом:

1) Первый исход – «хороший»: это когда в какой-то момент МТ останавливается (попадает на такт останова). В таком случае говорят, что МТ **применима** к заданному входному слову. А то слово, которое получено на ленте, считается выходным словом, т.е. результатом работы МТ, ответом.

2) Второй исход – «плохой»: это когда МТ зацикливается , никогда не попадает на такт останова. В этом случае говорят, что МТ **неприменима** к заданному входному слову.

Замечание.

1) Если надо указать, что после выполнения некоторого такта МТ должна остановиться, то это состояние будем обозначать $\mathbf{q_0}$. Например, такт $\mathbf{b,L,q_0}$ означает следующие действия: запись символа **b** в видимую клетку ленты, сдвиг влево и останов.

Таким образом множество состояний автомата: $\mathbf{Q=\{q_0,q_1,\ldots,q_m\}}$ – это внутренняя его память. Состояние $\mathbf{q_1}$ называется начальным, а состояние $\mathbf{q_0}$ называется заключительным (состояние останова).

2) Если заранее известно, что в процессе выполнения программы не может появиться некоторая конфигурация, то в соответствующей ячейке таблицы будем ставить прочерк.

1.2 Примеры на составление программ для МТ

Введём следующие два соглашения: буквой **P** будем обозначать входное слово; буквой **A** будем обозначать алфавит входного слова, т.е. набор тех символов, из которых и только которых может состоять **P**.

Пример1 (перемещение автомата, замена символов)

A={0,1,2,3,4,5,6,7,8,9}. Пусть **P** – непустое слово; значит, **P** – это последовательность из десятичных цифр. Требуется получить на ленте запись числа, которе на 1 больше числа **P.**

Решение. Для этой задачи предлагается выполнить следующие действия:

1. Перегнать автомат под последнюю цифру числа.
2. Если это цифра от 0 до 8, то заменить её цифрой на 1 больше и остановиться.
3. Если же это цифра 9, тогда заменить её на 0 и сдвинуть автомат к предыдущей цифре, после чего таким же способом увеличить на 1 эту предпоследнюю цифру.
4. Особый случай: в P только девятки (например, 99). Тогда автомат будет сдвигаться влево, заменяя девятки на нули, и в конце концов окажется под пустой клеткой. В эту пустую клетку надо записать 1 и остановиться (ответом будет 100).

В виде программы для МТ эти действия описываются следующим образом:

	0	**1**	**2**	**3**	**4**	**…**	**7**	**8**	**9**	**λ**	
q_1	$0Rq_1$	$1Rq_1$	$2Rq_1$	$3Rq_1$	$4Rq_1$	…	$7Rq_1$	$8Rq_1$	$9Rq_1$	λLq_2	**Под последнюю цифру**
q_2	$1Nq_0$	$2Nq_0$	$3Nq_0$	$4Nq_0$	$5Nq_0$	…	$8Nq_0$	$9Nq_0$	$0Lq_2$	$1Nq_0$	**Видимая цифра +1**

Пояснения.

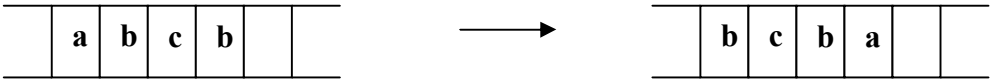
q_1 – это состояние, в котором автомат «бежит» под последнюю цифру числа.

q_2 – это состояние, в котором автомат прибавляет 1 к той цифре, которую видит в данный момент.

Пример2 (анализ символов)

A={a,b,c}. Перенести первый символ непустого слова **P** в его конец.

Например:



Решение. Для решения этой задачи предлагается выполнить следующие действия:

1.Запомнить первый символ слова P, а затем стереть этот символ.

2. Перегнать автомат вправо под первую пустую клетку за P и записать в неё запомненный символ.

В МТ нет другого запоминающего устройства, кроме ленты, а запоминать символ в какой-то клетке на ленте бессмысленно: как только автомат сдвинеться влево или вправо от этой клетки, он тут же забудет данный символ.

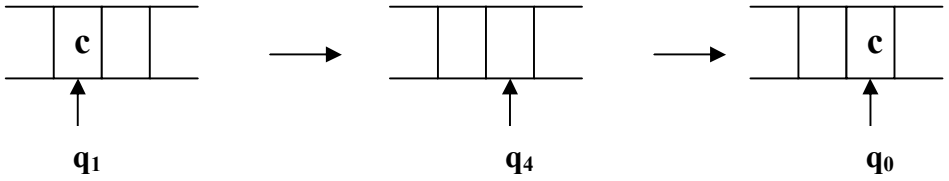
Выход один, надо использовать разные состояния автомата. Если первый символ – это **a**, то надо перейти в состояние q_2, в котором автомат бежит вправо и записывает в конце **a**. Если первый символ **b**, тогда надо перейти в состояние q_3, где делается всё то же самое, только в конце записывается символ **b**. Если первым был символом **c**, тогда переходим в состояние q_4, в котором автомат дописывает за входным словом символ **c**.

С учетом сказанного программа будет такой:

	a	b	c	λ	
q_1	$\lambda R q_2$	$\lambda R q_3$	$\lambda R q_4$	$\lambda R q_1$	**анализ 1-го символа, удаление его, развевление**
q_2	$a R q_2$	$b R q_2$	$c R q_2$	$a N q_0$	**запись a справа**
q_3	$a R q_3$	$b R q_3$	$c R q_3$	$b N q_0$	**запись b справа**
q_4	$a R q_4$	$b R q_4$	$c R q_4$	$c N q_0$	**запись c справа**

Замечание. При пустом слове программа зациклится – автомат, находясь в состоянии q_1 и попадая все время на пустые клетки, будет бесконечно перемещаться вправо. Поэтому в этом случае программу надо остановить.

Если во входном слове ровно один символ, тогда автомат сотрёт этот символ, сдвинется на одну клетку вправо и запишет в неё данный символ.



Пример3 (сравнение символов, стирание слова)
A={a,b,c}. Если первый и последний символы (непустого) слова **P** одинаковы, тогда это слово не менять, а иначе заменить его на пустое слово.

Решение. Для решения этой задачи предлагается выполнить следующие действия:

1. Запомнить первый символ входного слова, не стирая его.

2. Переместить автомат под последний символ и сравнить его с запомненным. Если они равны, то больше ничего не делать.

3. В противном случае уничтожить всё входное слово.

Как запомнить символ и как перегонять автомат под последний символ слова, мы уже знаем из предыдущих примеров. Стирание входного слова реализуется заменой всех символов на символ **λ**. При этом, раз автомат оказался в конце слова, будем перемещать автомат справа налево до первой пустой клетки.

Эти действия описываются следующей программой для МТ. Напомним, что прочерк означает невозможность появления соответствующей конфигурации.

	a	b	c	λ	
q_1	aNq_2	bNq_4	cNq_6	λNq_0	**анализ 1-го символа, разветвление**
q_2	aRq_2	bRq_2	cRq_2	λLq_3	**идти к последнему символу при 1-м символе a**
q_3	aNq_0	bNq_8	cNq_8	-	**сравнить посл. символ с a, не равны-на q_8 (стереть P)**
q_4	aRq_4	bRq_4	cRq_4	λLq_5	**аналогично при 1-м символе b**
q_5	aNq_8	bNq_0	cNq_8	-	
q_6	aRq_6	bRq_6	cRq_6	λLq_7	**аналогично при 1-м символе c**
q_7	aNq_8	bNq_8	cNq_0	-	
q_8	λLq_8	λLq_8	λLq_8	λNq_0	**стереть всё слово, двигаясь справа налево**

Пример4 (удаление символа из слова)

A={a,b}. Удалить из слова **P** его второй символ, если такой есть.

Решение. Кажется, что эта задача решается просто: надо сдвинуть автомат под клетку со вторым символом и затем очистить эту клетку. Однако это не так. Внутри выходного слова не должно быть пустых клеток. Поэтому после удаления второго символа надо «сжать» слово, перенеся первый символ на одну клетку вправо. Сразу запоминаем первый символ, стираем его и записываем вместо второго символа.

В виде программы для МТ всё это записывается так:

	a	b	λ	
q_1	λRq_2	λRq_3	λNq_0	**анализ и удаление 1-го символа, разветвление**
q_2	aNq_0	aNq_0	aNq_0	**замена 2-го символа на a**
q_3	bNq_0	bNq_0	bNq_0	**замена 2-го символа на b**

Пример5 (сжатие слова)

A={a,b,c}. Удалить из слова **P** первое вхождение символа **a**, если такое есть.

Решение. Будем в цикле переносить вправо все начальные символы **b** и **c** входного слова – до первого символа a или до пустой клетки.

	a	b	c	λ	
q_1	λRq_0	λRq_2	λRq_3	λNq_0	**q_λ: стереть 1-й символи и перенести его вправо**
q_2	bNq_0	bRq_2	bRq_3	bNq_0	**q_b: запись b, перенос ранее видимого символа вправо**
q_3	cNq_0	cRq_2	cRq_3	cNq_0	**q_c: запись c, перенос ранее видимого символа вправо**

Пример6 (вставка символа в слово)

A={a,b,c}. Если **P** – непустое слово, то за его первым символом вставить символ **a**.

Решение. В состояниях q_2, q_3, q_4 автомат может видеть только пустую клетку, а в состоянии q_5 он обязательно видит первый символ входного слова, но не пустую клетку.

	a	b	c	λ	
q_1	aLq_2	bLq_3	cLq_4	λNq_0	анализ 1-го символа для переноса его влево
q_2	-	-	-	aRq_5	приписать a слева
q_3	-	-	-	bRq_5	приписать b слева
q_4	-	-	-	cRq_5	приписать c слева
q_5	aNq_0	aNq_0	aNq_0	-	Заменить бывший 1-й символ на a

Пример7 (раздвижка слова)
A={a,b,c}. Вставить в слово **P** символ **a** за первым вхождением символа **c**, если такое есть.

Решение.

	a	b	c	λ	
q_1	aRq_1	bRq_1	aLq_4	λLq_0	вправо до c, вставка a вместо c, перенос c влево
q_2	aLq_2	aLq_3	aLq_4	aNq_0	перенос a справа
q_3	bLq_2	bLq_3	bLq_4	bNq_0	перенос b справа
q_4	cLq_2	cLq_3	cLq_4	cNq_0	перенос c справа

Пример8 (формирование слова на новом месте)
A={a,b,c}. Удалить из **P** все вхождения символа **a.**

Решение. Конкретно предлагается выполнить следующие действия:

1. Выходное слово будем строить справа от входного, отделим их знаком “=”.

2. После этого возвращаемся к началу входного слова.

3. Теперь надо перенести в цикле все символы входного слова, кроме **a**, вправо за знак = в формируемое выходное слово.

Для этого анализируем первый символ входного слова. Если это **a**, тогда стираем его и переходим к следующему символу. Если же первый символ – это **b** или **c**, тогда стираем его и «бежим» вправо до первой пустой клетки, куда и записываем этот символ. Снова возвращаемся налево к тому символу, который стал первым во входном слове, и повторяем те же самые действия, но уже по отношению к этому символу.

4. Этот цикл завершается, когда при возврате налево мы увидим в качестве первого символа знак =. Надо этот знак стереть, сдвинуться вправо под выходное слово и остановиться.

	a	b	c	=	λ	
q_1	aRq_1	bRq_1	cRq_1	-	$=Nq_2$	запись справа знак=
q_2	aLq_2	bLq_2	cLq_2	$=Lq_2$	λRq_3	влево к 1-му символу слова
q_3	λRq_3	λRq_4	λRq_5	λRq_0	-	анализ и удаление его, разветвление
q_4	aRq_4	bRq_4	cRq_4	$=Rq_4$	bNq_2	запись b справа, возврат налево (в цикл)
q_5	aRq_5	bRq_5	cRq_5	$=Rq_5$	cNq_2	запись c справа, возврат налево (в цикл)

Пример9 (фиксирование места на ленте)

A={a,b}. Удвоить слово **P**, поставив меду ним и его копией знак =.

Решение. Надо выполнить следующие действия:

1. Вначале записываем знак = за входным словом.

2. Затем возвращаемся под первый символ входного слова.

3. Далее заменяем видимый символ **a** на двойник **A**, «бежим» вправо до первой свободной клетки и записываем в неё символ **a**. После этого возвращаемся влево к клетке с двойником **A**, восстанавливаем прежний символ **a** и сдвигаемся вправо к следующему символу.

Теперь аналогичным образом копируем второй символ (заменяем его на **A**, в конец дописываем **a** и т.д.). и все последующие символы входного слова.

4. Когда мы скопируем последний символ входного слова и вернемся к его двойнику, то затем после сдвига на одну позицию вправо мы попадем на знак =. Это сигнал о том, что входное слово полностью скопировано, поэтому работу МТ надо завершить.

	a	b	=	A	B	λ	
q_1	aRq_1	bRq_1	-	-	-	$=Lq_2$	поставить = справа от слова
q_2	aLq_2	bLq_2	-	-	-	λRq_3	налево под 1-й символ
q_3	aRq_4	BRq_5	$=Nq_0$	-	-	-	анализ и замена очередного символа
q_4	aRq_4	bRq_4	$=Rq_4$	-	-	aNq_6	запись a справа
q_5	aRq_5	bRq_5	$=Rq_5$	-	-	bNq_6	запись b справа
q_6	aLq_6	bLq_6	$=Lq_6$	aRq_3	bRq_3	-	возврат, восстановление, к след. символу

Отметим, что в этой программе можно избавиться от состояния q_6, если объединить его с состоянием q_2, предусмотрев в q_2 возврат влево как до пустой клетки, так и до символов **A** и **B**.

	a	b	=	A	B	λ	
			…				
q_2	aLq_2	bLq_2	$=Lq_2$	aRq_3	bRq_3	λRq_3	налево до λ, A или B
			…				

1.3 Задачи для самостоятельного решения

1.1 A={a,b,c}. Приписать слева к слову P символ b (P→bP).

1.2 A={a,b,c}. Приписать справа к слову P символы bc (P→Pbc).

1.3 A={a,b,c}. Заменить на a каждый второй символ в слове P.

1.4 A={a,b,c}. Оставить в слове P только первый символ (пустое слово не менять).

1.5 A={a,b,c}. Оставить в слове P только последний символ (пустое слово не менять).

1.6 A={a,b,c}. Определить, является ли P словом ab. Ответ (выходное слово): слово ab, если является, или пустое слово иначе.

1.7 A={a,b,c}. Определить, входит ли в слово P символ a. Ответ: слово из символа слово a (да, входит) или пустое слово (нет).

1.8 A={a,b,c}. Если в слово P не входит символ a, то заменить в P все символы b на c, иначе в качестве ответа выдать слово из одного символа a.
1.9 A={a,b,0,1}. Определить, является ли слово P идентификатором (непустым словом, начинающимся с буквы). Ответ: слово a (да) или пустое слово (нет).
1.10 A={a,b,0,1}. Определить, является ли слово P записью числа в двоичной системе счисления (непустым словом, состоящем только из цифр 0 и 1.
Ответ: слово 1 (да) или слово 0.
1.11 A={0,1}. Считая непустое слово P записью двоичного числа, удалить из него незначащие нули, если такие есть.
1.12 A={0,1}. Для непустого слова P определить, является ли оно записью степени двойки (1,2,4,8,..) в двоичной системе счисления. Ответ: слово 1 (является) или слово 0.
1.13 A={0,1,2,3}. Считая непустое слово P записью числа в четверичной системе счисления, определить, является оно чётным числом или нет. Ответ:1 (да) или 0.
1.14 A={0,1}. Считая непустое слово P записью числа в двоичной системе, получить двоичное число, равное учетверенному числу P (например: 101→10100).
1.15 A={0,1}. Считая непустое слово P записью числа в двоичной системе, получить двоичное число, равное неполному частному от деления числа P на 2 (например:1011 →101).

2. Нормальные алгоритмы Маркова

2.1 Краткое описание нормальных алгоритмов Маркова

Подстановки.

В нормальных алгоритмах Маркова (НАМ) используется только одно элементарное действие – подстановка.

Формулой подстановки называется запись вида α → β (читается «α заменить на β»), где α и β – любые символы (возможно, и пустые). При этом α называется левой частью формулы, а β – правой частью.

Сама **подстановка** (как действие) задается формулой подстановки и применяется к некоторому слову P. Суть операции сводится к тому, что в слове P отыскивается часть, совпадающая с левой частью этой формулы (т.е. с α), и она заменяется на правую часть формулы (т.е. на β). При этом остальные части слова P (слева и справа от α) не меняются. Получившееся слово R называют **результатом подстановки.** Условно это можно изобразить так:

Необходимые уточнения:
1. Если левая часть формулы подстановки входит в слово P, то говорят, что эта формула **применима к P**. Но если α не входит в P, то формула считается **неприменимой к P**, и подстановка не выполняется.

2. Если левая часть α входит в P несколько раз, то на правую часть β, по определению, заменяется только первое вхождение α в P.
3. Если правая часть формулы подстановки – пустое слово, то подстановка α→ сводится к вычеркиванию части α из P.
4. Если в левой части формулы подстановки указано пустое слово, то подстановка →β сводится, по определению, к приписыванию β слева к слову P.

Определение НАМ

Нормальным алгоритмом Маркова (НАМ) называется непустой конечный упорядоченный набор формул подстановки:

$$\begin{cases} \alpha_1 \to \beta_1 \\ \alpha_2 \to \beta_2 \\ \quad \dots \qquad (k \geq 1) \\ \alpha_k \to \beta_k \end{cases}$$

В этих формулах могут использоваться два вида стрелок: обычная стрелка (→) и стрелка «с хвостиком» (|→). Формула с обычной стрелкой называется **обычной формулой**, а формула со стрелкой «с хвостиком» - **заключительной формулой.**

Записать алгоритм в виде НАМ – значит предъявить такой набор формул.

Правила выполнения НАМ.

Задается некоторое **входное слово** P. Работа НАМ сводится к выполнению последовательности шагов. На каждом шаге формулы подстановки просматриваются сверху вниз и выбирается первая из формул, применимых к входному слову P. Далее выполняется подстановка согласно найденной формуле. Получается новое слово P' и так далее.

Необходимые уточнения:

1. Если на очередном шаге была применена обычная формула (α→β), то работа НАМ продолжается.

2. Если на очередном шаге была применена заключительная формула (α|→β), то после её применения работа НАМ прекращается. То слово, которое получилось и есть **выходное слово**.

3. Если на очередном шаге к текущему слову неприменима ни одна формула, то и в этом случае работа НАМ прекращается.

2.2 Примеры на составление НАМ

Пример1 (вставка и удаление символов)
A={a,b,c,d}. В слове **P** требуется заменить первое вхождение подслова **bb** на **ddd** и удалить все вхождения символа c.
Например: abbcabbca → adddabba

Решение.На первый взгляд кажется, что эту задачу решает следующий НАМ:

$$\begin{cases} bb \to ddd \quad (1) \\ c \to \qquad (2) \end{cases}$$

Однако это не так. Проверим НАМ на входном слове abbcabbca (над стрелками указаны номера применённых формул, а в словах подчёркнуты те части, к которым были применены эти формулы):

$$abbcabbca \xrightarrow{1} adddcabbca \xrightarrow{1} adddcadddca \xrightarrow{2} adddadddca \rightarrow \ldots$$

Видно, что второе вхождение bb заменилось на ddd.

Если переставить формулы, НАМ тоже будет работать неправильно (легко проверить на этом же входном слове).

Правильная будет следующая НАМ:

$$\begin{cases} c \rightarrow & (1) \\ bb \mapsto ddd & (2) \end{cases}$$

Надо переставить формулы в первой НАМ и после вторую формулу заменить на заключительную. Проверим НАМ на этом же входном слове:

$$abbcabbca \xrightarrow{1} abbabbca \xrightarrow{1} abbabba \xrightarrow{2} adddabba$$

Проверим наш НАМ ещё на входном слове, в которое не входит bb:

$$dcacb \xrightarrow{1} dacb \xrightarrow{1} dab$$

Пример2 (перестановка символов)

A={a,b}. Преобразовать слово **P** так, чтобы в его начале оказались все символы **a**, а в конце – все символы **b**.

Например: babba → aabbb

Решение. Кажется, что для этой задачи нужен сложный НАМ. Однако это не так, задача решается с помощью одной формулы:

$$\begin{cases} ba \rightarrow ab \end{cases}$$

Например: babba → abbba → abbab → ababb → aabbb

Пример3 (использование спецзнака)

A={a,b}. Удалить из непустого слова **P** его первый символ. Пустое слово не менять.

Решение. На первый взгляд кажется, что эту задачу решает следующий НАМ:

$$\begin{cases} a \mapsto & (1) \\ b \mapsto & (2) \end{cases}$$

Однако это неправильный алгоритм, в чём можно убедиться, применив его к слову bbaba:

bbaba ↦ bbba

Как видно, НАМ удалил не первый символ слова, а первое вхождение символа **a**. Ясно, что перестановка формул тоже ничего не даст. Надо как-то зафиксировать (пометить) первый символ слова, например: ***ξ** ↦.

Итого, получаем следующий НАМ:

$$\begin{cases} \rightarrow * & (1) \\ *a \mapsto & (2) \\ *b \mapsto & (3) \end{cases}$$

Проверим его на том же входном слове:

$$bbaba \xrightarrow{1} *bbaba \xrightarrow{1} **bbaba \xrightarrow{1} ***bbaba$$

Постоянно буде приписываться *. Отсюда вытекает очень важное правило: если в НАМ есть формула с пустой левой частью (→β), то её место – только в самом конце НАМ.

$$\begin{cases} *a \mapsto & (1) \\ *b \mapsto & (2) \\ \to * & (3) \end{cases}$$

Проверим данный алгоритм:

$$bbaba \xrightarrow{3} \underline{*b}baba \mapsto_{2} baba$$

Для непустого слова всё в порядка, но наш алгоритм зациклится на пустом входном слове. Правильный будет следующий алгоритм:

$$\begin{cases} *a \mapsto & (1) \\ *b \mapsto & (2) \\ * \mapsto & (3) \\ \to * & (4) \end{cases}$$

Пример4 (фиксация спецзнаком заменяемого символа)
A={0,1,2,3}. Пусть **P** – непустое слово. Трактуя его как запись неотрицательного целого числа в четверичной системе счисления, требуется получить запись этого же числа, но в двоичной системе.
Например: 0123 → 00011011

Решение. Надо заменить: 0→00, 1→01, 2→10, 3→11. Кажется, что эту запись реализует следующая НАМ:

$$\begin{cases} 0 \to 00 & (1) \\ 1 \to 01 & (2) \\ 2 \to 10 & (3) \\ 3 \to 11 & (4) \end{cases}$$

Но этот алгоритм неправильный, в чём можно убедиться на входном слове

$$0123: \underline{0}123 \xrightarrow{1} \underline{0}0123 \xrightarrow{1} \underline{0}00123 \xrightarrow{1} \dots$$

Ошибка здесь в том, что после замены четверичной цифры на двоичную, уже нельзя отличить двоичные цифры от четверичных. Предлагается пометить слева спецзнаком * ту четверичную цифру, которая должна быть заменена на пару соответствующих двоичных цифр, а потом спецзнак надо поместить перед следующей четверичной цифрой:

0123 → *0123 → 00*123 → 0001*23 → 000110*3 → 00011011*

Итого, получаем следующий алгоритм перевода чисел из четверичной системы в двоичную:

$$\begin{cases} *0 \to 00* & (1) \\ *1 \to 01* & (2) \\ *2 \to 10* & (3) \\ *3 \to 11* & (4) \\ * \ \text{I}\!\to & (5) \\ \quad \to * & (6) \end{cases}$$

Проверим этот НАМ на входном слове 0123:

$$0123 \overset{6}{\to} \underline{*0}123 \overset{1}{\to} 00\underline{*1}23 \overset{2}{\to} 0001\underline{*2}3 \overset{3}{\to} 000110\underline{*3} \overset{4}{\to} 00011011\underline{*} \overset{5}{\text{I}\!\to} 00011011$$

Пример5 (перемещение спецзнака)

A={a,b}. Требуется приписать символ **a** к концу слова **P**.

Например: bbab → bbaba

Решение. Воспользуемся спецзнаком, который поместим в конец P, а затем заменим его на a. Чтобы поместить спецзнак в конец слова, сначала спецзнак * приписываем слева к слову P, а затем «перегоняем» звёздочку через все буквы слова. «Перепрыгивание» звёздочки – это формула *ξ→ξ*.

С учётом всего сказанного получаем следующий НАМ:

$$\begin{cases} *a \to a* \\ *b \to b* \\ * \ \text{I}\!\to a \\ \quad \to * \end{cases}$$

Отметим, что при пустом входном слове этот НАМ сначала введёт звёздочку, а затем тут же заменит её на символ a и остановится.

Пример6 (смена спецзнака)

A={a,b}. В слове **P** заменить на **aa** последнее вхождение символа **a**, если такое есть. Например: bababb → babaabb

Решение. Удвоение символа **a** реализуется формулой **a**I→**aa**. Чтобы она применялась не к первому вхождению символа **a**, а к последнему, надо поставить справа от последнего символа **a** спецзнак * и применить формулу удвоение символа.

Получаем следующий НАМ:

$$\begin{cases} *a \to a* & (1) \\ *b \to b* & (2) \\ b* \to *b & (3) \\ a* \ \text{I}\!\to aa & (4) \\ * \ \text{I}\!\to & (5) \\ \quad \to * & (6) \end{cases}$$

Проверим этот алгоритм на входном слове bababb (двойная стрелка означает несколько шагов применения формул (1) и (2)):

$$bababb \overset{6}{\to} * bababb \overset{1,2}{\Rightarrow} bababb\underline{*} \overset{3}{\to} babab\underline{*b} \overset{2}{\to} bababb\underline{*} \overset{3}{\to} babab\underline{*b} \overset{2}{\to} \ldots$$

Ошибка произошла из-за того, что мы используем спецзнак * как при движении вправо так и влево. Надо ввести еще один спецзнак #. Пусть * движется вправо, а # - влево. Тогда получим следующий НАМ:

$$\begin{cases} *a \to a* & (1) \\ *b \to b* & (2) \\ * \to \# & (3) \\ b\# \to \#b & (4) \\ a\# \vdash\!\to aa & (5) \\ \# \vdash\!\to & (6) \\ \to * & (7) \end{cases}$$

Проверим этот алгоритм на прежнем входном слове:

$$\text{bababb} \overset{7}{\to} *\,\text{bababb} \overset{1,2}{\Rightarrow} \text{bababb}\underline{*} \overset{3}{\to} \text{babab}\underline{b\#} \overset{4}{\to} \text{baba}\underline{b\#}\text{b} \overset{4}{\to} \text{bab}\underline{a\#}\text{bb} \overset{5}{\vdash\!\to} \text{babaabb}$$

Если во входное слово не входит символ a, тогда имеем:

$$\text{bb} \overset{7}{\to} \underline{*b}\text{b} \overset{2}{\to} \text{b}\underline{*b} \overset{2}{\to} \text{bb}\underline{*} \overset{3}{\to} \text{b}\underline{b\#} \overset{4}{\to} \underline{b\#}\text{b} \overset{4}{\to} \underline{\#}\text{bb} \overset{6}{\vdash\!\to} \text{bb}$$

Пример7 (перенос символа через слово)

A={a,b}. Перенести в конец непустого слова **P** его первый символ. Пустое слово не менять.

Например: bbaba → babab

Решение. Для решения этой задачи предлагается выполнить следующие действия.

1. Помечаем первый символ слова P спецзнаком *.

2. Заменяем * и этот символ на новый символ: a на A, b на B. Этим мы фактически вводим два новых спецзнака A и B, которые нужны, чтобы отличить первый символ слова от остальных символов при его переносе в конец слова.

3. Перегоняем новый символ A или B через все символы слова P в его конец. Такое перемещение реализуется аналогично перегону звёздочки – с помощью формул вида $A\xi \to \xi A$ и $B\xi \to \xi B$

4. Наконец, заменяем A или B в конце слова на прежний символ и останавливаем алгоритм.

Все действия реализуются в виде следующего НАМ:

$$\begin{cases} *a \to A & (1) \\ *b \to B & (2) \\ Aa \to aA & (3) \\ Ab \to bA & (4) \\ Ba \to aB & (5) \\ Bb \to bB & (6) \\ A \vdash\!\to a & (7) \\ B \vdash\!\to b & (8) \\ * \vdash\!\to & (9) \\ \to * & (10) \end{cases}$$

Проверим этот алгоритм на входном слове bbaba и на входном слове из одного символа:

$$\text{bbaba} \overset{10}{\to} \underline{*b}\text{baba} \overset{2}{\to} \underline{Bb}\text{aba} \overset{6}{\to} \text{b}\underline{Ba}\text{ba} \overset{5}{\to} \text{ba}\underline{Bb}\text{a} \overset{6}{\to} \text{bab}\underline{Ba} \overset{5}{\to} \text{baba}\underline{B} \overset{8}{\vdash\!\to} \text{babab}$$

10 1 7
a → *a → A ׀→ a

Замечание. В этом НАМ можно уменьшить число формул, если не вводить новые символы A и B, а использовать вместо них пары *a и *b. Это позволит исключить из НАМ формулы (1), (2), (7), (8). Что касается «перепрыгивания» через какой-то символ ξ, то оно реализуется формулами вида *aξ→ξ*a и *bξ→ξ*b.

Таким образом возможен следующий НАМ, решающий эту задачу:

*aa → a*a (1)
*ab → b*a (2)
*ba → a*b (3)
*bb → b*b (4)
* ׀→ (5)
→ * (6)

Проверим этот алгоритм на прежнем входном слове bbaba:

6 4 3 4 3 5
bbaba → *bbaba →b*baba → ba*bba → bab*ba → baba*b ׀→babab

Пример8 (использование нескольких спецзнаков)

A={a,b}. Удвоить слово **P**, т.е. приписать к **P** (слева или справа) его копию. Например: abb → abbabb

Решение. Предлагается следующий план решения задачи:

1. Приписываем к концу слова P символ =, справа от которого будем строить копию P.

2. Просматриваем по очереди все символы слова P и, не уничтожая их, переносим копию каждого символа в конец.

3. Удаляем символ =, который отделял слово P от его копии, и останавливаем алгоритм.

Предлагается следующий НАМ:

a → a (1)
b → b (2)
* → = (3)
Aa → aA (4)
Ab → bA (5)
A= → =A (6)
A → a (7)
Ba → aB (8)
Bb → bB (9)
B= → =B (10)
B → b (11)
#a → a#A (12)
#b → b#B (13)
#= ׀→ (14)
→ #* (15)

Проверим данный алгоритм на двух входных словах – на пустом и на abb:

15 3 14
<пустое слово> → #* → #= |→ (т.е получили <пустое слово><пустое слово>)

15 1,2 3 12 4-6 8 12 8-10
abb → #*abb → #abb* → #abb= → a#Abb= → a#bb=A → a#bb=a → ab#Bb= →

11 12 8-10 11 14
ab#b=aB → ab#b=ab → abb#B=ab → abb#=abB → abb#=abb |→ abbabb

Эту задачу можно решить другим способом:
1. Сначала за каждой (малой) буквой входного слова вставляем её двойник – соответствующую большую букву.
2. В новом слове переставляем малые и большие буквы так, чтобы слева оказались все малые буквы, а справа – все большие. Потом заменяем все большие буквы на малые.
Эти действия описываются в виде следующего НАМ:

a → aA
b → bB
* → #
Aa → aA
Ab → bA
Ba → aB
Bb → bB
A# → #a
B# → #b
|→
→ *

2.3 Задачи для самостоятельного решения

2.1 A={f,h,p}. В слове P заменить все пары ph на f.
2.2 A={f,h,p}. В слове P заменить на f только первую пару ph, если такая есть.
2.3 A={a,b,c}. Приписать слово bac слева к слову P.
2.4 A={a,b,c}. Заменить слово P на пустое слово, т.е удалить все символы.
2.5 A={a,b,c}. Заменить любое входное слово на слово a.
2.6 Выписать НАМ, не меняющий входное слово (при любом алфавите A).
2.7 A={|}. Считая слово P записью числа в единичной системе счисления, получить остаток от деления этого числа на 2, т.е. получить слово из одной палочки, если число нечётно, или пустое слово, если число чётно.
2.8 A={|}. Считая слово P записью положительного числа в единичной системе счисления, уменьшить это число на 1.
2.9 A={|}. Считая слово P записью числа в единичной системе счисления, увеличить это число на 2.
2.10 A={0,1,2}. Считая слово P записью числа в троичной системе счисления, получить остаток от деления этого числа на 2, т.е. получить слово 1, если число нечётно, или слово 0, если число чётно. (Замечание: в чётном троичном числе должно быть чётное количество цифр 1.)

2.11 A={a,b,c}. Определить, входит ли символ a в слово P. Ответ (выходное слово): слово a, если входит, или пустое слово, если не входит.
2.12 A={a,b}. Если в слово P входит больше символов a, чем символов b, то в качестве ответа выдать слово из одного символа a, если в P равное количество a и b, то в качестве ответа выдать пустое слово, а иначе выдать ответ b.
2.13 A={0,1,2,3}. Преобразовать слово P так, чтобы сначала шли все чётные цифры (0 и 2), а затем – все нечётные.
2.14 A={a,b,c}. Преобразовать слово P так, чтобы сначала шли все символы a, затем – все символы b и в конце – все символы c.
2.15 A={a,b,c}. Определить, из скольких различных символов составлено слово P; ответ получить в единичной системе счисления (например: acaac→||).

3 Рекурсивные функции

Вычислимые функции – числовые функции, значения которых можно вычислять посредством единого для данной функции алгоритма.

Арифметические функции – функции, области определения и значений которых целые неотрицательные числа, то есть натуральный ряд + число ноль.

Частичные арифметические функции – арифметические функции с ограниченной областью определения, остальные **– всюду определенными.**

Примитивно-рекурсивные функции

В качестве простейших функций в теории рекурсивных функций приняты следующие:

1. $O(x) = 0$ – константа «ноль».

2. $S(x) = x + 1$ – « последователь ».

3. $I_m^n(x_1, x_2, ..., x_n) = x_m; (m \le n)$ – функция тождества или выбора аргумента, проекция.

Оператор суперпозиции (подстановки) S_m^n **–** подстановка в функцию от m переменных m функций от n переменных, что дает новую функцию от n переменных.

Суперпозицией функций $g(y_1, y_2, ..., y_m)$ и $f_1, f_2, ..., f_m$ называют функцию:

$$h(x_1, x_2, ..., x_n) = g\left[f_1(x_1, x_2, ..., x_n), ..., f_m(x_1, x_2, ..., x_n)\right];$$

$$h = S_m^n(g, f_1, f_2, .., f_m).$$

Оператор примитивной рекурсии R_n**,** определяющий значение функции f, записывается в виде следующей схемы:

$$R_n : \begin{cases} f(x_1, x_2, ..., x_n; 0) = g(x_1, x_2, ..., x_n) \\ f(x_1, x_2, ..., x_n; y+1) = h[x_1, x_2, ..., x_n; y; f(x_1, x_2, ..., x_n; y)]. \end{cases}$$

Примитивно-рекурсивная функция – арифметическая функция, которая может быть получена из простейших с помощью конечного числа применений операторов суперпозиции и примитивной рекурсии.

Примитивно-рекурсивные функции являются всюду определенными.

Пример 1. Константа a получается путем суперпозиции функций $S(x)$ и $O(x)$:

$$a = \underbrace{S(S(\ldots S}_{a}(0))\ldots).$$

Пример 2. Операция сложения $f_+(x,y) = x + y$ может быть определена с помощью оператора примитивной рекурсии:

$$\begin{cases} f_+(x,0) = x = I_1^2(x,y) \\ f_+(x,y+1) = x + (y+1) = (x+y) + 1 = f_+(x,y) + 1 = S(f_+(x,y)). \end{cases}$$

Таким образом, функция $f_+(x,y)$ получена из простейших $I_1^2(x,y)$ и $S(x)$ путем применения оператора примитивной рекурсии, что соответствует определению примитивно-рекурсивной функции.

Пример 3. Примитивная рекурсивность операции умножения $f_\times(x,y) = x \times y$ доказывается с использованием сложения:

$$\begin{cases} f_\times(x,0) = 0 = O(x,y); \\ f_\times(x,y+1) = x \times (y+1) = x \times y + x = f_\times(x,y) + x \\ = f_+(f_\times(x,y), I_1^2(x,y)) \end{cases}$$

Пример 4. Примитивная рекурсивность операции возведения в степень $f_\wedge(x,y) = x^y$ доказывается следующим образом:

$$\begin{cases} f_\wedge(x,0) = 1; \\ f_\wedge(x,y+1) = x^{y+1} = x^y \cdot x^1 = f_\wedge(x,y) \cdot x = f_\times(f_\wedge(x,y), I_1^2(x,y)) \end{cases}$$

Пример 5. Операция вычитания не является примитивно-рекурсивной, т.к. она не всюду определена: результат операции a-b при $b > a$ не определен в области натуральных чисел. Однако примитивно-рекурсивной является так называемое арифметическое (усеченное) вычитание или разность.

Арифметическое вычитание:

$$f_{\dot{-}}(x,y) = x \dot{-} y = \begin{cases} x - y, & x > y \\ 0, & x \le y. \end{cases}$$

Для доказательства примитивной рекурсивности $f_{\dot{-}}(x,y)$ вначале рассмотрим операцию $x \dot{-} 1$: $f_{\dot{-}1}(x) = x \dot{-} 1 = \begin{cases} x-1, & x > 1 \\ 0, & x \le 1 \end{cases} \Rightarrow$;

$$\Rightarrow \begin{cases} f_{\dot{-}1}(0) = x \dot{-} 1\big|_{x=0} = 0 \dot{-} 1 = 0 \\ f_{\dot{-}1}(x+1) = \begin{cases} (x+1)-1, & x+1 > 1 \\ 0, & x+1 \le 1 \end{cases} = \begin{cases} x, & x > 0 \\ 0, & x = 0 \end{cases} = x = I_1^1(x) \end{cases}$$

$$\Rightarrow \quad \begin{cases} f_{\dot{-}1}(0) = 0 \dot{-} 1 = 0 \\ f_{\dot{-}1}(x+1) = x. \end{cases}$$

т.е. операция $x \dot{-} 1$ – примитивно-рекурсивна.

Дополнительное свойство:

$x \dot{-} (y+z) = (x \dot{-} y) \dot{-} z$.

$$\begin{cases} f_{\dot{-}}(x,0) = x = I_1^2(x,y); \\ f_{\dot{-}}(x,y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = f_{\dot{-}1}(x,y), \end{cases}$$

арифметическое вычитание – примитивно-рекурсивно.

Пример 6. Функция $sg(x)$ – аналог функции $sign(x)$ для натуральных чисел.

$$sg(x) = \begin{cases} 0, & x = 0, \\ 1, & x \neq 0. \end{cases}$$

Функция $sg(x)$ примитивно-рекурсивна:

$$\begin{cases} sg(0) = 0, \\ sg(x+1) = 1. \end{cases}$$

$\overline{sg(x)}$ – антисигнум, функция обратная $sg(x)$.

$$\overline{sg(x)} = 1 \dot{-} sg(x) = \begin{cases} 1, & x = 0 \\ 0, & x > 0 \end{cases}.$$

Пример 7. Примитивная рекурсивность функций $max(x,y)$, $min(x,y)$ и модуль двух чисел доказывается с помощью арифметического вычитания:

$max(x,y) = y + (x \dot{-} y) = x + (y \dot{-} x)$,

$min(x,y) = x \dot{-} (x \dot{-} y) = y \dot{-} (y \dot{-} x)$,

$|x - y| = (x \dot{-} y) + (y \dot{-} x)$.

Отношение $R(x_1, x_2, ..., x_n)$ называется **примитивно-рекурсивным**, если примитивно-рекурсивна его характеристическая функция χ_R:

$$\chi_{R(x_1,x_2,...,x_n)} = \begin{cases} 1, (x_1,x_2,...,x_n) \in R \\ 0, (x_1,x_2,...,x_n) \notin R. \end{cases}$$

Пример 8. Отношение $x = y$ – примитивно-рекурсивно.

Действительно, $\chi_{=}(x,y) = \begin{cases} 1, & x = y \\ 0, & x \neq y \end{cases} = \overline{sg}|(x-y)|$.

Отношение $x > y$ примитивно-рекурсивно.

Действительно, $\chi_{>}(x,y) = \begin{cases} 1, & x > y \\ 0, & x \leq y \end{cases} = sg(x \dot{-} y)$.

Отношение $x \geq y$ примитивно-рекурсивно.

Действительно, $\chi_{\geq}(x,y) = \chi_{>}(x,y) + \chi_{=}(x,y)$..

Оператор минимизации (μ-оператор, оператор наименьшего корня) определяет новую арифметическую функцию $f(x_1,x_2,...,x_n)$ от n переменных с помощью ранее построенной арифметической функции $g(x_1,x_2,...,x_n;y)$ от n+1 переменных. Пусть существует некоторый механизм вычисления функции g, причем значение функции f неопределенно, если этот механизм работает бесконечно, не выдавая никакого определенного значения.

Зафиксируем набор значений аргументов $x_1,x_2,...,x_n$ и рассмотрим уравнение относительно y: $g(x_1,x_2,...,x_n;y) = 0$; чтобы найти решение этого уравнения, натуральное y, будем вычислять последовательность значений:

$g(x_1,x_2,...,x_n;y) = 0$ для $y = 0,1,....$

Наименьшее целое неотрицательное значение $y = a$, удовлетворяющее этому уравнению: $g(x_1,x_2,...,x_n,a) = 0$ обозначим:

$\mu_y(g(x_1,x_2,...,x_n;y) = 0)$.

Говорят, что функция $f(x_1,x_2,...,x_n)$ получена из функции $g(x_1,x_2,...,x_n;y)$ **операцией минимизации**, если:

$f(x_1,x_2,...,x_n) = \mu_y(g(x_1,x_2,...,x_n;y) = 0)$..

Оператор минимизации работает бесконечно в одном из следующих случаев:

1) значение $g(x_1,x_2,...,x_n,0)$ не определено;

2) значение $g(x_1,x_2,...,x_n,y)$ для $y = 0,1,2,....,i-1$ определены, но не равны нулю, а значение $g(x_1,x_2,...,x_n,i)$ – не определено;

3) значение $g(x_1,x_2,...,x_n,y)$ определены для всех $y=0,1,2,....i-1$, но не равны нулю.

Оператор минимизации является удобным средством получения обратных функций: вычитание, деление, извлечение корня и так далее.

Пример 9. Определение операции «вычитание»:

$f_{-}(x,y) = x - y = \mu_z(y + z = x)$.

Пример 10. Определение операции «деление»:

$f_{/}(x,y) = x/y = \mu_z(y \cdot z = x)$.

Пример 11. Определение операции « извлечение корня »:

$f_{\sqrt{\ }}(x,y) = \sqrt[x]{y} = \mu_z(z^x = y)$.

Пример 12. Определение операции « логарифм »:

$f_{log}(x,y) = {}^{log_x}y = \mu_z(x^z = y)$

Пример 13. Процесс вычисления функции с помощью оператора минимизации приведен ниже:

$$f(x,y) = x^2 \dot{-} \mu_z \left(\left[\frac{x+y}{x+z} \right] - z = 0 \right)$$

$$f(6,34) = ?$$

$$f(6,34) = 36 \dot{-} \mu_z \left(\left[\frac{40}{6+z} \right] - z = 0 \right)$$

$$z = 0$$

$$f(6,34) = 36 \dot{-} \mu_z \left(\left[\frac{40}{6+0} \right] - 0 = 4 - 0 \neq 0 \right)$$

$$z = 1$$

$$f(6,34) = 36 \dot{-} \mu_z \left(\left[\frac{40}{6+1} \right] - 1 \neq 0 \right)$$

$$z = 2$$

$$f(6,34) = 36 \dot{-} \mu_z \left(\left[\frac{40}{6+2} \right] - 2 \neq 0 \right)$$

$$z = 3$$

$$f(6,34) = 36 \dot{-} \mu_z \left(\left[\frac{40}{6+3} \right] - 3 \neq 0 \right)$$

$$z = 4$$

$$f(6,34) = 36 \dot{-} \mu_z \left(\left[\frac{40}{6+4} \right] - 4 = 0 \right)$$

$$f(6,34) = 36 \dot{-} 4 = 32$$

Частично-рекурсивная функция – функция, которая может быть построена из простейших с помощью конечного числа применений оператора суперпозиции, примитивной рекурсии и минимизации.

Частично-рекурсивная функция является не всюду определенной, причем там, где она не определена, процесс ее вычисления продолжается бесконечно.

Общерекурсивная функция – всюду определенная частично-рекурсивная функция.

Связь между алгоритмами и рекурсивными функциями выражается **тезисом Черча**: какова бы ни была вычислимая неотрицательная целочисленная функция от неотрицательных целочисленных аргументов, существует тождественно равная ей частично-рекурсивная функция.

3.1 Задачи для самостоятельного решения

1. Разработать алгоритм вычисления $f(n)$ в виде рекурсивной функции.
2. Проверить модель алгоритма на множестве тестовых примеров.

3. Определить к какому классу рекурсивных функций принадлежит $f(n)$: примитивно-рекурсивна, частично-рекурсивна или общерекурсивна.

Варианты заданий

1. Сумма всех четных делителей числа n.
2. Количество всех нечетных делителей числа n.
3. Количество нулей в двоичной записи n.
4. Сумма цифр в двоичной записи n.
5. Количество взаимно-простых с n чисел, $\le n$.
6. Максимальная цифра в 8-ричной записи числа n.
7. Минимальная цифра в 8-ричной записи числа n.
8. Количество четных цифр в 8-ричной записи числа n.
9. Количество нечетных цифр в 8-ричной записи числа n.
10. Сумма простых делителей числа n.
11. Количество простых делителей числа n.
12. Количество простых чисел, $\le n$.
13. Количество чисел, являющихся полными квадратами, $\le n$.
14. Сумма чисел, являющихся степенью двойки, $\le n$.
15. Максимальная цифра в 16-ричной записи числа n.

4. Задачи теоретического характера

Примем следующие обозначения:

Последовательность из **n** подряд идущих символов **a** будем обозначать $\mathbf{a^n}$; например $\mathbf{a^0}$ – это пустое слово, $\mathbf{a^1}$ – это a, $\mathbf{a^4}$ – это аааа и т.д.

4.1 Применимость алгоритма

Алгоритм называется **применимым** к слову, если, начав работать над этим словом как входным, он остановится через конечное число шагов. Если алгоритм зацикливается, то он **неприменим** к этому слову Если алгоритм Н применим к слову Р, тогда результат применения Н к Р будем обозначать Н(Р).

Область применимости алгоритма относительно некоторого алфавита – это множество всех таких слов в этом алфавите, к которым применим алгоритм.

Пример1

Определить область применимости следующего НАМ относительно алфавита {a,b}:

$$\begin{cases} b \to b & (1) \\ a \mapsto b & (2) \end{cases}$$

Решение. Формула (1) применима к любому входному слову, в которое входит хотя бы один символ b, причем она не меняет это слово. Поэтому на таких словах данный НАМ зацикливается. Если во входном слове нет символов b, но есть хотя бы один символ a, тогда формула (1) не будет работать, а сработает заключительная формула (2), которая остановит алгоритм. Следовательно, НАМ останавливается на словах, состоящих, только из символов a. Для пустого слова обе формулы неприменимы, поэтому НАМ сразу остановтися.Ответ: область применимости НАМ – все слова вида a^n ($n \ge 0$).

Пример2

Построить НАМ, который применим ко всем словам в алфавите {a,b,c}, кроме двух слов – abc и baac.

Решение. Из условия задачи следует, что НАМ должен зацикливаться на двух указанных словах. Однако это не значит, что задачу решает следующий НАМ

$$\begin{cases} abc \to abc \\ baac \to baac \end{cases}$$

Дело в том, что данный алгоритм зацикливается не только на этих двух словах, но и на любых словах, в которые они входят как подслова, например, на слове ccabcbb. Обычно в такой ситуации поступают следующим образом: начало и конец входного слова P помечают какими-то спецзнаками (например, *P#), а затем используют формулы, в левой части которых указывают нужные слова и эти спецзнаки (*abc# и *baac#). Такие формулы будут применимы только к нужным входным словам.

В нашем примере эти формулы должны зацикливать алгоритм, а для останова его на других словах можно использовать, например, формулу *|→. Итого, получаем:

$$\begin{cases} \#a \to a\# \\ \#b \to b\# \\ \#c \to c\# \\ *abc\# \to *abc\# \\ *baac\# \to *baac\# \\ * \;\; |\to \\ \quad \to *\# \end{cases}$$

4.2 Самоприменимость алгоритма

Записью НАМ называется слово, состоящее из последовательно записанных через точку с запятой формул подстановки этого алгоритма (точки с запятой отделяют друг от друга соседние формулы). Точки с запятой не входят в формулы.

Пусть имеются следующие алгоритмы Н1 и Н2:

$$H1: \begin{cases} \#a \to \# \quad (1) \\ \# \;\; |\to \quad (2) \\ \quad \to \# \quad (3) \end{cases} \qquad H2: \begin{cases} a \to b \quad (4) \\ b \to bb \quad (5) \end{cases}$$

Тогда записью алгоритма Н1 является слово

#a → # ; # |→ ; → # а записью алгоритма Н2 – слово a → b ; b → bb

Понятно, что алгоритм однозначно определяет свою запись и наоборот.

Алгоритм называется **самоприменимым**, если он применим к своей записи, и **несамоприменимым** в противном случае.

Например, алгоритм Н1 самоприменим, т.к. начав работать над своей записью как входным словом, он остановится:

$$\underline{\#a}\to\#;\#|\to;\to\# \;\xrightarrow{1}\; \underline{\#}\to\#;\#|\to;\to\# \;\xrightarrow{2}\; |\to \;\to\#;\#|\to;\to\#$$

Алгоритм Н2 несамоприменим, т.к. он зацикливается на своей записи:

$$\underline{a}\to b;b\to bb \xrightarrow{4} \underline{b}\to b;b\to bb \xrightarrow{5} \underline{b}b\to b;b\to bb \xrightarrow{5} \underline{b}bb\to b;b\to bb \xrightarrow{5} \dots$$

Отметим, что если применимость зависит как от алгоритма, так и от слова, то самоприменимость зависит только от алгоритма, от того, применим ли этот алгоритм к конкретному слову – к своей собственной записи, которая однозначно определяется данным алгоритмом.

Пример3

Построить НАМ, который несамоприменим, но применим к любому слову в алфавите {a,b}.

Решение. Поскольку искомый НАМ применим ко всем словам, составленным из символов a и b, то зацикливаться он может лишь на словах, содержащих какой-то дополнительный символ, скажем *. Следовательно, чтобы НАМ оказался несамоприменимым (зацикливался на своей записи), символ * должен входить в запись алгоритма и на этом символе НАМ должен циклиться.

Например: { * → *

Пример4

Известно, что проблема самоприменимости алгоритмически неразрешима, т.е. не существует единого способа (алгоритма), который позволял бы определять для любого алгоритма, самоприменим он или нет. Однако в частных случаях (для некоторых классов алгоритмов) эта проблема может оказаться разрешимой. Например, проблема самоприменимости для НАМ, содержащих только одну формулу подстановки, разрешима. Требуется доказать это утверждение.

Доказательство. Пусть НАМ имеет вид {α|→β или {α→β. Тогда можно использовать следующий метод проверки самоприменимости: если единственная формула алгоритма заключительная или если это обычная формула, левая часть (α) которой не входит в правую часть (β), то такой НАМ самоприменим, иначе он несамоприменим.

Действительно, какой бы ни была единственная формула - заключительной или обычной, на первом шаге применения НАМ к его записи (к слову α|→β или α→β) часть α в этой записи будет заменена на β, в результате чего получится слово β|→β или β→β. Если формула заключительная, то на этом НАМ и остановится, а это значит, что он самоприменим. Если формула обычная и α не входит в β, то формула неприменима к получившемуся слову и работа НАМ прекращается, поэтому и в данном случае алгоритм самоприменим. Но если формула обычная и α входит в β, то в получившемся слове β→β будет присутствовать α, поэтому наша формула снова применяется, причем в новом слове по-прежнему окажется α. Получаем бесконечный процесс подстановок, а это и значит, что НАМ несамоприменим.

4.3 Эквивалентность алгоритмов

Эквивалентными называют алгоритмы, которые предлагают разные способы решения одной и той же задачи. Это значит, что на одинаковых входных словах они выдают одинаковые результаты. При этом надо учиты-

вать, если один алгоритм зацикливается, то и эквивалентный ему алгоритм должен зацикливаться.

Точное определение: алгоритмы Н1 и Н2 **эквивалентны относительно алфавита** А, если области применимости Н1 и Н2 совпадают и для любого слова Р из этой области выполняется равенство Н1(Р)=Н2(Р).

Замечание. Одни и те же алгоритмы могут быть эквивалентными относительно одного алфавита и не эквивалентными относительно другого алфавита. Например алгоритмы

Н1: $\{a \mapsto a$ Н2: $\begin{cases} a \mapsto a \\ b \to b \end{cases}$

эквивалентны относительно алфавита {a} и не эквивалентны относительно алфавита {a,b}: слова в алфавите {a} они не меняют, но если в входное слово входят только символы b, то Н1 остановится, а Н2 зациклится.

Пример5

Определить, эквивалентны ли следующие пары НАМ относительно алфавита {a,b}

1) $H1: \{a \mapsto b \qquad H2: \begin{cases} *b \to b* \\ *a \mapsto b \\ \to * \end{cases}$

2) $H3: \begin{cases} a \to b \\ b \to a \end{cases} \qquad H4: \begin{cases} a \to aa \\ b \to bb \end{cases}$

3) $H5: \{aa \to a \qquad H6: \begin{cases} *aa \to a* \\ *b \to b* \\ * \mapsto \\ \to * \end{cases}$

Решение. При проверке алгоритмов на эквивалентность надо прежде всего установить, совпадают ли области их применимости.

В случае алгоритмов Н1 и Н2, которые первое вхождение символа а заменяют на b, это условие не выполняется: Н1 применим ко всем словам из символов а и b, а Н2 зацикливается на словах, не содержащих а. Значит, алгоритмы Н1 и Н2 не эквивалентны.

Если области применимости совпадают, тогда надо показать, что на одних и тех же словах из данной области эти алгоритмы выдают одинаковые результаты.

У пары алгоритмов Н3 и Н4 одна и та же область применимости – она состоит только из одного пустого слова. На непустых же словах эти алгоритмы зацикливаются, причём зацикливаются по разному: Н3 постоянно меняет а на b и b на а, тогда как Н4 всё время добавляет новый символ а или b. Однако такое различное поведение при зацикливании не играет никакой роли при определении эквивалентности алгоритмов. Важно лишь, чтобы в случае останова алгоритмы выдавали одинаковые выходные слова. А для пары Н3 и

Н4 это условие выполняется: на единственном слове (пустом), к которому они применимы, они выдают один и тот же ответ – пустое слово. Значит, алгоритмы Н3 и Н4 эквивалентны.

Теперь рассмотрим пару алгоритмов Н5 и Н6. У них одна и та же область применимости – это множество всех слов в алфавите {a,b}. Однако второе условие эквивалентности (одинаковые результаты при одинаковых исходных данных) не выполняется. Чтобы доказать это, достаточно привести лишь одно слово, на котором алгоритмы выдают разные ответы. Таким словом может быть, например, слово аааа:

Н5: ааааа → ааа → аа → а

Н6: аааа → *аааа → а*аа → аа* ↦ аа

Итак, алгоритмы Н5 и Н6 не эквивалентны.

4.4 Композиция алгоритмов

Как известно, к результату одной функции можно применить другую функцию, например: sin(ctgx). Точно так же выходное слово одного алгоритма Н1 можно подать на вход другому алгоритму Н2. Такое последовательное выполнение сначала алгоритма Н1, а затем алгоритма Н2 называется композицией этих алгоритмов. При этом надо учитывать, что любой из этих алгоритмов может зацикливаться, тогда должна зацикливаться и их композиция.

Композицией алгоритмов Н1 и Н2 относительно алфавита А называется такой алгоритм Н (обозначается Н1°Н2 или Н2(Н1)), что для любого слова Р в алфавите А выполняются следующие условия:

1) если Н1 неприменим к Р, то и Н неприменим к Р;

2) если Н1 применим к Р, но Н2 неприменим к слову Н1(Р), то и Н неприменим к Р;

3) если Н1 применим к Р и Н2 применим к слову Н1(Р), то Н применим к Р, причём выполняется равенство Н(Р)=Н2(Н1(Р)).

Доказана следующая теорема: для любых нормальных алгоритмов Н1 и Н2 существует нормальный алгоритм Н, который является (относительно соответствующего алфавита) композицией Н1 и Н2. (Аналогичная теорема верна и для машины Тьюринга.)

Пример6

Построить нормальный алгоритм Н, являющийся композицией указанных нормальных алгоритмов Н1 и Н2 (Н=Н2(Н1)) относительно алфавита {a,b}:

$$H1:\begin{cases} *a \to * \\ *b \to b* \\ * \mapsto \\ \to * \end{cases} \qquad H2:\begin{cases} b \to a \end{cases}$$

Решение. Прежде всего отметим, что в общем случае нельзя строить композицию Н простым выписыванием друг за другом формул подстановки из алгоритмов Н1 и Н2. Например, если сначала выписать формулы из Н1, а затем из Н2, то получим алгоритм Н12, а если сначала выписать формулы из Н2, а затем из Н1, то получим алгоритм Н21:

$$H12:\begin{cases} *a \to * \\ *b \to b* \\ * \mapsto \\ \quad \to * \\ b \to a \end{cases} \qquad H21:\begin{cases} b \to a \\ *a \to * \\ *b \to b* \\ * \mapsto \\ \quad \to * \end{cases}$$

Таким образом, ни H12, ни H21 не является композицией алгоритмов H1 и H2. Например, для входного слова abb имеем:

H12: abb → *abb → *bb → b*b → bb* ↦ bb

H21: abb → aab → aaa → *aaa → *aa → *a → * ↦

тогда как H2(H1(abb))=H2(bb)=aa.

Замечание. Существует способ построения списка формул для композиции H по спискам формул из H1 и H2, что и доказывает указанную выше теорему. Однако в виду громоздкости, мы будем использовать другой подход.

В нашем примере алгоритм H1 удаляет из входного слова все символы a, алгоритм H2 заменяет все b на a. Значит, последовательное применение сначала H1, а затем H2 приводит к тому, что из входного слова удаляются все символы a, после чего оставшиеся символы b заменяются на a (без последующего удаления их). НАМ, решающий такую задачу, может быть, например, следующим:

$$H:\begin{cases} *a \to * \\ *b \to a* \\ * \mapsto \\ \quad \to * \end{cases}$$

Это и есть композиция заданных алгоритмов H1 и H2.

4.5 Задачи для самостоятельного решения

4.1 Из указанного НАМ вычеркнуть ровно одну формулу подстановки так, чтобы получился алгоритм, применимый ко всем словам в алфавите {a,b}:

$$\begin{cases} a \to b \\ ba \to aba \\ b \to a \end{cases}$$

4.2 Определить область применимости указанного НАМ относительно алфавита {a,b,c}:

$$а)\begin{cases} a \to b \\ b \to c \\ c \to a \end{cases} \qquad б)\begin{cases} a \to \\ bb \to b \\ ccc \to cc \end{cases} \qquad в)\begin{cases} a \to \\ b \to b \\ c \to \end{cases} \qquad г)\begin{cases} a \to c \\ b \to a \\ cc \to \\ c \to c \end{cases} \qquad д)\begin{cases} ba \to ab \\ ca \to ac \\ cb \to bc \\ abc \mapsto \\ \quad \to \end{cases}$$

4.3 Определить область применимости указанного НАМ относительно алфавита {a,b}:

$$а)\begin{cases} *ab \to ab \\ *a \to * \\ *b \to * \\ * \quad \mapsto \\ \quad \to * \end{cases} \qquad б)\begin{cases} ab \to ba \\ ba \mapsto ab \\ b \quad \to bb \end{cases} \qquad в)\begin{cases} aa \to ba \\ b \quad \to a \end{cases}$$

4.4 Построить НАМ, который из всех слов в алфавите {a,b,c} применим только к двум словам – пустому слову и слову abccba.

4.5 Построить НАМ, в котором не более 5 формул подстановки и который из всех слов в алфавите {a,b} применим только к словам, имеющим следующий вид:

а) $a^n b^n$, где $n \geq 0$ б) $a^n b^m$, где $n \neq m$, $n \geq 0$, $m \geq 0$

в) $a^n b^m$, где $n \geq m \geq 0$ г) $a^n b^m$, где $n > m \geq 0$

4.6 Построить НАМ, в котором не более 5 формул подстановки и который из всех слов в алфавите {a,b,c} применим только к тем словам, длина которых:

а) кратна 5 б) не кратна 5

в) больше 4 г) меньше 4

д) равна 3 е) не равна 3

4.7 Построить НАМ, в котором не более 3 формул подстановки и который из всех слов в алфавите {a,b} применим только к тем словам, для которых выполняется хотя бы одно из следующих двух условий: 1) в слове меньше трёх символов a; 2) число символов b кратно 3.

4.8 Пусть в каждой формуле подстановки некоторого НАМ число символов в левой части строго больше числа символов в правой части. Доказать, что такой НАМ применим к любому слову.

4.9 Определить, самоприменим ли указанный НАМ:

$$а)\{\to \qquad б)\{\mapsto \qquad в)\begin{cases} a \to b \\ b \to a \end{cases} \qquad г)\begin{cases} b \to a \\ ab \to ab \\ aa \mapsto \end{cases} \qquad д)\begin{cases} b \to a \\ aa \to b \end{cases}$$

4.10 A={a,b}. Построить НАМ, в котором не более 4 формул подстановки и который:

а) самоприменим и применим ко всем словам в алфавите A;

б) самоприменим, но неприменим ко всем словам в алфавите A;

в) самоприменим и применим только к какому-то одному слову в алфавите A;

г) самоприменим и неприменим только к какому-то одному слову в алфавите A;

д) несамоприменим и неприменим ко всем словам в алфавите A;

е) несамоприменим и применим только к какому-то одному слову в алфавите A;

ж) несамоприменим и неприменим только к какому-то одному слову в алфавите A.

4.11 A={a,b}. Построить НАМ, в котором не более 3 формул подстановки и который:

а) самоприменим и применим только к тем словам в алфавите А, которые содержат хотя бы один символ а;

б) самоприменим и неприменим только к тем словам в алфавите А, которые содержат хотя бы один символ а;

в) несамоприменим и применим только к тем словам в алфавите А, которые содержат хотя бы один символ а;

г) несамоприменим и неприменим только к тем словам в алфавите А, которые содержат хотя бы один символ а.

4.12 Пусть в некотором НАМ все формулы являются обычными и есть формула с пустой левой частью. Что можно сказать об области применимости этого НАМ? Может ли такой НАМ быть самоприменимым? (Ответ обосновать.)

4.13 Пусть в левых и правых частях всех формул подстановки некоторого НАМ используются только символы а. Установить, может ли такой НАМ быть:

а) самоприменимым и неприменимым ни к одному слову в алфавите {а};

б) несамоприменимым и применимым ко всем словам в алфавите {а}.

4.14 Для каждой указанной пары алгоритмов Н1 и Н2 определить, эквивалентны ли они относительно алфавита {a,b}:

$$а)\ H1:\{a \to a \qquad H2:\begin{cases} *a \to aa \\ a \ \ \to *a \end{cases}$$

$$б)\ H1:\begin{cases} ba \to ab \\ ab \to \end{cases} \qquad H2:\begin{cases} ab \to \\ ba \to \end{cases}$$

$$в)\ H1:\{ab \to \qquad H2:\begin{cases} *ab \to * \\ *a \ \to a* \\ *b \ \to b* \\ * \ \ \mapsto \\ \ \ \to * \end{cases}$$

$$г)\ H1:\{ab \to \qquad H2:\begin{cases} *ab \to \\ *a \ \to a* \\ *b \ \to b* \\ * \ \ \mapsto \\ \ \ \to * \end{cases}$$

$$д)H1:\begin{cases} *b \to b* \\ *a \mapsto \\ * \mapsto \\ a \to a* \end{cases} \qquad H2:\begin{cases} *a \to a\# \\ *b \to b* \\ \#a \mapsto \\ \#b \to b* \\ \# \mapsto \\ \to * \end{cases}$$

4.15 Для указанного НАМ построить такой НАМ, который эквивалентен ему относительно алфавита {a,b} и содержит только одну формулу подстановки:

$$а)\begin{cases} +a \to a+ \\ +b \to b+ \\ + \to * \\ a* \to *a \\ b* \to *b \\ * \mapsto a \\ \to + \end{cases} \qquad б)\begin{cases} *a \to b* \\ *b \to b* \\ * \mapsto \\ \to * \end{cases} \qquad в)\begin{cases} *a \to * \\ *b \to b* \\ * \to \\ \to * \end{cases}$$

Список литературы

1. Алферова З.В. Теория алгоритмов. – М.: Статистика, 1973. – 164с.
2. Мальцев А.И. Алгоритмы и рекурсивные функции. – М.: Наука, 1960. – 392с.
3. Марков А.А., Нагорный Н.М. Теория алгорифмов. – М.: ФАЗИСТ, 1996. – 448с.