МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
О.Г. Локтионова
«6» 09 2021 г.

ТЕОРИЯ АЛГОРИТМОВ

Методические рекомендации
к практическим занятиям
по дисциплине «Математическая логика и теория алгоритмов»
для студентов направления подготовки
09.03.01 Информатика и вычислительная техника

Курск 2021

УДК 510.6
Составители: С. В. Дегтярев, Е.Н. Иванова

Рецензент
Доцент кафедры программной инженерии,
кандидат технических наук
Ю.А. Халин

Теория алгоритмов : методические указания к практическим занятиям по дисциплине «Математическая логика и теория алгоритмов» для студентов направления подготовки 09.03.01 Информатика и вычислительная техника / Юго-Зап. гос. ун-т; сост.: С.В. Дегтярев, Е.Н. Иванова. – Курск, 2021. - 29 с. - Библиограф.: с. 29.

Рассмотрены теоретические основы теории алгоритмов: примитивно и частично-рекурсивные функции, нормальный алгоритмы Маркова, машина Тьюринга-Поста, вопросы вычислительной сложности алгоритмов. Теоретический материал поясняется примерами, завершается вопросами для самопроверки, заданиями для самостоятельного выполнения.

Предназначены для студентов направления 09.03.01 Информатика и вычислительная техника очной и заочной форм обучения.

Текст печатается в авторской редакции

Подписано в печать 6.08.21 . Формат 60x84 1/16.
Усл.печ.л. Уч.-изд.л. . Тираж 20 экз. Заказ 1044 . Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

Цель практических занятий

Изучить основные понятия теории алгоритмов; овладеть методами идентификации примитивно-рекурсивных функций, составления алгоритмов для решения задач, определения сложности алгоритмов; приобрести навыки составления нормального алгоритма, программирования машины Тьюринга.

1 Элементы теории алгоритмов

Алгоритм есть точное предписание, в соответствии с которым по любому входному объекту из данного класса входных объектов можно эффективно получать выходные объекты. Примеры алгоритмов:

– сортировка массива чисел в порядке возрастания;

– вычисление таблицы значений булевой функции, заданной формулой;

– вычисление чисел Фибоначчи по рекуррентному соотношению;

– решение системы линейных алгебраических уравнений методом исключения Гаусса.

Основные требования к алгоритмам

1. Алгоритм применяется к *исходным данным* и дает *результаты*. Кроме того, в процессе работы алгоритма могут появляться промежуточные данные. Итак, каждый алгоритм имеет дело с *данными*: исходными, промежуточными и выходными. Данными могут быть числа, векторы, матрицы, массивы, формулы, рисунки (в графических системах). Данные можно эффективно кодировать в виде слов в некотором алфавите. Таким образом, с каждым алгоритмом A связан некоторый алфавит Σ – внешний алфавит A, алгоритм получает на вход слова в алфавите Σ и вырабатывает в качестве результатов также слова в алфавите Σ.

2. Алгоритм является единой инструкцией, на вход которой можно подавать любое из бесконечного списка входных слов. В этом состоит свойство массовости алгоритм. Как результат работы алгоритма может получиться выходное слово – результат, в этом случае алгоритм сходится, а может случиться бесконечная

последовательность действий, и тогда алгоритм A не определен на данном входном слове.

3. Данные для своего размещения требуют *памяти.* Память состоит из ячеек, так что каждая ячейка может содержать один символ алфавита данных. Таким образом, объем данных и требуемая память согласованы.

4. Алгоритм состоит из отдельных *элементарных шагов* или *дискретных предписаний.* Причем множество различных шагов, из которых составлен алгоритм, конечно.

5. Последовательность шагов алгоритма *детерминирована*, т.е. после каждого шага либо указывается, какой шаг делать дальше, либо дается команда остановки, после чего работа алгоритма считается законченной.

6. Алгоритм должен обладать свойством *замкнутости*, т.е. выполнение вычислений определяется только предписанием, и для вычисления не требуется внешних процессов или вычисляющих устройств.

Алфавит – непустое конечное множество символов, элементы алфавита называются его буквами. Слово в алфавите Σ есть конечная последовательность (может быть и пуская) его букв. Слово в алфавите Σ имеет вид $a_0a_1\ldots a_n$, где $a_i \in \Sigma$. Множество всех слов в алфавите Σ обозначим через Σ^*.

Основная операция на словах – операция приписывания слова к слову: если дано слово A, имеющее вид $a_0a_1\ldots a_n$, и слово B вида $b_0b_1\ldots b_m$, то можно образовать новое слово AB вида $a_0a_1\ldots a_nb_0b_1\ldots b_m$, полученное приписыванием (или соединением, конкатенацией) слов A и B.

Пустое слово обозначается Λ. Справедливо:
$A\Lambda = \Lambda A = A$.

Каждый алгоритм задает функцию, поскольку по набору исходных данных выдает результат применения алгоритма к этим данным. Функция, значения которой могут находиться с помощью некоторого алгоритма – вычислимая функция.

Функция $f : X \to Y$ называется частичной, если она определена не для каждого значения $x \in X$. Множество тех $x \in X$, для которых

однозначно указано соответствующее значение функции f, называется областью определения функции.

Ограничимся только функциями, заданными на множестве натуральных чисел *N*.

Введем следующие функции:

$S^1(x) = x + 1$ – функция следования;

$O^n(x_1, \ldots, x_n) = 0$ – функция нуля;

$I_m^n(x_1, \ldots, x_n) = x_m \ (1 \le m \le n;\ n = 1, 2, \ldots)$ – функция проекции,

называемые простейшими.

Пусть даны частичные функции:

$g : N^n \to N$,

$h : N^{n+2} \to N$.

Частичная функция $f : N^{n+1} \to N$ получена из функций g, h примитивной рекурсией. Если:

$f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n)$,

$f(x_1, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y))$.

Для $n = 0$ уравнения принимают вид:

$f(0) = a$,

$f(x+1) = h(x, f(x))$.

Задав исходные данные $(x_1, \ldots, x_n, 0)$, можно шаг за шагом найти значения функции для $(x_1, \ldots, x_n, 1)$, $(x_1, \ldots, x_n, 2)$, …, $(x_1, \ldots, x_n, m)$, …

Символически задание f через примитивную рекурсию можно записать как

$f = \Re(g, h)$.

Частичная функция $f : N^{n+1} \to N$ называется примитивно рекурсивной относительно множества частичных функций Ф, если f получается из функций множества Ф и простейших функций конечным числом операций подстановки (суперпозиции) и примитивной рекурсии.

Если $\Phi = \varnothing$, то примитивно рекурсивная относительно множества частичных функций Ф функция получается только из

простейших функций и поэтому ее называют просто примитивно рекурсивной.

Примитивно рекурсивные функции являются всюду определенными функциями.

Пример

Функция $f(x, y) = x + y$ примитивно рекурсивна:

$$f(x, 0) = x + 0 = x = I_1^1(x),$$

$$f(x, y+1) = x + (y+1) = (x+y) + 1 = S^1(x+y) = S^1(f(x, y)).$$

Это означает, что функция $f(x, y) = x + y$ строится из примитивно рекурсивных функций I_1^1, $h(x, y, z) = z + 1 = S^1(z)$ с помощью примитивной рекурсии. Поэтому $f(x, y) = x + y$ примитивно рекурсивная функция.

Пример

Функция $f(x, y) = x \cdot y$ примитивно рекурсивна:

$$f(x, 0) = x \cdot 0 = 0 = O^1(x),$$

$$f(x, y+1) = x \cdot (y+1) = x \cdot y + x = f(x, y) + x.$$

Если взять:

$g(x) = O^1(x)$,

$h(x, y, z) = z + x$ – примитивно рекурсивная функция,

то функция $f(x, y) = x \cdot y$ примитивно рекурсивна.

Пусть дана частичная функция $f : N^n \to N$. Зафиксируем данные $(x_1, \ldots, x_n)$. Введем функцию:

$$\mathrm{M}f(x_1, \ldots, x_n) = \min_{y \in N} \{ f(x_1, \ldots, x_{n-1}, y) = x_n \}.$$

M – это операция на множестве частичных функций. Результатом является новая частичная функция с тем же числом аргументов. Называется эта операция минимизацией.

Пример

Функция

$g(x)=\begin{cases} x-1,\ x>0, \\ \text{не определено},\ x=0. \end{cases}$ – частично рекурсивна

Действительно,

$g(x)=\mathrm{M}S^1(x)=\min\limits_{y\in N}\{S^1(y)=y+1=x\}$

Следовательно, g получена из простейшей функции S^1 с помощью операции минимизации.

Частичная функция $f: N^{n+1}\to N$ называется частично рекурсивной относительно множества частичных функций Σ, если f получается из функций множества Σ и простейших функций конечным числом операций подстановки (суперпозиции), примитивной рекурсии и минимизации.

Если $\Sigma=\varnothing$, то частично рекурсивная относительно множества частичных функций Σ функция получается только из простейших функций, и поэтому ее называют просто частично рекурсивной.

Каждая примитивно рекурсивная функция является частично рекурсивной. Обратное утверждение неверно, поскольку в класс частично рекурсивных функций в соответствии с определением попадают частичная функция $\mathrm{M}S^1$ и, например, нигде не определенная функция

$f(x)=\min\limits_{y\in N}\{x+1+y=0\}$.

Минимизация может быть организована как последовательный (алгоритмический) процесс:

$f(x_1,\ldots,x_{n-1},0),\ f(x_1,\ldots,x_{n-1},1),\ \ldots,\ f(x_1,\ldots,x_{n-1},y),\ \ldots$

Наименьшее a, для которого

$f(x_1,\ldots,x_{n-1},a)=x_n$,

– это значение для $\mathrm{M}f(x_1,\ldots,x_n)$

Тезис Черча.

Класс алгоритмически (машинно) вычислимых частичных функций совпадает с классом всех частично рекурсивных функций.

Вопросы для самопроверки

1. Дайте определение алгоритма.
2. Укажите основные требования, предъявляемые к алгоритмам.

3. Что такое алфавит, буква алфавита?
4. Какая функция называется частичной?
5. Какая функция называется примитивно рекурсивной?
6. Какая функция называется частично рекурсивной?
7. Перечислите базовые рекурсивные функции.

Задания для выполнения

1. Доказать, что следующие функции являются примитивно рекурсивными:

1) $f(x) = x + 4$;
2) $f(x) = 2x + 1$;
3) $f(x) = 3^{x+1}$;
4) $f(x) = 4^{x^2}$;
5) $f(x) = \left|x^2 + 2x + 1\right|$;
6) $f(x) = \left|6x^2 + 2x - 8\right|$;
7) $f(x, y) = 3^{x+y}$;
8) $f(x, y) = x^y$;
9) $f(x, y) = y^{x+2}$;
10) $f(x) = x^2 - 10x + 25$;
11) $f(x) = 3x^2 - 4x + 1$;
12) $f(x, y) = x^{y-12}$.

2. Написать примитивно рекурсивное описание функций:

1) $f(x) = x - 3$;
2) $f(x) = 3x - 4$;
3) $f(x, y) = 2x - 4y$;
4) $f(x, y) = x + y$.

3. Какая функция получается из функции $g(x)$ и $h(x, y, z)$ с помощью операции примитивной рекурсии:

1) $g(x) = x,\ h(x, y, z) = z^x$
2) $g(x) = x,\ h(x, y, z) = z^x$
3) $g(x) = x,\ h(x, y, z) = xy$
4) $g(x) = 1,\ h(x, y, z) = z \cdot (x + 1)$.

2 Нормальные алгоритмы Маркова

Теория нормальных алгоритмов (или алгорифмов, как называл их создатель теории) была разработана советским математиком А. А. Марковым (1903 – 1979 гг.) в конце 1940-х – начале 1950-х гг. XX в. Эти алгоритмы представляют собой некоторые правила по переработке слов в каком-либо алфавите, так что исходные данные и искомые результаты для алгоритмов являются словами в некотором алфавите.

Марковские подстановки

Алфавитом (как и прежде) называется любое непустое множество. Его элементы называются буквами, а любые последовательности букв словами в данном алфавите. Для удобства рассуждений допускаются пустые слова (они не имеют в своем составе ни одной буквы). Пустое слово будем обозначать Λ. Если A и B – два алфавита, причем $A \subseteq B$, то алфавит B называется расширением алфавита A.

Слова будем обозначать латинскими буквами: P, Q, R (или этими же буквами с индексами). Одно слово может быть составной частью другого слова. Тогда первое называется подсловом второго или вхождением во второе. Например, если A – алфавит русских букв, то можем рассмотреть такие слова: $P_1 = \text{параграф}$, $P_2 = \text{граф}$, $P_3 = \text{ра}$. Слово P_2 является подсловом слова P_1, а P_3 – подсловом P_1 и P_2, причем в P_1 оно входит дважды. Особый интерес представляет первое вхождение.

Определение

Марковской подстановкой называется операция над словами, задаваемая с помощью упорядоченной пары слов (P, Q), состоящая в следующем. В заданном слове R находят первое вхождение слова P (если таковое имеется) и, не изменяя остальных частей слова R, заменяют в нем это вхождение словом P. Полученное слово

называется результатом применения марковской подстановки (P, Q) к слову R. Если же первого вхождения P в слово R нет (и, следовательно, вообще нет ни одного вхождения P в R), то считается, что марковская подстановка (P, Q) неприменима к слову R.

Частными случаями марковских подстановок являются подстановки с пустыми словами:
$(\Lambda, Q), (P, \Lambda), (\Lambda, \Lambda)$

Пример

Примеры марковских подстановок рассмотрены в таблице, в каждой строке которой сначала дается преобразуемое слово, затем применяемая к нему марковская подстановка и, наконец, получающееся в результате слово:

Преобразуемое слово	Марковская подстановка	результат
1325846	(258, 00)	130046
графит	(ит, ика)	графика
функция	$(\Lambda, \lambda -)$	$\lambda -$ функция
логика	(ика, Λ)	лог
книга	(Λ, Λ)	книга
икра	(кар, а)	неприменимо

Для обозначения марковской подстановки (P, Q) используется запись $P \to Q$. Она называется формулой подстановки (P, Q). Некоторые подстановки (P, Q) будем называть заключительными. Для обозначения таких подстановок будем использовать запись $P \to \overline{\cdot} Q$, называя ее формулой заключительной подстановки. Слово P называется левой частью, а Q – правой частью в формуле подстановки.

Упорядоченный конечный список формул подстановок

$$\begin{cases} P_1 \to (\overline{\cdot}) Q_1, \\ P_2 \to (\overline{\cdot}) Q_2, \\ \dots \\ P_r \to (\overline{\cdot}) Q_r, \end{cases}$$

в алфавите A называется схемой (или записью) нормального алгоритма в A.

нормальным алгоритмом (Маркова) в алфавите A называется следующее правило построения последовательности V_i слов в алфавите A, исходя из данного слова V в этом алфавите. В качестве начального слова V_0 последовательности берется слово V. Пусть для некоторого $i \geq 0$ слово V_i построено и процесс построения рассматриваемой последовательности еще не завершился. Если при этом в схеме нормального алгоритма нет формул, левые части которых входили бы в V_i, то V_{i+1} полагают равным V_i, и процесс построения последовательности считается завершившимся. Если же в схеме имеются формулы с левыми частями, входящими в V_i, то в качестве V_{i+1} берется результат марковской подстановки правой части первой из таких формул вместо первого вхождения ее левой части в слово V_i; процесс построения последовательности считается завершившимся, если на данном шаге была применена формула заключительной подстановки, и продолжающимся – в противном случае. Если процесс построения упомянутой последовательности обрывается, то рассматриваемый нормальный алгоритм применим к слову V. Последний член W последовательности называется результатом применения нормального алгоритма к слову V. Таким образом, нормальный алгоритм перерабатывает V в W.

Последовательность V_i будем записывать следующим образом:

$V_0 \Rightarrow V_1 \Rightarrow V_2 \Rightarrow \ldots \Rightarrow V_{m-1} \Rightarrow V_m$, где $V_0 = V$, $V_m = W$

Пример

Пусть $A = \{a, b\}$ – алфавит. Рассмотрим следующую схему нормального алгоритма в A:

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

Всякое слово V в алфавите A, содержащее хотя бы одно вхождение буквы a, он перерабатывает в слово, получающееся из V вычеркиванием в нем самого левого (первого) вхождения буквы a. Пустое слово он перерабатывает в пустое. Алгоритм не применим к таким словам, которые содержат только букву b.

$aabab \Rightarrow abab$,

$ab \Rightarrow b$,
$aa \Rightarrow a$,
$bbab \Rightarrow bbb$,
$baba \Rightarrow bba$.

Пример

Пусть $A = \{a_0, a_1, \ldots, a_n\}$ – алфавит. Рассмотрим следующую схему нормального алгоритма в A:

$$\begin{cases} a_0 \to \Lambda, \\ a_1 \to \Lambda, \\ \ldots \\ a_n \to \Lambda, \\ \Lambda \to \neg\Lambda. \end{cases}$$

Она определяет нормальный алгоритм, перерабатывающий всякое слово (в алфавите A) в пустое слово.

$a_1a_2a_1a_3a_0 \Rightarrow a_1a_2a_1a_3 \Rightarrow a_2a_1a_3 \Rightarrow a_2a_3 \Rightarrow a_3 \Rightarrow \Lambda$,

$a_0a_2a_2a_1a_3a_1 \Rightarrow a_2a_2a_1a_3a_1 \Rightarrow a_2a_2a_3a_1 \Rightarrow a_2a_2a_3 \Rightarrow a_2a_3 \Rightarrow a_3 \Rightarrow \Lambda$.

Нормальные алгоритмы не производят вычислений: они лишь производят преобразования слов, заменяя в них одни буквы другими по предписанным им правилам. Результат применения правил интерпретируется как вычисления.

Пример

В алфавите $A = \{A\}$ схема $\Lambda \to \neg 1$ определяется нормальный алгоритм, который к каждому слову в алфавите $A = \{A\}$ приписывает слева 1. Следовательно, алгоритм реализует (вычисляет) функцию $f(x) = x + 1$.

Пример

Дана функция

$$\varphi_3(11\ldots1) = \begin{cases} 1, \text{ если } n \text{ делится на 3}, \\ \Lambda, \text{ если } n \text{ не делится на 3}, \end{cases}$$

где n – число единиц в слове 11…1. Рассмотрим нормальный алгоритм в алфавите $A=\{1\}$ со следующей схемой:

$$\begin{cases} 111\to\Lambda, \\ 11\to\neg\Lambda, \\ 1\to\neg\Lambda, \\ \Lambda\to\neg 1. \end{cases}$$

Этот алгоритм работает по такому принципу: пока число букв «1» в слове не меньше 3, алгоритм последовательно стирает по три буквы. Если число букв меньше 3, но больше 0, то оставшиеся буквы «1» или «11» стираются заключительно; если слово пусто, оно заключительно переводится в слово «1»:

$1111111\Rightarrow 1111\Rightarrow 1\Rightarrow\Lambda$,

$111111111\Rightarrow 111111\Rightarrow 111\Rightarrow\Lambda\Rightarrow 1$.

Таким образом, рассмотренный алгоритм реализует (или вычисляет) данную функцию

Определение

Функция f, заданная на некотором множестве слов алфавита A, называется нормально вычислимой, если найдется такое расширение B данного алфавита $(A\subseteq B)$ и такой нормальный алгоритм в B, что каждое слово V (в алфавите A) из области определения функции f этот алгоритм перерабатывает в слово $f(V)$

Таким образом, нормальные алгоритмы, рассмотренные в примерах, показывают, что функции $f(x)=x+1$ и $\varphi_3(x)$ нормально вычислимы.

Пример, демонстрирующий нормальный алгоритм в расширенном алфавите, вычисляющий данную функцию.

Построим нормальный алгоритм для вычисления функции $f(x)=x+1$ не в одноичной системе, а в десятичной. В качестве алфавита возьмем перечень арабских цифр $A=\{0,1,\ 2,3,4,5,6,7,8,9\}$, а нормальный алгоритм будем строить в его расширении $B=A\cup\{a,b\}$. Схема алгоритма:

$0b \to 1$ $\quad a0 \to 0a$

$1b \to 2$ $\quad a1 \to 1a$ $\quad 1a \to 1b$

$2b \to 3$ $\quad a2 \to 2a$ $\quad 2a \to 2b$

$3b \to 4$ $\quad a3 \to 3a$ $\quad 3a \to 3b$

$4b \to 5$ $\quad a4 \to 4a$ $\quad 4a \to 4b$

$5b \to 6$ $\quad a5 \to 5a$ $\quad 5a \to 5b$

$6b \to 7$ $\quad a6 \to 6a$ $\quad 6a \to 6b$

$7b \to 8$ $\quad a7 \to 7a$ $\quad 7a \to 7b$

$8b \to 9$ $\quad a8 \to 8a$ $\quad 8a \to 8b$

$9b \to b0$ $\quad a9 \to 9a$ $\quad 9a \to 9b$

$b \to 1$ $\quad 0a \to 0b$ $\quad \Lambda \to a$

Применим алгоритм к пустому слову Λ. На каждом шаге должна будет применяться самая последняя формула данной схемы. Получается бесконечный процесс:

$\Lambda \Rightarrow a \Rightarrow aa \Rightarrow aaa \Rightarrow aaaa \Rightarrow \ldots$

Это означает, что к пустому слову данный алгоритм не применим.

Применим алгоритм к слову «499»:

$499 \Rightarrow 499a \Rightarrow 499b \Rightarrow 49b0 \Rightarrow 4b00 \Rightarrow 500.$

Применим алгоритм к слову «326»:

$326 \Rightarrow 326a \Rightarrow 326b \Rightarrow 327.$

В рассмотренном примере нормальный алгоритм построен в алфавите B, являющемся существенным расширением алфавита A, но данный алгоритм слова в алфавите A перерабатывает снова в слова в алфавите A. В таком случае говорят, что алгоритм задан над алфавитом A.

Вопросы для самопроверки

1. Что такое марковская подстановка? Приведите примеры.
2. Что такое заключительная подстановка? Приведите примеры.
3. Что такое подслово?
4. Что такое нормальный алгоритм в алфавите A?

5. Как задается нормальный алгоритм?

6. Что такое нормальный алгоритм над алфавитом A?

Задания для выполнения

1. Решить задачи:

1) Дан алфавит $A=\{b, c, d\}$. Перенести в начало все «c».

2) Дан алфавит $A=\{b, c, d\}$. Упорядочить буквы в слове по алфавиту.

3) Дан алфавит $A=\{b, c, d\}$. Если во входное слово входит буква b, то заменить все вхождения букв b на c.

4) Дан алфавит $A=\{b, c, a\}\}$. Заменить во входном слове все вхождения ba, на a.

5) Дан алфавит $A=\{b, c, d\}$. Заменить во входном слове все вхождения bc, на b.

6) Дан алфавит $A=\{b, c, d\}$. Перенести в начало все d.

7) Дан алфавит $A=\{b, c, a\}$. В непустом слове P удвоить первый символ, т.е. приписать этот символ слева к P.

8) Дан алфавит $A=\{b, c, a\}$. Удвоить каждый символ в слове P (например: $bacb \rightarrow bbaaccbb$).

9) Дан алфавит $A=\{b, a\}$. Перенести первый символ непустого слова P в конец слова.

10) Дан алфавит $A=\{b, a\}$. Перенести последний символ непустого слова P в начало слова.

11) Дан алфавит $A=\{b, a\}$. В непустом слове P переставить первый и последний символы.

12) Дан алфавит $A=\{b, a\}$. Если в непустом слове P совпадают первый и последний символы, то удалить оба этих символа, а иначе слово не менять.

13) Дан алфавит $A=\{b, a\}$. Определить, является ли слово P палиндромом (перевёртышем, симметричным словом).

14) Дан алфавит $A=\{b, a\}$. Пусть слово P имеет нечётную длину. Удалить из него средний символ.

2. Решить задачи:

1) Написать алгоритм Маркова для вычисления функции $f(x)=x-6$, где x задано в шестеричной системе счисления и не содержит незначащих нулей, показать правильность его работы для $x=1000_6$, $x=212_6$, $x=101_6$. Ведущие нули в результате нужно удалить.

2) Написать алгоритм Маркова для вычисления функции $f(x)=x+8$, где x задано в восьмеричной системе счисления и не содержит незначащих нулей, показать правильность его работы для $x=1000_8$, $x=212_8$, $x=101_8$. Ведущие нули в результате нужно удалить.

3) Пусть A – русский алфавит. Построить нормальный алгоритм над алфавитом A, который преобразует слово «муха» в слово «слон», а любое другое слово в алфавите A в пустое слово. При этом, если слово «муха» входит в некоторое слово Q, например Q=черемуха, то слово Q алгоритм должен переработать в пустое слово.

4) Пусть A – русский алфавит. Построить нормальный алгоритм над алфавитом A, который преобразует слово «слон» в слово «муха», а любое другое слово в алфавите A в пустое слово. При этом, если слово «слон» входит в некоторое слово Q, например Q=заслон, то слово Q алгоритм должен переработать в пустое слово.

3 Машина Тьюринга-Поста

Машины Тьюринга-Поста – это пример алгоритма. Придуман этот алгоритм независимо Аланом Тьюрингом и Эмилем Постом.

Машина Тьюринга (МТ) состоит из:

– управляющего устройства, которое может находиться в одном из фиксированного множества внутренних состояний $Q=\{q_0, q_1,\ldots, q_n\}$. Среди элементов множества Q выделены два подмножества: начальное $Q_B=\{q_0\}$ и заключительное $Q_E=\{q_k\}$; $k=\overline{1, m}$. В начальном состоянии машина находится перед началом работы, перейдя в заключительное состояние, машина останавливается;

– бесконечной ленты, разбитой на ячейки; в каждой ячейке может быть записан один символ внешнего алфавита $A=\{a_1,\ldots, a_m\}$; один из символов внешнего алфавита называется пустым (для обозначения будем использовать Λ; в каждый момент на ленте записано конечное число непустых символов.

Использование пустого символа удобно тем, что операцию стирания символа в некоторой клетке можно рассматривать как запись в эту клетку пустого символа Λ, поэтому вместо длинной фразы «записать символ в клетку или стереть находящийся там символ» можно говорить просто «записать символ в клетку»;

– считывающей и записывающей головки, которая обозревает одну ячейку ленты, записывает в нее новый символ из внешнего алфавита (он может совпадать с прежним), сдвигается влево или вправо на одну ячейку или остается на месте. Управляющее устройство при этом может изменить свое внутреннее состояние.

Активную часть МТ (управляющее устройство и считывающая/записывающая головка) можно рассматривать как автомат. 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: q_1, 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.

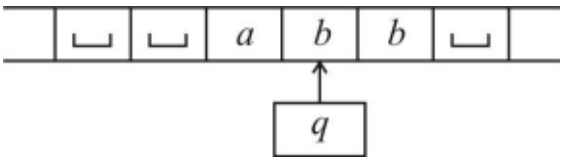

Рисунок 1 – Представление машины Тьюринга

Управляющее устройство будем обозначать прямоугольником, в котором записано внутреннее состояние, а обозреваемую ячейку стрелкой так, как это указано на рис. 1. Машина Тьюринга работает детерминировано, тактами. Такт состоит в выполнении одной команды – выражения вида

$q_i a_j \to q_k a_l d_m$,

где $d_m \in \{L, R, H\}$.

Эта команда содержит условие (левая часть команды до стрелки) и действие (правая часть команды после стрелки). Команда выполняется следующим образом. Если на начало очередного такта

внутреннее состояние машины q_i, обозреваемый символ a_j, то после выполнения команды $q_i a_j \to q_k a_l d_m$, т.е. к концу такта, внутреннее состояние становится q_k, в обозреваемую ячейку записывается символ a_l, и головка сдвигается на одну ячейку вправо, если $d_m = R$, сдвигается влево, если $d_m = L$, остается на месте, если $d_m = H$.

Пару из видимого символа (S) и текущего состояния автомата (q) будем называть конфигурацией и обозначать $\langle S, q \rangle$. Автомат может выполнять три элементарных действия:

1) раписывать в видимую клетку новый символ (менять содержимое других клеток автомат не может, 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 или вправо, («перепрыгивать» сразу через несколько клеток автомат не может), либо остается неподвижным;

3) переходить в новое состояние (в частности, может остаться в прежнем состоянии).

Ничего другого делать автомат не умеет, поэтому все более сложные операции так или иначе должны быть сведены к этим трём элементарным действиям.

Машина начинает работу, обозревая самую левую непустую ячейку и находясь в начальном внутреннем состоянии.

Работа машины Тьюринга не всегда заканчивается. Рассмотрим примеры. Предположим, что программа машины содержит следующие команды:

$q_1 a \to q_2 a R$,

$q_2 b \to q_1 b L$,

$q_3 \Lambda \to q_3 \Lambda R$.

Если машина на начало очередного такта находится в состоянии q_1, обозревает ячейку, в которой записан символ a, а в следующей записан символ b, то произойдет зацикливание на участке ленты, содержащем эти символы, и машина будет работать бесконечно. Зацикливание работы машины может произойти еще в одном случае. Если машина находится в состоянии q_3, обозревает пустую ячейку (с

символом Λ) и все следующие ячейки так же пусты, то машина также зацикливается, но уже на бесконечном участке ленты.

Сама по себе МТ ничего не делает. Для того, чтобы заставить её работать, надо написать для неё программу. Программа – это множество команд с различными условиями, т.е. различными левыми частями.

Программа записывается в виде следующей таблицы:

Таблица 1 – Алгоритм работы машины Тьюринга

	q_0	q_1	...	q_n
s_1			...	
s_2			...	
...	...	...	...	...
s_i		$s_j[L,R,H]q_k$		
...	...	...	...	...
s_n			...	
_			...	

Сверху перечисляются все состояния, в которых может находиться автомат, слева – все символы (в том числе и Λ), которые автомат может видеть на ленте. (Какие именно символы и состояния указывать в таблице – определяет автор программы.) На пересечениях же (в ячейках таблицы) 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.

До выполнения программы нужно проделать следующие предварительные действия. Во-первых, надо записать на ленту входное слово, к которому будет применена программа. Входное слово – это конечная последовательность символов, записанных в соседних клетках ленты; 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. Во-вторых, надо установить автомат в состояние q_0 (указанное в таблице первым) и разместить его под первым символом входного

слова. Если входное слово пустое, то автомат может смотреть в любую клетку, т.к. все они пусты. После этих предварительных действий начинается выполнение программы. В таблице отыскивается ячейка на пересечении первого столбца (т.к. автомат находится в состоянии q_0) и той строки, который соответствует первому символу входного слова (это необязательно верхняя строка таблицы), и выполняется такт, указанный в этой ячейке. В результате автомат окажется в новой конфигурации. 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? Введём 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. 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 к этому моменту получено на ленте, считается выходным словом, т.е. результатом работы МТ, ответом. В момент останова должны быть выполнены следующие обязательные условия:
– внутри выходного слова не должно быть пустых клеток (отметим, что во время выполнения программы внутри обрабатываемого слова пустые клетки могут быть, но в конце их уже не должно остаться);
– автомат обязан остановиться под одним из символов выходного слова (под каким именно – не играет роли), а если слово пустое – под любой клеткой ленты;
2) второй исход – «плохой»: это когда МТ зацикливается, никогда не попадая на такт останова (например, автомат на каждом шаге сдвигается вправо и потому не может остановиться, т.к. лента бесконечна). В этом случае говорят, что МТ неприменима к заданному входному слову. Ни о каком результате при таком исходе не может идти и речи. Один и тот же алгоритм (программа МТ) может быть применимым к одним входным словам (т.е.

останавливаться) и неприменимым к другим (т.е. зацикливаться). Таким образом, применимость/неприменимость зависит не только от самого алгоритма, но и от входного слова. На каких входных словах алгоритм должен останавливаться? На тех, которые относятся к допустимым исходным данным решаемой задачи, для которых задача осмысленна. Но на ленте могут быть записаны любые входные слова, в том числе и те, для которых задача не имеет смысла; на таких словах поведение алгоритма не фиксируется, он может остановиться (при любом результате), а может и зациклиться.

Рассмотрим примеры на составление программ для МТ, чтобы продемонстрировать некоторые типичные приёмы программирования на МТ. Для сокращения формулировки задач введём следующие два соглашения:

– буквой *P* будем обозначать входное слово;

– буквой *A* будем обозначать алфавит машины Тьюринга, т.е. набор тех символов, из которых может состоять *P*.

Пример (перемещение автомата, замена символов)

Алфавит $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Пусть *P* – непустое слово, значит, *P* – это последовательность из десятичных цифр, т.е. запись неотрицательного целого числа в десятичной системе.

Требуется получить на ленте запись числа, которое на 1 больше числа *P*.

Решение

Для решения этой задачи предлагается выполнить следующие действия:

– перегнать автомат под последнюю цифру числа;

– если это цифра от 0 до 8, то заменить её цифрой на 1 больше и остановиться (рис. 2);

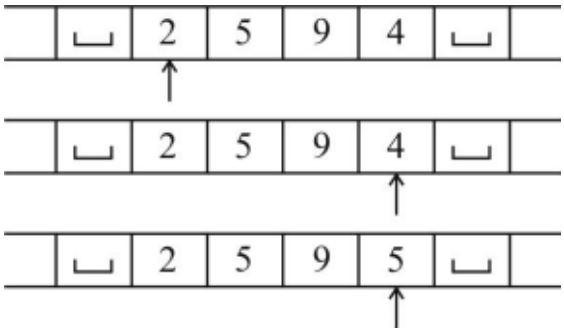

Рисунок 2 – Выполнение инкремента для цифр от 0 до 8

– 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 (рис. 3);

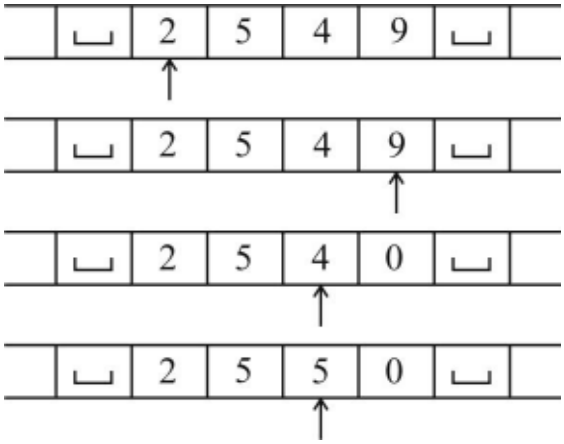

Рисунок 3 – Выполнение инкремента для цифры 9

– 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) (рис.4).

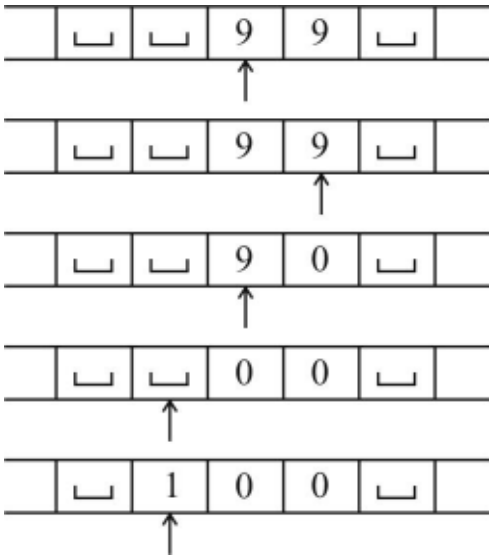

Рисунок 4 – Выполнение инкремента для числа 99

В виде программы для МТ эти действия описываются следующим образом

Таблица 2 – Операция инкремента

	q_0	q_1
0	$0, R, q_0$	$1, H, q_k$
1	$1, R, q_0$	$2, H, q_k$
2	$2, R, q_0$	$3, H, q_k$
3	$3, R, q_0$	$4, H, q_k$
4	$4, R, q_0$	$5, H, q_k$
5	$5, R, q_0$	$6, H, q_k$
6	$6, R, q_0$	$7, H, q_k$
7	$7, R, q_0$	$8, H, q_k$
8	$8, R, q_0$	$9, H, q_k$
9	$9, R, q_0$	$0, L, q_1$
_	$\_, L, q_1$	$1, H, q_k$

Будем рассматривать функции f от одной или нескольких переменных, заданных на множестве $N = \{0, 1, \ldots, n, \ldots\}$ натуральных чисел или его подмножествах (частичные функции) и принимающие значения на множестве N.

Определение

Функция $f(x_1, \ldots, x_n)$ называется *вычислимой*, если существует алгоритм, позволяющий вычислять ее значения для тех переменных, для которых она определена, и работающий бесконечно, если функция для данного набора переменных не определена.

Определение

Функция $f(x_1, \ldots, x_n)$ называется *вычислимой по Тьюрингу*, если существует машина Тьюринга, вычисляющая эту функцию.

Тезис Тьюринга

Всякий алгоритм можно реализовать машиной Тьюринга.

Тезис Тьюринга доказать нельзя. Это утверждение означает, что математическое понятие вычислимой по Тьюрингу функции является идеальной моделью интуитивного понятия алгоритма. Этот тезис подтверждается опытом. По своему характеру тезис Тьюринга напоминает математические законы механики, которые точно так же не могут быть доказаны, но, открытые Ньютоном, многократно подтверждены опытом. В силу тезиса Тьюринга невозможность

построения машины Тьюринга означает отсутствие алгоритма решения данной проблемы.

Изучение машин Тьюринга закладывает фундамент алгоритмического мышления, сущность которого состоит в том, что нужно уметь разделять процесс вычисления на простые составляющие шаги. В машине Тьюринга такое разделение доведено до предельной простоты. В современной ЭВМ алгоритмический процесс разделяется не на столь мелкие составляющие, как в машине Тьюринга. Наоборот, есть стремление укрупнить выполняемые машиной процедуры. Например, операция сложения в машине Тьюринга – целая программа, а в ЭВМ это простейшая функция.

Вопросы для самопроверки

1. Обоснуйте свойства машины Тьюринга как алгоритма.
2. Что такое алфавит машины Тьюринга?
3. Каковы составные части машины Тьюринга?
4. Что представляет собой программа функционирования машины Тьюринга?
5. Какие операции может выполнять автомат машины Тьюринга?
6. Что такое входное слово?
7. Какими могут быть результаты работы машины Тьюринга?
8. Как обозначается начальное и конечные состояния на диаграмме, представляющей машину Тьюринга?

Задания для выполнения

1. Составьте программы для работы МТ

1) Дан алфавит $A = \{b, c, d\}$. Перенести в начало все «*c*».

2) Дан алфавит $A = \{b, c, d\}$. Упорядочить буквы в слове по алфавиту.

3) Дан алфавит $A = \{b, c, d\}$. Если во входное слово входит буква *b*, то заменить все вхождения букв *b* на *c*.

4) Дан алфавит $A = \{b, c, a\}\}$. Заменить во входном слове все вхождения *ba*, на *a*.

5) Дан алфавит $A=\{b, c, d\}$. Заменить во входном слове все вхождения *bc*, на *b*.

6) Дан алфавит $A=\{b, c, d\}$. Перенести в начало все d.

7) Дан алфавит $A=\{b, c, a\}$. В непустом слове *P* удвоить первый символ, т.е. приписать этот символ слева к *P*.

8) Дан алфавит $A=\{b, c, a\}$. Удвоить каждый символ в слове *P* (например: *bacb*→*bbaaccbb*).

9) Дан алфавит $A=\{b, a\}$. Перенести первый символ непустого слова *P* в конец слова.

10) Дан алфавит $A=\{b, a\}$. Перенести последний символ непустого слова *P* в начало слова.

11) Дан алфавит $A=\{b, a\}$. В непустом слове *P* переставить первый и последний символы.

12) Дан алфавит $A=\{b, a\}$. Если в непустом слове *P* совпадают первый и последний символы, то удалить оба этих символа, а иначе слово не менять.

13) Дан алфавит $A=\{b, a\}$. Определить, является ли слово *P* палиндромом (перевёртышем, симметричным словом).

14) Дан алфавит $A=\{b, a\}$. Пусть слово *P* имеет нечётную длину. Удалить из него средний символ.

4 Сложность вычислений

Любая практическая задача требует предварительного задания исходных данных. Как правило, можно задать некоторое характерное число *n*. Например, для задачи сортировки массива чисел по возрастанию *n* – число чисел в массиве, для задачи решения системы линейных уравнений *n* – число уравнений. Характерное число задачи определяет *размерность задачи* как величину массива исходных данных.

С ростом характерного числа размерность задачи возрастает. Введем понятие скорости роста для функций, зависящих от целочисленного параметра *n*.

Определение

Функции $f(n)$ и $g(n)$ имеют *одинаковую скорость роста*, если при достаточно больших n, начиная с некоторого n_0, выполняется условие:

$C_1 g(n) \le f(n) \le C_2 g(n)$,

где C_1, C_2 – некоторые константы.

Определение

Скорость роста функции $f(n)$ *ограничена снизу* скоростью роста функции $g(n)$, если при достаточно больших n, начиная с некоторого n_0, выполняется условие:

$C_1 g(n) \le f(n)$.

Определение

Скорость роста функции $f(n)$ *ограничена сверху* скоростью роста функции $g(n)$, если при достаточно больших n, начиная с некоторого n_0, выполняется условие:

$f(n) \le C_2 g(n)$.

Определение

Скорость роста функции $f(n)$ *больше* скорости роста функции $g(n)$, если для любой сколь угодно большой константы C_2 существует некоторое n_0, начиная с которого выполняется условие:

$f(n) \le C_2 g(n)$.

Для того чтобы более наглядно представить скорости роста функций, их сравнивают со скоростями роста хорошо известных функций. В качестве таковых чаще всего используют степенные функции n^a. При $a = 1$ скорость роста функции n^a называют *линейной*, при $a = 2$ – квадратичной, при $a = 3$ – кубической и т. д. Скорость роста вида n^a называют *полиномиальной*. Очевидно, что при возрастании a скорость роста тоже увеличивается. Для некоторых функций скорости роста превосходят в пределе при $n \to \infty$ любую полиномиальную скорость. Такими функциями являются, например, 2^n, e^n, $n!$. Скорости такого типа называют *экспоненциальными*.

Обозначим через $r(n)$ скорость роста размерности задачи. В задаче вычисления таблицы значений булевой функции n

переменных скорость роста определяется таблицей значений переменных. Так как различных наборов переменных 2^n, а каждый набор состоит из n символов, то размерность задачи равна $n2^n$ и скорость роста будет экспоненциальной. В задаче решения системы линейных алгебраических уравнений методом исключения Гаусса наиболее быстро растет число элементов матрицы системы уравнений размером $n \times n$. Поэтому скорость роста размерности этой задачи будет квадратичной, $r(n) = n^2$.

Размерность задачи определяет память, необходимую для представления исходных данных в ЭВМ, Кроме того, необходима дополнительная память для размещения промежуточных данных. Величина этой памяти зависит от конкретного алгоритма и ее, как правило, нетрудно рассчитать.

Рассмотрим время реализации алгоритма – время счета.

Пусть при выполнении некоторого алгоритма выполняются элементарные операции $t_1, t_2, \ldots, t_k$ (арифметические, логические и другие). Среднее время выполнения этих операций обозначим через $t_1, t_2, \ldots, t_k$. По аналогии с размерностью задачи введем понятие скорости роста числа выполняемых операций в зависимости от характерного числа n. Обозначим их для операций $t_1, t_2, \ldots, t_k$ через $g_1(n)$, $g_2(n)$, …, $g_k(n)$. Без доказательства приведем следующее утверждение.

Утверждение

При $n \to \infty$ скорость роста общего времени счета $T(n)$ равна максимальной из скоростей роста числа элементарных операций $g_1(n)$, $g_2(n)$, …, $g_k(n)$ *независимо* от среднего времени их выполнения $t_1, t_2, \ldots, t_k$.

Определение

Скорость роста общего времени счета $T(n)$ называется *вычислительной сложностью* или просто *сложностью алгоритма*.

Обозначим сложность алгоритма через $f(n)$. В зависимости от сложности все алгоритмы делятся на несколько классов.

Определение

Полиномиальными называются алгоритмы, сложность которых ограничена некоторым полиномом.

Пример

Рассмотрим задачу определения максимального элемента в массиве из n чисел. Поскольку число операций сравнения постоянно и равно $n - 1$, сложность алгоритма $f(n) = n$.

Определение

Экспоненциальными называются алгоритмы, сложность которых при возрастании n превышает полином любой степени.

Пример

Рассмотрим задачу коммивояжёра. Необходимо обойти n городов и вернуться в исходный пункт, так чтобы суммарный путь был минимальным. Количество всех возможных вариантов обхода равно $0{,}5n!$. Следовательно, сложность точного решения, основанного на переборе всех вариантов, равна $f(n) = n!$

Пример

Рассмотрим задачу вычисления конъюнктивной нормальной формы (КНФ) булевой функции n переменных. Количество всех наборов переменных равно 2^n. Количество всех операций при переборе всех дизъюнкций пропорционально $n2^n$, Следовательно, сложность алгоритма $f(n) = n2^n$.

Экспоненциальные алгоритмы практически могут быть реализованы только при малых значениях n (обычно при $n < 10$).

Задачи, для которых существуют точные алгоритмы решения полиномиальной сложности, называются *задачами класса P*.

Задачи, для которых не удается найти точные алгоритмы решения полиномиальной сложности, составляют *класс NP*.

Для многих задач класса *NP* выполняется свойство сводимости, состоящее в том, что данный алгоритм выражается при помощи полиномиального числа операций через другой алгоритм, имеющий полиномиальную сложность.

Список использованных источников

1. Колмогоров, А.Н. Математическая логика / А.Н. Колмогоров, А.Г. Драгалин. – М. : КомКнига, 2006. – 240 с. – Текст : непосредственный.

2. Чень, Ч. Математическая логика и автоматическое доказательство теорем / Чень Ч., Ли Р. – М.: Наука, 1983. – Текст : непосредственный.