5. ЭЛЕМЕНТЫ ТЕОРИИ АЛГОРИТМОВ

В данном разделе изучаются формализация понятия алгоритма в виде машины Тьюринга, рекурсивные функции и алгоритмически неразрешимые проблемы. Дополнительные сведения по тематике данного раздела студенты могут найти в [11,14].

После завершения работы с теоретическим материалом следует ответить на вопросы для самопроверки по каждой теме и выполнить тестовые задания, приведенные в разделе 4.2. Практические занятия служат для закрепления изучаемого материала и изучения особенностей программирования на машине Тьюринга.

5.1. Понятие алгоритма

Интуитивное представление об алгоритме как о формальном предписании, которое определяет совокупность операций и порядок их выполнения для решения всех задач какого-либо типа, существует в математике с давних времен. В современных вычислительных системах алгоритмический подход к решению задач реализуется как *императивное программирование*. Термин "алгоритм" происходит от имени средневекового узбекского математика аль-Хорезми, который еще в IX веке сформировал правила выполнения арифметических действий, которые мы изучаем в школе. Под алгоритмом обычно понимается точное предписание, определяющее процесс переработки исходных данных в требуемый результат.

При этом требуется:

- чтобы исходные данные были заданы в конкретном алфавите и могли принимать значения из некоторого множества, т.е. носили *массовый* характер;

- чтобы процесс переработки данных состоял из отдельных *дискретных* шагов и был *детерминированным*;
- чтобы были четко указаны условия остановки процесса, и что следует считать *результатом* процесса.

Таким образом, любой алгоритм характеризуется *массовостью, детерминированностью и результативностью*. Для решения любой массовой задачи требуется построить соответствующий алгоритм, поэтому важность понятия алгоритма трудно переоценить.

Математика накопила большое число алгоритмов для решения разнообразных задач. В то же время попытки решения ряда задач натолкнулись на трудности, которые не удалось преодолеть. Возникла необходимость доказать *существование алгоритма* (*алгоритмическую разрешимость*) или *принципиальную невозможность построения алгоритма* (*алгоритмическую неразрешимость*) для ряда важных задач.

Между тем, сформулированное выше понятие алгоритма нельзя считать точным определением, моделью алгоритма. Поэтому были предложены точные математические модели, уточняющие понятие алгоритма: машина Тьюринга, система рекурсивных функций Клини, нормальный алгоритм А.А.Маркова, схема Колмогорова-Успенского, лямбда-конверсии Черча, финитные комбинаторные процессы Поста.

А.Черч впервые обосновал положение о том, что все уточнения понятия алгоритма эквивалентны ("тезис Черча"), т.е. правильно отражают интуитивное представление об алгоритмах, сложившееся в математике.

С помощью этих моделей алгоритма доказана алгоритмическая неразрешимость ряда важных задач математики и вычислительной техники и, в частности, неразрешимость *проблемы останова* универсальной вычислительной машины, реализующей любой алгоритм. Например, из алгоритмической неразрешимости проблемы останова следует важный практический вывод: *невозможно создать универсальную отладочную*

программу для обнаружения возможности зацикливания отлаживаемой программы.

Алгоритмически неразрешимой оказалась проблема выводимости в исчислении предикатов: для любых формул P и Q узнать, существует ли дедуктивная цепочка, ведущая от P к Q. Однако для исчисления высказываний и одноместных предикатов проблема выводимости разрешима.

Алгоритмическая неразрешимость некоторой задачи означает, что не существует общего алгоритма, решающего любую задачу рассматриваемого класса, однако для отдельных подклассов алгоритмически неразрешимой задачи может существовать алгоритм. Не следует смешивать алгоритмическую неразрешимость какой-либо проблемы с положением, когда некоторая проблема еще не решена. Для нерешенных проблем остается надежда найти разрешающий алгоритм, тогда как для алгоритмически неразрешимых проблем любые попытки поиска алгоритма бесполезны.

Прикладная теория алгоритмов базируется на выводах теории алгоритмов об алгоритмической разрешимости тех или иных проблем, но занимается, главным образом, разработкой наиболее эффективных с точки зрения практики алгоритмов, способов их описания, преобразования и реализации на современных ЭВМ.

Алгоритм рассматривается как совокупность определенным образом связанных между собой операторов, представляющих элементарные операции, которые производятся над множеством подвергающихся переработке объектов. Способы реализации операторов предполагаются известными (как правило, операторы сами являются некоторыми стандартными алгоритмами). При решении конкретной задачи задаются также значения исходных данных и параметров, входящих в описание алгоритма.

Для описания алгоритмов используются различные методы, отличающиеся степенью детализации и формализации. Теоретическое описание обычно дается в повествовательно-формальном изложении, цель которого – обосновать без

лишних подробностей процедуру, предлагаемую в качестве алгоритма. Для наглядного представления структуры алгоритмов широко применяются графические средства: графы, блок-схемы, сети. Формальное и полное описание алгоритмов осуществляется на специально разработанных алгоритмических языках (BASIC, FORTRAN, PASCAL, C и др.); такое описание содержит всю необходимую для реализации алгоритма информацию, но не связано непосредственно со специфическими особенностями вычислительных машин.

Вопросы для самопроверки по теме 5.1

1. Определите понятие алгоритма, эффективности алгоритмов и алгоритмической разрешимости.
2. Для чего используются математические модели алгоритма? Приведите примеры таких моделей.
3. Приведите примеры алгоритмически неразрешимых проблем.

5.2. Машина Тьюринга

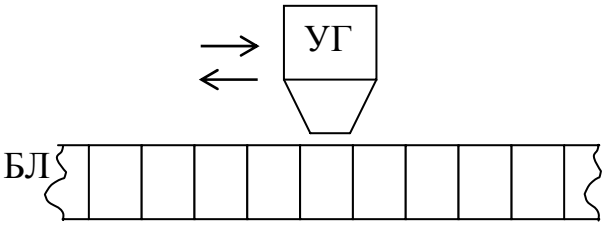

Рис. 5.1. Машина Тьюринга

Модель алгоритма, называемая машиной Тьюринга [1,5,11,14-19], состоит из бесконечной ленты (БЛ), разделенной на ячейки, и управляющей головки (УГ), которая перемещается по ленте и способна считывать символ в ячейке, против которой она находится, а также замещать обозреваемый символ новым (рис.5.1). В каждой ячейке может быть записан один символов из *ленточного алфавита A*. Головка может находиться в одном из внутренних состояний, принадлежащих конечному множеству (*алфавиту состояний*) *Q*. Работа машины происходит в дискретном времени в соответствии с программой, задаваемой набором команд вида

$$qa \to a^{+}Dq^{+}.$$

В зависимости от состояния головки $q \in Q$ и символа $a \in A$, против которого она стоит, головка записывает на ленте новый символ a^{+} (или оставляет старый), переходит в новое состояние q^{+} (или остается в старом) и передвигается: вправо (П), влево (Л) или остается в прежнем положении (Н).

Определение 5.1. Назовем *конфигурацией машины Тьюринга* K_t в момент t содержимое ее ленты, состояние головки $q \in Q$ и обозреваемый ею символ a.

Пусть на ленте записана цепочка символов $\ldots\Lambda a_1 a_3 a_1 a_2 a_3 a_1\Lambda\ldots$, слева и справа от которой свободные ячейки (содержат символ Λ), причем головка, находясь в состоянии q_i, обозревает символ a_2. Соответствующую конфигурацию будем записывать, помещая обозначение состояния головки перед обозреваемым символом: $a_1 a_3 a_1 q_i\, a_2 a_3 a_1$.

Определение 5.2. Конфигурация машины Тьюринга называется *заключительной*, если головка машины Тьюринга находится в состоянии останова q_0.

Работу машины Тьюринга можно описать как последовательную смену ее конфигураций, причем машина переходит от конфигурации K_t к конфигурации K_{t+1} в соответствии со своей программой. Любая начальная конфигурация K_0, которой соответствует состояние q_1, порождает последовательность конфигураций $K_0, K_1, K_2, \ldots, K_t, \ldots$.

Эта последовательность обрывается, если машина оказывается в заключительной конфигурации. В этом случае будем говорить, что машина Тьюринга *применима к конфигурации* K_0.

Если последовательность конфигураций $K_0, K_1, K_2, \ldots, K_t, \ldots$ никогда не обрывается, т.е. машина работает вечно (“зацикливается”), будем говорить, что машина Тьюринга *неприменима к конфигурации* K_0.

Для решения задачи исходные данные должны быть закодированы некоторым “естественным” образом символами некоторого алфавита A и

записаны в виде слова X на ленте машины, причем головка в начальном состоянии $q_1 \in Q$ обозревает самый левый символ слова X, т.е. начальная конфигурация имеет вид q_1X. Результирующая конфигурация имеет вид $q_0f(X)$.

В этом случае будем говорить, что машина Тьюринга *вычисляет словарную функцию f*, причем слово $f(X)$ есть значение этой функции для аргумента X. Числовые функции – это частный случай словарных, поскольку конкретный вид символов, которыми оперирует машина, несущественен, также как и тип данных: цифровых, алфавитно-цифровых и т.д.

В теории алгоритмов рассматриваются примеры специализированных машин Тьюринга с ленточным алфавитом $A=\{\Lambda,+,1\}$, алфавитом состояний $\{q_0,q_1,q_2, \ldots, q_n\}$ и алфавитом перемещений $D \in \{П,Л,Н\}$. Символ Λ играет роль разделителя. Символы q_1, q_0 – соответственно начальное и заключительное состояние машины (останов).

Рассматриваемые машины выполняют арифметические операции над неотрицательными целыми числами, для представления которых используется унитарный код. Число x представляется $(x+1)$-й единицей, причем отдельно записанная единица представляет нулевое значение x.

Пример 5.1 (Прибавление единицы). В табл. 5.1 приведен пример программы машины с алфавитом состояний $\{q_1,q_2,q_0\}$. Машина увеличивает значение числа на единицу.

Таблица 5.1. Программа прибавления единицы

a	q	
	q_1	q_2
Λ	-	$1Нq_0$
1	$1Пq_2$	-

Например, увеличение числа три на единицу машина осуществляет за два шага:

$$\ldots\Lambda\Lambda q_1 1111\Lambda\ldots \Rightarrow \ldots\Lambda q_2\Lambda 1111\Lambda\ldots \Rightarrow$$

$$\ldots\Lambda\, q_0 11111\Lambda\ldots\,.$$

Для исходной конфигурации $\ldots\Lambda q_1\Lambda 11\Lambda\ldots$ поведение машины не определено. Это означает, что рассматриваемая машина реализует *частичную словарную функцию*.

Можно показать, что машины Тьюринга способны реализовать основные программные структуры:

- суперпозицию программ;
- композицию программ;
- ветвление и циклические структуры.

Другими словами, машины Тьюринга обладают универсальными вычислительными возможностями. В том числе можно показать возможность построения универсальной машины Тьюринга.

Вопросы для самопроверки по теме 5.2

1. Как устроена машина Тьюринга и для чего она предназначена?
2. Что называется конфигурацией машины Тьюринга?
3. Как можно описать работу машины Тьюринга?
4. Каким образом машина Тьюринга вычисляет словарную функцию?
5. Каким образом задается программа машины Тьюринга?

5.3. Универсальные машины Тьюринга и алгоритмическая разрешимость

Проблема останова произвольной машины Тьюринга T с лентой t – это вопрос о возможности установить с помощью некоторой алгоритмической процедуры, что вычисления произвольной машины Тьюринга T с лентой t завершаются за конечное время, т.е. не происходит зацикливания.

Теорема 5.1. Проблема останова машины Тьюринга T с лентой t алгоритмически неразрешима.

Доказательство. Предположим, что существует машина Тьюринга D, которая решает проблему останова для машины Тьюринга T с лентой t, если на ленту машины D поместить описание (d_T) машины T (рис. 5.2).

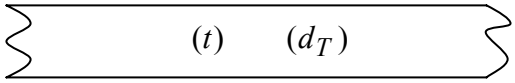

Рис. 5.2. Лента для проблемы останова

Диаграмма работы машины D представлена на рис. 5.3. Машина D решает проблему останова для всех пар машина-лента (T,t). Следовательно, она решает эту проблему для частного случая, когда лента t содержит описание самой машины T (самоанализ!), т.е. для случая, когда $t = d_T$. Для этого случая легко построить машину Тьюринга E, которой достаточно только одного описания d_T.

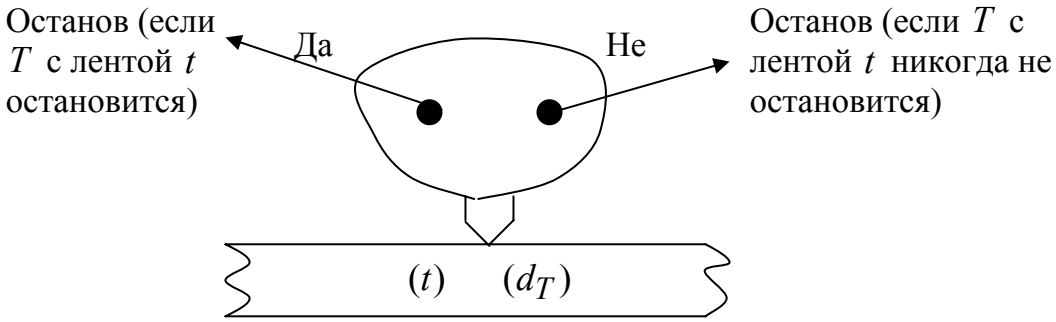

Рис. 5.3. Диаграмма работы машины *D*

Машина E преобразует это описание в $(d_T)(d_T)$ и далее работает как D. Диаграмма состояний машины E имеет два выхода, как показано на рис.5.4.

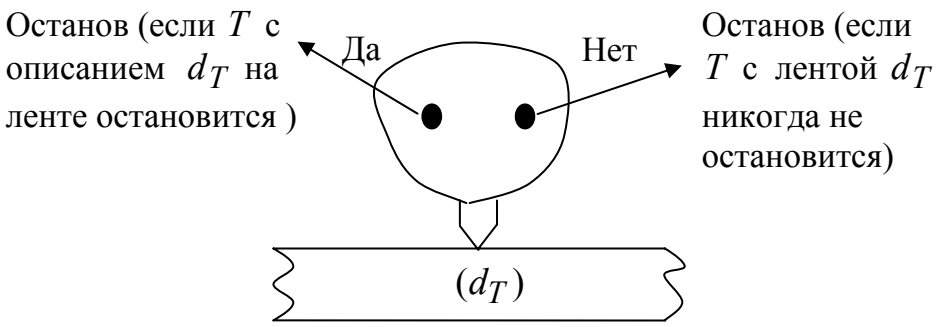

Рис. 5.4. Диаграмма работы машины *E*

Модифицируем E следующим образом: добавим два новых состояния s' и s''. Получим машину E^*, показанную на рис. 5.5.

Машина E^* с лентой d_T останавливается, если T с лентой d_T никогда не остановится, и наоборот.

Поместим на ленту машины E^* описание $d_T = d_{E^*}$ самой этой машины. Тогда машина E^* с лентой d_{E^*} останавливается, если машина E^* с лентой d_{E^*} никогда не останавливается, и наоборот. Но этого не может быть!

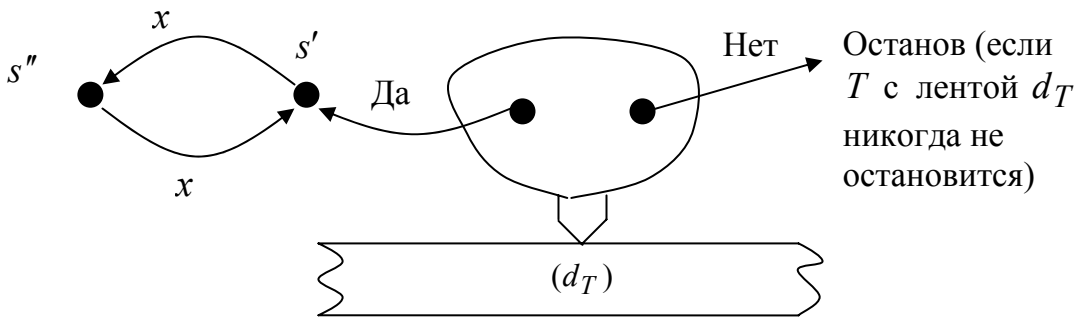

Рис. 5.5. Диаграмма работы машины *E**

Таким образом, машина E^*, а значит и E, а значит и D, не может существовать. Следовательно, проблема останова не разрешима. *Что и требовалось доказать.*

Вопросы для самопроверки по теме 5.3

1. Как формулируется проблема останова?
2. Какое значение имеет неразрешимость проблемы останова в теории вычислительных систем?
3. Как доказывается неразрешимость проблемы останова с помощью модели алгоритма в виде машины Тьюринга?

5.4. Элементы теории рекурсивных функций

Рекурсия – способ определения функций, являющийся одним из основных объектов изучения в теории алгоритмов. Смысл рекурсии в том, что значение функции f в точке x определяется через значения этой функции в "предшествующих" точках.

Определение 5.3. *Оператор примитивной рекурсии* определяет $(n+1)$-местную функцию $f(x_1,...,x_n;y)$ через n-местную функцию $g(x_1,...,x_n)$ и $(n+2)$-местную функцию $h(x_1,...,x_n; y; z)$ следующим образом:

$f(x_1,...,x_n;0)=g(x_1,...,x_n)$,

$f(x_1,...,x_n; 1)=h(x_1,...,x_n; 0; f(x_1,...,x_n; 0))$,

$f(x_1,...,x_n; 2)=h(x_1,...,x_n; 1; f(x_1,...,x_n; 1))$,

...

$f(x_1,...,x_n; m+1)=h(x_1,...,x_n; m; f(x_1,...,x_n; m))$.

Пример 5.2. Пусть переменные $x_1,...,x_n$ отсутствуют, причем рекурсивная схема задана соотношениями

$$\begin{cases} g = 0, \\ h(y;z) = 2 * y + z. \end{cases}$$

Тогда $f(0)=0$, $f(1)=2*0+0=0$, $f(2)=2*1+0=2$, $f(3)=2*2+2=6$, $f(4)=2*3+6=12$, $f(5)=2*4+12=20$, ..., $f(x+1)=2*x+f(x)$.

Функцию определенную данной рекурсивной схемой можно выразить аналитически:

$f(x+1)=2*x+f(x)=2*x+2*(x-1)+f(x-1)=...=$

$2*x+2*(x-1)+2*(x-2)+...+2*2+2*1=$

$2*[x+(x-1)+(x-2)+...+2+1]$.

Выражение в прямоугольных скобках [ ] представляет сумму чисел от 1 до x, равную по формуле арифметической прогрессии $(x+1)*x/2$, поэтому получаем $f(x+1)=(x+1)*x$, или $f(x)=x*(x-1)$.

Определение 5.4. *Примитивно-рекурсивная функция* (ПР-функция) это одна из простейших функций:

- *следования* $S(x)=x+1=x'$,
- *константы ноль* $0(x)=0$,
- *тождества* $Im(x_1,...,x_n)=x_m$ $(1\le m\le n)$,
- либо функция, которая может быть получена из простейших функций с помощью применения конечного числа операторов *подстановки и примитивной рекурсии*.

Все ПР-функции всюду определенные. В предыдущем примере мы имели ПР-функцию $f(x)=x*(x-1)$. Ниже мы рассмотрим и другие примеры ПР-функций. В дальнейшем изложении и примерах мы предполагаем, что области определения и значения ПР-функций – множества неотрицательных целых чисел (что не уменьшает общности выводов).

Примеры ПР-функций

***Функция сложения*:**

$f_c(x; 0)=x$,

$f_c(x; 1)=h(x; 0; f_c(x; 0))=S(f_c(x; 0))$,

...

$f_c(x; m+1)=S(f_c(x; m))$.

***Функция умножения*:**

$f_y(x; 0)=0,$

$f_y(x; 1)=h(x; 0; f_y(x; 0))=f_c(x; f_y(x; 0)),$

...

$f_y(x; m+1)=f_c(x; f_y(x; m)).$

Функции сигнум и антисигнум. Поскольку мы не используем отрицательных чисел, определим функцию сигнум следующим образом

$$\operatorname{sgn}(x)=\begin{cases}0, \text{ если } x=0;\\ 1, \text{ если } x>0,\end{cases}$$

и антисигнум как

$$\overline{\operatorname{sgn}}(x)=\begin{cases}1, \text{ если } x=0;\\ 0, \text{ если } x>0.\end{cases}$$

Обе эти функции рекурсивны и могут быть заданы схемами:

$$\begin{cases}\operatorname{sgn}(0) = 0,\\ \operatorname{sgn}(x+1)=1;\end{cases} \quad \text{и} \quad \begin{cases}\overline{\operatorname{sgn}}(0) = 1,\\ \overline{\operatorname{sgn}}(x+1)=0.\end{cases}$$

Усеченная разность. Разность $x - y$ не определена для неотрицательных целых x и y, т.е. является частичной функцией. Введем у*сеченную разность* $x\neg y$, доопределив $x - y$ следующим образом

$$x\neg y=\begin{cases}x-y, \text{ если } x-y\geq 0,\\ 0 \text{ в противном случае.}\end{cases}$$

Усеченная разность определяется рекурсивной схемой

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

Модуль разности $|x - y|$ можно выразить через усеченную разность:

$$|x - y| = (x \neg y) + (y \neg x).$$

Оператор минимизации

Примитивной рекурсии недостаточно, чтобы задать любую вычислимую функцию. В общем случае, требуется еще *оператор минимизации*

$$\mu_t[f(x_1,...,x_{n-1}; t)=x_n],$$

который обозначает вычисление минимального значения t, при котором выполняется равенство $f(x_1,...,x_{n-1}; t)=x_n$. Могут быть следующие случаи, когда значение оператора минимизации не определено:

1) значение $f(x_1,...,x_{n-1}; 0)$ не определено;

2) значения $f(x_1,...,x_{n-1}; y)$ при $y=0,1,2,...,k$ определены, но

отличны от x_n, а при $y=k+1$ не определены;

3) значения $f(x_1,...,x_{n-1}; y)$ при всех $y=0,1,2,3,...$ определены,

но отличны от x_n.

Определение 5.5. Функция называется *частично-рекурсивной* (ЧР), если она может быть получена из $S(x)$, $0(x)$ и Im с помощью применения конечного числа операторов подстановки, примитивной рекурсии и минимизации.

Определение 5.6. ЧР-функция называется *общерекурсивной* (ОР), если она всюду определена.

Примеры ЧР-функций

Целая часть **$[x/y]$**. Функция x/y является частичной, поскольку для $y=0$ не определена. Введем в рассмотрение функцию $[x/y]$, полученную доопределением x/y так, что $[x/0]=x$. Можно показать, что $[x/y]= \mu_z[(x \neg z)((x + +1) \neg y(z + 1))=0]$, следовательно, $[x/y]$ – ОР-функция [5].

Функция **$[x \bmod y]$.** Функция $x \bmod y$ (остаток от деления x на y) является частичной, поскольку для $y=0$ не определена. Введем в рассмотрение функцию $[x \bmod y]$, полученную доопределением $x \bmod y$ так, что $[x \bmod 0]=x$.

Можно показать, что $[x \bmod y]= x \neg y[x/ y]$, следовательно, $[x \bmod y]$ – ОР-функция [5].