Е.Н. Дронова

Основные алгоритмические модели

Об издании

Основной титульный экран

Дополнительный титульный экран непериодического издания – 1

Дополнительный титульный экран непериодического издания – 2

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Алтайский государственный педагогический университет»

Е.Н. Дронова

ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ МОДЕЛИ

Учебное пособие

Барнаул

ФГБОУ ВО "АлтГПУ"
2016

Об издании - 1, 2, 3.

ISBN 978–5–88210–814–3

УДК 002(075)+510(075)
ББК 32.97я73+22.1я73
Д758

Дронова, Е.Н.
Основные алгоритмические модели [Электронный ресурс] : учебное пособие / Е.Н. Дронова. – Барнаул : АлтГПУ, 2016. – Систем. требования: ПК с Intel® x86-совместимый процессором, Pentium® 4 или новее ; 512 Мб ОЗУ ; Windows XP и более поздние ; Adobe Acrobat Reader ; SVGA видеоплата и монитор (1024x768, 16 млн цв.) ; мышь.
ISBN 978–5–88210–814–3

Рецензенты:
Алтухов Ю.А., доктор физико-математических наук, профессор (АлтГТУ);
Афонина М.В., кандидат педагогических наук, доцент (АлтГПУ)

В пособии представлено описание таких алгоритмических моделей, как класс рекурсивных функций, машина Тьюринга, машина Поста, машины произвольного доступа, нормальные алгоритмы Маркова. Особое внимание уделено разработке вычислительных алгоритмов в указанных алгоритмических моделях.
Пособие предназначено студентам педагогических вузов, изучающих теорию алгоритмов.
Рекомендовано к изданию редакционно-издательским советом АлтГПУ 28.01.2016 г.

Деривативное электронное издание.
Текстовое (символьное) электронное издание.

Системные требования:

ПК с Intel® x86-совместимый процессором, Pentium® 4 или новее ; 512 Мб ОЗУ ; Windows XP и более поздние ; Adobe Acrobat Reader ; SVGA видеоплата и монитор (1024x768, 16 млн цв.) ; мышь.

Об издании - 1, 2, 3.

Электронное издание создано при использовании программного обеспечения Sunrav BookOffice.

Объём издания - 8 166 КБ.

Дата подписания к использованию: 30.03.2016

Федеральное государственное бюджетное образовательное учреждение высшего образования «Алтайский государственный педагогический университет» (ФГБОУ ВО «АлтГПУ»)

ул. Молодежная, 55, г. Барнаул, 656031

Тел. (385-2) 36-82-71, факс (385-2) 24-18-72

e-mail: rector@altspu.ru, http://www.altspu.ru

Об издании - 1, 2, 3.

Содержание

Предисловие

Данное учебное пособие предназначено для тех, кто изучает теорию алгоритмов в высших учебных заведениях. Автор знакомит будущих специалистов с основными понятиями теории алгоритмов, описывает основные алгоритмические модели и правила их функционирования, показывает взаимосвязь теории алгоритмов со школьным курсом информатики.

В *первой главе* представлен материал о сущности понятия алгоритма, о развитии и использовании его в науке, о становлении и задачах теории алгоритмов, об основных направлениях ее применения. *Главы 2, 3, 4, 5, 6* посвящены различным алгоритмическим моделям; здесь описаны назначение, устройство и функционирование следующих формальных моделей алгоритма: класс частично-рекурсивных функций, машина Тьюринга, машина Поста, машины произвольного доступа, нормальные алгоритмы Маркова.

Все главы в пособии завершаются списком вопросов и заданий. Вопросы, приведенные в конце каждой главы, помогут вам понять, хорошо ли вы усвоили новый материал, и ускорят процесс его повторения. Подобранные нами задачи к главам весьма разнообразны – среди них есть простые, а есть и такие, для решения которых нужно проявить смекалку. Мы надеемся, что каждый здесь найдет себе задачи по вкусу.

В нашем пособии, как и в любом другом, вам встретятся новые термины. Для удобства они напечатаны курсивом. Ключевые идеи и слова в тексте также выделены курсивом. Мы не считаем важным заучивать их наизусть, но понимать их смысл и уметь применять на практике вы должны научиться. С этой целью изложенный нами в пособии материал иллюстрирован большим количеством примеров. Часть этих примеров позволит вам более глубоко усвоить сущность новых терминов, часть – освоить правила оперирования с ними и научиться применять их при решении задач.

Часть ключевых задач, позволяющих усвоить новый материал на более высоком уровне, приведена не в содержании соответствующей главы, а вынесена в список заданий к ней. Решение этих задач предлагается найти самостоятельно, что потребует не только понимания изученного материала, но и проявления творчества. Однако если решение какой-либо задачи вам не удастся найти – не огорчайтесь: подробное решение ключевых задач представлено нами в конце пособия, там же вы сможете найти ответы и указания к решению большинства задач, приведенных в конце всех глав.

В завершении подчеркнем, что данное пособие не является исчерпывающим руководством по изучению основных алгоритмических моделей. Оно призвано помочь студентам более тщательно разобраться с соответствующими основными понятиями, научиться решать типовые задачи и позволит в дальнейшем (при написании курсовых, дипломных работ) приступить уже со знанием дела к изучению более подробной специальной литературы.

Глава 1. «Алгоритм» как центральное понятие теории алгоритмов

1.1. Развитие понятия алгоритма

Понятие «алгоритм» давно является привычным не только для математиков. Оно является концептуальной основой разнообразных процессов обработки информации, возможность автоматизации таких процессов обеспечивается наличием соответствующих алгоритмов. В упрощенном понимании «алгоритм» – это то, что можно запрограммировать на ЭВМ.

Термин «алгоритм» происходит от *algorithmi* – латинской формы написания великого математика средневекового Востока *Аль-Хорезми*. Он жил приблизительно с 783 по 850 гг. в городе Ургенче – областном центре современной Хорезмской области Узбекистана. Его научные работы посвящены правилам выполнения арифметических действий в десятичной системе счисления. Роль их в развитии математики огромна. Связано это во многом с тем, что в XI веке еще не была разработана математическая символика (знаки операций, скобки, буквенные обозначения и т. п.). В связи с этим, Аль-Хорезми в своих работах стремился выработать такой стиль четкого, строгого словесного предписания, который бы не позволил читателям уклониться от предписанных действий или что-либо пропустить в них. Разработанные этим ученым правила выполнения арифметических действий над десятичными числами начинались (в латинском варианте) словами «Алгоризми сказал». С течением времени это выражение преобразовалось во фразу «алгоритм гласит».

Таким образом, слово «алгоритм» происходит от имени ученого Аль-Хорезми и как научный термин первоначально обозначало лишь правила выполнения арифметических действий в десятичной системе счисления.

С течением времени слово «алгоритм» приобрело более широкий смысл и стало обозначать не только правила цифровых вычислений десятичной позиционной арифметики, но и любые точные правила действий в произвольных процессах, следуя которым искомые величины решаемых задач можно было найти из исходных данных.

Вплоть до 30-х годов XX века понятие алгоритма оставалось интуитивно понятным: под *алгоритмом* понимали конечную последовательность элементарных действий, направленных на решение поставленной задачи. К этому времени были известны такие яркие примеры алгоритмов, как алгоритм Евклида для нахождения наибольшего общего делителя двух натуральных чисел, алгоритм Гаусса для решения системы линейных уравнений над полем, алгоритм нахождения рациональных корней многочленов одного переменного с рациональными коэффициентами, алгоритм разложения многочлена одного переменного над конечным полем на неприводимые множители и т. д.

Указанные математические проблемы решены путем указания конкретных разрешающих процедур (алгоритмов). Иными словами, для доказательства *алгоритмической разрешимости* той или иной проблемы достаточно представить алгоритм ее решения.

Наряду с алгоритмически разрешимыми проблемами к началу XX века были сформулированы такие проблемы, алгоритмическая разрешимость которых была маловероятна. В связи с этим, возник вопрос «Как доказать, что та или иная проблема (задача) не имеет алгоритма решения (*алгоритмически неразрешима*)»?

Данный вопрос позволил немного прояснить сложившуюся на тот период времени ситуацию: для того, чтобы можно было строго математически доказать отсутствие алгоритма решения некоторой задачи, необходимо сначала строго математически определить само понятие алгоритма.

Таким образом, в среде ученых обострился следующий вопрос: «Как формализовать понятие "алгоритм"?»

1.2. Возникновение и основные этапы развития теории алгоритмов как науки

Начальной точкой отсчета современной теории алгоритмов можно считать теорему о неполноте символических логик, доказанную немецким математиком *Куртом Геделем* в 1931 году. В этой работе было показано, что некоторые математические проблемы не могут быть решены алгоритмами из определенного класса. Важность результата Геделя связана с вопросом о том, совпадает ли использованный им класс алгоритмов с классом всех алгоритмов в интуитивном понимании этого термина. Эта работа дала толчок к поиску и анализу различных формализаций понятия «алгоритм».

Первые фундаментальные работы по теории алгоритмов были опубликованы *в середине 1930-х годов Аланом Тьюрингом, Алоизом Черчем и Эмилем Постом.* Предложенные ими машина Тьюринга, машина Поста и класс частично-рекурсивных функций Черча были первыми формальными описаниями алгоритма, использующими строго определенные модели вычислений (**алгоритмическими моделями**). Сформулированные гипотезы Тьюринга, Поста и Черча постулировали эквивалентность предложенных ими моделей вычислений и интуитивного понятия алгоритма. Важным развитием этих работ стала формулировка и доказательство существования алгоритмически неразрешимых проблем.

В 1950-е годы существенный вклад в развитие теории алгоритмов внесли работы *А.Н. Колмогорова и А.А. Маркова.* Формальные модели Поста, Тьюринга и Черча, равно как и модели А.Н. Колмогорова и А.А. Маркова, оказались эквивалентными в том смысле, что любой класс проблем, разрешимых в одной модели, разрешим и в другой.

Появление доступных ЭВМ и существенное расширение круга решаемых на них задач привели в *1960–70-х годах* к практически значимым исследованиям алгоритмов и вычислительных задач. На этой основе в данный период оформились следующие разделы в теории алгоритмов:

- классическая теория алгоритмов (формулировка задач в терминах формальных языков, понятие задачи разрешения, описание сложностных классов задач, формулировка в 1965 году Эдмондсом проблемы $P = NP$, открытие класса NP-полных задач и его исследование и др.);
- теория асимптотического анализа алгоритмов (понятие сложности и трудоемкости алгоритма, критерии оценки алгоритмов, методы получения асимптотических оценок, в частности для рекурсивных алгоритмов, асимптотический анализ трудоемкости или времени выполнения, получение теоретически нижних оценок сложности задач);
- теория практического анализа вычислительных алгоритмов (получение явных функций трудоемкости, практически значимые критерии качества алгоритмов, методики выбора рациональных алгоритмов).

Обобщая исследования в различных разделах теории алгоритмов, можно выделить следующие *основные задачи и направления развития, характерные для современной теории алгоритмов:*

- формализация понятия «алгоритм» и исследование формальных алгоритмических систем (моделей вычислений);
- доказательство алгоритмической неразрешимости задач;
- формальное доказательство правильности и эквивалентности алгоритмов;
- классификации задач, определение и исследование сложностных классов;
- доказательство теоретических нижних оценок сложности задач;

- получение методов разработки эффективных алгоритмов;
- асимптотический анализ сложности итерационных алгоритмов;
- исследование и анализ рекурсивных алгоритмов;
- получение явных функций трудоемкости алгоритмов;
- разработка классификаций алгоритмов;
- исследование емкостной (по ресурсу памяти) сложности задач и алгоритмов;
- разработка критериев сравнительной оценки ресурсной эффективности алгоритмов и методов их сравнительного анализа.

Полученные в теории алгоритмов результаты находят сегодня достаточно широкое практическое применение, в рамках которого можно выделить два аспекта: теоретический и практический.

Теоретический аспект: при исследовании некоторой задачи результаты теории алгоритмов позволяют ответить на вопрос – является ли эта задача в принципе алгоритмически разрешимой? В случае алгоритмической разрешимости задачи следующим важным теоретическим вопросом является вопрос о принадлежности этой задачи к классу *NP*-полных задач. При утвердительном ответе можно говорить о существенных временных затратах для получения точного решения этой задачи для больших размерностей исходных данных, иными словами – об отсутствии быстрого точного алгоритма ее решения.

Практический аспект: методы и методики теории алгоритмов, в основном асимптотического и практического анализа, позволяют осуществить:

- рациональный выбор из известного множества алгоритмов решения данной задачи, учитывающий особенности их применения в разрабатываемой программной системе;
- получение временных оценок решения сложностных задач на основе функции трудоемкости;
- получение достоверных оценок невозможности решения некоторой задачи за определенное время;
- разработку и совершенствование эффективных алгоритмов решения практически значимых задач в области обработки информации.

1.3. Алгоритмы: интуитивное представление об алгоритмах, исполнитель алгоритма, свойства и способы записи алгоритмов

Понятие «алгоритм» является центральным понятием теории алгоритмов, поэтому рассмотрим его более подробно.

3.1. Интуитивное представление об алгоритмах

3.2. Исполнитель алгоритма

3.3. Свойства алгоритмов

3.4. Способы записи алгоритмов

1.3.1. Интуитивное представление об алгоритмах

Первоначально под словом «алгоритм» понимали способ выполнения арифметических действий над десятичными числами (*см. 1.1*). В дальнейшем это понятие стали использовать для обозначения любой последовательности действий, приводящей к решению поставленной задачи. Современный человек понимает под *алгоритмом* четкую систему инструкций о выполнении в определенном порядке некоторых действий для решения всех задач какого-то данного класса.

Многочисленные и разнообразные алгоритмы окружают нас буквально во всех сферах жизни и деятельности. Многие наши действия доведены до бессознательного автоматизма, мы порой и не осознаем, что они регламентированы определенным алгоритмом четкой системой инструкций. Например, наши действия при входе в продуктовый магазин (сдать свою сумку, получить корзину с номером, пройти в торговый зал, заполнить корзину продуктами, оплатить покупку в кассе, предъявить чек контролеру, взять свою сумку, переложить в нее продукты, сдать корзину, покинуть магазин).

Вместе с тем, есть немало таких действий, выполняя которые, мы тщательно следуем той или иной инструкции. Это непривычные действия, профессионально нам не свойственные. Например, если вы никогда раньше не пекли торт, то, получив рецепт его приготовления, вы будете стараться выполнить в указанной последовательности все его предписания.

Говоря об алгоритмах, нельзя не подчеркнуть их огромную роль в математике. Каждый из вас буквально с первых классов школы при изучении математики встретился с большим количеством алгоритмов. Это, прежде всего, алгоритмы выполнения четырех арифметических действий над различными числами натуральными, целыми, дробными, комплексными; алгоритмы геометрических построений с помощью циркуля и линейки (деление пополам отрезка и угла, опускание и восстановление перпендикуляров, проведение параллельных прямых); алгоритмы вычисления площадей и объемов различных геометрических фигур и т. д. При изучении математики в вузе вами были освоены алгоритм вычисления наибольшего общего делителя двух натуральных чисел (алгоритм Евклида), алгоритм нахождения определителей различных порядков, алгоритм вычисления интегралов от рациональных функций и т. д.

Иными словами, алгоритмы широко распространены как в практике, так и в науке, и требуют более внимательного к себе отношения и тщательного изучения.

Следует выделить следующую особенность понятия «алгоритм»: оно не является формальным математическим понятием, так как в его определении используются такие неуточняемые понятия, как «точные правила действий», «последовательность элементарных действий» и т. д. Вместе с тем, описание понятия алгоритма раскрывает его суть; употребляя термин «алгоритм», мы можем пояснить его смысл. Поэтому принято говорить, что *«алгоритм» это интуитивное понятие*.

Из разнообразных вариантов словесного описания сущности понятия «алгоритм» приведем, на наш взгляд, наиболее удачные [3]:

алгоритм (по А.Н. Колмогорову) это всякая система вычислений, выполняемых по строго определенным правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи;

алгоритм (по А.А. Маркову) это точное предписание, определяющее вычислительный процесс, идущий от варьируемых исходных данных к искомому результату.

Подчеркнем, что, несмотря на существование различных определений понятия «алгоритм», в явной или неявной форме все эти определения постулируют следующий ряд *требований к алгоритму:*

− алгоритм должен содержать конечное количество элементарно выполнимых предписаний, т. е. удовлетворять требованию конечности записи;

− алгоритм должен выполнять конечное количество шагов при решении задачи, т. е. удовлетворять требованию конечности действий;

− алгоритм должен быть единым для всех допустимых исходных данных, т. е. удовлетворять требованию универсальности;

− алгоритм должен приводить к правильному по отношению к поставленной задаче решению, т. е. удовлетворять требованию правильности.

1.3.2. Исполнитель алгоритма

Любой алгоритм существует не сам по себе, а предназначен для определенного исполнителя действий. Так, алгоритм вычисления производной для полинома фиксированной степени вполне ясен тем, кто знаком с основами математического анализа, но для прочих он может оказаться совершенно непонятным.

Исполнитель алгоритма — это тот объект (или субъект), для управления которым составляется алгоритм. Иными словами, исполнитель алгоритма — это некоторая абстрактная или реальная система (техническая, биологическая или биотехническая), способная выполнить действия, предписываемые алгоритмом. Исполнителем алгоритма может быть человек, группа людей, робот, станок, компьютер, язык программирования и т. д. Его характеризуют:

- среда;
- система команд;
- элементарные действия;
- отказы.

Среда (или обстановка) — это «место обитания» исполнителя.

Система команд исполнителя (СКИ) — это конечное множество команд, которые понимает исполнитель, т. е. умеет их выполнять. Для каждой такой команды должны быть заданы *условия применимости* (в каких состояниях среды может быть выполнена команда) и описаны *результаты выполнения команды*.

Элементарное действие совершается исполнителем после вызова какой-либо команды.

Отказы исполнителя возникают, если команда вызывается при недопустимом для нее состоянии среды.

В качестве примера рассмотрим широко известного учебного исполнителя *Кенгуренка*.

Среда исполнителя. На экране присутствуют три основных элемента среды учебного исполнителя: строка меню, поле программы и поле рисунка, на котором находится Кенгуренок. На поле рисунка неявно (т. е. ее не видно) нанесена прямоугольная сетка. Длину стороны одной квадратной ячейки этой сетки назовем шагом. Размер всего поля — 15 шагов по горизонтали и 19 шагов по вертикали.

Система команд исполнителя. Команды делятся на команды установки режимов (определенного состояния учебного исполнителя, в котором могут выполняться определенные действия) и команды управления Кенгуренком.

Команды установки режимов следующие:

- пуск (запуск на исполнение готовой программы в пошаговом автоматическом режиме);
- отладка (выполнение программы в отладочном режиме с остановкой после каждой команды);
- установка (очистка поля и установка положения Кенгуренка с помощью клавиш перемещения курсора);
- разное (содержит подменю с дополнительными командами работы с файлами — чтение, запись, печать программы, печать рисунка, стереть программу);

– результат (мгновенное получение результата работы программы автоматический режим исполнения).

Команды управления Кенгуренком следующие:

- шаг (перемещение Кенгуренка на один шаг вперед с рисованием линии);
- поворот (поворот Кенгуренка на 90^0 против часовой стрелки);
- прыжок (перемещение Кенгуренка на один шаг вперед без рисования линии);
- пока <условие> повторять <тело цикла> конец цикла (цикл с предусловием);
- если <условие> то <серия 1> иначе <серия 2> конец ветвления (полное ветвление);
- если <условие> то <серия> конец ветвления (неполное ветвление);
- сделай <имя процедуры> (обращение к процедуре).

Отметим еще одну важнейшую характеристику исполнителя алгоритма: исполнитель не вникает в смысл того, что он делает! Он действует *формально*, т. е. отвлекается от содержания поставленной задачи и только строго согласно алгоритму выполняет некоторые инструкции, действия. Целесообразность же предусматриваемых алгоритмом действий обеспечивается точным анализом со стороны того, кто составляет алгоритм.

Раскрытое нами понятие исполнителя алгоритма позволяет привести следующее определение термина «алгоритм»: «*Алгоритм* понятное и точное предписание исполнителю выполнить конечную последовательность команд, приводящих от исходных данных к искомому результату».

В приведенном определении содержатся основные понятия, связанные с алгоритмом и его свойствами. Взаимосвязь этих понятий отражена нами на рис. 1.

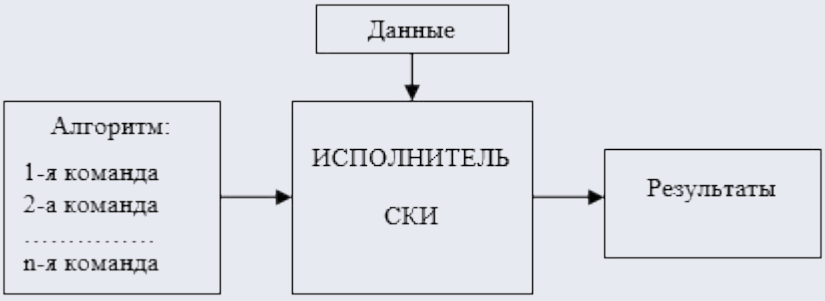


Рис. 1. Схема функционирования исполнителя алгоритмов

1.3.3. Свойства алгоритмов

Несмотря на то, что понятие алгоритма имеет интуитивный характер, его определение нестрогое, можно выделить следующие характерные черты алгоритма.

1. *Дискретность.* Описываемый алгоритмом процесс должен быть разбит на последовательность отдельных шагов. Возникающая в результате такого разбиения запись представляет собой упорядоченную совокупность четко разделенных друг от друга предписаний, образующих прерывную (дискретную) структуру алгоритма.

2. *Понятность.* Алгоритм не должен содержать предписаний, смысл которых может восприниматься исполнителем неоднозначно, т. е. запись алгоритма должна быть настолько четкой и полной, чтобы у исполнителя не возникало потребности в принятии каких-либо самостоятельных решений.

3. *Детерминированность* (или *определенность*). Запись алгоритма должна быть четкой, полной и продуманной в деталях, чтобы у исполнителя не могло возникнуть потребности в принятии решений. Кроме того, в алгоритмах недопустимы также ситуации, когда после выполнения очередной команды алгоритма исполнителю неясно, какая из команд алгоритма должна выполняться на следующем шаге.

4. *Результативность* (или *направленность, конечность*). Выполнение алгоритма обязательно должно привести к решению поставленной задачи, либо к сообщению о том, что при заданных исходных величинах задачу решить невозможно. Алгоритмический процесс не может обрываться безрезультатно.

5. *Массовость.* Алгоритм пригоден для решения любой задачи из некоторого класса задач, т. е. алгоритм правильно работает на некотором множестве исходных данных, которое называется областью применимости алгоритма.

Вопрос: «Возможна ли ситуация, когда способ решения задачи есть, но он не является алгоритмом?»

Ответ. Не каждый способ, приводящий к решению задачи, является алгоритмом. Например, опишем следующий способ (метод) проведения перпендикуляра к прямой *MN*, проходящего через заданную точку *A*:

1) отложить в обе стороны от точки *A* на прямой *MN* циркулем отрезки равной длины с концами *B* и *C*;

2) увеличить раствор циркуля до радиуса, в полтора-два раза большего длины отрезков *AB* и *AC*;

3) провести указанным раствором циркуля дуги окружностей с центрами *B* и *C* так, чтобы они охватили точку *A* и образовали две точки пересечения друг с другом (*D* и *E);*

4) взять линейку, приложить ее к точкам *D* и *E* и соединить их отрезком.

При правильном построении отрезок пройдет через точку *A* и будет искомым перпендикуляром.

Указанный способ рассчитан на исполнителя-человека. Применяя его, человек, разумеется, построит искомый перпендикуляр. Но, тем не менее, этот способ алгоритмом не является. Прежде всего, оно не обладает свойством детерминированности. Так, в пункте 1 требуется от исполнителя сделать выбор

отрезка *произвольной* длины (для построения точек B и C можно провести окружность произвольного радиуса r с центром в точке A). В пункте 2 требуется сделать выбор отрезка в полтора-два раза большего длины отрезков AB и AC. В пункте 3 надо провести дуги, которые также однозначно не определены. Человек-исполнитель, применяющий данный способ к одним и тем же исходным данным (прямой MN и точке A) повторно, получит несовпадающие промежуточные результаты. Это противоречит требованию детерминированности алгоритма.

Вышесказанное позволяет дать уточненное понятие алгоритма, которое опять же не является определением в математическом смысле слова, но более формально описывает понятие алгоритма, раскрывает его сущность.

Алгоритм это конечная система команд, сформулированная на языке исполнителя, которая определяет последовательность перехода от допустимых исходных данных к конечному результату и которая обладает свойствами дискретности, понятности, детерминированности, результативности и массовости.

1.3.4. Способы записи алгоритмов

Алгоритм, составленный для некоторого исполнителя, можно представить различными способами: с помощью графического или словесного описания, в виде таблицы, последовательности формул, записанных на алгоритмическом языке, и т. д. Рассмотрим следующие наиболее распространенные формы представления алгоритма:

- *словесная* (запись на естественном языке);
- *графическая* (изображения из графических символов);
- *псевдокоды* (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
- *программная* (тексты на языках программирования).

Опишем эти способы записи алгоритма более подробно.

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.

Например, алгоритм нахождения наибольшего общего делителя двух натуральных чисел может быть следующим:

1) задать два числа;
2) если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;
3) определить большее из чисел;
4) заменить большее из чисел разностью большего и меньшего из чисел;
5) повторить алгоритм с шага 2.

Укажем недостатки использования словесной формулировки алгоритма:

- описание действий строго неформализуемы;
- многословность записей;
- неоднозначность толкования отдельных предписаний.

Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным.

Примером графического представления алгоритма может служить его представление в виде *блок-схемы*, когда алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.

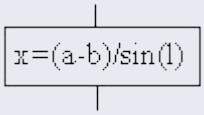
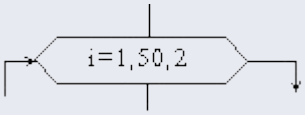
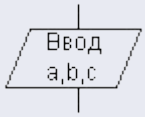
В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений,

проверки условий, управлению повторением действий и т. д.) соответствует геометрическая фигура, представленная в виде *блочного символа*. Блочные символы соединяются *линиями переходов*, определяющими очередность выполнения действий. В таблице 1 приведены наиболее часто употребляемые блочные символы.

Блок *«процесс»* применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных. Для улучшения наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.

Таблица 1

Блочные символы

Название символа	*Обозначение и пример заполнения*	*Пояснение*
Процесс		Вычислительное действие или последовательность действий
Условие		Проверка условий
Модификация		Начало цикла с заранее известным количеством повторов
Предопределенный процесс		Вычисления по подпрограмме
Ввод-вывод		Ввод и вывод данных
Пуск-останов		Начало, конец алгоритма
Печать		Вывод результатов на печать

Блок *«условие»* используется для обозначения переходов управления по условию. В каждом таком блоке должны быть указаны вопрос, условие или сравнение, которые он определяет.

Блок *«модификация»* используется для организации циклических конструкций. (Слово «модификация» означает видоизменение, преобразование). Внутри блока записывается параметр цикла, для которого

указываются его начальное значение, конечное значение и шаг изменения значения параметра для каждого повторения.

Блок *«предопределенный процесс»* используется для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.

Недостаток блок-схем в том, что при описании сложных алгоритмов они превращаются в весьма запутанную сеть.

Псевдокод представляет собой систему правил, предназначенную для записи алгоритмов с помощью текстовых структур. Он занимает промежуточное место между естественными и формальными языками.

С одной стороны, псевдокод близок к естественному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой стороны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.

В псевдокоде не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя.

Однако в псевдокоде обычно имеются некоторые конструкции, присущие формальным языкам, что облегчает переход от записи на псевдокоде к записи алгоритма на формальном языке. В частности, в псевдокоде, как и в формальных языках, есть *служебные слова*, смысл которых строго определен. Они выделяются в печатном тексте жирным шрифтом, а в рукописном тексте подчеркиваются.

Формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций.

Примером псевдокода является *школьный алгоритмический язык*, описанный в учебнике А.Г. Кушниренко «Основы информатики и вычислительной техники». Общий вид алгоритма на этом языке можно представить следующим образом:

```
алг название алгоритма (аргументы и результаты)
    дано условия применимости алгоритма
    надо цель выполнения алгоритма
нач описание промежуточных величин
|   последовательность команд (тело алгоритма)
кон
```

Приведем пример записи конкретного алгоритма на школьном алгоритмическом языке.

```
алг сумма квадратов (арг цел n, рез цел S)
    дано n > 0
    надо S = 1*1 + 2*2 + 3*3 + ... + n*n
нач цел i
|   ввод n; S := 0
|   нц для i от 1 до n
|   |   S := S + i*i
|   кц
|   вывод "S = ", S
кон
```

Описание алгоритма в словесной, графической формах, на псевдокоде допускает некоторый произвол при изображении команд. Вместе с тем, эти способы позволяют человеку понять суть дела и исполнить алгоритм. Однако для компьютера алгоритм должен быть записан на «понятном» ему языке, на языке программирования. *Программа* это описание структуры алгоритма на языке программирования.

Выбор того или иного способа записи алгоритма зависит от нескольких причин. Так, если для вас наиболее важна наглядность записи алгоритма, то разумно использовать блок-схему. Если алгоритм небольшой, то его можно записать в текстовой форме, при этом команды могут быть пронумерованы или записаны в виде сплошного текста. Если же разрабатываемый вами алгоритм предназначен для исполнения на компьютере, то нужно использовать формальный язык для его записи (язык программирования) и в качестве формы записи алгоритма применять программу. Псевдокод можно, например, использовать как промежуточное звено при переходе к представлению алгоритма в виде программы.

1.4. Базовые алгоритмические структуры

Вне зависимости от выбранной формы записи элементарные шаги алгоритма (команды) при укрупнении объединяются в алгоритмические конструкции: последовательные, ветвящиеся, циклические, рекурсивные. В 1969 году Эдсгер В. Дейкстра в статье «Структуры данных и алгоритмы» доказал, что для записи любого алгоритма достаточно трех основных алгоритмических конструкций: *последовательных, ветвящихся, циклических.*

Иными словами, алгоритмы можно представлять как некоторые структуры, состоящие из отдельных *базовых элементов*, а поэтому изучение основных принципов конструирования алгоритмов необходимо начинать с раскрытия сущности этих базовых элементов.

Логическая структура любого алгоритма может быть представлена комбинацией трех базовых структур: *следование, ветвление, цикл.*

Характерной особенностью базовых структур является наличие в них одного входа и одного выхода.

1. Базовая структура *следование* предписывает выполнение последовательности действий, следующих одно за другим, без пропусков и повторений (таблица 2).

Таблица 2

Базовая структура ***следование***

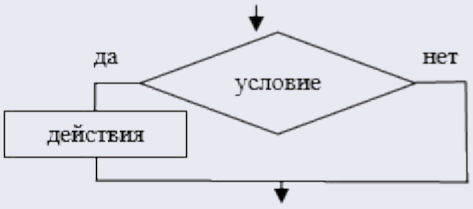
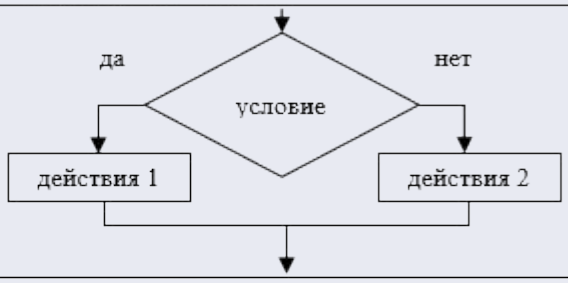
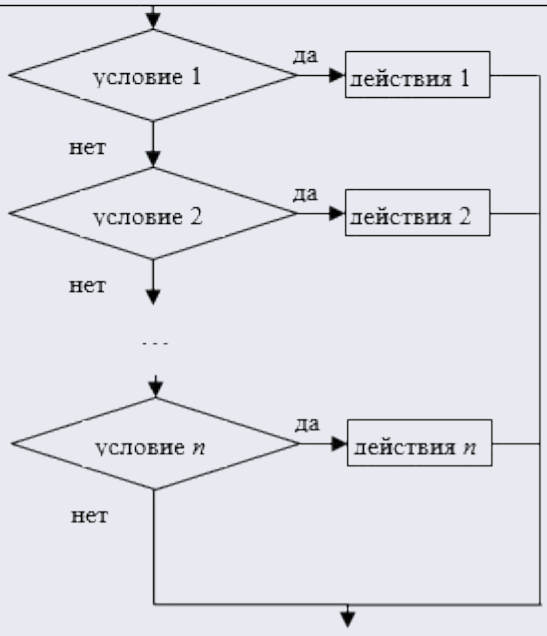
Псевдокод	*Блок-схема*
действие 1 действие 2 ..... действие *n*	

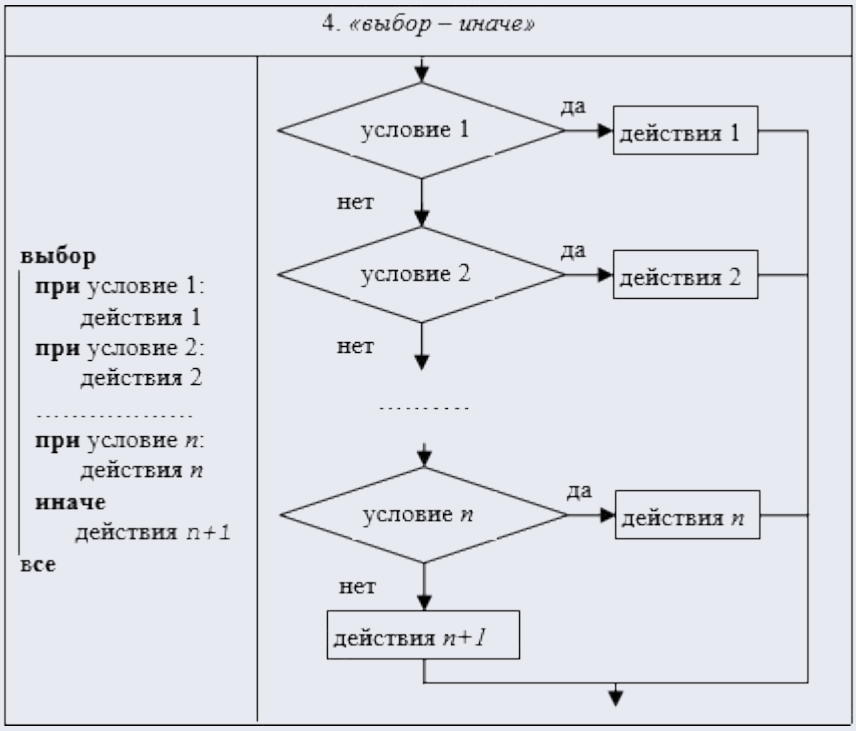
2. Базовая структура *ветвление* обеспечивает в зависимости от результата проверки условия (да или нет) выбор одного из альтернативных путей работы алгоритма. Каждый из путей ведет к общему выходу, так что работа алгоритма будет продолжаться независимо от того, какой путь будет выбран.

Структура ветвления существует в четырех основных вариантах (таблица 3): «если то», «если то иначе», «выбор», «выбор иначе».

Таблица 3

Базовая структура ***ветвление***

<table>
<tr><th>Псевдокод</th><th>Блок-схема</th></tr>
<tr><td colspan="2">1. «если – то»</td></tr>
<tr><td>если условие
| то действия
все</td><td> 
</td></tr>
<tr><td colspan="2">2. «если – то – иначе»</td></tr>
<tr><td>если условие
| то действия 1
| иначе действия 2
все</td><td> 
</td></tr>
<tr><td colspan="2">3. «выбор»</td></tr>
<tr><td>выбор
| при условие 1:
| действия 1
| при условие 2:
| действия 2
| .....
| при условие n:
| действия n
все</td><td> 
</td></tr>
</table>



3. Базовая структура *цикл* обеспечивает многократное выполнение некоторой совокупности действий, которая называется *телом цикла.*

Существует три основные разновидности циклов (таблица 4):

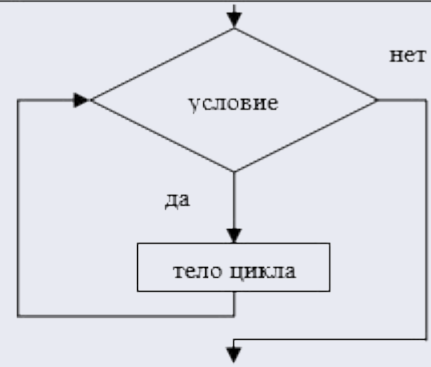
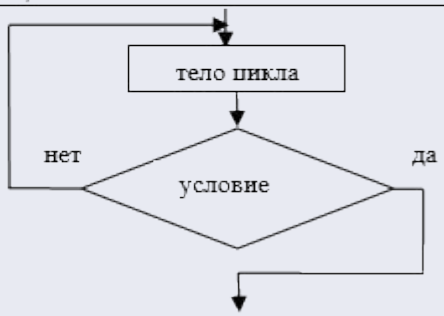
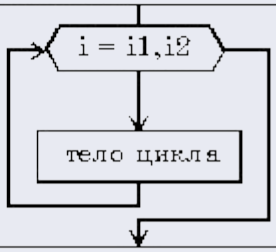
– цикл типа *«пока»* (или цикл с предусловием) предписывает выполнять тело цикла до тех пор, пока выполняется условие, записанное после слова «пока». Чтобы цикл не повторялся бесконечно необходимо, чтобы в теле цикла осуществлялись действия, приводящие к ситуации, когда условие перестанет быть истинным;

– цикл типа *«до»* (или цикл с постусловием) предписывает выполнять тело цикла до тех пор, пока условие не станет истинным. Чтобы не происходило зацикливания необходимо, чтобы в теле цикла осуществлялись действия, приводящие к ситуации, когда условие станет истинным;

– цикл типа *«для»* (или цикл с параметром) предписывает выполнять тело цикла для всех значений некоторой переменной (параметра цикла) в заданном диапазоне с заданным шагом.

Таблица 4

Базовая структура ***цикл***

Псевдокод	*Блок-схема*
1. Цикл типа «пока»	
нц пока условие | тело цикла **кц**	
2. Цикл типа «до»	
нц | тело цикла | **пока не** условие **кц**	
3. Цикл типа «для»	
нц для i от i1 до i2 | тело цикла **кц**	

Описанные нами базовые алгоритмические структуры при составлении различных алгоритмов могут использоваться и в различных комбинациях. Например, в случае, когда внутри цикла необходимо повторить некоторую последовательность действий, т. е. организовать внутренний цикл, используют структуру «цикл в цикле».

Вопросы к главе 1

1. Раскройте сущность понятия «алгоритм».
2. С именем какого ученого связано происхождение термина «алгоритм»?
3. Какие алгоритмы были разработаны самыми первыми?
4. В каких науках, по вашему мнению, алгоритмы играют важную роль? Ответ обоснуйте.
5. Приведите примеры алгоритмов, которые вы узнали при обучении в школе, в вузе.
6. Приведите примеры алгоритмов, которыми вы пользуетесь в повседневной жизни.
7. Как вы думаете, существуют ли задачи, которые человек, вообще говоря, умеет решать, не зная при этом алгоритм их решения? Ответ обоснуйте.
8. Что означают слова «алгоритмически разрешимая проблема», «алгоритмически неразрешимая проблема»?
9. Какой вопрос возник перед учеными при их попытке доказать алгоритмическую неразрешимость некоторой проблемы?
10. В какой период времени возникла наука «теория алгоритмов»?
11. С именами каких ученых связано развитие теории алгоритмов?
12. Назовите и охарактеризуйте основные этапы развития теории алгоритмов.
13. Раскройте основные задачи и направления развития современной теории алгоритмов.
14. Почему принято говорить об интуитивном представлении понятия алгоритма?
15. Раскройте сущность понятия «исполнитель алгоритма».
16. Приведите примеры различных исполнителей алгоритма.
17. Что характеризует исполнителя алгоритма? Раскройте суть этих характеристик.
18. Назовите основные свойства алгоритма и раскройте их содержание.
19. Приведите пример алгоритма и покажите, что он удовлетворяет всем основным свойствам алгоритмов.
20. Почему кулинарный рецепт приготовления торта нельзя считать алгоритмом? Какими свойствами алгоритма он не обладает?
21. Какие способы записи алгоритма существуют? В чем их суть?
22. Приведите примеры различных способов записи алгоритмов.
23. Существует ли, по вашему мнению, наилучший способ записи алгоритма? Ответ обоснуйте.
24. Какие базовые алгоритмические структуры существуют? Запишите их на языке блок-схем.

Задания к главе 1

1. Составьте алгоритм сложения в столбик двух натуральных чисел. Предполагается, что операция сравнения двух натуральных чисел для человека является выполнимой.

2. Переформулируйте способ проведения перпендикуляра к прямой в заданной точке (*см. 1.3.3.*) так, чтобы он стал алгоритмом.

3. Является ли следующее правило нахождения наибольшего общего делителя двух натуральных чисел алгоритмом?

$$НОД(a,b) = ?,\ где\ a,b \in N$$

$$a = bq_1 + r_1,\ где\ 0 < r_1 < b,$$
$$b = r_1q_2 + r_2,\ где\ 0 < r_2 < r_1,$$
$$r_1 = r_2q_3 + r_3,\ где\ 0 < r_3 < r_2,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$r_i = r_{i+1}q_{i+2} + r_{i+2},\ где\ 0 < r_{i+2} < r_{i+1},$$
$$r_{i+1} = r_{i+2}q_{i+3}.$$

$$НОД(a,b) = r_{i+2}.$$

$$НОД(116,24) = ?$$

$$116 = 24 \cdot 4 + 20,$$
$$24 = 20 \cdot 1 + 4,$$
$$20 = 4 \cdot 5.$$

$$НОД(116,24) = 4.$$

Действительно:

$$116 = 2 \cdot 2 \cdot 29,$$
$$24 = 2 \cdot 2 \cdot 2 \cdot 3.$$

4. Рассмотрите правило деления десятичных дробей. Выясните, при всех ли значениях десятичных дробей данное правило приведет к точному результату (оборвется на n-м шаге)? Можно ли это правило назвать алгоритмом?

5. Является ли алгоритмом следующая последовательность шагов?

1. исходное число умножим на 2;
2. к полученному числу прибавим 1;
3. определим остаток «*y*» от деления полученной в п. 2 суммы на 3;
4. разделим исходное число на *y*; частное будет являться искомым результатом.

Здесь исходные данные все натуральные числа.

6. Есть двое песочных часов: на 3 минуты и на 8 минут. Для приготовления эликсира бессмертия его надо варить ровно 7 минут. Как это сделать? Придумайте систему команд исполнителя «Колдун» и запишите последовательность команд этого исполнителя для приготовления эликсира.

7. Составьте в виде блок-схемы алгоритм нахождения факториала числа *n*.

8. Рассмотрим способ выписывания всех простых чисел в интервале от 1 до некоторого *N*. Этот способ носит название *«Решето Эратосфена»*, по имени древнегреческого ученого, впервые предложившего данный алгоритм.

Для этого выпишем подряд все натуральные числа от 1 до *N*. Возьмем первое число, большее 1 (это

будет 2), и зачеркнем каждое второе число, начиная отсчет со следующего за двойкой числа. Затем возьмем первое незачеркнутое число, большее 2 (это будет 3), и зачеркнем каждое третье число, начиная отсчет с числа 3+1 (ранее зачеркнутые числа участвуют в отсчете). Далее возьмем первое незачеркнутое число, большее 3 (это будет 5), и зачеркнем каждое пятое число, начиная отсчет с числа 5 +1. Продолжая так действовать, остановимся тогда, когда первое незачеркнутое число окажется больше $\sqrt{N}$.

В результате применения этого алгоритма незачеркнутыми останутся все простые числа, не превосходящие *N*, и только они. Докажите это.

9. Постройте блок-схемы алгоритмов вычисления по известным x и n следующих выражений с использованием только арифметических действий +, , *, /. Входные параметры: x вещественное, n целое неотрицательное число. В построенных блок-схемах вложенных циклов быть не должно.

1. $1 - x + x^2 - x^3 + x^4 - \ldots + (-1)^n x^n$;
2. $1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^n}{n!}$;
3. $1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \ldots + (-1)^{\frac{n}{2}} \frac{x^n}{n!}$ (n четное натуральное число).

10. Дана последовательность целых чисел $a_1, a_2, \ldots, a_n$. Укажите все ошибки в предлагаемой записи алгоритма нахождения максимального и минимального элемента в заданной последовательности. Идея алгоритма такова: разбиваем последовательность на пары; числа в каждой паре упорядочиваем по возрастанию; максимальный элемент ищем среди четных (по месту расположения) элементов последовательности, а минимальный среди нечетных.

1. Разобьем исходную последовательность на пары. Последняя пара может быть неполной, если число n нечетно.
2. Положим $m = \frac{n}{2}$, если n четно, и $m = \frac{n+1}{2}$ в противном случае (m число пар, которые надо обработать).
3. Упорядочим числа в каждой паре по возрастанию. Будем элементы в паре обозначать $P_{k,1}$ и $P_{k,2}$, где k номер пары.
4. Положим $\min = P_{1,1}$ и $\max = P_{1,2}$.
5. Положим $k = 1$.
6. Если $P_{k,2} > \max$, тогда $\max = P_{k,2}$.
7. Если $P_{k,1} < \min$, тогда $\min = P_{k,1}$.
8. Увеличим k на единицу.
9. Если $k = m$, конец алгоритма; иначе переход на п. 6.

11. У исполнителя КВАДР две команды, которым присвоены номера:

1) прибавь 1,

2) возведи в квадрат.

Первая из этих команд увеличивает число на экране на 1, вторая – возводит в квадрат. Программа для исполнителя КВАДР – это последовательность номеров команд. Запишите программу для исполнителя КВАДР, которая преобразует число 5 в число 2500 и содержит не более 6 команд. Если таких программ более одной, то запишите любую из них.

12. У исполнителя ДВАПЯТЬ две команды, которым присвоены номера:

1) отними 2,

2) раздели на 5.

Выполняя первую из них, ДВАПЯТЬ отнимает от числа на экране 2, а выполняя вторую, делит это число на 5 (если деление нацело невозможно, ДВАПЯТЬ отключается).

Запишите порядок команд в программе, которая содержит не более 5 команд и переводит число 152 в число 2.

13. У исполнителя КАЛЬКУЛЯТОР две команды, которым присвоены номера:

1) отними 2,

2) раздели на 3.

Выполняя первую из них, КАЛЬКУЛЯТОР отнимает от числа на экране 2, а выполняя вторую, делит его на 3 (если деление нацело невозможно, КАЛЬКУЛЯТОР отключается).

Запишите порядок команд в программе получения из числа 37 числа 3, содержащей не более 5 команд, указывая лишь номера команд.

14. Исполнитель КУЗНЕЧИК живёт на числовой оси. Начальное положение КУЗНЕЧИКА – точка 0. Система команд Кузнечика:

Вперед 7: КУЗНЕЧИК прыгает вперёд на 7 единиц,

Назад 5: КУЗНЕЧИК прыгает назад на 5 единиц.

Какое наименьшее количество раз должна встретиться в программе команда «Назад 5», чтобы КУЗНЕЧИК оказался в точке 19?

15. Исполнитель РОБОТ ходит по клеткам бесконечной вертикальной клетчатой доски, переходя по одной из команд вверх, вниз, вправо, влево в соседнюю клетку в указанном направлении. РОБОТ выполнил следующую программу:

вправо
вниз
вправо
вверх
влево
вверх
вверх
влево

Укажите наименьшее возможное число команд, которое необходимо для того, чтобы РОБОТ вернулся в

ту же клетку, из которой начал движение.

16. На экране есть два окна, в каждом из которых записано по числу. Исполнитель СУММАТОР имеет только две команды, которым присвоены номера:

1. Запиши сумму чисел в первое окно.

2. Запиши сумму чисел во второе окно.

Выполняя команду номер 1, СУММАТОР складывает числа в двух окнах и записывает результат в первое окно, а выполняя команду номер 2, заменяет этой суммой число во втором окне. Напишите программу, содержащую не более 5 команд, которая из пары чисел 1 и 2 получает пару чисел 13 и 4. Укажите лишь номера команд.

17. Система команд исполнителя РОБОТ, «живущего» в прямоугольном лабиринте на клетчатой плоскости:

- *вверх,*
- *вниз,*
- *влево,*
- *вправо.*

При выполнении любой из этих команд РОБОТ перемещается на одну клетку соответственно (по отношению к наблюдателю): вверх ↑, вниз ↓, влево ←, вправо →.

Четыре команды проверяют истинность условия отсутствия стены у каждой стороны той клетки, где находится РОБОТ (также по отношению к наблюдателю):

- *сверху свободно*
- *снизу свободно*
- *слева свободно*
- *справа свободно*

Цикл

ПОКА < условие >
последовательность команд
КОНЕЦ ПОКА

выполняется, пока условие истинно.

В конструкции

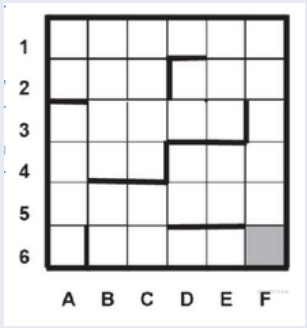
ЕСЛИ < условие >
ТО команда1
ИНАЧЕ команда2
КОНЕЦ ЕСЛИ

выполняется команда1 (если условие истинно) или команда2 (если условие ложно)

Если РОБОТ начнёт движение в сторону находящейся рядом с ним стены, то он разрушится и программа прервётся.

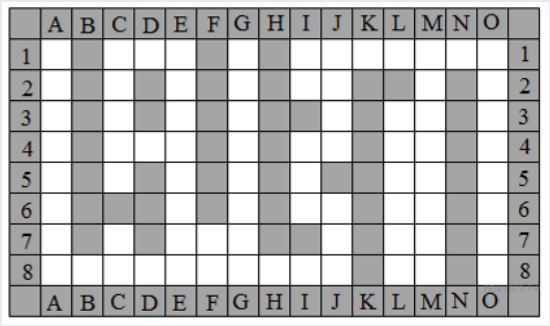
Сколько клеток лабиринта соответствуют требованию, что, начав движение в ней и выполнив предложенную программу, РОБОТ уцелеет и остановится в закрашенной клетке (клетка F6)?

НАЧАЛО
ПОКА *снизу свободно*
ИЛИ *справа свободно*
ПОКА *снизу свободно*
вниз
КОНЕЦ ПОКА
вправо
КОНЕЦ ПОКА
КОНЕЦ



1) 7 2) 12 3) 17 4) 21

18. Исполнитель МАШИНКА «живет» в ограниченном прямоугольном лабиринте на клетчатой плоскости, изображенном на рисунке. Серые клетки – возведенные стены, светлые – свободные клетки, по которым МАШИНКА может свободно передвигаться. По краю поля лабиринта также стоит возведенная стенка с нанесенными номерами и буквами для идентификации клеток в лабиринте.



Система команд исполнителя МАШИНКА:

- *вверх,*
- *вниз,*
- *влево,*
- *вправо.*

При выполнении любой из этих команд МАШИНКА перемещается на одну клетку соответственно (по отношению к наблюдателю): вверх ↑, вниз ↓, влево ←, вправо →.

Четыре команды проверяют истинность условия отсутствия стены у каждой стороны той клетки, где находится МАШИНКА (также по отношению к наблюдателю):

- *сверху свободно,*
- *снизу свободно,*

- *слева свободно,*
- *справа свободно.*

Цикл

ПОКА <условие> команда

выполняется, пока условие истинно, иначе происходит переход на следующую строку.

При попытке передвижения на любую серую клетку МАШИНКА разбивается о стенку.

Сколько клеток приведенного лабиринта соответствуют требованию, что, стартовав в ней и выполнив предложенную ниже программу, МАШИНКА не разобьется?

НАЧАЛО
ПОКА <снизу свободно> вниз
ПОКА <справа свободно> вправо
вверх
вправо
КОНЕЦ

1) 0 2) 7 3) 1 4) 3

19. Автомат получает на вход трёхзначное число. По этому числу строится новое число по следующим правилам.

1. Складываются первая и вторая, а также вторая и третья цифры исходного числа.

2. Полученные два числа записываются друг за другом в порядке убывания (без разделителей).

Пример: исходное число 348; суммы: 3+4=7 и 4+8=12; результат: 127.

Сколько существует чисел, в результате обработки которых автомат выдаст число 1715?

20. Алгоритм вычисления значения функции F(n), где n – натуральное число, задан следующими соотношениями:

F(1)=1; F(2)=1;

F(n)=F(n- 2)(n-1), при n>2.*

Чему равно значение функции F(8)? В ответе запишите только натуральное число.

21. Алгоритм вычисления значения функции F(n) и G(n), где n–натуральное число, задан следующими соотношениями:

F(1)=0

F(n)=F(n–1)+n, при n>1

G(1)=1

*G(n)=G(n–1)*n, при n>1*

Чему равно значение функции F(5) + G(5)?

22. Последовательность чисел Фибоначчи задается рекуррентным соотношением:

F(1)=1

F(2)=1

F(n)= F(n–2) + F(n–1), при n>2, где n – натуральное число.

Чему равно восьмое число в последовательности Фибоначчи?

Дополнительные задания

Составьте точный план действий, приводящий к решению следующих задач. Можно ли каждый конкретный план назвать алгоритмом? Ответ обоснуйте.

1. Волк, коза и капуста. На берегу реки стоит крестьянин с лодкой, а рядом с ним волк, коза и капуста. Крестьянин должен переправиться сам и перевезти волка, козу и капусту на другой берег. Однако в лодку, кроме крестьянина, помещается либо только волк, либо только коза, либо только капуста. Оставлять же волка с козой или козу с капустой без присмотра нельзя волк может съесть козу, а коза капусту. Как должен вести себя крестьянин?

2. Два встречных поезда, в каждом из которых паровоз и 21 вагон, встретились на дороге с одним тупиком (рис. 2). Тупик вмещает 11 вагонов или 10 вагонов и паровоз. Как поездам разъехаться (т. е. как должны маневрировать машинисты, чтобы каждый поезд продолжил движение в своем направлении)?

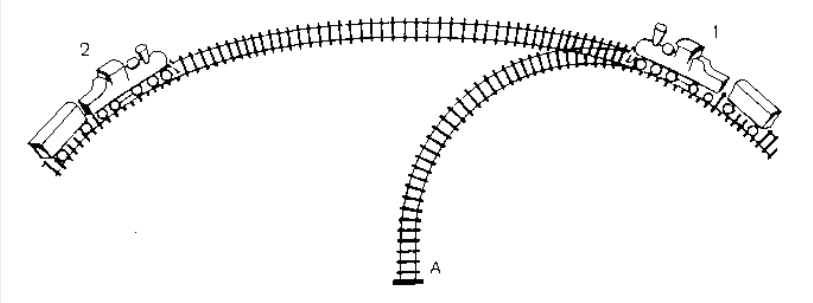


Рис. 2

3. Ханойская башня. На подставке укреплены три стержня, на левый стержень нанизано несколько колец уменьшающегося размера внизу самое большое кольцо, на нем поменьше, сверху еще меньше и т. п. (рис. 3).

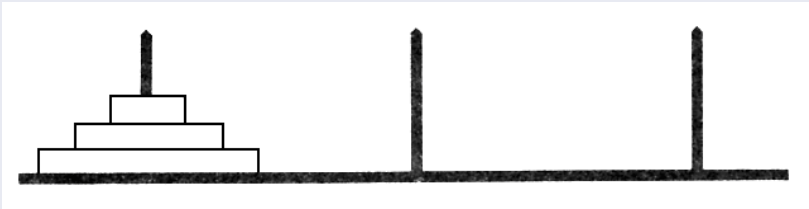


Рис. 3

Надо, перемещая по одному кольцу со стержня на стержень, переместить все кольца на правый

стержень, но при этом ни в какой момент времени большее кольцо на меньшее класть нельзя. Опишите, как надо перекладывать кольца, если в начальный момент на левом стержне: а) 1; б) 2; в) 3; г) 4; д*) 64 кольца. (По преданию перекладыванием 64 колец занимаются монахи в одном из буддийских монастырей. Согласно легенде, в момент, когда они кончат перекладывать кольца, наступит конец света. Прикиньте приблизительно, когда это произойдет, если считать, что монахи перекладывают примерно 1 кольцо в секунду.)

4. Фирма «Электронные приборы» выпустила автоматизированную ванну «Банный комплекс 10», управляемую с помощью 10 кнопок «долить 1 л», «долить 2 л», …, «долить 5 л»; «слить 1 л», «слить 2 л», …, «слить 5 л», при нажатии на которые доливается или сливается указанное количество литров воды. Однако в результате ошибки фирмы все кнопки, кроме «долить 5 л» и «слить 3 л», не работают. Как долить в ванну 3 л воды? Сколько воды при этом пропадет впустую из-за брака фирмы?

5. Два солдата подошли к реке, по которой на лодке катаются двое мальчиков. Как солдатам переправиться на другой берег, если лодка вмещает только одного солдата либо двух мальчиков, а солдата и мальчика уже не вмещает?

6. Петя и Коля играют в следующую игру: на стол кладется 15 спичек. Ребята по очереди берут их со стола, причем за один ход разрешается взять 1, 2 или 3 спички. Выигрывает тот, кто возьмет последнюю спичку. Первым ходит Петя. Как он должен играть, чтобы выиграть?

7. Автоматическое устройство имеет 2 кнопки и экран. При включении на экране загорается число 0. При нажатии на одну кнопку число на экране удваивается (вместо x появляется $2x$). При нажатии на другую кнопку число увеличивается на 1 (вместо x появляется $x+1$). Как надо нажимать на кнопки, чтобы на экране появилось:

1) число 5;

2) число 99;

3) число 99, если разрешается нажимать на кнопки не более 10 раз?

8. Придумайте способ нахождения самой легкой и самой тяжелой из 100 монет различной массы, если можно сделать не более 150 взвешиваний на чашечных весах без гирь.

9. Имеется а) 3, б) 4, в) 5, г) 6 монет, среди которых одна фальшивая (легче других). Придумайте способ нахождения фальшивой монеты за минимальное число взвешиваний на чашечных весах без гирь.

10. Имеется 1000 монет, из которых одна фальшивая (легче других). Придумайте способ нахождения фальшивой монеты за 7 взвешиваний на чашечных весах без гирь. Докажите, что нельзя придумать способ, который гарантирует нахождение фальшивой монеты за 6 взвешиваний.

Глава 2. Класс рекурсивных функций

В первой главе мы говорили о том, что центральным вопросом в теории алгоритмов явился вопрос формализации интуитивного понятия «алгоритм». Попытки различных ученых дать точный математический эквивалент для общего интуитивного представления об алгоритме привели к тому, что были предложены несколько моделей алгоритма (машина Тьюринга, машина Поста, рекурсивные функции, нормальные алгоритмы Маркова и др.). В данной главе мы рассмотрим такую модель алгоритма, как частично-рекурсивные функции.

2.1. Введение в теорию рекурсивных функций

Всякий алгоритм однозначно сопоставляет допустимым начальным данным результат. Это означает, что с каждым алгоритмом однозначно связана функция, которую он вычисляет. Кроме того, возникают естественные вопросы: «Для всякой ли функции существует вычисляющий ее алгоритм?», «Если нет, то для каких функций существует вычисляющий их алгоритм, как описать такие, как говорят, алгоритмически или эффективно вычислимые функции?».

Исследование данных вопросов привело ученых (Геделя, Клини, Черча) к созданию в 1930-х гг. теории рекурсивных функций. При этом класс вычислимых функций (названных здесь рекурсивными) получил такое описание, которое весьма напоминает процесс построения аксиоматической теории на базе некоторой системы аксиом. Сначала были выбраны простейшие функции, вычислимость которых очевидна (своего рода аксиомы). Затем были сформулированы некоторые правила, названные операторами, на основе которых можно строить новые функции из уже имеющихся (своего рода «правила вывода»). Тогда требуемым классом функций будет совокупность всех функций, получающихся из простейших с помощью выбранных операторов.

Прежде чем приступить к описанию класса рекурсивных функций, приведем ряд вспомогательных определений.

Определение 1

Пусть A, B некоторые множества. Совокупность всех упорядоченных пар вида (a,b), где $a \in A$, $b \in B$ называется *декартовым (прямым) произведением* A на B и обозначается $A \times B$.

Определение 2

Пусть X, Y некоторые множества. Если *некоторым* элементам множества X поставлены в соответствие однозначно определенные элементы множества Y, то говорят, что задана *частичная функция* из X в Y.

Совокупность тех элементов множества X, у которых есть соответствующие в Y, называется *областью определения* функции.

Совокупность тех элементов множества Y, которые соответствуют некоторым элементам множества X, называется *областью значений* функции.

Если область определения функции из X в Y совпадает с множеством X, то функция называется *всюду определенной.*

Определение 3

Частичные функции из $\underbrace{X \times X \times \ldots \times X}_{n} = X^{(n)}$ в Y называются частичными функциями от n переменных или n *-местными функциями* из X в Y.

Для записи функций и изучения их свойств пользуются особым формальным языком. Алфавит этого языка состоит из символов, разбитых на три группы:

1. предметные символы это буквы $a, b, x, y, \ldots$ или буквы с нижними индексами $a_0, a_1, x_1, y_0, \ldots$;
2. функциональные символы это буквы с верхними и, возможно, нижними индексами:

$f^{(1)}, g^{(2)}, f_0^{(1)}, \dots$;

3. символы третьей группы это символы левой, правой скобок и запятой: «(», «)», «,».

Конечные последовательности символов, записанные в этом функциональном алфавите, (слова) называются *термами*.

Пример

x терм длины 1;

$f(a)$ терм длины 4;

$g(x,a)$ терм длины 6.

2.2. Примитивно рекурсивные функции

Подчеркнем, что здесь и в дальнейшем (во всей главе 2) будут рассматриваться функции, заданные на множестве натуральных чисел и принимающие натуральные значения. Функции предполагается брать частичные, т. е. определенные, вообще говоря, не для всех значений аргументов.

Определение 1

Функции:

1) $O(x) = 0$ (нуль-функция),

2) $S(x) = x + 1$ (функция следования),

3) $I_m^n(x_1, x_2, ..., x_n) = x_m$ (функция проекции, $1 \le m \le n$)

называются *простейшими (базисными) функциями*.

Определение 2

Пусть 1) f – m-местная функция на множестве натуральных чисел N ; 2) $g_1, g_2, ..., g_m$ – n-местные функции на множестве N (считаем, что все функции $g_1, g_2, ..., g_m$ зависят от одних и тех же переменных $x_1, x_2, \dots x_n$).

Оператор G, ставящий в соответствие функциям $f, g_1, g_2, ..., g_m$ n-местную функцию h, удовлетворяющую тождеству $h(x_1, ..., x_n) = f(g_1(x_1, ..., x_n), g_2(x_1, ..., x_n), ..., g_m(x_1, ..., x_n))$, называется *оператором суперпозиции (подстановки)*.

Причем функция $h \overset{\text{обозначение}}{=} G(f, g_1, g_2, ..., g_m)$ является, очевидно, суперпозицией функций f и $g_1, g_2, ..., g_m$.

Замечания

1. Если среди функций f , $g_1, g_2, ..., g_m$ из определения 2 имеются частичные функции, то и функция h будет частичной.

2. Функция h на наборе переменных $x_1, x_2, \dots x_n$ определена тогда и только тогда, когда определены все функции $g_1(x_1, ..., x_n), g_2(x_1, ..., x_n), ..., g_m(x_1, ..., x_n)$ и функция f определена на наборе $g_1(x_1, ..., x_n), g_2(x_1, ..., x_n), ..., g_m(x_1, ..., x_n)$.

Пример 1

Найдем значение термов $G(I_1^2, I_3^4, I_2^4)$ и $G(I_1^2, I_1^3, I_2^2)$.

Решение

1) $G(I_1^2, I_3^4, I_2^4) = I_1^2(I_3^4, I_2^4) =$ <перейдем к полной форме записи функций I_3^4 и I_2^4> $= I_1^2(I_3^4(x_1, x_2, x_3, x_4), I_2^4(x_1, x_2, x_3, x_4)) = \ = I_1^2(x_3, x_2) = x_3$.

2) $G(I_1^2, I_1^3, I_2^2) = I_1^2(I_1^3, I_2^2) = I_1^2(I_1^3(x_1, x_2, x_3), I_2^2(x_1, x_2))$ – не определено, так как функции I_1^3 и I_2^2 имеют различную местность, а не одинаковую.

Определение 3

Оператор примитивной рекурсии R каждой $(n+2)$-местной функции $f(x_1, \ldots, x_n, y, z)$ и n-местной функции $g(x_1, \ldots, x_n)$ на множестве N ставит в соответствие $(n+1)$-местную функцию $h \overset{\text{обозначение}}{=} R(f, g)$, удовлетворяющую следующей *схеме примитивной рекурсии*:

$$\begin{cases} h(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n), \\ h(x_1, \ldots, x_n, y+1) = f(x_1, \ldots, x_n, y, h(x_1, \ldots, x_n, y)). \end{cases}$$

Замечания

1. Важно отметить, что независимо от числа аргументов в h, рекурсия ведется только по одной переменной y; остальные n переменных $x_1, \ldots, x_n$ на момент применения схемы примитивной рекурсии зафиксированы и играют роль параметров.

2. Очевидно, что схема примитивной рекурсии однозначно определяет функцию h. Причем, схема примитивной рекурсии выражает каждое значение функции h не только через данные функции f и g, но и через так называемые предыдущие значения определяемой функции h: прежде чем получить значение $h(x_1, \ldots x_n, k)$, придется проделать $k+1$ вычисление по указанной схеме для $y = 0, 1, 2, \ldots, k$.

3. Про функцию h говорят, что она получена рекурсией из функций f и g. (Напомним, что *рекурсией* называется способ задания функции, при котором значение функции при определенных значениях аргументов выражается через уже заданные значения функции при других значениях аргументов.)

4. Если функции g и f частичные, то $h(x_1, \ldots x_n, y+1)$ считается определенной в том и только том случае, когда определены $h(x_1, \ldots x_n, y)$ и $f(x_1, \ldots x_n, y, t)$ при $t = h(x_1, \ldots x_n, y)$. Иными словами, если $h(x_1, \ldots x_n, y_0)$ неопределенно, то и $h(x_1, \ldots x_n, y)$ неопределенно при $y > y_0$.

5. Оператор примитивной рекурсии в соответствии с определением 3 мы будем применять и при $n = 0$. В этом случае схема примитивной рекурсии будет иметь следующий вид:

$$\begin{cases} h(0) = const, \\ h(y+1) = f(y, h(y)), \end{cases}$$

где g – постоянная одноместная функция, равная числу $const$.

Пример 2

Покажем, что функция $s(x,y)=x+y$ может быть получена из простейших с помощью оператора примитивной рекурсии.

Решение

Переобозначим: $s^2(x,y)=h^2(x,y)=x+y$.

Будем искать: $f^3(x,y,z)$ и $g^1(x)$. Из схемы примитивной рекурсии имеем:

$$\begin{cases} h^2(x,0) \overset{опр}{=} x+0=x=I_1^1(x), \\ h^2(x,y+1) \overset{опр}{=} x+(y+1)=\underline{x+y+1} \overset{\text{схема пр. рек.}}{=} f^3(x,y,h(x,y)) \overset{опр}{=} \end{cases}$$

$$\overset{опр}{=} f^3(x,y,\underbrace{x+y}_{z}) .$$

Следовательно, $g^1(x)=I_1^1(x)$, а $f^3(x,y,z)=z+1=S(z)$.

Определение 4

Функция называется *примитивно рекурсивной*, если она может быть получена из простейших функций O,S,I_m^n с помощью конечного числа применений операторов суперпозиции и примитивной рекурсии.

Пример 3

Функция сложения $s(x,y)=x+y$ является примитивно рекурсивной.

2.3. Частично рекурсивные функции

Определение 1

Будем говорить, что n-местная функция $\varphi(x_1, x_2, \dots x_n)$ получается из $(n+1)$-местных функций $f_1(x_1, x_2, \dots x_n, y)$ и $f_2(x_1, x_2, \dots x_n, y)$ с помощью *оператора минимизации* M (или *оператора наименьшего числа*), если:

для любых $x_1, x_2, ..., x_n, y$ равенство $\varphi(x_1, x_2, ..., x_n) = y$ выполнено тогда и только тогда, когда значения функций $f_i(x_1, x_2, ..., x_n, 0), ..., f_i(x_1, x_2, ..., x_n, y-1)$ $(i = 1,2)$ определены и попарно неравны [т. е. $f_1(x_1, x_2, ..., x_n, 0) \neq f_2(x_1, x_2, ..., x_n, 0), ...,$ $f_1(x_1, x_2, ..., x_n, y-1) \neq f_2(x_1, x_2, ..., x_n, y-1)$], а значение $f_1(x_1, x_2, ..., x_n, y) = f_2(x_1, x_2, ..., x_n, y)$.

Замечание 1

1. Функции φ, f_1, f_2 заданы на множестве натуральных чисел и принимают натуральные значения.

2. Величина $\varphi(x_1, x_2, ..., x_n)$ в определении 1 равна наименьшему значению аргумента y, при котором выполняется равенство $f_1(x_1, x_2, ..., x_n, y) = f_2(x_1, x_2, ..., x_n, y)$.

3. Используют следующее обозначение:

$$\varphi(x_1, x_2, ..., x_n) \overset{обозн.}{=} \mu_y[f_1(x_1, x_2, ..., x_n, y) = f_2(x_1, x_2, ..., x_n, y)].$$

4. В частном случае может быть $f_2(x_1, x_2, ..., x_n, y) = 0$. Тогда:

$$\varphi(x_1, x_2, ..., x_n) = \mu_y[f_1(x_1, x_2, ..., x_n, y) = 0].$$

5. Оператор минимизации называют также μ *-оператором.*

Пример 1

Рассмотрим функцию $d(x, y) = \mu_z[y + z = x]$. Вычислим $d(7,2)$ и $d(3,4)$.

Решение

1). Вычислим $d(7,2)$. Для этого нужно положить $y = 2$ и, придавая переменой z значения 1, 2, 3…, каждый раз вычислять сумму $y + z$. Как только она станет равной 7, то соответствующее значение z принять за $d(7,2)$.

$z = 1$: $2 + 1 = 3 \neq 7$;

$z = 2$: $2 + 2 = 4 \neq 7$;

$z = 3$: $2 + 3 = 5 \neq 7$;

$z = 4$: $2 + 4 = 6 \neq 7$;

$z = 5$: $2 + 5 = 7$.

Следовательно, $d(7,2) = 5$.

2). Аналогично вычислим $d(3,4)$.

$z = 1$: $4 + 1 = 5 > 3$;

$z = 2$: $4 + 2 = 6 > 3$;

……

Видим, что данный процесс будет продолжаться бесконечно. Следовательно, $d(3,4)$ не определено.

Заметим, что вычисления в 1) и 2) можно осуществить более рационально. Действительно: $d(x,y) = \mu_z[y + z = x] \overset{\text{замечание 1(2)}}{=} x - y$. Поэтому $d(7,2) = 7 - 2 = 5$, а $d(3,4) = 3 - 4 \notin N$, т. е. не определено.

Замечание 2

Подчеркнем, что операция минимизации может давать частичные функции даже при применении к всюду определенным функциям.

Действительно, в примере 1 функции $f_1(x,y,z) = y + z$ и $f_2(x,y,z) = x$ всюду определены (т. е. определены на всем множестве $N^{(3)}$), а функция $d(x,y) = \mu_z[y + z = x] = x - y$ определена только при $x > y$.

В отличие же от оператора минимизации операторы суперпозиции и примитивной рекурсии, примененные к всюду определенным функциям, дают также всюду определенные функции.

Замечание 3

Оператор минимизации является удобным средством для построения обратных функций к одноместным функциям.

Действительно, функция $f^{-1}(x) = \mu_y[f(y) = x]$ (наименьший y, такой, что $f(y) = x$) является обратной для функции $f(x)$. Поэтому в применении к одноместным функциям оператор минимизации иногда называют *оператором обращения*.

Пример 2

Найдем функцию, обратную к функции следования $S(x) = x + 1$.

Решение

Переобозначим: $S(x) = f(x) = x + 1$.

Тогда $f^{-1}(x) = \mu_y[y + 1 = x] = x - 1$.

Итак, $S^{-1}(x) = x - 1$, $x \in \{2,3,4,...\}$.

Определение 2

Функция называется *частично рекурсивной*, если она может быть получена из простейших функций O, S, I_m^n с помощью конечного числа применений операторов суперпозиции, примитивной рекурсии и μ -оператора.

Определение 3

Если функция частично рекурсивна и всюду определена, то она называется *общерекурсивной.*

Пример 3

Определим, является ли функция $d(x, y) = \mu_z[y + z = x]$ частично рекурсивной.

Решение

Заметим, что функция $d(x, y) = \mu_z[y + z = x]$ получена с помощью оператора минимизации из функций $f_1(x, y, z) = y + z$ и $f_2(x, y, z) = x$.

В свою очередь: 1) функция $f_1(x, y, z) = y + z$ является примитивно рекурсивной (поясните, почему), т. е. может быть получена с помощью конечного числа применений операторов суперпозиции и примитивной рекурсии; 2) функция $f_2(x, y, z) = x$ равна $I_1^1(x)$, т. е. может быть получена из простейших функций.

Таким образом, функция $d(x, y) = \mu_z[y + z = x]$ может быть получена из простейших функций с помощью конечного числа применений операторов суперпозиции, примитивной рекурсии и μ -оператора, поэтому данная функция является частично рекурсивной.

2.4. Взаимосвязь между различными классами рекурсивных функций

Обозначим: $Ч$ класс частично рекурсивных функций; $Ч_о$ класс общерекурсивных функций, $Ч_{пр}$ класс примитивно рекурсивных функций.

Рассмотрим вопрос о соотношении введенных классов $Ч$, $Ч_о$, $Ч_{пр}$.

Очевидны следующие соотношения:

1. Класс частично рекурсивных функций $Ч$ шире класса примитивно рекурсивных функций $Ч_{пр}$ (т. е. $Ч_{пр} \subset Ч$), поскольку для построения частично рекурсивных функций из простейших используется больше средств, чем для построения примитивно рекурсивных функций.

2. Класс примитивно рекурсивных функций $Ч_{пр}$ включается в класс общерекурсивных функций $Ч_о$ (т. е. $Ч_{пр} \subset Ч_о$), так как все примитивно рекурсивные функции всюду определены.

3. Класс общерекурсивных функций $Ч_о$ включается в класс частично рекурсивных функций $Ч$ (т. е. $Ч_о \subset Ч$), поскольку среди частично рекурсивных функций встречаются функции, как всюду определенные, так и не всюду определенные (например, $d(x,y) = \mu_z[y+z=x]$), и даже нигде не определенные (например, $f(x) = \mu_y[x+1+y=0]$).

Из положений 1–3 можно сделать вывод, что $Ч_{пр} \subset Ч_о \subset Ч$ (1).

Замечание

При выяснении соотношений между классами $Ч$, $Ч_о$, $Ч_{пр}$ ученые долгое время не могли обосновать справедливость следующего включения: $Ч_{пр} \subset Ч_о$ (1). Дело в том, что на тот момент очевидным являлось нестрогое включение ($Ч_{пр} \subseteq Ч_о$), а вот строгое включение ($Ч_{пр} \subset Ч_о$) оставалось спорным. Ученые не могли указать функцию, которая являлась бы общерекурсивной, но не являлась бы примитивно рекурсивной.

Первый пример общерекурсивной функции, не являющейся примитивно рекурсивной, был дан Аккерманом в 1928 г. Построенная им функция в теории алгоритмов получила название *функции Аккермана.* Раскроем ее суть.

Функция Аккермана $\varphi(x,y)$ задается соотношениями:

$$\begin{cases} \varphi(x,0) = y+1, & (1) \\ \varphi(x+1,0) = \varphi(x,1), & (2) \\ \varphi(x+1,y+1) = \varphi(x,\varphi(x+1,y)). & (3) \end{cases}$$

Можно доказать, что данные соотношения однозначно определяют функцию $\varphi(x,y)$.

Попробуем вычислить некоторые значения функции Аккермана в соответствии с ее определением:

$\varphi(0,0) \overset{1}{=} 1$;

$\varphi(0,1) \overset{1}{=} 2$;

$\varphi(1,0) \overset{2}{=} \varphi(0,1) = 2$;

$\varphi(1,1) \overset{3}{=} \varphi(0,\varphi(1,0)) = \varphi(0,2) \overset{1}{=} 3$;

$\varphi(2,0) \overset{2}{=} \varphi(1,1) = 3$;

$\varphi(1,2) \overset{3}{=} \varphi(0,\varphi(1,1)) = \varphi(0,3) \overset{1}{=} 4$;

$\varphi(3,0) \overset{2}{=} \varphi(2,1) \overset{3}{=} \varphi(1,\varphi(2,0)) = \varphi(1,3) \overset{3}{=} \varphi(0,\varphi(1,2)) = \varphi(0,4) \overset{1}{=} 5.$

Результаты приведенных вычислений убеждают, что найдется алгоритм вычисления значений функции $\varphi(x,y)$. При этом в процессе вычисления какого-либо значения функции $\varphi(x,y)$ в некоторой точке используются вычисленные ранее ее значения в неких предыдущих точках. Этим соотношения (1)–(3) похожи на схему примитивной рекурсии. Но примитивная рекурсия ведется по одному аргументу, а в соотношениях (1)–(3) рекурсия ведется сразу по двум аргументам. Причем существенно усложняется характер упорядочения точек, а, следовательно, и понятие предшествующей точки. Это упорядочение не предопределено заранее, как в схеме примитивной рекурсии, где число n всегда предшествует числу $n+1$, а выясняется в ходе вычислений.

Возникает вопрос, можно ли вычисление функции Аккермана свести к вычислению по схеме примитивной рекурсии, т. е. будет ли функция Аккермана примитивно рекурсивной. Оказывается, нет, в 1928 г. этот факт доказал Аккерман (идея доказательства того, что функция Аккермана не является примитивно рекурсивной, состоит в обосновании того, что функция Аккермана растет быстрее, чем любая примитивно рекурсивная функция, и поэтому не может быть примитивно рекурсивной).

2.5. Тезис Черча

Понятие частично рекурсивной функции оказалось исчерпывающей формализацией понятия вычислимой функции. При построении аксиоматической теории высказываний исходные формулы (аксиомы) и правила вывода выбирались так, чтобы полученные в теории формулы исчерпали бы все тавтологии алгебры высказываний. К чему же стремимся мы в теории рекурсивных функций, почему именно так выбрали простейшие функции и операторы для получения новых функций? Рекурсивными функциями мы стремимся исчерпать все мыслимые функции, поддающиеся вычислению с помощью какой-нибудь определенной процедуры механического характера. В теории рекурсивных функций выдвигается соответствующая естественнонаучная гипотеза, носящая название тезис Черча.

Тезис Черча

Числовая функция тогда и только тогда алгоритмически вычислима, когда она частично рекурсивна.

Эта гипотеза не может быть доказана строго математически, она подтверждается практикой, опытом, ибо призвана увязать практику и теорию. Все рассматривавшиеся в математике конкретные функции, признаваемые вычислимыми в интуитивном смысле, оказывались частично рекурсивными.

Вопросы к главе 2

1. Когда и кем была создана теория рекурсивных функций? Какова ее роль в теории алгоритмов?
2. Раскройте идею построения всех примитивно рекурсивных (частично рекурсивных) функций.
3. Какие функции называются простейшими? Какова область определения каждой простейшей функции?
4. Дайте определение оператора суперпозиции.
5. Приведите пример функции, полученной с помощью оператора суперпозиции.
6. Как вы считаете, какие функции (частичные или всюду определенные) могут быть получены из простейших с помощью оператора суперпозиции?
7. Дайте определение оператора примитивной рекурсии.
8. Приведите пример функции, полученной с помощью оператора примитивной рекурсии.
9. Как вы считаете, какие функции (частичные или всюду определенные) могут быть получены из простейших с помощью оператора примитивной рекурсии?
10. Какие функции называются примитивно рекурсивными. Приведите пример примитивно рекурсивной функции.
11. Дайте определение оператора минимизации.
12. Приведите пример функции, полученной с помощью оператора минимизации.
13. Как вы считаете, какие функции (частичные или всюду определенные) могут быть получены из простейших с помощью оператора минимизации?
14. Проиллюстрируйте, как с помощью оператора минимизации можно построить функцию, обратную к одноместной функции.
15. Какие функции называются частично рекурсивными? Приведите пример частично рекурсивной функции.
16. Какие функции называются общерекурсивными? Приведите пример общерекурсивной функции.
17. Как связаны между собой класс частично рекурсивных функций, класс общерекурсивных функций, класс примитивно рекурсивных функций?
18. Сформулируйте тезис Черча и раскройте его роль в теории алгоритмов.

Задания к главе 2

1. Докажите, что простейшие функции вычислимы.

2. Найдите значения следующих термов:

а) $G(I_1^2, I_1^3, I_2^3)$;

б) $G(I_1^2, I_1^3, I_2^2)$;

в) $G(S, G(S, I_1^1))$;

г) $G(I_1^1, O)$;

д) $G(S, G(S, G(S, G(S, I_2^2))))$;

е) $G(S, G(S, G(S, G(S, G(I_2^2, S, O)))))$.

3. Определите, можно ли получить функцию $f(x) = 1$ из простейших функций с помощью конечного числа применений оператора суперпозиции? А функции $f(x) = 2$, $f(x) = 3$ и т. д.?

4. Найдите результат применения операции суперпозиции $G(j, f_1, f_2)$, если $j(x, y) = x + y$ и:

а) $f_1(u, v, z) = u^2 vz$, $f_2(u, v, z) = 2^u vz$;

б) $f_1(x, y, z) = xyz$, $f_2(x, y) = xy^3$;

в) $f_1(u, v, z) = u^2 v 2^z$, $f_2(u, v, t) = uvt^2$

5. Найдите результат применения операции суперпозиции $G(\varphi, f_1, f_2, f_3)$, если $\varphi(x, y, z) = xy + z$ и:

а) $f_1(u, v, z) = uvz^2$, $f_2(u, v, z) = 2vz^u$, $f_3(u, v, z) = u + v + z$;

б) $f_1(x, y, z) = xy^3 z$, $f_2(x, y) = xy$, $f_3(x, z) = x + z^2$;

в) $f_1(u, v, z) = uv2^z$, $f_2(u, v, t) = u^2 + vt$, $f_3(v, z, t) = v^2 + 2z + 5^t$.

6. Покажите, что следующие функции могут быть получены из простейших с помощью оператора примитивной рекурсии:

а) $f(x, y) = xy$;

б) $f(x, y) = x^y$;

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

г) $\overline{sg}(x) = \begin{cases} 1, \text{если } x = 0, \\ 0, \text{если } x > 0. \end{cases}$

д) Поскольку рассматриваются лишь функции с целыми неотрицательными аргументами и значениями, то вводиться функция *«усеченного вычитания»*. Она определяется следующим образом:

$$x \dot{-} y \overset{\text{определение}}{=} \begin{cases} 0, \text{если } x < y, \\ x - y, \text{если } x \geq y. \end{cases}$$

В этом контексте покажите, что функции $f(x) = x \dot{-} 1$ и $f(x, y) = x \dot{-} y$ могут быть получены из простейших с помощью оператора примитивной рекурсии.

е) $f(x, y) = |x - y|$.

7. Найдите арифметическую функцию $f = R(h, g)$, полученную операцией примитивной рекурсии, если $g(x) = I_1^1(x)$, $h(x, y, z) = I_3^3(x, y, z) + 1$. Вычислите $f(2,5)$.

8. Найдите результат применения операции примитивной рекурсии $h = R(f, g)$, если $g(u) = 2u$, $f(u, y, z) = uyz$.

9. Найдите результат применения операции примитивной рекурсии $h = R(f, g)$, если $f(u, v, y, z) = 2^u vz$ и $g(u, v) = u^2 v$.

10. Какая функция получается из функций g и h с помощью схемы примитивной рекурсии, если:

а) $g(x) = x$, $h(x, y, z) = z^x$;

б) $g(x) = x$, $h(x, y, z) = x^z$;

в) $g(x) = 1$, $h(x, y, z) = z \cdot (x + 1)$;

г) $g(x) = x$, $h(x, y, z) = x \cdot y + f(x,0)$?

11. Докажите, что следующие функции примитивно рекурсивны:

а) $f(x) = x + 6$;

б) $f(x) = 2x + 1$;

в) $f(x) = x + x^2$;

г) $f(x) = x^3 + 3$;

д) $f(x) = 2^x$;

е) $f(x) = x^x$;

ё) $f(x) = 3^{x^2+1}$;

з) $f(x) = x^2 + 3x + 2$;

ж) $f(x) = |3x - 5|$;

и) $f(x) = |6x^2 + 2x - 8|$;

к) $f(x, y) = 2x + 3y$;

л) $f(x, y) = x^2 y$;

м) $f(x, y) = xy + 5$;

н) $f(x, y) = 2x - y$.

12. Верны ли следующие равенства:

а) $\mu_z[y + z = x] = x - y$;

в) $\mu_y[y - x = 0] = 0$;

б) $\mu_z[sg\, z = 1] = 1$; г) $\mu_y[y(y-(x+1)) = 0] = 0$?

13. Найдите результат применения операции минимизации $\varphi(x) = \mu_y[f(x,y) = 0]$ к следующим функциям:

а) $f(x,y) = |x - 2y|$;

б) $f(x,y) = y - x$;

в) $f(x,y) = 8$;

г) $f(x,y) = y + x + 1$;

д) $f(x,y) = x - y + 1$;

е) $f(x,y) = xy$.

14. Докажите, что функция $q(x,y) = \frac{x}{y}$ является частично рекурсивной.

15. Докажите, что функция $h(x,y) = \mu_z[y + z \geq x]$ является общерекурсивной.

16. Докажите, что функция $f(x) = \mu_y[x + 1 + y = 0]$ является частично рекурсивной функцией, нигде не определенной.

17. Найдите функции, обратные к следующим функциям:

а) $sg\, x = \begin{cases} 0, \textit{если } x = 0, \\ 1, \textit{если } x > 0 : \end{cases}$

б) $f(x) = 3x$;

в) $f(x) = x^2$.

18. Изобразите с помощью кругов Эйлера взаимосвязь между следующими классами рекурсивных функций:

1) частично рекурсивные функции всюду определенные (общерекурсивные функции);

2) частично рекурсивные функции не всюду определенные;

3) частично рекурсивные функции нигде не определенные;

4) примитивно рекурсивные функции.

Глава 3. Машина Тьюринга

3.1. Назначение и предпосылки создания машины Тьюринга

Возникновение машины Тьюринга связано с поиском ученых дать точный математический эквивалент для интуитивного представления об алгоритмах. Данная машина является еще одной широко известной моделью алгоритма.

Машина Тьюринга была предложена в 1936 г. (за 9 лет до появления первой ЭВМ) английским математиком Аланом Тьюрингом как *абстрактная вычислительная конструкция.* Целью ее создания было получение возможности доказательства существования или несуществования алгоритмов решения различных задач. Руководствуясь этой целью, Тьюринг искал как можно более простую, «бедную» алгоритмическую схему, лишь бы она была универсальной.

Прежде чем мы начнем знакомиться с машиной Тьюринга, необходимо сделать два общих замечания относительно объектов, с которыми работают алгоритмы.

Замечание 1

Одной из причин расплывчивости интуитивного понятия алгоритма является разнообразие объектов, с которыми работают алгоритмы. В вычислительных алгоритмах объектами являются числа. В алгоритме шахматной игры объектами являются фигуры и их позиции на шахматной доске. В алгоритме форматирования текста слова некоторого языка и правила переноса слов. Однако во всех этих и других случаях можно считать, что алгоритм имеет дело не с объектами реального мира, а с некоторыми изображениями этих объектов.

Например, есть алгоритм сложения двух целых чисел. Результатом сложения числовых объектов 26 и 22 будет числовой результат 48. Но мы можем считать, что объектом этого алгоритма является входная последовательность, состоящая из пяти символов: «$26+22$», а результатом является последовательность, состоящая из двух символов «48».

При этом мы исходили из того, что имеется набор из 11 различных символов $\{0,1,2,3,4,5,6,7,8,9,+\}$. Используемые символы будем называть *буквами*, а их набор *алфавитом*. В общем случае буквами могут служить любые символы, требуется только, чтобы они были различны между собой, а их число было конечным.

Определение 1

Произвольная конечная совокупность символов называется *алфавитом.*

Определение 2

Любая конечная последовательность букв (символов) из некоторого алфавита называется *словом* в этом алфавите. Количество букв в слове называется *длиной* слова. Слово, в котором нет букв, называется *пустым* словом.

Пример

$A = \{0,1,2,3,4,5,6,7,8,9,+\}$ алфавит,

48 слово в алфавите A, длина которого равна 2;

$26+22$ слово в алфавите A, длина которого равна 5.

Алгоритм сложения двух целых чисел перерабатывает слово «$26+22$» в слово «48».

Итак, объекты реального мира можно изображать словами в различных алфавитах. Это позволяет считать, что объектами работы алгоритмов могут быть только слова.

Определение 3

Слово, к которому применяется алгоритм, называется *входным словом*; слово, получаемое в результате работы алгоритма, называется *выходным*. Совокупность слов, к которым применим алгоритм, называется *областью применимости алгоритма.*

К сожалению, нельзя доказать, что все возможные объекты можно описать словами, так как само понятие объекта не было формально (т. е. строго) определено. Но можно проверить, что для любого наугад взятого алгоритма, работающего не над словами, его объекты можно выразить так, что они становятся словами, а суть алгоритма от этого не меняется.

Замечание 2

Любой алфавит можно заменить другим. Такая замена называется *кодированием.*

Например, пусть каждой букве из первого алфавита ставится в соответствие код, представляющий собой слово во втором алфавите. В качестве второго алфавита достаточно иметь алфавит из двух букв, так как любое слово из любого алфавита можно закодировать в двухбуквенном алфавите с гарантией однозначного восстановления исходного слова. Следовательно, любой алгоритм можно свести к алгоритму над словами в алфавите $\{0,1\}$, а перед применением алгоритма входное слово следует закодировать, после применения алгоритма выходное слово надо раскодировать.

Будем считать, что алгоритмы работают со словами, и мы формально описываем объекты-слова, над которыми работают алгоритмы, в некотором алфавите.

В дальнейшем для уточнения понятия алгоритма следует формально описать действия над объектами-словами и порядок выполнения этих действий. В качестве такой формальной схемы мы и рассмотрим машину Тьюринга.

3.2. Устройство машины Тьюринга

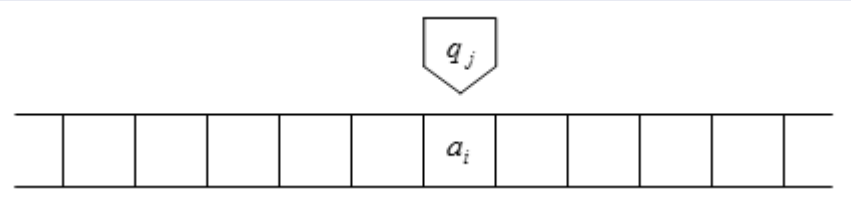
Машина Тьюринга является абстрактной машиной, т.е. существует не реально, а лишь в воображении. Машина Тьюринга это строгое математическое построение, математический аппарат (аналогичный, например, аппарату дифференциальных уравнений), созданный для решения определенных задач. Этот математический аппарат 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 же 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.

Машина Тьюринга представляет собой систему, работающую в дискретные моменты времени $t = 0,1,2,3,\ldots$ и состоящую из следующих частей (рис. 4):

1. *Ленты*, разбитой на ячейки и *бесконечной* в обе стороны. В каждой ячейке может быть записан один символ из конечного алфавита $A = \{a_0, a_1, a_2, \ldots, a_m\}$, называемого *внешним алфавитом*.

2. *Управляющего устройства*, которое может находиться в одном из конечного числа *внутренних состояний* $Q = \{q_0, q_1, q_2, \ldots, q_n\}$. Число элементов в Q характеризует объем внутренней памяти машины.

3. *Считывающей/пишущей головки (автомата)*, которая может перемещаться вдоль ленты и в каждый момент времени обозревает (считывает) одну из ячеек ленты.



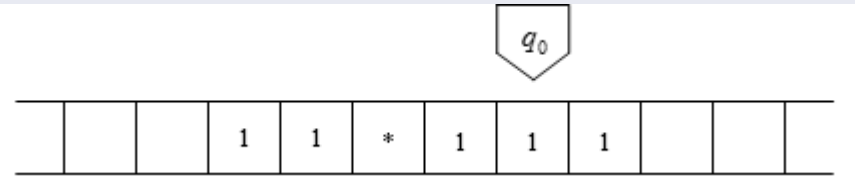
С каждой машиной Тьюринга связаны два конечных алфавита:

1) алфавит входных символов $A = \{a_0, a_1, a_2, \ldots, a_m\}$, в котором условимся считать a_0 *символом пустой ячейки;*

2) алфавит внутренних состояний $Q = \{q_0, q_1, q_2, \ldots, q_n\}$, в котором условимся считать состояние q_1 *начальным* (в этом состоянии машина всегда начинает работать), состояние q_0 *– заключительным* (в этом состоянии машина всегда останавливается).

Входное слово размещается на ленте по одной букве в расположенных подряд ячейках. Слева и справа от входного слова находятся только пустые ячейки (в алфавит A всегда входит «пустая» буква a_0 символ того, что ячейка пуста).

Пример



На ленте машины Тьюринга записано слово 11*111; автомат обозревает второй символ входного слова, считая справа.

3.3. Команды и порядок работы машины Тьюринга

Опишем, как машиной Тьюринга осуществляются алгоритмы по переработке входных слов.

Функционирование машины Тьюринга происходит в дискретные моменты времени $t = 0,1,2,3,\ldots$ и заключается в следующем. В зависимости от того, какая буква a_i обозревается автоматом, а также в зависимости от своего внутреннего состояния q_j, автомат машины Тьюринга может выполнять следующие действия:

- записать новую букву в обозреваемую ячейку;
- выполнить сдвиг по ленте на одну ячейку вправо или влево, остаться на месте;
- перейти в новое внутреннее состояние.

Таким образом, работа машины Тьюринга определяется системой *команд* вида:

$$q_j a_i \to q_k a_l X \quad (1)$$, где

q_j исходное внутреннее состояние машины;

a_i считываемый символ;

q_k новое внутреннее состояние машины;

a_l новый записываемый символ в обозреваемую изначально ячейку ленты машины Тьюринга;

$X \in \{Л, П, С\}$ направление движения автомата, обозначаемое одним из символов: $Л$ (влево), $П$ (вправо), $С$ (стоим на месте).

Предполагается, что для каждой пары $q_j a_i$, где $j = \overline{1,n}$, $i = \overline{0,m}$ имеется точно одна команда вида (1).

Как же работает машина Тьюринга? Как этот универсальный исполнитель осуществляет переработку входного слова в выходное слово в соответствии с определенным алгоритмом? Опишем эту работу.

Находясь в какой-либо момент времени в незаключительном состоянии (т. е. в состоянии, отличном от q_0), машина совершает шаг, который полностью определяется ее текущим состоянием q_j и символом a_i, воспринимаемым ею в данный момент на ленте. При этом содержание шага регламентировано соответствующей командой $q_j a_i \to q_k a_l X$, где $X \in \{Л, П, С\}$. Шаг заключается в том, что:

1) содержимое a_i обозреваемой на ленте ячейки стирается и на его место записывается символ a_l;

2) машина переходит в новое состояние q_k;

3) машина переходит к обозреванию следующей правой ячейки от той, которая обозревалась только что, если $X = П$, или к обозреванию следующей левой ячейки, если $X = Л$, или же продолжает обозревать ту же ячейку, если $X = С$.

В следующий момент времени (если $q_k \neq q_0$) машина делает шаг, регламентированный командой $q_k a_l \to q_r a_s X$ и т. д. до тех пор, пока не будет достигнуто состояние останова q_0.

Как только будет достигнуто заключительное состояние q_0 машина Тьюринга остановится, при этом последовательность символов, записанных в этот момент на ленте машины Тьюринга, будет являться выходным словом.

Ниже приведем несколько определений основных понятий, связанных с функционированием машины Тьюринга.

Определение 1

Совокупность всех команд машины Тьюринга, описывающей определенный алгоритм, называется *программой* (или *функциональной схемой*) машины Тьюринга.

Замечание

Программа машины Тьюринга может записываться либо «цепочкой» команд через запятую, либо в виде таблицы (см. табл. 5), в каждой клетке которой записана команда. Отметим, что таблица, определяющая порядок работы машины Тьюринга, не является в прямом смысле слова программой, так как ее предписания выполняются не последовательно, одно за другим, а описывают преобразования символов входного слова, находящегося на ленте.

Кроме того, подчеркнем, что программа машины Тьюринга с внешним алфавитом $A = \{a_0, a_1, a_2, \ldots, a_m\}$ и алфавитом внутренних состояний $Q = \{q_0, q_1, q_2, \ldots, q_n\}$ содержит ровно $n(m+1)$ команд (поясните почему).

Таблица 5

Общий вид записи программы машины Тьюринга

Q \ A	a_0	a_1	…	a_i	…	a_m
q_0	…	…	…	…	…	…
q_1	…	…	…	…	…	…
…	…	…	…	…	…	…
q_j	…	…	…	$q_k a_l \begin{cases} Л \\ П \\ С \end{cases}$	…	…
…	…	…	…	…	…	…
q_n	…	…	…	…	…	…

Определение 2

Под k *-й конфигурацией* будем понимать изображение ленты машины с информацией, сложившейся на ней к началу k-го шага (или слово в алфавите A, записанное на ленту к началу k-го шага), с указанием того, какая ячейка обозревается в этот шаг и в каком внутреннем состоянии находится машина.

Конфигурация называется заключительной, если состояние, в которой при этом находится машина, заключительное.

Определение 3

Активной зоной конфигурации назовем минимальную часть ленты машины, содержащую обозреваемую ячейку, а также все ячейки, в которых записаны непустые символы.

Определение 4

Будем говорить, что непустое слово α в алфавите $A \setminus \{a_0\} = \{a_1, a_2, \ldots, a_m\}$ воспринимается машиной в *стандартном положении,* если оно записано в последовательных ячейках ленты, все другие ячейки пусты, и машина обозревает крайнюю справа ячейку из тех, в которых записано слово α.

Определение 5

Стандартное положение называется *начальным*, если машина, воспринимающая слово в стандартном положении, находится в начальном состоянии q_1.

Определение 6

Будем говорить, что *слово* α *перерабатывается машиной в слово* β, если от слова α, воспринимаемого в начальном стандартном положении, машина после выполнения конечного числа команд приходит к слову β, воспринимаемому в положении останова (в заключительном состоянии).

Определение 7

Если в программе машины Тьюринга нет команды, приводящей машину в состояние останова, или машина в процессе работы не приходит в состояние q_0, то говорят, что машина Тьюринга *неприменима* к данному входному слову.

Машина Тьюринга *применима* к входному слову только в том случае, если, начав работу над этим входным словом, она рано или поздно дойдет до состояния останова.

Пример

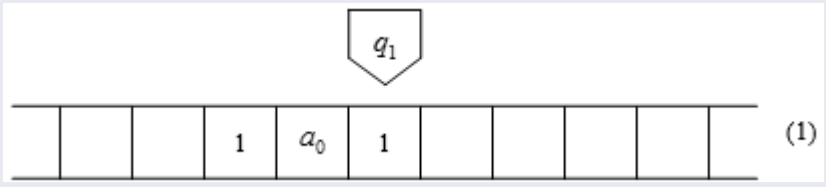
Дана машина Тьюринга с внешним алфавитом $A = \{a_0, 1\}$, алфавитом внутренних состояний $Q = \{q_0, q_1, q_2\}$ и со следующей функциональной схемой:

$q_1a_0 \to q_2a_0\Pi; \quad q_2a_0 \to q_0 1C; \quad q_1 1 \to q_1 1\Pi; \quad q_2 1 \to q_2 1\Pi.$

Определим, в какое слово перерабатывает эта машина слово 1 a_0 1, исходя из стандартного начального положения.

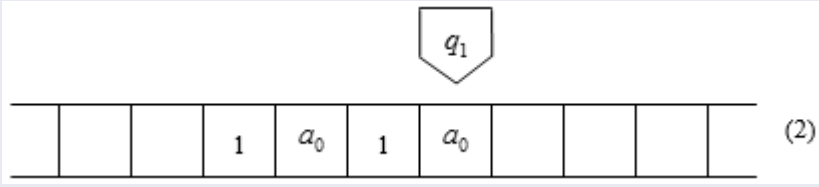
Решение

Будем последовательно выписывать конфигурации машины при переработке ею данного слова.



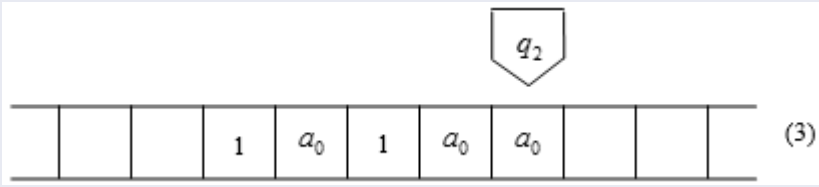
(1)

Конфигурация (1) является начальной. Здесь автомат обозревает крайний правый символ входного слова, а машина находится в состоянии q_1. Поэтому применяем команду $q_1 1 \to q_1 1\Pi$, в результате чего будет осуществлен переход к следующей конфигурации (2).



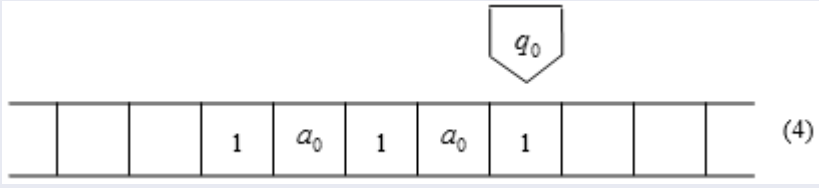
(2)

В конфигурации (2) автомат обозревает пустую ячейку (символ a_0), а машина находится в состоянии q_1. Поэтому используем команду $q_1a_0 \to q_2a_0\Pi$. После ее применения состояние машины Тьюринга будет характеризоваться конфигурацией (3).



(3)

В конфигурации (3) автомат обозревает символ a_0, а машина находиться в состоянии q_2, поэтому применяем команду $q_2a_0 \to q_0 1C$. В результате мы будем иметь конфигурацию (4).



(4)

В конфигурации (4) машина находится в состоянии q_0, т. е. в состоянии останова. Поэтому последовательность символов внешнего алфавита на ленте в этот момент времени образует выходное

слово. Иными словами, исходное слово $1\,a_0\,1$ переработано данной машиной Тьюринга в выходное слово $1\,a_0\,1\,a_0\,1$.

Полученную последовательность конфигураций (1)–(4) можно записать более коротким способом:

$$1a_0q_11 \Rightarrow 1a_01q_1a_0 \Rightarrow 1a_01a_0q_2a_0 \Rightarrow 1a_01a_0q_01 \quad (5)$$

Подчеркнем, что в записи (5) важно следить за тем, чтобы в каждый момент времени внутреннее состояние машины Тьюринга q_i было записано перед символом обозреваемой ячейки.

3.4. Вычислимые по Тьюрингу функции

В связи с тем, что машина Тьюринга осуществляет переработку каждого входного слова в выходное слово в соответствии с определенным алгоритмом, можно сказать, что машине Тьюринга соответствует некоторая словарная функция, которая каждому элементу из области определения (входному слову) ставит в соответствие единственный элемент из области значений (выходное слово). Здесь областью определения и областью значения словарной функции являются множества конечных слов в некоторых алфавитах, причем в случае, когда будут существовать слова из области определения, к которым данная машина Тьюринга неприменима, словарная функция будет являться частичной.

Вышесказанное раскрывает взаимосвязь машины Тьюринга с понятием функции.

Определение 1

Функция называется *вычислимой по Тьюрингу*, если существует машина Тьюринга, вычисляющая ее, т. е. такая машина Тьюринга, которая вычисляет ее значения для тех наборов значений аргументов, для которых функция определена, и работающая вечно, если функция для данного набора значений аргументов не определена.

Замечания

1. Подчеркнем, что в определении 1 речь идет о функциях (возможно частичных, т. е. не всюду определенных), заданных на множестве натуральных чисел и принимающих также натуральные значения.

2. Условимся, как записывать на ленте машины Тьюринга значения $x_1, x_2, \ldots, x_n$ аргументов функции $f(x_1, x_2, \ldots, x_n)$, из какого положения начинать переработку исходного слова и, наконец, в каком положении получать значение функции. Это можно делать, например, следующим образом.

Значения $x_1, x_2, \ldots, x_n$ аргументов будем располагать на ленте в виде следующего слова: $0\underbrace{1\ldots1}_{x_1}0\underbrace{1\ldots1}_{x_2}0\ldots0\underbrace{1\ldots1}_{x_n}0$ (1). Здесь полезно ввести следующие обозначения: $\underbrace{1\ldots1}_{x} = 1^x$, $\underbrace{0\ldots0}_{x} = 0^x$. Таким образом, предыдущее слово (1) можно представить следующим образом: $01^{x_1}01^{x_2}0\ldots01^{x_n}0$.

Далее начинать переработку данного слова будем из стандартного начального положения, т. е. из положения, при котором в состоянии q_1 обозревается крайняя правая единица записанного слова. Если функция $f(x_1, x_2, \ldots, x_n)$ определена на данном наборе значений аргументов, то в результате на ленте должно быть записано подряд $f(x_1, x_2, \ldots, x_n)$ единиц, т. е. $1^{f(x_1, x_2, \ldots, x_n)}$; в противном случае машина должна работать бесконечно, т. е. никогда не приходить в состоянии останова q_0.

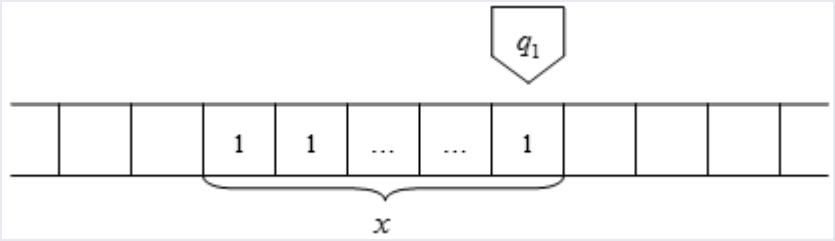
При выполнении всех перечисленных условий будем говорить, что *машина Тьюринга вычисляет данную функцию.* Таким образом, сформулированное определение 1 становится строгим.

Пример 1

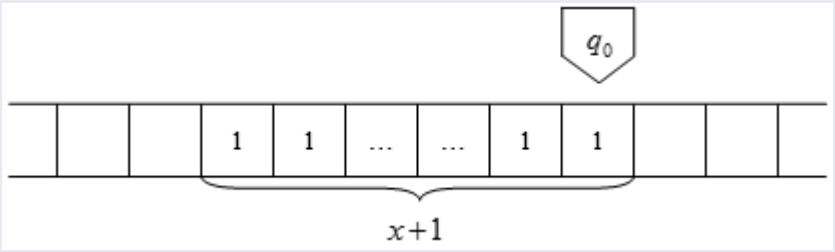
Разработаем программу машины Тьюринга, вычисляющую функцию $S(x) = x + 1$.

Решение

Запишем исходную конфигурацию машины Тьюринга.



Для того чтобы входное слово 01^x0 было переработано машиной Тьюринга в слово $01^{x+1}0$, нужно применить следующие команды: $q_1 1 \to q_1 1 П$; $q_1 0 \to q_0 1 С$. В результате заключительная конфигурация будет иметь вид:



Отметим, что приведенное решение не единственное.

Замечание

При составлении программы машины Тьюринга, вычисляющей некоторую функцию f, мы не очень строго относимся к тому, в каком начальном положении машина начинает работать (часто это бывает стандартное начальное положение), в каком завершает работу и как эта работа протекает.

В связи с этим, рассматривают более сильное понятие вычислимости функции на машине Тьюринга понятие правильной вычислимости.

Определение 2

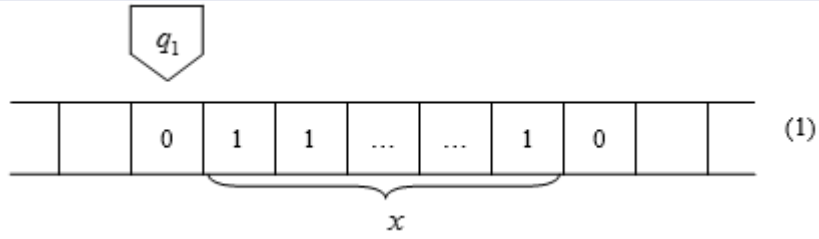
Будем говорить, что машина Тьюринга *правильно вычисляет* функцию $f(x_1, x_2, \ldots, x_n)$, если начальное слово $q_1 01^{x_1} 01^{x_2} 0 \ldots 01^{x_n} 0$ она переводит в слово $q_0 01^{f(x_1, x_2, \ldots, x_n)} 0 \ldots 0$ и при этом в процессе работы не пристраивает к начальному слову новых ячеек на ленте ни слева, ни справа. Если же функция f не определена на данном наборе значений аргументов, то, начав работать из указанного положения, она никогда в процессе работы не будет надстраивать ленту слева.

Пример 2

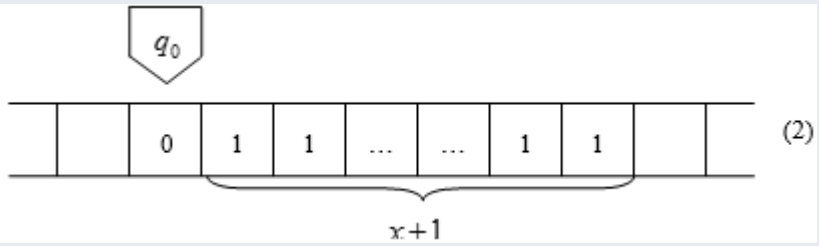
Разработаем программу машины Тьюринга, правильно вычисляющую функцию $S(x) = x + 1$.

Решение

Запишем исходную конфигурацию машины Тьюринга (1).



Для того чтобы входное слово 01^x0 было переработано машиной Тьюринга в слово $01^{x+1}0$ (причем заключительная конфигурация должна иметь вид $q_0 01^{x+1}0$), нужно применить следующие команды: $q_1 0 \to q_2 0П$, $q_2 1 \to q_2 1П$, $q_2 0 \to q_3 1Л$, $q_3 1 \to q_3 1Л$, $q_3 0 \to q_0 0С$. В итоге получим заключительную конфигурацию (2).



3.5. Основные операции над машинами Тьюринга

Прямое построение машин Тьюринга для решения даже простых задач может оказаться затруднительным. Однако существуют приемы, которые облегчают данный процесс, если использовать способы сочетания программ нескольких машин в результирующую программу. Рассмотрим основные способы сочетания машин Тьюринга.

1. Суперпозиция машин

Пусть даны две машины Тьюринга T_1 и T_2, которые вычисляют соответственно словарные функции $f_1(P)$ и $f_2(P)$ в одном и том же алфавите. Тогда существует машина Тьюринга T, которая вычисляет функцию $f(P) = f_2(f_1(P))$. При этом для любого слова P функция $f(P)$ определена в том и только том случае, когда $f_1(P)$ определена и $f_2(f_1(P))$ определена.

Программа машины T строится так: состояния машины T_2 переобозначим так, чтобы они отличались от состояний машины T_1. Начальное состояние q_1^1 машины T_1 объявим начальным q_1 для машины T, заключительное состояние q_0^2 машины T_2 объявим заключительным q_0 для машины T. Заключительное состояние q_0^1 машины T_1 отождествим с начальным состоянием q_1^2 машины T_2. Полученные T_2 команды для обеих машин объединяем в одну программу.

Рассмотрим начальную конфигурацию q_1P (в состоянии q_1 обозревается первый символ входного слова P). Поскольку $q_1 = q_1^1$ начальное состояние машины T_1, то вначале машина T будет работать как T_1. Если машина T_1 будет применима к q_1^1P, то на некотором шаге получим конфигурацию $q_0^1f_1(P)$. А так как $q_0^1 = q_1^2$ начальное состояние для машины T_2, то теперь машина T будет действовать как T_2. Если машина T_2 будет применима к $q_0^1f_1(P)$, то на некотором шаге будет получена конфигурация $q_0^2f_2(f_1(P))$, которая является заключительной для машины T, так как $q_0^2 = q_0$. Если машина T_1 окажется неприменимой к q_1^1P или машина T_2 окажется неприменимой к $q_0^1f_1(P)$, то машина T будет неприменимой к q_1P.

Машина T называется *суперпозицией машин* T_1 и T_2 и обозначается T_1T_2. Схематически суперпозицию изображают так:

$$P \underset{T_1}{\rightarrow} f_1(P) \underset{T_2}{\rightarrow} f_1(f_2(P)).$$

2. Соединение машин

Пусть даны машины Тьюринга T_1 и T_2, вычисляющие словарные функции $f_1(P)$ и $f_2(P)$ соответственно. Тогда существует машина T, которая начальную конфигурацию q_1P переводит в заключительную конфигурацию $q_0f_1(P)*f_2(P)$, если $f_1(P)$ и $f_2(P)$ определены, и неприменима в противном случае. Здесь $*$ новый символ, не входящий во внешний алфавит машин T_1 и T_2. Машина T называется *соединением машин* T_1 и T_2 и обозначается T_1*T_2. Существование машины T вытекает из следующих неформально описываемых конструкций.

Лента машины T является двухэтажной. В качестве внешнего алфавита T берут двухэтажные буквы $\begin{pmatrix} b \\ a \end{pmatrix}$, где a и b буквы внешнего алфавита машин T_1 и T_2. Каждой букве a внешнего алфавита машин T_1, T_2 ставится в соответствие двухэтажная буква $\begin{pmatrix} a_0 \\ a \end{pmatrix}$, где a_0 символ пустой буквы. Тогда слову $P = a_{i_1} \dots a_{i_k}$ ставится в соответствие двухэтажное слово $\begin{pmatrix} a_0 \dots a_0 \\ a_{i_1} \dots a_{i_k} \end{pmatrix}$. Машина T будет работать так: $\begin{pmatrix} a_0 \dots a_0 \\ a_{i_1} \dots a_{i_k} \end{pmatrix} \Rightarrow \begin{pmatrix} a_{i_1} \dots a_{i_k} \\ a_{i_1} \dots a_{i_k} \end{pmatrix} \Rightarrow \begin{pmatrix} a_{i_1} \dots a_{i_k} \\ f_1(P) \end{pmatrix} \Rightarrow \begin{pmatrix} f_2(P) \\ f_1(P) \end{pmatrix} \Rightarrow \Rightarrow \begin{pmatrix} a_0 \dots a_0 \\ f_1(P) * f_2(P) \end{pmatrix}$. Причем здесь существование машин Тьюринга для реализации каждого шага очевидно.

3. Ветвление машин

Пусть даны машины Тьюринга T_1 и T_2, вычисляющие словарные функции $f_1(P)$ и $f_2(P)$ соответственно, которые заданы в одном и том же алфавите. Тогда существует машина Тьюринга T, которая начальную конфигурацию $q_1 \varepsilon * P$, где $\varepsilon \in \{0,1\}$, переводит в заключительную конфигурацию $q_0 f_1(P)$, если $\varepsilon = 0$ и в $q_0 f_2(P)$, если $\varepsilon = 1$.

Машина T называется *разветвлением машин* T_1 и T_2 и обозначается $T_1 v T_2$.

Схематически разветвление представлено на рис. 6.

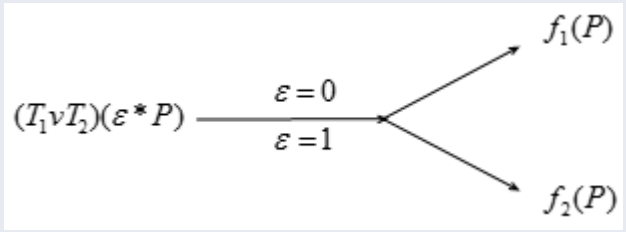


Рис. 6. Схематическое представление операции разветвления машин Тьюринга

Существование машины T вытекает из следующих конструкций. Пусть q_1^1 и q_1^2 начальные состояния машин T_1 и T_2 соответственно. Считаем, что множества внутренних состояний машин не пересекаются. Объединим программы машин T_1 и T_2, добавив новое начальное состояние q_1 и следующие команды: $q_1 0 \to q_1^1 a_0 П$, $q_1^1 * \to q_1^1 a_0 П$, $q_1^1 1 \to q_1^2 a_0 П$, $q_1^2 * \to q_1^2 a_0 П$.

Теперь заключительные состояния q_0^1 и q_0^2 машин T_1 и T_2 объединим, а полученное состояние q_0 считаем заключительным для машины T. Если $q_1 \varepsilon * P$ начальная конфигурация, то машина T через 2 шага перейдет в конфигурацию $q_1^1 P$, если $\varepsilon = 0$, и в конфигурацию $q_1^2 P$, если $\varepsilon = 1$, а затем будет работать как машина T_1 или T_2 соответственно.

4. Реализация цикла

Важным приемом в программировании является разбиение решаемой задачи на циклы. После выполнения каждого «прохода» цикла проверяется выполнимость некоторого условия. Если условие выполнено, то выдается результат, если нет, то цикл повторяется. Точнее, процедура задается так.

Пусть имеем словарные функции f_1 и f_2 и некоторый предикат Ф на словах. (Напомним, что n-*местным предикатом*, определенным на множествах $M_1,\dots,M_n$, называют выражение, содержащее n переменных $x_1,\dots,x_n$, превращающееся в высказывание при подстановке вместо этих переменных конкретных элементов из множеств $M_1,\dots,M_n$ соответственно.) Условимся значения предиката Ф обозначать 0, 1. Для произвольного слова P проверяется верно ли, что $\Phi(P)=1$, если да, то выдается ответ $f_1(P)$. Если $\Phi(P)=0$, то вычисляется $P'=f_2(P)$. Затем проверяется верно ли, что $\Phi(P')=1$, если да, то выдается ответ $f_1(P')$; если же окажется, что $\Phi(P')=0$, то вычисляется $P''=f_2(P')$ и т. д.

Оказывается, существует машина Тьюринга T, реализующая данную процедуру. Пусть существуют машины Тьюринга для вычисления функций f_1 и f_2 и предиката Ф. Обозначим их T_1, T_2, T_Φ соответственно. Пусть T_0 машина, которая оставляет всякое слово P без изменения. Машина T строится в соответствии со схемой, представленной на рис. 7.

Дадим некоторые пояснения к представленному рисунку. Заключительные состояния q_0^1 и q_0^2 машин T_1 и T_2 не объединяются, а считаются различными. Состояние q_0^1 объявляется заключительным для T_1, а q_0^2 отождествляется с начальным состоянием q_1 для T. Заключительное состояние для машины $T_\Phi * T_0$ объявляется начальным для $T_1 v T_2$. Из изложенного следует, что если $T_1 v T_2$ работает как T_1, то полученное ею значение является выходом T, если же $T_1 v T_2$ работает как T_2, то полученное ею значение снова подается на вход машины T.

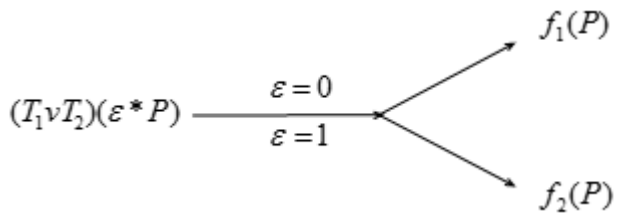


Рис. 7. Схематическое представление реализации цикла в машине Тьюринга

Описанные выше операции над машинами Тьюринга (суперпозиция машин, соединение машин, ветвление машин, реализация цикла), являются удобным инструментом для конструирования машин Тьюринга. Ниже приведем список машин Тьюринга, которые часто используются в качестве составляющих для других машин:

1. A (перенос нуля): $q_1 001^x 0 \Rightarrow q_0 01^x 00$.
2. $Б^+$ (сдвиг вправо): $q_1 a_i 1^x 0 \Rightarrow a_i 1^x q_0 0$.
3. $Б^-$ (сдвиг влево): $01^x q_1 a_i \Rightarrow q_0 01^x a_i$.

4. B (транспозиция): $q_1 01^x 01^y 0 \Rightarrow q_0 01^y 01^x 0$.
5. K (копирование): $q_1 01^x 00^x 0 \Rightarrow q_0 01^x 01^x 0$.
6. $Л$ (стирающая машина): $q_1 01^x 0 \Rightarrow q_0 00^x 0$.
7. R (удаление 1): $q_1 01^{x+1} 0 \Rightarrow q_0 01^x 00$.
8. S (добавление 1): $q_1 01^x 0 \Rightarrow q_0 01^{x+1} 0$.

В качестве упражнения читателю предлагается написать программы для вышеуказанных машин Тьюринга. Мы же приведем пример, как могут быть использованы эти машины для конструирования более сложной машины Тьюринга.

Пример

Разработаем машину Тьюринга, правильно вычисляющую функцию $I_2^3(x_1, x_2, x_3) = x_2$.

Решение

В соответствии с определением функции, правильно вычислимой на машине Тьюринга, мы должны переработать начальное слово $q_1 01^{x_1} 01^{x_2} 01^{x_3} 0$ в слово $q_0 01^{x_2} 0$. Для этого будем использовать машины Тьюринга из списка (1)–(8).

$\underbrace{q_1 01^{x_1} 01^{x_2}}_{(4)} 01^{x_3} 0$ (указано, что будем применять машину (4));

B : $\underbrace{q_{i_1} 01^{x_2} 01^{x_1}}_{(2)} 01^{x_3} 0$ (результат применения машины (4));

$Б^+$: $01^{x_2} \underbrace{q_{i_2} 01^{x_1}}_{(6)} 01^{x_3} 0$ (результат применения машины (2));

$Л$: $01^{x_2} \underbrace{q_{i_3} 00^{x_1}}_{(2)} 01^{x_3} 0$ (результат применения машины (6));

$Б^+$: $01^{x_2} 00^{x_1} \underbrace{q_{i_4} 01^{x_3}}_{(6)} 0$ (результат применения машины (2));

$Л$: $01^{x_2} \underbrace{00^{x_1} q_{i_5}}_{(3)} 00^{x_3} 0$ (результат применения машины (6));

$Б^-$: $\underbrace{01^{x_2} q_{i_6}}_{(3)} 00^{x_1} 00^{x_3} 0$ (результат применения машины (3));

$Б^-$: $q_0 01^{x_2} 00^{x_1} 00^{x_3} 0$ (результат применения машины (3)).

Таким образом, функция $I_2^3(x_1, x_2, x_3) = x_2$ вычисляется путем последовательного применения машин $В$, $Б^+$, $Л$, $Б^+$, $Л$, $Б^-$, $Б^-$. Подчеркнем, что нами приведена лишь общая схема реализации машины Тьюринга, правильно вычисляющей данную функцию (для написания же полной программы нужно строго следить за внутренним состоянием машины на каждом шаге ее работы).

3.6. Тезис Тьюринга

Вернемся к интуитивному представлению об алгоритмах. Напомним, что одно из свойств алгоритма заключается в том, что он представляет собой единый способ, позволяющий для каждой задачи из некоего бесконечного множества задач за конечное число шагов найти ее решение.

На понятие алгоритма можно взглянуть и с несколько иной точки зрения. Каждую задачу из бесконечного множества задач можно выразить (закодировать) некоторым словом некоторого алфавита, а решение задачи — каким-то другим словом того же алфавита. В результате получим функцию, заданную на некотором подмножестве множества всех слов выбранного алфавита и принимающую значения на множестве всех слов того же алфавита. Решить какую-либо задачу — значит найти значение этой функции на слове, кодирующем данную задачу. А иметь алгоритм для решения всех задач данного класса — значит иметь единый способ, позволяющий за конечное число шагов «вычислить» значения построенной функции для любых значений аргументов из ее области определения. Таким образом, алгоритмическая проблема — по существу, проблема о вычислении значений функции, заданной в некотором алфавите.

Остается уточнить, что значит уметь вычислять значения функции. Это значит вычислять значения функции с помощью подходящей машины Тьюринга. Для каких же функций возможно их тьюрингово вычисление? Многочисленные исследования ученых, обширный опыт показали, что такой класс функций чрезвычайно широк. Каждая функция, для вычисления значений которой существует какой-либо алгоритм, оказывалась вычислимой посредством некоторой машины Тьюринга. (Это можно обосновать тем, что язык Тьюрингова программирования содержит основные операторы программирования и позволяет осуществлять последовательное выполнение программ, параллельное их соединение, использовать условные переходы, реализовать цикл). Это является основанием для предположения о том, что для всех процедур, претендующих называться алгоритмическими, существует (при подходящем кодировании) реализующая их машина Тьюринга. Данное предположение носит название *тезиса Тьюринга*.

Тезис Тьюринга

Для нахождения значений функции, заданной в некотором алфавите, тогда и только тогда существует какой-нибудь алгоритм, когда функция является вычислимой по Тьюрингу, т. е. когда она может вычисляться на подходящей машине Тьюринга.

Это означает, что строго математическое понятие вычислимой (по Тьюрингу) функции является по существу идеальной моделью взятого из опыта понятия алгоритма. Данный тезис есть не что иное, как аксиома, постулат, выдвигаемый нами, о взаимосвязях нашего опыта с той математической теорией, которую мы под этот опыт хотим подвести. Конечно же, данный тезис в принципе не может быть доказан средствами математики, потому что он не имеет внутриматематического характера (одна сторона в тезисе — понятие алгоритма — не является точным математическим понятием). Он выдвинут исходя из опыта, и именно опыт подтверждает его состоятельность.

Впрочем, не исключается принципиальная возможность того, что тезис Тьюринга будет опровергнут. Для этого должна быть указана функция, которая вычислима с помощью какого-нибудь алгоритма, но не вычислима ни на какой машине Тьюринга. Но такая возможность представляется маловероятной (в этом одно из значений тезиса): всякий алгоритм, который будет открыт, может быть реализован на

машине Тьюринга.

3.7. Машины Тьюринга и современные электронно-вычислительные машины

Изучение машин Тьюринга и практика составления программ для них закладывают фундамент алгоритмического мышления, сущность которого состоит в том, что нужно уметь разделять тот или иной процесс вычисления или какой-либо другой деятельности на простые составляющие шаги. В машине Тьюринга расчленение (анализ) вычислительного процесса на простейшие операции доведено до предельной возможности: распознавание единичного рассмотренного вхождения символа, перемещение точки наблюдения данного ряда символов в соседнюю точку и изменение имеющейся в памяти информации. Конечно, такое мелкое дробление вычислительного процесса, реализуемого в машине Тьюринга, значительно его удлиняет. Но логическая структура процесса, расчлененного, образно выражаясь, до атомарного состояния, значительно упрощается и предстает в некотором стандартном виде, весьма удобном для теоретических исследований (именно такое расчленение на простейшие составляющие вычислительного процесса на машине Тьюринга дает еще один косвенный аргумент в пользу тезиса Тьюринга: всякая функция, вычисляемая с помощью какого-либо алгоритма, может быть вычислена на машине Тьюринга, потому что каждый шаг данного алгоритма можно расчленить на еще более мелкие операции, которые реализуются в машине Тьюринга.) Таким образом, понятие машины Тьюринга есть теоретический инструмент анализа алгоритмического процесса, а значит, анализа существа алгоритмического мышления.

В современных ЭВМ алгоритмический процесс расчленен не на столь мелкие составляющие, как в машинах Тьюринга. Наоборот, создатели ЭВМ стремятся к известному укрупнению выполняемых машиной процедур (на этом пути, конечно, есть свои ограничения). Так, для выполнения операции сложения на машине Тьюринга составляется целая программа, а в современной ЭВМ такая операция является простейшей.

Машина Тьюринга обладает бесконечной внешней памятью (неограниченная в обе стороны лента, разбитая на ячейки). Но ни в одной реально существующей машине бесконечной памяти быть не может. Это говорит о том, что машины Тьюринга отображают потенциальную возможность неограниченного увеличения объема памяти современных ЭВМ.

Можно провести более подробный сравнительный анализ работы современной ЭВМ и машина Тьюринга. В большинстве ЭВМ принята трехадресная система команд, обусловленная необходимостью выполнения бинарных операций, в которых участвует содержимое сразу трех ячеек памяти. Например, число из ячейки *a* умножается на число из ячейки *b*, и результат отправляется в ячейку *c*. Существуют ЭВМ двухадресные и одноадресные. Так, одноадресная ЭВМ работает следующим образом: вызывается (в сумматор) число из ячейки *a*; в сумматоре происходит, например, умножение этого числа на число из ячейки *b*; результат отправляется из сумматора в ячейку *c*. Машину Тьюринга можно считать одноадресной машиной, в которой система одноадресных команд упрощена еще больше: на каждом шаге работы машины команда предписывает замену лишь единственного знака, хранящегося в обозреваемой ячейке, а адрес обозреваемой ячейки при переходе к следующему такту может меняться лишь на единицу (обозрение соседней справа или слева ячейки ленты) или не меняется вовсе. Это удлиняет процесс, но в то же время резко унифицирует его, делает стандартным.

Подводя итоги, можно сказать, что современные ЭВМ есть некие реальные физические модели машин Тьюринга, огрубленные с точки зрения теории, но созданные в целях реализации конкретных вычислительных процессов. В свою очередь, понятие машины Тьюринга и теория таких машин есть теоретический фундамент и обоснование современных ЭВМ.

Вопросы к главе 3

1. Кем и когда была разработана машина Тьюринга?
2. Сформулируйте цель создания машины Тьюринга.
3. Что называется алфавитом? Приведите пример какого-либо алфавита.
4. Что называется словом? Какое слово считается пустым? Приведите пример слова, длина которого равна 7.
5. Поясните, что применительно к алгоритму называют входным словом, выходным словом.
6. Поясните сущность операции кодирования. Приведите пример какой-либо кодировки.
7. Как вы считаете, машина Тьюринга это реальная или абстрактная машина? Ответ поясните.
8. Опишите основные составные части машины Тьюринга. Ответ проиллюстрируйте.
9. Раскройте суть алфавитов (внешнего и внутреннего), с которыми работает машина Тьюринга.
10. В каком внутреннем состоянии машина Тьюринга начинает работать? В каком останавливается?
11. В чем суть следующей команды машины Тьюринга: $q_s a_n \to q_t a_m X$, где $X \in \{П, Л, С\}$?
12. Что представляет собой программа (функциональная схема) машины Тьюринга?
13. Сколько команд может содержать программа машины Тьюринга, если ее внешний алфавит $A = \{0,1,2,3,4,5,6,7,8,9\}$, а алфавит внутренних состояний $Q = \{q_0, q_1, q_2, q_3\}$?
14. Что представляет собой конфигурация машины Тьюринга на i -шаге ее работы?
15. Что понимают под активной зоной конфигурации машины Тьюринга?
16. Что означают слова «слово p воспринимается машиной Тьюринга в стандартном положении»?
17. Чем характеризуется стандартное начальное положение машины Тьюринга?
18. Что означают слова «слово p_1 перерабатывается машиной Тьюринга в слово p_2 »?
19. Когда говорят, что машина Тьюринга применима (неприменима) к входному слову?
20. Какая функция называется вычислимой по Тьюрингу?
21. Что означают слова «машина Тьюринга работает вечно»?
22. Запишите начальную и заключительную конфигурацию для машины Тьюринга, вычисляющей функцию $f(x, y, z) = x + y + z$, в том числе и для вычисления ее значения в точке $(4,1,5)$.
23. Какая функция называется правильно вычислимой по Тьюрингу? Является ли правильно вычислимая по Тьюрингу функция вычислимой по Тьюрингу?
24. Перечислите, какие основные операции можно выполнять с машинами Тьюринга.
25. Раскройте суть суперпозиции машин Тьюринга.

26. Раскройте суть соединения машин Тьюринга.

27. Раскройте суть ветвления в машине Тьюринга.

28. Раскройте суть реализации цикла в машине Тьюринга.

29. Назовите, какие машины Тьюринга часто используются в качестве составляющих для других машин Тьюринга.

30. Сформулируйте тезис Тьюринга и раскройте его роль в теории алгоритмов.

31. Выделите общее и различное между машинами Тьюринга и современными электронно-вычислительными машинами.

Задания к главе 3

1. Опишите, какой алгоритм выполняет данная машина Тьюринга.

Q \ A	a_0	0	1
q_1	q_0a_0C	$q_11\Pi$	$q_10\Pi$

Известно, что в начальном состоянии автомат обозревает самый левый символ входного слова. К каким словам, составленным из символов внешнего алфавита применима эта машина Тьюринга?

2. Опишите, какой алгоритм выполняет данная машина Тьюринга.

Q \ A	a_0	0	1
q_1	q_0a_0C	$q_1a_0\Pi$	$q_1a_0\Pi$

К каким словам, составленным из символов внешнего алфавита, применима машина? Какое условие надо наложить на начальное положение автомата для того, чтобы результат применения машины к произвольному слову из алфавита был одним и тем же?

3. Опишите, какой алгоритм выполняет данная машина Тьюринга. Автомат в начальном состоянии обозревает самый правый символ входного слова.

Q \ A	a_0	+	–
q_1	q_0a_0C	$q_2+\Pi$	$q_1-\Pi$
q_2	q_0a_0C	$q_1-\Pi$	$q_1-\Pi$

К каким словам, составленным из символов внешнего алфавита применима машина?

4. Покажите, что машина Тьюринга обладает всеми свойствами алгоритма.

5. Дана машина Тьюринга с внешним алфавитом $A=\{a_01\}$, алфавитом внутренних состояний $Q=\{q_0,q_1,q_2,q_3,q_4,q_5,q_6,q_7\}$ и со следующей функциональной схемой (программой):

Q \ A	q_1	q_2	q_3	q_4	q_5	q_6	q_7
a_0	$q_4a_0\Pi$	$q_6a_0\Pi$	$q_6a_0\Pi$	q_01C	$q_4a_0\Pi$	q_0a_0C	$q_6a_0\Pi$
1	$q_21\Pi$	$q_31\Pi$	$q_11\Pi$	q_5a_0C	q_5a_0C	q_7a_0C	q_7a_0C

Изображая на каждом такте работы машины получающуюся конфигурацию, определите, в какое слово перерабатывает машина каждое из следующих слов, исходя из начального стандартного положения:

а) 11111;

е) $1\,a_0\,111\,a_0\;\;a_0\,1111$;

б) 111111;
в) 1111;
г) 1111111;
д) 111;
ж) 11 a_0 a_0 111111;
з) 11 a_0 111.

6. Остановится ли когда-нибудь заданная машина Тьюринга, если она начнет перерабатывать следующее слово, начав в состоянии q_1 обозревать ячейку, в которой записана самая левая буква перерабатываемого слова:

а) 1111 a_0 1; б) 11111; в) 1 a_0 1 a_0 1?

A \ Q	q_1	q_2	q_3
a_0	$q_1a_0П$	$q_3a_0Л$	q_0a_0C
1	$q_1 1П$	$q_1a_0П$	$q_2 1Л$

Если машина остановится, то какова ее заключительная конфигурация?

7. Машина Тьюринга с внешним алфавитом $A=\{a_0,1\}$ определяется следующей функциональной схемой:

A \ Q	q_1	q_2	q_3
a_0	$q_2a_0П$	$q_2a_0П$	q_0a_0C
1	$q_1 1П$	$q_3 1П$	$q_3 1П$

Остановится ли когда-нибудь эта машина, если она начнет перерабатывать следующее слово (в начальный момент в состоянии q_1 машина обозревает ячейку, в которой записана самая левая буква перерабатываемого слова):

а) 111 a_0 a_0 1; б) 11 a_0 a_0 11 a_0 1; в) 111111?

Если остановка происходит, то какое слово получается в результате, какая ячейка и в каком (перед остановкой) состоянии обозревается?

8. Остановится ли когда-нибудь машина Тьюринга с внешним алфавитом $A=\{a_0,1\}$ и функциональной схемой:

A \ Q	q_1	q_2	q_3	q_4
a_0	$q_2a_0П$	$q_3a_0Л$	$q_1 1Л$	q_0a_0C
1	$q_2 1П$	$q_4a_0П$	$q_0 1C$	q_2a_0C

при переработке следующих слов (в начальный момент головка машины обозревает ячейку ленты, в которой записана самая левая буква перерабатываемого слова):

а) 111 a_0 1 a_0 1; б) 1111; в) 1 a_0 1 a_0 1 a_0 1?

9. Машина Тьюринга определяется следующей функциональной схемой.

A \ Q	q_1	q_2	q_3
a_0		$q_3 1 П$	$q_1 a_0 Л$
1	$q_2 a_0 Л$	$q_2 1 Л$	$q_3 1 П$
*	$q_0 a_0 С$	$q_2 * Л$	$q_3 * П$

Определите, в какое слово перерабатывает машина каждое из следующих слов, исходя из начального стандартного состояния. После этого постарайтесь усмотреть общую закономерность в работе машины:

а) 111*111;
б) 1111*11;
в) 111*1;
г) 1*11;
д) 11*111;
е) 11111*;
ж) *1111

10. Машина Тьюринга определяется следующей функциональной схемой.

A \ Q	q_1	q_2	q_3	q_4
a_0	$q_1 a_0 П$	$q_3 a_0 П$	$q_3 a_0 Л$	$q_1 a_0 Л$
1	$q_2 a_0 Л$	$q_2 1 Л$	$q_4 a_0 П$	$q_4 1 П$
*	$q_0 a_0 С$	$q_3 * Л$		$q_4 * П$

Определите, в какое слово перерабатывает машина каждое из следующих слов, исходя из стандартного начального состояния:

а) 111*11;
б) 11*11;
в) 1111*1;
г) 11111*111;
д) 11111*1111.

Постарайтесь выявить общую закономерность в работе машины.

11. На ленте записаны два числа в двоичной системе счисления, разделенные звездочкой:

		1	0	1	1	*	1	0	1		

Определите, какую операцию проделает с ними машина Тьюринга, начиная из стандартного начального положения, если программа машины задается таблицей:

	q_1	q_2	q_3	q_4	q_5	q_6

a_0	q_0a_0C		$q_1 1Л$	$q_5a_0П$	$q_6a_0Л$	
1	$q_2 0Л$	$q_2 1Л$	$q_4 0Л$	$q_4 1Л$	$q_5 1П$	$q_6 0Л$
0		$q_2 0Л$	$q_3 0Л$	$q_4 0Л$	$q_5 0П$	$q_6 0Л$
*		$q_3 * Л$			$q_5 * П$	$q_3 * Л$

12. Вопрос, аналогичный вопросу из предыдущей задачи, для ленты

		1	1	0	1	*	1	0	0	1	

и для машины Тьюринга с программой:

$q_1 1 \to q_1 0Л$, $q_1 0 \to q_1 0Л$,

$q_2 1 \to q_2 0Л$, $q_2 0 \to q_3 0Л$,

$q_3 1 \to q_4 1Л$, $q_1 * \to q_2 * Л$,

$q_4 1 \to q_1 0Л$, $q_1 a_0 \to q_0 a_0 C$.

13. Постройте такую машину Тьюринга, которая из n записанных подряд единиц оставляла бы на ленте $(n-2)$ единицы, также записанные подряд, если $n \geq 2$, и работала бы вечно, если $n = 0$ или $n = 1$.

14. Известно, что на ленте записано слово $\underbrace{11...11}_{n}$, $n \geq 1$. Постройте машину Тьюринга с внешним алфавитом $A = \{a_0, 1\}$, которая отыскивала бы левую единицу этого слова (т. е. приходила бы в состояние, при котором обозревалась бы ячейка с самой левой единицей данного слова, и в этом положении останавливалась), если в начальный момент головка машины обозревает одну из ячеек с буквой данного слова.

15. Сконструируйте машину Тьюринга с внешним алфавитом $A = \{a_0, 1\}$, которая каждое слово в алфавите $A_1 = \{1\}$ перерабатывает в пустое слово, исходя из стандартного начального положения.

16. Сконструируйте машину Тьюринга с внешним алфавитом $A = \{a_0, 1\}$, которая каждое слово длиной n в алфавите $A_1 = \{1\}$ перерабатывает в слово длиной $n+1$ в том же алфавите A_1.

17. Постройте машину Тьюринга для подсчета на ленте количества штрихов, которые располагаются

подряд и образуют входное слово, при этом требуется стереть все штрихи и записать на ленте их количество в десятичной системе. Кроме самой программы-таблицы опишите словами, что выполняется машиной в каждом состоянии.

18. Постройте машину Тьюринга, которая во входном слове все буквы «а» заменит на буквы «б». Кроме самой программы-таблицы опишите словами, что выполняется машиной в каждом состоянии.

19. Дано число в восьмеричной системе счисления. Разработайте машину Тьюринга, которая увеличивала бы заданное число на 1. Автомат в состоянии q_1 обозревает некую цифру входного слова. Кроме самой программы-таблицы опишите словами, что выполняется машиной в каждом состоянии.

20. Дано натуральное число $n > 1$. Разработайте машину Тьюринга, которая уменьшала бы заданное число n на 1, при этом в выходном слове старшая цифра не должна быть 0. Например, если входным словом было «100», то выходным словом должно быть «99», а не «099». Автомат в состоянии q_1 обозревает правую цифру числа. Кроме самой программы-таблицы опишите словами, что выполняется машиной в каждом состоянии.

21. Даны два натуральных числа m и n, представленные в унарной системе счисления. Соответствующие наборы символов (последовательности единиц) разделены пустой клеткой. Автомат в состоянии q_1 обозревает самый правый символ входной последовательности. Разработайте машину Тьюринга, которая на ленте оставит сумму чисел m и n. Кроме самой программы-таблицы опишите словами, что выполняется машиной в каждом состоянии.

22. Даны два натуральных числа m и n, представленные в унарной системе счисления. Соответствующие наборы символов (последовательности единиц) разделены пустой клеткой. Автомат в состоянии q_1 обозревает самый правый символ входной последовательности. Разработайте машину Тьюринга, которая на ленте оставит разность чисел m и n. Известно, что $m > n$. Кроме самой программы-таблицы опишите словами, что выполняется машиной в каждом состоянии.

23. На ленте машины Тьюринга записаны два набора единиц. Они разделены *. Составьте функциональную схему машины так, чтобы она выбрала больший из этих наборов, а меньший стерла, исходя из стандартного начального положения. Звездочка должна быть сохранена, чтобы было видно, какой из массивов выбран.

24. На ленте машины Тьюринга находится десятичное число. Определите, делится ли это число на 5 без остатка. Если делится, то запишите справа от числа слово «да», иначе – «нет». Автомат обозревает некую цифру входного числа. Кроме самой программы-таблицы опишите словами, что выполняется машиной в каждом состоянии.

25. Дан массив из открывающихся и закрывающихся скобок. Постройте машину Тьюринга, которая удаляла бы пары взаимных скобок, т. е. расположенных подряд «( )». Автомат в состоянии q_1 обозревает крайний левый символ строки. Кроме самой программы-таблицы опишите словами, что выполняется машиной в каждом состоянии.

26. Дана строка из букв a и b. Разработайте машину Тьюринга, которая переместит все буквы a в левую, а буквы b в правую часть строки. Каретка находится над крайним левым символом строки. Кроме самой программы-таблицы опишите словами, что выполняется машиной в каждом состоянии.

27. На ленте машины Тьюринга находится число, записанное в десятичной системе счисления. Умножьте это число на 2. Автомат в состоянии q_1 обозревает крайнюю левую цифру числа. Кроме самой программы-таблицы опишите словами, что выполняется машиной в каждом состоянии.

28. Алфавит $\{0, 1, 2, 3\}$. Стереть четвертую слева цифру «3». Если это удастся (т. е. если четвертая слева тройка существует), записать в этой клетке символ, который стоит непосредственно после нее. Если нет, то стереть всё слово.

29. Алфавит $\{0, 1, 2\}$. Определить, есть ли в слове два подряд вхождения символа «2». Если да, то каждую пару символов «2» заменить на пару символов «1». Если нет, то стереть все слово и записать вместо него символ «*».

30. Алфавит $\{0, 1\}$. Стереть вторую справа цифру «0». Если это удастся, то после выполнения данной операции записать в этой клетке символ, который стоит непосредственно перед ней. Если нет, стереть слово.

31. Алфавит $\{0, 1, 2\}$. Определить, есть ли в слове последовательность «012». Если есть одна или больше, то все такие последовательности заменить на последовательность «***». Если нет ни одной искомой последовательности, то приписать к концу слова символ «–».

32. Постройте следующие машины Тьюринга:

а) A (перенос нуля): $q_1 001^x 0 \Rightarrow q_0 01^x 00$.

б) $Б^+$ (сдвиг вправо): $q_1 a_i 1^x 0 \Rightarrow a_i 1^x q_0 0$.

в) $Б^-$ (сдвиг влево): $01^x q_1 a_i \Rightarrow q_0 01^x a_i$.

г) B (транспозиция): $q_1 01^x 01^y 0 \Rightarrow q_0 01^y 01^x 0$.

д) K (копирование): $q_1 01^x 00^x 0 \Rightarrow q_0 01^x 01^x 0$.

е) $Л$ (стирающая машина): $q_1 01^x 0 \Rightarrow q_0 00^x 0$.

ж) R (удаление 1): $q_1 01^{x+1} 0 \Rightarrow q_0 01^x 00$.

з) S (добавление 1): $q_1 01^x 0 \Rightarrow q_0 01^{x+1} 0$.

33. Докажите, что следующие функции вычислимы по Тьюрингу, для чего постройте машины Тьюринга, вычисляющие их:

а) $f(x, y) = x + y$;

б) $f(x) = x \dot{-} 1 = \begin{cases} 0, & \text{если } x = 0, \\ x - 1, & \text{если } x > 0; \end{cases}$

в) $sg(x) = \begin{cases} 0, & \text{если } x = 0, \\ 1, & \text{если } x > 0; \end{cases}$

г) $\overline{sg}(x) = \begin{cases} 0, & \text{если } x > 0, \\ 1, & \text{если } x = 0; \end{cases}$

д) $f(x, y) = x \dot{-} y = \begin{cases} 0, & \text{если } x \le y, \\ 1, & \text{если } x > y; \end{cases}$

е) $f(x, y) = x - y$;

ж) $f(x) = \frac{x}{2}$;

з) $f(x) = \left[\frac{x}{2}\right]$ целая часть числа $\frac{x}{2}$;

и) $f(x, y) = НОД(x, y)$;

к) $f(x) = 2x + 1$;

л) $f(x) = \left[\frac{1}{x}\right]$;

м) $f(x) = \frac{1}{x - 2}$;

н) $f(x) = \begin{cases} 1, & \text{если } x \text{ делится на } 3, \\ 0, & \text{если } x \text{ не делится на } 3. \end{cases}$

34. Машина Тьюринга имеет следующую функциональную схему:

	q_1	q_2	q_3
0	$q_2 1Л$	$q_3 1П$	$q_0 0С$
1	$q_1 1П$	$q_2 1Л$	$q_0 1С$

Найдите формульное выражение функции $f(x)$, вычисляемой машиной. Изначально машина находится в стандартном начальном положении.

35. По программе машины Тьюринга напишите формульное выражение функции $f(x,y)$, вычисляемой этой машиной:

	q_1	q_2	q_3	q_4	q_5	q_6
0	$q_2 0П$	$q_1 0Л$	$q_4 0Л$	$q_4 0Л$	$q_6 0П$	$q_0 0С$
1	$q_1 1П$	$q_3 0П$	$q_3 0П$	$q_5 1Л$	$q_5 1Л$	$q_0 1С$

Изначально машина находится в состоянии q_1 и обозревает самый левый символ входного слова.

36. Какую функцию $f(x)$ вычисляет машина Тьюринга со следующей программой: $q_1 0 \to q_2 0П$, $q_1 1 \to q_0 1С$, $q_1 0 \to q_3 1С$, $q_2 1 \to q_2 1П$, $q_3 0 \to q_0 0С$, $q_3 1 \to q_3 1Л$?

37. Пусть машина Тьюринга имеет следующую программу: $q_1 0 \to q_0 0С$. Какие функции $f(x)$, $f(x_1, x_2)$, …, $f(x_1, \ldots, x_n)$ вычисляет эта машина?

38. Постройте машины Тьюринга для правильного вычисления функций: а) $O(x) = 0$; б) $f(x) = x \dot{-} 1$; в) $f(x) = sg(x)$; г) $f(x) = \overline{sg}(x)$; д) $f(x) = x \dot{-} y$; е) $f(x) = x - y$.

39. Обоснуйте, что машина Тьюринга является конечным автоматом.

(*Конечным автоматом* называется шестерка объектов:

$A = \{S, X, Y, s0, d, v\}$, где

S – конечное непустое множество (состояний);

X – конечное непустое множество входных сигналов (входной алфавит);

Y – конечное непустое множество выходных сигналов (выходной алфавит);

$s0 \in S$ – начальное состояние;

$d: S \times X \rightarrow S$ – функция переходов;

$v: S \times X \rightarrow Y$ – функция выходов.)

40. Покажите на конкретном примере, как машину Тьюринга (по аналогии с конечным автоматом) можно задать в виде ориентированного графа.

Глава 4. Машина Поста

4.1. Назначение и устройство машины Поста

Почти одновременно с Тьюрингом американский математик Эмиль Пост в 1937 году предложил иную абстрактную машину (алгоритмическую модель), характеризующуюся еще большей простотой, чем машина Тьюринга. Это строгое математическое построение было также предложено в качестве уточнения понятия алгоритма.

Идеи Эмиля Поста оказались настолько фундаментальны, что к ним вернулись через 30 лет. В 1967 г. профессор В.А. Успенский пересказывает их с новых позиций, именно он вводит термин машины Поста, трактуя ее как абстрактную машину, которая работает по алгоритмам, разработанным человеком.

Машина Поста, как и машина Тьюринга, представляет собой универсального исполнителя, являющегося полностью детерминированным, позволяющему «вводить» начальные данные и после выполнения программ «читать» результат. Машина Поста является менее популярной, чем машина Тьюринга, хотя она устроена проще в том отношении, что ее элементарные действия проще, чем элементарные действия машины Тьюринга, и способы записи менее разнообразны. Именно по этим причинам запись и переработка информации на машине Поста требует, вообще говоря, большего объема «памяти» и большего числа шагов, чем на машине Тьюринга.

В основе создания машины Поста лежат следующие *идеи*:

- вся информация, которая должна быть обработана по существу, должна быть обработана по форме, т. е. с помощью двоичного алфавита;
- вся информация должна обрабатываться побуквенно.

В соответствии с указанными идеями машина Поста была разработана как абстрактная конструкция, представляющая собой *бесконечную ленту,* разделенные на одинаковые клетки, каждая из которых может быть либо пустой, либо заполненной меткой «V» (рис. 8). Подчеркнем, что это ограничение не влияет на универсальность машины Поста, так как любой алфавит может быть закодирован двумя знаками.

Кроме ленты в машине Поста имеется *каретка (головка чтения/записи),* которая:

- умеет перемещаться вдоль ленты на одну клетку вправо или влево;
- умеет наносить в обозреваемую клетку метку, если этой метки там ранее не было;
- умеет стирать метку, если она была в обозреваемой ячейке;
- умеет проверять наличие в клетке метки.

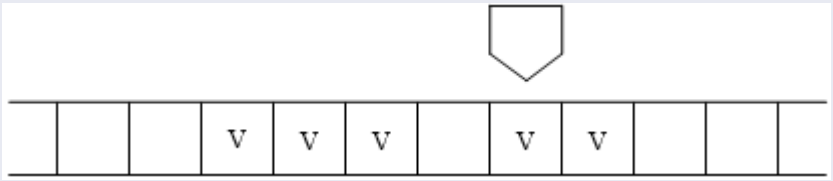


Рис. 8

Информация о заполненных метками клетках ленты характеризует состояние ленты, которое может меняться в процессе работы машины. В каждый момент времени каретка находится над одной из клеток ленты (обозревает ее). Информация о местоположении каретки вместе с состоянием ленты

характеризует *состояние машины Поста.*

Замечание

Состояние машины Поста отличается от состояния машины Тьюринга:

- в машине Тьюринга состояние определяет, что следует записать в обозреваемую ячейку для каждого определенного символа, задает характер движения каретки и, наконец, указывает новое состояние машины;
- в машине Поста состояние описывает местонахождение каретки и состояние ленты.

4.2. Команды и порядок работы машины Поста

Все *команды* машины Поста имеют следующую *структуру*:

nKm , где

n номер текущей команды;

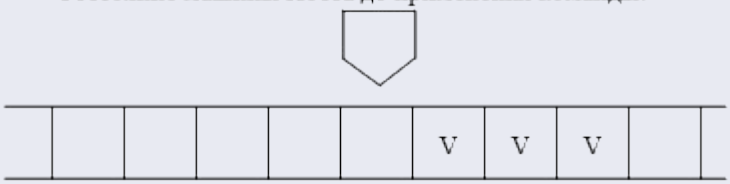
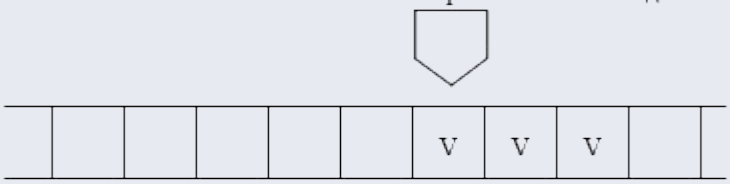
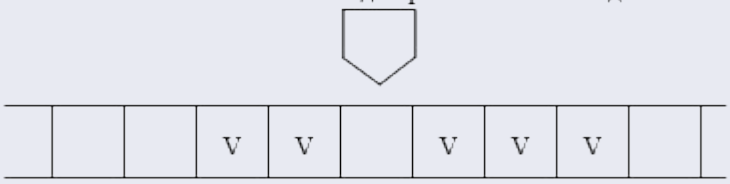
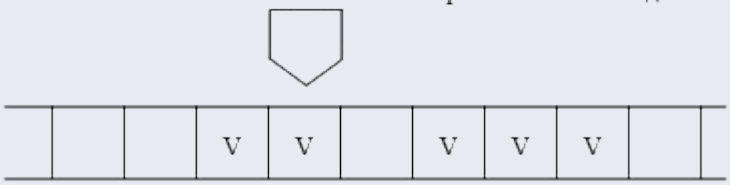
K действие, выполняемое кареткой;

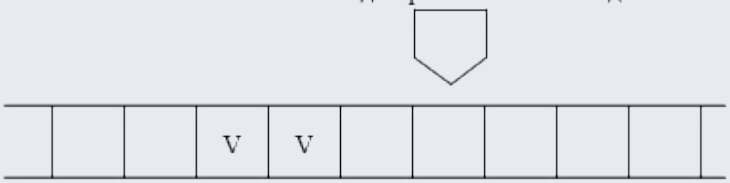
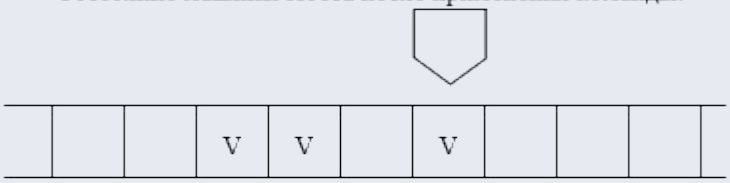
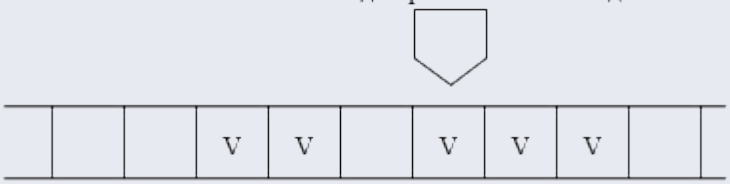
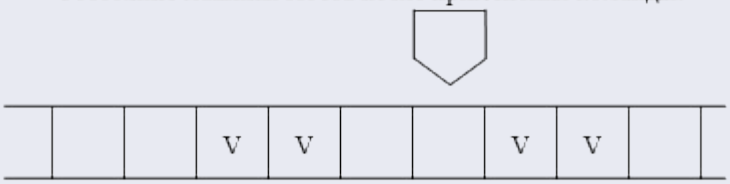
m номер следующей команды, подлежащей выполнению.

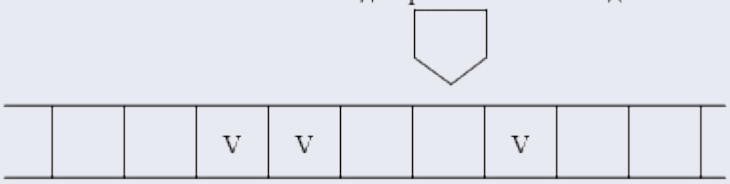
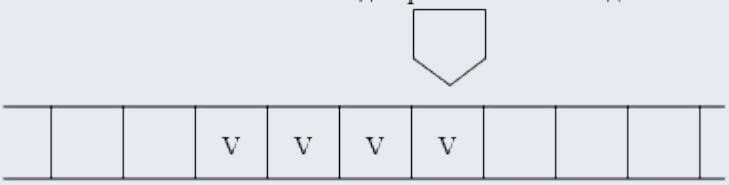
Существует всего шесть действий, выполняемых кареткой машины Поста, а поэтому существует всего шесть команд машины Поста (таблица 6).

Таблица 6

Система команд машины Поста

№	*Команда и состояние машины Поста до и после ее применения*
1	$n \rightarrow m$ *Сдвиг каретки на одну клетку вправо, содержимое ленты не меняется* Состояние машины Поста до применения команды:  Состояние машины Поста после применения команды: 
2	$n \leftarrow m$ *Сдвиг каретки на одну клетку влево, содержимое ленты не меняется* Состояние машины Поста до применения команды:  Состояние машины Поста после применения команды: 

3	*nVm* *В обозреваемую ячейку ленты ставится метка «V». Выполнение этой команды возможно только в том случае, если обозреваемая ячейка пустая, в противном случае команда считается невыполнимой.* Состояние машины Поста до применения команды:  Состояние машины Поста после применения команды: 
4	*nXm* *Каретка стирает метку в обозреваемой ячейки. Выполнение этой команды возможно только в том случае, если обозреваемая ячейка содержит метку, в противном случае команда считается невыполнимой.* Состояние машины Поста до применения команды:  Состояние машины Поста после применения команды: 

5	$n?m1,m2$ *Команда передачи. Проверяется содержимое текущей ячейки, если метки нет, то происходит передача управления команде с номером $m1$, иначе, если метка есть – команде с номером $m2$. Содержимое ленты не меняется.* Состояние машины Поста до применения команды:  Состояние машины Поста после применения команды: 
6	$n![m]$ *Команда останова машины. Содержимое ленты не меняется. У команды останова отсылка не обязательна.* Состояние машины Поста до применения команды:  Состояние машины Поста после применения команды: 

Подчеркнем, что ситуации, в которых каретка должна наносить метку там, где она уже есть, или, наоборот, стирать метку там, где ее нет, являются *аварийными (недопустимыми)*.

Программой для машины Поста будем называть непустой список команд, такой, что: 1) на n-ом месте команда с номером n; 2) номер m каждой команды совпадает с номером какой-либо команды списка.

Работа машины Поста состоит в том, что каретка передвигается вдоль ленты и печатает или стирает метки. Чтобы машина Поста работала, надо задать некоторую программу и некоторое состояние машины (т. е. нужно как-то расставить метки по ячейкам ленты, в частности, можно все секции оставить пустыми и поставить каретку против одной из ячеек).

Работа машины на основании заданной программы происходит следующим образом: машина приступает к выполнению первой команды программы; эта команда выполняется за один шаг, после чего машина приступает к выполнению той команды, номер которой равен отсылке первой команды. Эта команда также выполняется за один шаг, после чего начинается выполнение той команды, номер команды которой равен отсылке предыдущей команды и т. д.

Вообще каждая команда выполняется за один шаг, а переход от выполнения одной команды к выполнению другой происходит по следующему правилу: пусть на k -ом шаге выполнялась команда с номером n, тогда:

1) если эта команда имеет единственную отсылку m, то на $(k+1)$ -ом шаге выполняется команда с номером m;

2) если эта команда имеет две отсылки $m1$ и $m2$, то на $(k+1)$ -ом шаге выполняется одна из двух команд с номером $m1$ или с номером $m2$;

3) если же выполняющаяся на k -ом шаге команда вовсе не имеет отсылки, то на $(k+1)$ -ом шаге и на всех последующих шагах не выполняется никакая команда – машина останавливается.

Возможны следующие *случаи останова* машины Поста:

1) в ходе выполнения программы машина дойдет до выполнения команды остановки; программа в этом случае считается выполненной, машина останавливается происходит *результативная остановка;*

2) в ходе выполнения программы машина дойдет до выполнения невыполнимой команды; выполнение программы прекращается, машина останавливается происходит *безрезультативная остановка.*

Кроме того, возможна программа, при выполнении которой машина не останавливается никогда; в этом, как и в предыдущем случае, мы имеем дело с некорректным алгоритмом (программой).

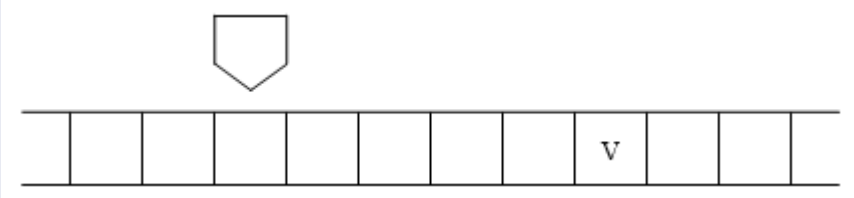
4.3. Примеры типичных программ машины Поста

Пример 1

На ленте проставлена отметка в одной-единственной ячейке. Каретка стоит на некотором расстоянии слева от этой ячейки. Необходимо подвести каретку к ячейке, стереть отметку и остановить каретку слева от нее.

Решение

Исходное состояние машины следующее:



Сначала попробуем описать алгоритм обычным языком. Поскольку нам известно, что каретка стоит напротив пустой ячейки, но неизвестно, сколько шагов нужно совершить до непустой ячейки, мы можем сразу сделать шаг вправо, проверить, заполнена ли ячейка, после чего повторять эти действия до тех пор, пока не наткнемся на заполненную ячейку. Как только мы ее найдем, мы выполним операцию стирания, после чего нужно будет лишь сместить каретку влево и остановить выполнение программы.

Программа для машины Поста:

$1 \rightarrow 2$

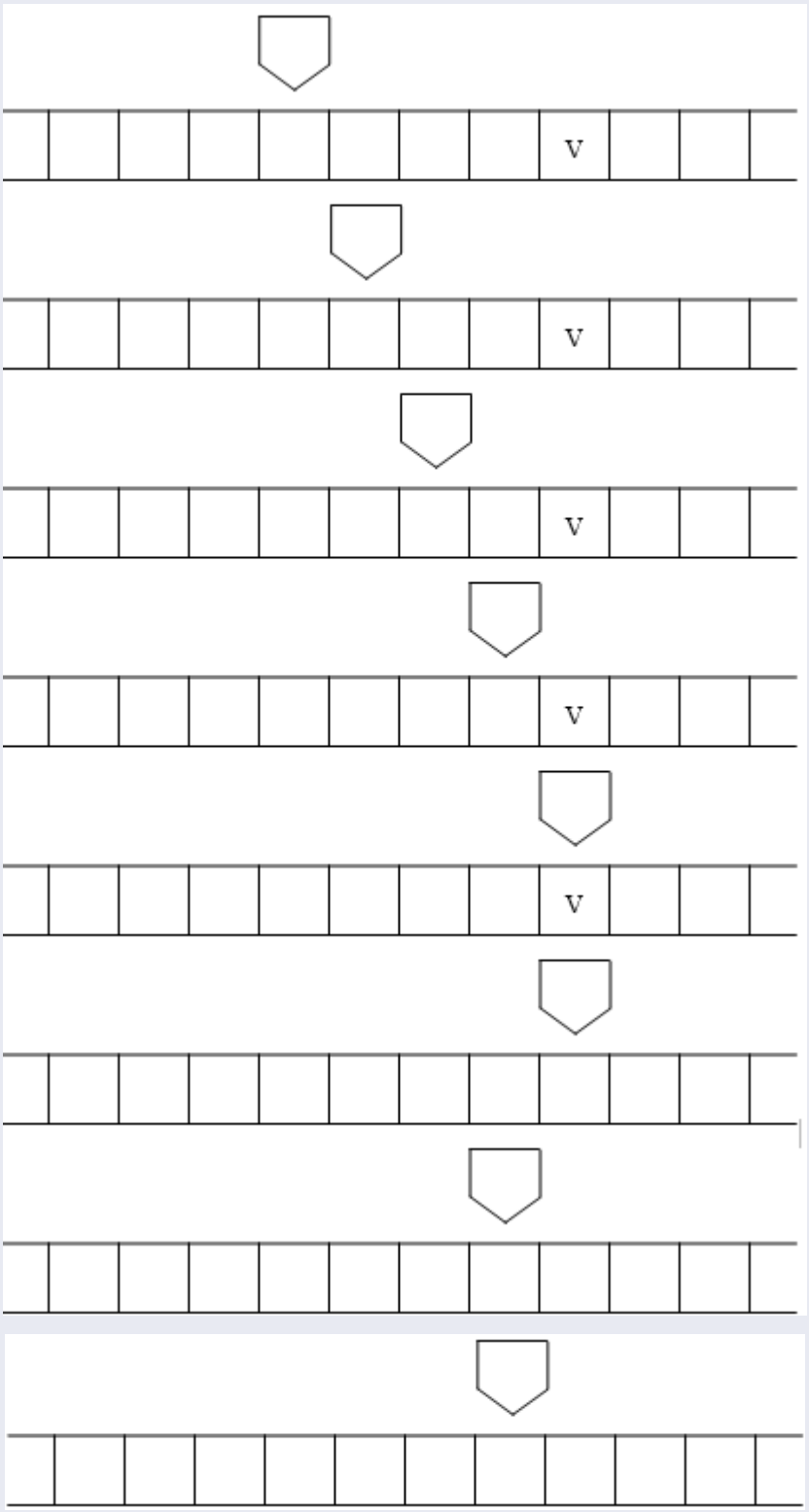
$2\,?1,3$

$3\,X\,4$

$4 \leftarrow 5$

$5!$

Проиллюстрируем состояние машины Поста на каждом шаге ее работы:



Замечание

Команда передачи (условного перехода) $n?m1,m2$ является одним из основных средств организации циклических процессов, например, для нахождения первой метки справа (или слева) от каретки, расположенной над пустой ячейкой; нахождение слева (или справа) от каретки пустой клетки, если она расположена над меткой и т. д.

Остановимся на представлении чисел на ленте машины Поста и выполнении операций над ними.

Число k представляется на ленте машины Поста идущими подряд k метками. Между двумя числами делается интервал, как минимум, из одной пустой ячейки на ленте. Например, запись чисел 3 и 4 на ленте машины Поста будет выглядеть так:

Пример 2

Составим программу для прибавления к произвольному числу единицы. Предположим, что на ленте записано только одно число и головка находится над одной из клеток, в которой находится метка, принадлежащая этому числу:

Решение

Для решения задачи можно перенести каретку влево (или вправо) до первой пустой клетки, а затем нанести метку.

Программа, добавляющая к числу метку слева, имеет вид:

$1 \leftarrow 2$

$2?3,1$

$3V4$

$4!$

Программа, добавляющая к числу метку справа, имеет вид:

$1 \rightarrow 2$

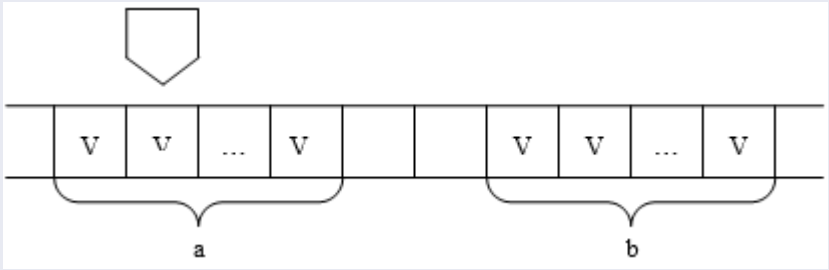
$2?3,1$

$3V4$

$4!$

Пример 3

Приведем программу для сложения двух натуральных чисел a и b. Между числами – несколько неотмеченных клеток. Каретка находится где-то над первым числом:



Решение

Реализуем следующий алгоритм сложения двух чисел: будем двигаться к первой левой метки первого числа, затем первую левую метку сотрем, и будем двигаться к крайней правой метки первого числа, затем припишем недостающую у первого числа метку справа (теперь первое число будет придвинуто ко второму на одну ячейку); так будем действовать до тех пор, пока числа a и b не сольются на ленте машины Поста.

Приведем соответствующую программу машины Поста с кратким комментарием сущности каждой используемой в ней команды:

Команда	*Комментарий к команде*
$1 \leftarrow 2$	Поиск начала первого числа: сдвигаемся влево
2?3,1	до тех пор, пока не встретим неотмеченную ячейку.
$3 \rightarrow 4$	Сдвигаемся вправо, на первую метку первого числа и
$4 X 5$	удаляем ее.
$5 \rightarrow 6$	Ищем конец первого числа: сдвигаемся вправо
6?7,5	пока каретка не встанет на неотмеченную ячейку и
$7 V 8$	ставим метку.
$8 \rightarrow 9$	Проверяем, заполнился ли промежуток между числами:
9?1,10	если не заполнился – переход на первую команду, иначе –
10!	конец.

4.4. Тезис Поста

Тезис Поста

Всякий алгоритм представим в форме машины Поста.

Этот тезис одновременно является формальным определением алгоритма: алгоритм (по Посту) программа для машины Поста, приводящая к решению поставленной задачи.

Тезис Поста является гипотезой. Его невозможно строго доказать (так же, как и тезис Тьюринга), потому что в нем фигурирует, с одной стороны, интуитивное понятие алгоритма, а с другой точное понятие «машина Поста».

Обоснование тезиса Поста сегодня происходит не путем строго математического доказательства, а путем эксперимента действительно, всякий раз, когда указывается какой-либо алгоритм, его можно перевести в форму программы машины Поста, приводящей к тому же результату.

Замечание

Машину Поста, как и машину Тьюринга, можно рассматривать как упрощенную модель ЭВМ. В самом деле, как ЭВМ, так и машина Поста имеют:

– неделимые носители информации (клетки биты), которые могут быть заполненными или незаполненными;

– ограниченный набор элементарных действий команд, каждая из которых выполняется за один такт (шаг).

Обе машины работают на основе программы. Однако в машине Поста информация располагается линейно и читается подряд, а в ЭВМ можно читать информацию по адресу; набор команд ЭВМ значительно шире и выразительнее, чем команды машины Поста и т. д.

Вопросы к главе 4

1. Кем и когда была разработана машина Поста?
2. Сформулируйте идеи, которые легли в основу создания машины Поста.
3. Опишите основные составные части машины Поста.
4. Что называют состоянием машины Поста?
5. Чем состояние машины Поста отличается от состояния машины Тьюринга?
6. Сколько существует основных типов команд машины Поста? Раскройте их суть.
7. При выполнении каких команд машины Поста может произойти аварийная ситуация (остановка)? Как этого избежать?
8. Что представляет собой программа машины Поста?
9. Опишите, как происходит работа машины Поста на основе программы.
10. Какие случаи останова машины Поста возможны? Какие из них являются некорректными?
11. Сформулируйте тезис Поста.
12. Раскройте роль машины Поста в теории алгоритмов

Задания к главе 4

1. Начальное состояние: лента машины Поста пуста. Что будет находиться на ленте в результате работы следующей программы?

$1 V 2$
$2 \to 3$
$3 \leftarrow 1$

2. Известно, что на ленте машины Поста находится метка. Напишите программу, которая находит ее.

3. На ленте имеется массив из n отмеченных ячеек. Каретка обозревает крайнюю левую отметку. Справа от данного массива на расстоянии в m ячеек находится еще одна отметка. Составьте для машины Поста программу, придвигающую данный массив к данной ячейке.

4. На ленте расположены два массива разной длины. Они разделены одной пустой ячейкой. Каретка обозревает крайний элемент одного из них. Составьте программу для машины Поста, сравнивающую длины массивов и стирающую больший из них. Отдельно продумайте случай, когда длины массивов равны.

5. Дан массив из n меток (т. е. идущих подряд отмеченных ячеек). Каретка обозревает крайнюю левую ячейку. Составьте для машины Поста программу, расставляющую эти метки на ленте так, чтобы между каждой парой было по одной пустой ячейке.

6. На ленте машины Поста расположен массив в n ячейках. Необходимо справа от данного массива через одну пустую ячейку разместить массив, вдвое больший (он должен состоять из $2n$ меток). При этом исходный массив может быть стерт.

7. На ленте машины Поста расположен массив из n меток (метки расположены через пробел). Нужно сжать массив так, чтобы все n меток занимали n расположенных подряд ячеек.

8. На ленте машины Поста расположено n массивов меток, отделенных друг от друга свободной ячейкой. Каретка находится над крайней левой меткой первого (левого) массива. Определите количество массивов.

9. На ленте машины Поста расположен массив из n меток. Составьте программу, действуя по которой, машина выяснит, делится ли число n на 3. Если да, то после массива через одну пустую секцию поставьте метку.

10.На ленте задан массив. Вычислите остаток от деления длины заданного массива на 3. Каретка располагается над первой ячейкой массива.

11.На ленте машины Поста расположен массив из $2n-1$ меток. Составьте программу нахождения средней метки массива и стирания ее.

12.На ленте машины Поста расположен массив из $2n$ отмеченных секций. Составьте программу, по которой машина Поста раздвинет на расстояние в 1 секцию две половины данного массива.

13.На ленте машины Поста находятся два массива из m и n меток. Составьте программу выяснения, одинаковы ли массивы по длине.

14.На ленте задан массив меток. Нужно увеличить длину массива в 2 раза. Каретка находится либо слева от массива, либо над одной из ячеек самого массива.

15.На ленте заданы два массива – m и n, $m > n$. Вычислить разность этих массивов. Каретка располагается над левой ячейкой правого массива.

16.На ленте имеется некоторое множество меток (общее количество меток не менее 1). Между метками множества могут быть пропуски, длина которых составляет одну ячейку. Заполнить все пропуски метками.

17.Дан массив меток. Каретка располагается где-то над массивом, но не над крайними метками. Стереть все метки, кроме крайних, и поставить каретку в исходное положение.

18.Дано N массивов меток. Массивы разделены тремя пустыми ячейками. Количество меток в массиве не меньше двух. Если количество меток в массиве кратно трем, то стереть метки в этом массиве через одну, в противном случае стереть весь массив. Каретка находится над крайней левой меткой первого массива.

Глава 5. Машины произвольного доступа

В данной главе рассмотрим такую алгоритмическую модель (уточнение понятия алгоритма), как машины произвольного доступа. Они были предложены достаточно недавно (в 1970-х годах) с целью моделирования реальных вычислительных машин и анализа сложности вычислений.

5.1. Устройство и порядок работы машины произвольного доступа

Машина произвольного доступа (МПД) состоит из бесконечного числа регистров R_1, R_2, R_3…, в каждом из которых может быть записано натуральное число или нуль.

Определение

Пусть r_n – число, записанное в регистре R_n, $n \in N$.

Состоянием машины произвольного доступа или *конфигурацией* назовем последовательность чисел $(r_1, r_2, r_3, \ldots)$.

Функционирование машины произвольного доступа заключается в изменении конфигураций путем выполнения команд в порядке их написания.

Машина произвольного доступа имеет следующие типы *команд*:

1. Команды обнуления. Для всякого $n \in N$ имеется команда $Z(n)$. Действие команды $Z(n)$ заключается в замене содержимого регистра R_n на 0. Содержимое других регистров не меняется. Обозначение действия $Z(n)$-r_n: 0.

2. Команды прибавления единицы. Для всякого $n \in N$ имеется команда $S(n)$. Действие команды $S(n)$ заключается в увеличении содержимого регистра R_n на 1. Содержимое других регистров не меняется. Обозначение действия $S(n)$-r_n: r_n+1.

3. Команды переадресации. Для всех $m,n \in N$ имеется команда $T(m,n)$. Действие команды $T(m,n)$ заключается в замене содержимого регистра R_n числом r_m, хранящемся в регистре R_m. Содержимое других регистров не меняется (включая и регистр R_m). Обозначение действия: $T(m,n)$-r_n: r_m или $r_m \rightarrow R_n$.

4. Команды условного перехода. Для всяких $m,n,q \in N$ имеется команда $J(m,n,q)$. Действие этой команды заключается в следующем: «Сравнивается содержимое регистров R_m и R_n, затем:

- если $r_n = r_m$ то МПД переходит к выполнению команды с номером q в списке команд;
- если $r_n \neq r_m$ то МПД переходит к выполнению следующей команды в списке команд».

Конечная, упорядоченная последовательность команд данных типов составляет *программу МПД*.

Рассмотрим порядок работы машины произвольного доступа.

Пусть зафиксирована начальная конфигурация $K_0(a_1, a_2, \ldots)$ чисел и программа $P = I_1 I_2 \ldots I_s$. Тогда однозначно определена последовательность конфигураций:

$$K_0, K_1, K_2, \ldots, K_t, K_{t+1}, \ldots, \quad (1)$$ где

K_1 есть конфигурация, полученная из K_0 применением команды I_1.

Пусть на некотором шаге выполнена команда I_t и получена конфигурация K_t. Тогда:

если I_t не есть команда условного перехода, следующая конфигурация K_{t+1} есть конфигурация, полученная из K_t применением команды I_{t+1};

если I_t есть команда условного перехода, т. е. $I_t = J(m,n,q)$, то K_{t+1} получается из K_t применением команды I_q, если $r_n = r_m$ в конфигурации K_t и команды I_{t+1}, если $r_n \neq r_m$.

Последовательность (1) будет обозначаться также $P(a_1,a_2,...)$ или $P(K_0)$ и называться *вычислением*.

Вычисление (работа машины) *останавливается* в следующих случаях:

1) выполнена последняя команда, т. е. $t = s$ и I_t не есть команда условного перехода;
2) если $I_t = J(m,n,q)$, $r_n = r_m$ в конфигурации K_t и $q > s$;
3) если $I_t = J(m,n,q)$, $r_n \neq r_m$ в конфигурации K_t и $t = s$.

Если вычисление остановилось, то последовательность $(r_1,r_2,...)$ содержимого регистров $R_1,R_2,...$ называется *заключительной конфигурацией.*

Если последовательность (1) конечна, то говорят, что МПД *применима* к начальной конфигурации $K_0(a_1,a_2,...)$ и пишут $P(a_1,a_2,...)\downarrow$ или $P(K_0)\downarrow$. В противном случае говорят, что МПД *неприменима* к начальной конфигурации $K_0(a_1,a_2,...)$ и пишут $P(a_1,a_2,...)\uparrow$ или $P(K_0)\uparrow$.

Замечание

В дальнейшем будем рассматривать только такие начальные конфигурации, в которых имеется конечное число элементов, отличных от нуля. Будем писать $(a_1,a_2,...,a_n)$ вместо $(a_1,a_2,...,a_n,0,...)$ для таких конфигураций. (Ясно, что для любого t конфигурация K_t будет содержать конечное число отличных от нуля элементов, если этим свойством обладает конфигурация K_0.)

Пример

Разработаем программу МПД, которая к каждому входному натуральному числу x прибавляет 2.

Решение

Идея разработки данной программы очевидна: к входному числу x, содержащемуся в некотором регистре, нужно дважды применить команду прибавления единицы $S(n)$.

Пусть входное число x записано в регистр R_1. Тогда искомая программа будет иметь следующий вид:

1) $S(1)$,

2) $S(1)$.

Проиллюстрируем работу машины на конкретном примере. Пусть $x = 37$, т. е. в регистр R_1 записано число 37 (таблица 7).

Таблица 7

Вычисление МПД искомого числа $x+2$

Номер команды	*Команда*	*Содержимое регистра* R_1
		37
1	*S(1)*	*38*
2	*S(1)*	*39*

Остановка работы машины будет происходить после выполнения последней (т. е. второй) команды, которая не является командой условного перехода.

5.2. Вычисление функций на машине произвольного доступа

Условимся, что будем понимать под вычислением функций на МПД.

Определение

Пусть $P = I_1I_2...I_s$ – фиксированная программа. Пусть $a_1,a_2,...,a_n,b \in N_0$.

Будем говорить, что *вычисление дает результат* b, если $P(a_1,a_2,...)\downarrow$ и в заключительной конфигурации $r_1 = b$. Обозначение: $P(a_1,a_2,...)\downarrow b$.

Будем говорить, что *программа* P *вычисляет функцию* f, если $\forall\, a_1,a_2,...,a_n,b \in N_0$ выполнимо $P(a_1,a_2,...)\downarrow b$ (f определена на $(a_1,a_2,...,a_n)$ и $f(a_1,a_2,...,a_n) = b$).

Назовем *функцию* f *вычислимой на МПД*, если существует программа P, которая вычисляет f.

Замечание

1. В определении рассматриваются частичные функции f: $N_0^n \to N_0$. $(N_0 = N \cup \{0\})$.

2. Любая программа P для любого $n \geq 1$ на начальных конфигурациях $K_0(a_1,a_2,...,a_n,0,...)$ определяет n-местную частичную функцию $f_p^n(x_1,...,x_n)$ такую, что $\forall\, a_1,...,a_n \in N_0$:

$$f_p^n(x_1,...,x_n) = \begin{cases} b, & \text{если } P(a_1,a_2,...)\downarrow \text{ и } P(a_1,a_2,...)\downarrow b, \\ \text{неопределена}, & \text{если } P(a_1,a_2,...)\uparrow . \end{cases}$$

3. Ясно, что разные программы могут вычислять одну и ту же функцию.

Пример 1

Программа МПД, разработанная в примере § 1:

1) $S(1)$,

2) $S(1)$,

вычисляет функцию $f(x) = x+2$, где $x \in N$.

Пример 2

Разработаем программу МПД, которая вычисляет функцию $f(x,y) = x+y$.

Решение

Условимся, что начальной конфигурацией МПД будет выступать последовательность чисел $(x,y,0,0,...)$, т. е. в регистре R_1 будет записано число x, в регистре R_2 будет записано число y, а в остальных регистрах R_3, R_4, … будет содержаться 0.

На данном этапе ясно, что в программе будут использоваться как минимум два регистра R_1 и R_2;

понадобятся ли нам еще регистры и какое количество – будет выяснено в ходе разработки программы.

Для того чтобы составить искомую программу МПД, нужно догадаться, как используя ее команды (команду обнуления, команду прибавления единицы, команду переадресации, команду условного перехода), можно найти сумму двух чисел x и y. Оказывается, сумму двух произвольных чисел можно найти, используя команду прибавления единицы. Действительно, для того чтобы найти сумму двух чисел x и y, нужно к числу x прибавить y раз единицу. Воспользуемся этим положением для составления искомой программы.

Кроме того, опишем назначение используемых регистров:

R_1 – предназначено для записи числа x, кроме того именно в этом регистре будет «накапливаться» искомая сумма;

R_2 – предназначено для записи числа y;

R_3 – предназначено для подсчета количества единиц, прибавленных к числу x.

Остановка программы должна происходить, когда в регистре R_1 будет записана искомая сумма $x+y$, т. е. когда к числу x будет прибавлено y единиц, а содержимое регистра R_3 будет равняться содержимому регистра R_2.

Составим словесный алгоритм нахождения искомой суммы:

1) сравниваем содержимое регистров R_3 с содержимым регистра R_2: если равно, то остановка МПД, если неравно, то переход к п. 2;

2) прибавляем к содержимому регистра R_1 единицу;

3) прибавляем к содержимому регистра R_3 единицу;

4) возвращаемся к п. 1.

Соответствующая программа МПД может быть, например, следующей:

1) $J(3,2,5)$,

2) $S(1)$,

3) $S(3)$,

4) $J(1,1,1)$.

Проиллюстрируем работу МПД на конкретном примере. Пусть $x = 5$, $y = 3$, т. е. в регистре R_1 записано число 5, а в регистре R_2 – число 3 (таблица 8).

Таблица 8

Вычисление МПД искомого числа $x+y$

Номер команды	*Команда*	*Содержимое регистров*		
		R_1	R_2	R_3
		5	*3*	*0*
1	*J(3,2,5)*	*5*	*3*	*0*
2	*S(1)*	*6*	*3*	*0*
3	*S(3)*	*6*	*3*	*1*
4	*J(1,1,1)*	*6*	*3*	*1*
1	*J(3,2,5)*	*6*	*3*	*1*
2	*S(1)*	*7*	*3*	*1*
3	*S(3)*	*7*	*3*	*2*
4	*J(1,1,1)*	*7*	*3*	*2*
1	*J(3,2,5)*	*7*	*3*	*2*
2	*S(1)*	*8*	*3*	*2*
3	*S(3)*	*8*	*3*	*3*
4	*J(1,1,1)*	*8*	*3*	*3*
1	*J(3,2,5)*	*8*	*3*	*3*
5	–			

Из таблицы видно, что как только в регистре *R1* будет «накоплена» искомая сумма $x+y$, машина перейдет к выполнению команды 5. А так как команды с таким номером в программе МПД нет, то машина остановится.

5.3. Композиция программ машин произвольного доступа

Поскольку доказательства вычислимости конкретных функций на МПД связаны с предъявлением конкретных программ, их вычисляющих, то следует ввести некоторые соглашения о составлении и записи программ на МПД. Аналогично композиции машин Тьюринга можно ввести композицию программ МПД.

Пусть программа $P = I_1I_2...I_s$. Будем говорить, что *программа P имеет стандартный вид*, если для всякой команды условного перехода $J(m,n,q)$ выполнимо $q \le s+1$.

Две *программы* P и P' назовем *эквивалентными*, если они определяют одни и те же n-местные функции, т. е. $f_P^n = f_{P'}^n$ для всех $n > 0$.

Теорема

Для всякой программы P существует эквивалентная ей программа стандартного вида P'.

Доказательство

Пусть $P = I_1I_2...I_s$. Тогда определим $P' = I'_1I'_2...I'_s$, где $\forall\ k \in \{1,...,s\}$:

$$I'_k = \begin{cases} I_k, & \text{если } I_k \text{ не есть команда условного перехода}, \\ I_k, & \text{если } I_k = J(m,n,q) \text{ и } q \le s+1, \\ J(m,n,s+1), & \text{если } I_k = J(m,n,q) \text{ и } q > s+1. \end{cases}$$

Ясно, что P' удовлетворяет нужным требованиям, что и требовалось доказать.

Пусть теперь даны две программы P и Q стандартного вида. Образуем программу $PQ = I_1I_2...I_sI_{s+1}...I_{s+t}$, где $P = I_1I_2...I_s$, $Q = I_{s+1}I_{s+2}...I_{s+t}$ с учетом нумерации, т. е. команды $J(m,n,q)$ заменены на $J(m,n,s+q)$. Тогда результат действия программы PQ совпадает с результатом вычисления по программе P, к которому применена программа Q.

Заметим, что для всякой программы P существует минимальное натуральное число $r(P)$, такое, что для всех $m,n \in N$, входящих в команды из P, т. е. $Z(n)$, $S(n)$, $T(m,n)$, $J(m,n,q)$ выполнено $m,n < r(P)$. Это число $r(P)$ называют *шириной* (или *рангом*) *программы* P.

Смысл ранга $r(P)$ состоит в том, что регистры R_t, где $t > r(p)$, в ходе вычисления по программе P не будут менять свое содержание и не будут влиять на содержимое регистров R_1, R_2, …, R_r, поэтому их можно использовать для других вычислений.

Заметим также, что можно организовать вычисление, используя программу P, в случае, когда входы программы находятся в регистрах R_{l1}, R_{l2}, …, R_{ln}, а результат заносится в регистр R_l. Пусть программа P вычисляет функцию f в стандартном понимании вычислимости. Тогда следующая программа $P[l1,...,ln \to l]$ будет вычислять $f(x_{l1},x_{l2},...,x_{ln})$ и результат запишет в R_l:

$T(l1,1)$

...

$T(ln,n)$

$Z(n+1)$

...

$Z(r(P))$

P

$T(1,l)$

Далее, будем считать, что регистры R_{l1}, R_{l2}, …, R_{ln} отличны от R_1, R_2, …, R_n.

Пример

Покажем, что следующая программа P вычисляет функцию $f(x,y) = xy$:

1) $J(2,3,6)$,

2) $S(3)$,

3) $H[1,4 \rightarrow 5]$, где H – программа, вычисляющая функцию $f(x,y) = x+y$ (*см. пример 2 из 5.2*)

4) $T(5,4)$,

5) $J(1,1,1)$,

6) $T(5,1)$.

Здесь начальной конфигурацией МПД выступает последовательность чисел $(x,y,0,0,...)$.

Решение

Прежде всего, отметим следующее:

«рабочими» регистрами в данной программе машины являются только регистры R_1,R_2,R_3,R_4,R_5;

запись $H[1,4 \rightarrow 5]$ означает, что будет вычисляться сумма чисел, записанных в регистрах R_1 и R_4, а найденная сумма будет помещаться в регистр R_5.

Для того чтобы понять принцип действия программы МПД, рассмотрим ее работу на конкретном примере. Пусть $x = 7$, $y = 3$, т. е. в регистре R_1 записано число 7, а в регистре R_2 – число 3 (таблица 9).

Таблица 9

Вычисление МПД искомого числа *xy*

Номер команды	*Команда*	*Содержимое регистров*				
		R_1	R_2	R_3	R_4	R_5
		7	*3*	*0*	*0*	*0*
1	*J(2,3,6)*	*7*	*3*	*0*	*0*	*0*
2	*S(3)*	*7*	*3*	*1*	*0*	*0*
3	*H[1,4 → 5]*	*7*	*3*	*1*	*0*	*7*
4	*T(5,4)*	*7*	*3*	*1*	*7*	*7*
5	*J(1,1,1)*	*7*	*3*	*1*	*7*	*7*
1	*J(2,3,6)*	*7*	*3*	*1*	*7*	*7*
2	*S(3)*	*7*	*3*	*2*	*7*	*7*
3	*H[1,4 → 5]*	*7*	*3*	*2*	*7*	*14*
4	*T(5,4)*	*7*	*3*	*2*	*14*	*14*
5	*J(1,1,1)*	*7*	*3*	*2*	*14*	*14*
1	*J(2,3,6)*	*7*	*3*	*2*	*14*	*14*
2	*S(3)*	*7*	*3*	*3*	*14*	*14*
3	*H[1,4 → 5]*	*7*	*3*	*3*	*14*	*21*
4	*T(5,4)*	*7*	*3*	*3*	*21*	*21*
5	*J(1,1,1)*	*7*	*3*	*3*	*21*	*21*
1	*J(2,3,6)*	*7*	*3*	*3*	*21*	*21*
6	*T(5,1)*	*21*	*3*	*3*	*21*	*21*

Из таблицы становится понятен принцип работы данной программы МПД: произведение двух чисел *x* и *y* находится через операцию суммирования, а именно число *x* суммируется *y* раз. Например, в случае, когда *x=7*, *y=3*, имеем:

$$7 \cdot 3 = 7+7+7 = 14+7 = 21.$$

Опишем теперь назначение используемых в программе регистров:

R_1 – предназначено для записи числа *x*, кроме того именно в этот регистр будет помещено искомое произведение *xy*;

R_2 – предназначено для записи числа y;

R_3 – предназначено для записи количества суммируемых чисел x;

R_4 – предназначено для записи очередного слагаемого для следующего суммирования с числом x;

R_5 – предназначено для записи очередного ответа суммирования.

Остановка программы происходит после выполнения последней (шестой) команды, которая не является командой условного перехода.

Замечание

Как следует из вышеизложенного, язык программ МПД содержит основные процедуры языков программирования и позволяет устраивать композицию (соединение) программ и использовать программы в качестве подпрограмм других программ. Это является основанием для предположения о том, что введенный класс вычислимых функций в точности отвечает классу алгоритмически вычислимых функций.

Данное предположение называется *тезисом Черча (для МПД)*. Как и тезисы Тьюринга, Поста, данный тезис доказать нельзя, однако принятие его позволяет истолковывать утверждения о несуществовании алгоритмов вообще.

Вопросы к главе 5

1. Когда была разработана МПД?
2. Раскройте назначение МПД.
3. Из каких элементов состоит МПД?
4. Что называется состоянием (конфигурацией) МПД?
5. Сколько типов команд МПД существует? Раскройте их суть.
6. Что называется программой МПД? Приведите пример программы МПД.
7. Опишите порядок работы МПД.
8. Когда происходит остановка работы МПД?
9. Что называется заключительной конфигурацией МПД?
10. Что означают слова «МПД применима к начальной конфигурации $(a_1, a_2, \ldots)$», «МПД неприменима к начальной конфигурации $(a_1, a_2, \ldots)$»?
11. Какая функция называется вычислимой на МПД?
12. Что означают слова «программа МПД имеет стандартный вид»?
13. Какие программы МПД называются эквивалентными? Приведите пример эквивалентных программ МПД.
14. Сформулируйте теорему о существовании для любой программы МПД эквивалентной ей программы.
15. Опишите суть композиции (соединения) двух программ МПД.
16. Что называется рангом программы МПД?
17. Что означает запись $P[l1, \ldots, ln \rightarrow l]$, где P – программа МПД?
18. Сформулируйте тезис Черча для МПД и раскройте его роль в теории алгоритмов.

Задания к главе 5

1. Разработайте программу МПД, которая вычисляет функцию $f(x)=\frac{x}{2}$, где x – четное натуральное число.

2. Запишите представленную программу МПД, вычисляющую функцию *f(x,y) = xy*, без использования в ней операции композиции машин МПД (т. е. без *H[1,4 → 5]*):

1) *J(2,3,6)*,

2) *S(3)*,

3) *H[1,4 → 5]*, где *H* – программа, вычисляющая функцию *f(x,y) = x+y* (*см. пример 2 из 5.2*)

4) *T(5,4)*,

5) *J(1,1,1)*,

6) *T(5,1)*.

3. Разработайте программу МПД, которая вычисляет следующие функции:

а) $f(x)=x+5$;

б) $f(x)=3x+5$;

в) $f(x)=x\dot{-}1=\begin{cases}0, \text{если } x\le 1\\ x-y, \text{если } x>1\end{cases}$;

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

д) $\overline{sg}(x)=\begin{cases}1, \text{если } x=0\\ 0, \text{если } x>0\end{cases}$;

е) $f(x,y)=I_2^2(x,y)+1$;

ё) $f(x,y,z)=I_2^3(x,y,z)\dot{-}1$;

ж) $f(x,y)=\max(x,y)=\begin{cases}x, \text{если } x>y\\ y, \text{если } x\le y\end{cases}$;

з) $f(x)=x!$

4. Покажите, что для каждой команды переадресации существует программа без команд переадресации, которая на всякой конфигурации МПД дает тот же результат, что и T(m, n). Это означает, что команды

переадресации на самом деле избыточны в нашем определении МПД. Тем не менее, представляется естественным и удобным иметь такие команды, облегчающие построение алгоритмов.

Глава 6. Нормальные алгоритмы Маркова

В данной главе рассмотрим еще один подход к уточнению понятия алгоритма, предложенный российским математиком А.А. Марковым (1903–1979) в конце 1940-х начале 1950-х годов, – *нормальные алгоритмы* (в авторской транскрипции – алгорифмы).

Данный подход связывает понятие алгоритма с переработкой слов в некотором алфавите в соответствии с определенными правилами. В качестве элементарного преобразования используется подстановка одного слова вместо другого. Множество таких подстановок определяют схему алгоритма. Правила, определяющие порядок применения подстановок, а также правила останова являются общими для всех нормальных алгоритмов.

6.1. Марковские подстановки

Алфавитом будем называть произвольный конечный набор различных символов. Составляющие его символы будем называть *буквами,* а любую конечную последовательность букв алфавита – *словом* в этом алфавите. Допускаются пустые слова (не содержащие ни одной буквы): *пустое слово* обозначают .

Одно слово может быть составной частью другого слова. Тогда первое называется *подсловом* второго или *вхождением* во второе.

Пример 1

$A = \{а,б,в,г,д,...,э,ю,я\}$ – алфавит.

α = параграф, β = граф, γ = ра – слова в алфавите A,

β – это подслово α,

γ – это подслово α и β, причем в α слово γ входит дважды.

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 *Q*».

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*.

Пример 2

Пусть *(78,0)* – марковская подстановка. Определим результат ее применения к словам *96747878*, *100078*, *78*.

Решение

9674<u>78</u>78 => 967407<u>8</u> => 967400,

100<u>78</u> => 10000,

<u>78</u> => 0.

Замечание

Частными случаями марковских подстановок являются подстановки с пустыми словами: *(,Q)*, *(P,)*, *(,)*. Их действие рассмотрим на конкретном примере.

Пусть есть марковские подстановки *(,д)*, *(у,)*, *(,)*. Определим результат их применения к слову

«удочка»:

$\_$*удочка* $\overset{(\wedge,\partial)}{\Rightarrow}$ *дудочка,*

удочка $\overset{(y,\wedge)}{\Rightarrow}$ *дочка,*

$\_$*удочка* $\overset{(\wedge,\wedge)}{\Rightarrow}$ *удочка.*

Для обозначения марковской подстановки *(P,Q)* используется запись $P \to Q$. Она называется *формулой подстановки (P,Q)*. Слово P называется левой частью, а Q – правой частью в формуле подстановки.

Подстановки *(P,Q)*, приводящие к остановке нормального алгоритма, называют заключительными. Для обозначения таких подстановок будем использовать запись $P \to .Q$, называя ее *формулой заключительной подстановки*.

(Здесь предполагается, что символы «→», «.» не входят в алфавит A входных слов.)

6.2. Нормальные алгоритмы и их применение к словам

Определение 1

Упорядоченный конечный список формул подстановок в алфавите A называется *схемой нормального алгоритма* и обозначается S_α.

Схема нормального алгоритма имеет вид:

S_α:	$P_1 \rightarrow (.)Q_1$
	$P_2 \rightarrow (.)Q_2$
	…….
	$P_r \rightarrow (.)Q_r$

(Запись *(.)* означает, что она может стоять в этом месте, а может отсутствовать.)

Данная схема S_α определяет алгоритм α, перерабатывающий слова в алфавите A. Этот алгоритм преобразования слов называют нормальным алгоритмом Маркова.

Определение 2

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 упомянутой последовательности обрывается, то говорят, что рассматриваемый *нормальный алгоритм применим к слову* V. Последний член W последовательности называется *результатом применения нормального алгоритма к слову* V.

Последовательность V_i будем записывать так: $V_0 => V_1 => V_2 = \ldots => V_{m-1} => V_m$, где $V_0 = V$, $V_m = W$.

Замечание 1

Используют следующие обозначения:

$S_\alpha : V$ (запись означает, что схема нормального алгоритма S_α неприменима к слову V);

$S_\alpha : V => W$ (запись означает, что схема нормального алгоритма S_α применима к слову V и результатом ее

применения является слово W).

Замечание 2

Мы определили понятие нормального алгоритма в алфавите A. Если же алгоритм задан в некотором расширении алфавита A, то говорят, что он есть *нормальный алгоритм над A*. (Алфавит B называется расширением алфавита A, если B A.)

Пример 1

Пусть $A = \{a_0, a_1, a_2 \ldots, a_n\}$ – алфавит.

Рассмотрим схему нормального алгоритма:

S_α:	$a_0 \rightarrow$
	$a_1 \rightarrow$
	…….
	$a_{n-1} \rightarrow$
	$\rightarrow$.

Определим результат применения данного нормального алгоритма к слову $a_1a_2a_1a_3a_0$.

Решение

$a_1a_2a_1a_3a_0 => a_1a_2a_1a_3 => a_2a_1a_3 = a_2a_3 => a_3 =>$ $=>$.

Данный нормальный алгоритм перерабатывает всякое слово (в алфавите A) в пустое слово.

Пример 2

Пусть $A = \{a_1, a_2\}$ – алфавит.

Рассмотрим схему нормального алгоритма:

S_α:	$a_1 \rightarrow$.
	$a_2 \rightarrow a_2$

Определим результат применения данного нормального алгоритма к словам, составленным из алфавита A.

Решение

Всякое слово P в алфавите A данный нормальный алгоритм переводит в слово Q, полученное из P путем вычеркивания первого вхождения буквы a_1. Данный нормальный алгоритм неприменим к словам, не содержащим вхождений буквы a_1.

Пример 3

Пусть $A = \{a_1, a_2, \ldots, a_n\}$ – алфавит.

Рассмотрим алфавит $B = A \quad \{\gamma, \beta\}$, $\gamma, \beta \notin A$, и соответственно схему нормального алгоритма S_α:

1. $\gamma\gamma \to \beta$

2. $\beta a \to a\beta$ для любого a из алфавита A

3. $\beta\gamma \to \beta$

4. $\beta \to .$

5. $\gamma ab \to b\gamma a$ для любых элементов a и b из алфавита A

6. $\quad \to \gamma$

Покажем, что данный алгоритм осуществляет обращение слов в алфавите A. (Обращением слова $P = a_{i0} \ldots a_{ik}$ назовем слово $P^* = a_{ik} \ldots a_{i0}$.)

Решение

Пусть $P = a_{j0} \ldots a_{jk}$ произвольное слово в алфавите A. Тогда:

$$P \overset{6}{\Rightarrow} \gamma P \overset{5}{\Rightarrow} a_{j1}\gamma a_{j0} \ldots a_{jk} \overset{5}{\Rightarrow} a_{j1}a_{j2}\gamma a_{j0}\, a_{j3} \ldots a_{jk} \overset{6}{\Rightarrow} a_{j1}a_{j2} \ldots a_{jk}\gamma a_{j0} \Rightarrow$$

$$\Rightarrow \gamma a_{j1}a_{j2} \ldots a_{jk}\gamma a_{j0} \overset{5}{\Rightarrow} \ldots \overset{5}{\Rightarrow} a_{j2}a_{j3} \ldots a_{jk}\gamma a_{j1}\gamma a_{j0}.$$

Далее, повторяя этот процесс получим:

$$\gamma a_{jk}\gamma a_{jk-1} \ldots \gamma a_{j1}\gamma a_{j0} \overset{6}{\Rightarrow} \gamma\gamma a_{jk}\gamma a_{jk-1} \ldots \gamma a_{j1}\gamma a_{j0} \overset{1}{\Rightarrow} \beta a_{jk}\gamma a_{jk-1} \ldots \gamma a_{j1}\gamma a_{j0} \Rightarrow$$

$$\Rightarrow a_{jk}\beta\gamma a_{jk-1} \ldots \gamma a_{j1}\gamma a_{j0} \overset{3}{\Rightarrow} a_{jk}\beta a_{jk-1} \ldots \gamma a_{j1}\gamma a_{j0} \Rightarrow \ldots \overset{4}{\Rightarrow} a_{jk}a_{jk-1} \ldots a_{j1}a_{j0}\beta \Rightarrow$$

$\Rightarrow .a_{jk}a_{jk-1} \ldots a_{j1}a_{j0}$, что и требовалось показать.

6.3. Нормально вычислимые функции

Определение

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

Пример 1

Пусть $A = \{a_0,a_1,a_2,\ldots,a_n\}$ – алфавит, схема нормального алгоритма S_α: $\rightarrow$ . . Данный алгоритм вычисляет функцию $f(P) = P$ для любого P.

Пример 2

В алфавите $A = \{1\}$ схема S_α: $\rightarrow 1$ определяет нормальный алгоритм, который к каждому слову в алфавите A приписывает слева 1. Следовательно, алгоритм вычисляет функцию $f(x) = x+1$ (в унарной системе счисления).

Пример 3

Составим нормальный алгоритм, вычисляющий значения функции $f(x) = x+1$ (x – целое положительное число) в десятичной системе счисления.

Решение

Определим алфавит A: $A = \{0,1,2,3,4,5,6,7,8,9,+\}$.

Условимся обозначать входное слово следующим образом: «$123+1$». Ясно, что на выходе нормального алгоритма в этом случае должно получиться число «124».

Рассмотрим два случая: 1) разряд единиц у числа x есть цифра из множества $\{0,1,2,3,4,5,6,7,8\}$ (иными словами, в этом случае при прибавлении 1 к x не происходит переполнение разряда единиц у числа x); 2) разряд единиц у числа x есть цифра 9 (иными словами, в этом случае при прибавлении 1 к x происходит переполнение разряда единиц у числа x, а поэтому нужно обращаться к разряду десятков).

Случай 1

Очевидно, что для нахождения суммы чисел x и 1, необходимо задать следующую схему нормального алгоритма S_α:

$0+1 \rightarrow 1$

$1+1 \rightarrow 2$

$2+1 \rightarrow 3$

$3+1 \rightarrow 4$

$4+1 \rightarrow 5$

5+1 → 6

6+1 → 7

7+1 → 8

8+1 → 9

→ . (данная подстановка обеспечивает остановку алгоритма).

Случай 2

Очевидно, что если в разряде единиц у искомой суммы будет стоять цифра «9», то разряд десятков нужно увеличить на *1*. Этого можно достичь, применив к входному слову следующую подстановку:

9+1 → +10

В результате, далее, нужно будет увеличить на *1* разряд десятков у исходного числа *x*. Этого можно добиться, применяя марковские подстановки, указанные в первом случае. Если при этом окажется, что увеличиваемый разряд числа *x* равен нулю (он отсутствует), то на его месте нужно написать *1* и остановить алгоритм. Это позволяет обеспечить следующая подстановка:

+1 → .1

Рассмотренные выше случаи исходного числа *x* позволяют записать схему искомого нормального алгоритма S_α:

1. 0+1 → 1	*5. 4+1 → 5*	*9. 8+1 → 9*
2. 1+1 → 2	*6. 5+1 → 6*	*10.9+1 → +10*
3. 2+1 → 3	*7. 6+1 → 7*	*11.+1 → .1*
4. 3+1 → 4	*8. 7+1 → 8*	*12. → .*

Проиллюстрируем работу приведенного нормального алгоритма на нескольких примерах.

1) *9+1 → +10 → .10*

2) *37+1 → _38 → .38*

3) *123+1 → _124 → .124*

4) *999+1 → 99+10 → 9+100 → +1000 → .1000*

5) *7899+1 → 789+10 → 78+100 → _7900 → .7900*

Пример 4

Составим нормальный алгоритм, вычисляющий значения функции *f(x) = 2x* (*x* – целое положительное число) в десятичной системе счисления.

Решение

Определим алфавит *A*: *A = {0,1,2,3,4,5,6,7,8,9,*}*.

Условимся обозначать входное слово следующим образом: «*23*2*». Ясно, что на выходе нормального

алгоритма в этом случае должно получиться число «*46*».

Для получения искомого произведения (также как и предыдущем примере) будем двигаться справо налево, от разряда единиц в исходном числе *x* к более высоким разрядам, умножая при этом цифры разрядов на *2*. При этом движении необходимо учесть два момента: 1) для продолжения умножения на *2* более высоких разрядов нужно смещать вправо символы «**2*»; 2) в случае, когда происходит переполнение разряда при умножении необходимо запоминать «в уме» *1* (это происходит в следующих случаях: *5*2 = 10, 6*2 = 12, 7*2 = 14, 8*2 = 16, 9*2 = 18*); эту «особенную» единицу мы будем фиксировать в ходе преобразований символом *a*.

С учетом вышесказанного запишем марковские подстановки, которые необходимо применить к разряду единиц исходного числа *x*.

*0*2 → *20* (правая часть этой подстановки означает, что в разряде единиц остается *0*, а далее будет происходить умножение на *2* разряда десятков.)

*1*2 → *22* (правая часть этой подстановки означает, что в разряде единиц остается *2*, а далее будет происходить умножение на *2* разряда десятков.)

*2*2 → *24* (правая часть этой подстановки означает, что в разряде единиц остается *4*, а далее будет происходить умножение на *2* разряда десятков.)

*3*2 → *26*

*4*2 → *28*

*5*2 → *2a0* (правая часть этой подстановки означает, что в разряде единиц остается *0*, далее символ *a* означает, что была запомнена *1*, которую нужно будет учесть при дальнейшем умножении на *2* разряда десятков.)

*6*2 → *2a2* (правая часть этой подстановки означает, что в разряде единиц остается *2*, далее символ *a* означает, что была запомнена *1*, которую нужно будет учесть при дальнейшем умножении на *2* разряда десятков.)

*7*2 → *2a4* (правая часть этой подстановки означает, что в разряде единиц остается *4*, далее символ, *a* означает, что была запомнена *1*, которую нужно будет учесть при дальнейшем умножении на *2* разряда десятков.)

*8*2 → *2a6*

*9*2 → *2a8*

Далее, запишем подстановки, которые необходимо применить к разряду десятков исходного числа *x*.

*0*2a → *21* (здесь к результату умножения *0* на *2*, т. е. к *0*, была прибавлена запомненная ранее *1*.)

*1*2a → *23* (здесь к результату умножения *1* на *2*, т. е. к *2*, была прибавлена запомненная ранее *1*.)

*2*2a → *25* (здесь к результату умножения *2* на *2*, т. е. к *4*, была прибавлена запомненная ранее *1*.)

*3*2a → *27*

*4*2a → *29*

*5*2a → *2a1*

*6*2a → *2a3*

*7*2a → *2a5*

*8*2a → *2a7*

*9*2a → *2a9*

Заметим, что вышеуказанные подстановки будут действовать нужным нам образом и на более высоких разрядах исходного числа (разрядах сотен, тысяч и т. д.), если таковые будут иметься.

Наконец, определим марковские подстановки, которые обеспечат остановку алгоритма:

**2a → . 1* **2 → .*

Итак, запишем схему искомого нормального алгоритма S_α.

1. *0*2a → *21*
2. *1*2a → *23*
3. *2*2a → *25*
4. *3*2a → *27*
5. *4*2a → *29*
6. *5*2a → *2a1*
7. *6*2a → *2a3*
8. *7*2a → *2a5*
9. *8*2a → *2a7*
10. *9*2a → *2a9*
11. *0*2 → *20*
12. *1*2 → *22*
13. *2*2 → *24*
14. *3*2 → *26*
15. *4*2 → *28*
16. *5*2 → *2a0*
17. *6*2 → *2a2*
18. *7*2 → *2a4*
19. *8*2 → *2a6*
20. *9*2 → *2a8*
21. **2a → . 1*
22. **2 → .*

Приведенный нормальный алгоритм нахождения значений функции $f(x) = 2x$ в десятичной системе счисления станет более понятным, если рассмотреть несколько конкретных примеров:

1) *0*2 → *20 → .0*

2) *1*2 → *22 → .2*

3) *5*2 → *2a0 → .10*

4) *10*2 → 1*20 → *220 → .20*

5) *16*2 → 1*2a2 → *232 → .32*

6) *99*2 → 9*2a8 → *2a98 → .198*

7) *100*2 → 10*20 → 1*200 → *2200 → .200*

8) *489*2 → 48*2a8 → 4*2a78 → *2978 → .978*

9) *92589*2 → 9258*2a8 → 925*2a78 → 92*2a178 → 9*25178 → → *2a85178 → .185178*

6.4. Способы сочетания нормальных алгоритмов

Способы сочетания нормальных алгоритмов позволяют строить новые алгоритмы из уже известных. Перечислим способы сочетания нормальных алгоритмов.

1. Суперпозиция алгоритмов. При суперпозиции двух алгоритмов A и B выходное слово первого алгоритма рассматривается как входное слово второго алгоритма B, результат суперпозиции C может быть представлен в виде $C(P) = B(A(P))$.

2. Объединение алгоритмов. Объединение алгоритмов A и B в одном и том же алфавите называется алгоритм C в том же алфавите, преобразующий любое слово P, содержащееся в пересечении областей определения алгоритмов A и B, в записанные рядом слова $A(P)$ и $B(P)$.

3. Разветвление алгоритмов. Разветвление алгоритмов представляет собой композицию D трех алгоритмов A, B и C, причем область определения алгоритма D является пересечением областей определения всех трех алгоритмов A, B и C, а для любого слова P из этого пересечения $D(P) = A(P)$, если $C(P) =$; $D(P) = B(P)$, если $C(P) \neq$, где – пустая строка.

4. Итерация алгоритмов. Итерация (повторение) представляет собой такую композицию C двух алгоритмов A и B, что для любого входного слова P соответствующее слово $C(P)$ получается в результате последовательного многократного применения алгоритма A до тех пор, пока не получится слово, преобразуемое алгоритмом B.

Пример

Составим нормальный алгоритм, вычисляющий значения функции $f(x) = 2x+1$ (x – целое положительное число) в десятичной системе счисления.

Решение

В предыдущем параграфе нами были составлены нормальные алгоритмы, вычисляющие значения функций $f(x) = 2x$ и $f(x) = x+1$, (x – целое положительное число) в десятичной системе счисления. Воспользуемся ими для составления искомого нормального алгоритма.

Определим алфавит A: $A = \{0,1,2,3,4,5,6,7,8,9,*,+\}$.

Условимся обозначать входное слово следующим образом: «*23*2+1*». Ясно, что на выходе нормального алгоритма в этом случае должно получиться число «*47*».

Очевидно, что для получения выходного слова, нужно сначала применить к входному слову x нормальный алгоритм, вычисляющий значение $2x$, далее нужно применить к слову $2x$ нормальный алгоритм, прибавляющий *1*. Для этого необходимо, чтобы после вычисления $2x$ нормальный алгоритм не останавливался, поэтому заменим марковские подстановки, приводящие к остановке алгоритма (**2a → . 1* и **2 → .*) на «аналогичные», но не останавливающие алгоритм – **2a → 1* и **2 →* .

Итак, запишем схему нормального алгоритма S_a, вычисляющего значения функции $f(x) = 2x+1$ (x – целое положительное число) в десятичной системе счисления:

1. $0*2a \to *21$
2. $1*2a \to *23$
3. $2*2a \to *25$
4. $3*2a \to *27$
5. $4*2a \to *29$
6. $5*2a \to *2a1$
7. $6*2a \to *2a3$
8. $7*2a \to *2a5$
9. $8*2a \to *2a7$
10. $9*2a \to *2a9$
11. $0*2 \to *20$
12. $1*2 \to *22$
13. $2*2 \to *24$
14. $3*2 \to *26$
15. $4*2 \to *28$
16. $5*2 \to *2a0$
17. $6*2 \to *2a2$
18. $7*2 \to *2a4$
19. $8*2 \to *2a6$
20. $9*2 \to *2a8$
21. $*2a \to 1$
22. $*2 \to$
23. $0+1 \to 1$
24. $1+1 \to 2$
25. $2+1 \to 3$
26. $3+1 \to 4$
27. $4+1 \to 5$
28. $5+1 \to 6$
29. $6+1 \to 7$
30. $7+1 \to 8$
31. $8+1 \to 9$
32. $9+1 \to +10$
33. $+1 \to .1$
34. $\to .$

Проиллюстрируем работу составленного нормального алгоритма на нескольких примерах.

1) $23*2+1 \to 2*26+1 \to *246+1 \to 46+1 \to \_\ 47 \to .47$

2) $999*2+1 \to 99*2a8+1 \to 9*2a98+1 \to *2a998+1 \to 1998+1 \to$

$\_1999 \to .1999$

Замечание

Отметим, что нормальные алгоритмы Маркова являются не только средством теоретических построений, но и основой специализированного языка программирования, применяемого как язык символьных преобразований при разработке систем искусственного интеллекта; это один из немногих языков, разработанных в России и получивший известность во всем мире.

6.5. Принцип нормализации Маркова

Создатель теории нормальных алгоритмов А.А. Марков выдвинул гипотезу, получившую название «принцип нормализации Маркова».

Принцип нормализации Маркова

Для нахождения значений функции, заданной в некотором алфавите, тогда и только тогда существует какой-нибудь алгоритм, когда функция нормально вычислима.

Иными словами, принцип нормализации заключается в том, что все алгоритмы исчерпываются нормальными алгоритмами или, то же самое, – всякий алгоритм нормализуем.

Сформулированный принцип, как и тезисы Черча, Тьюринга, Поста, носит внематематический характер (понятие произвольного алгоритма не является строго определенным) и не может быть строго доказан. Он основывается на том, что все известные в настоящее время алгоритмы являются нормализуемыми.

Замечание

В теории алгоритмов огромную роль играет тот факт, что все разработанные теории, уточняющие интуитивное понятие алгоритма, равносильны между собой. Другими словами, они описывают один и тот же класс алгоритмически вычислимых функций.

Полезно уяснить смысл и значение этого важного результата. В разное время в разных странах ученые независимо друг от друга, изучая интуитивное понятие алгоритма и алгоритмической вычислимости, создали теории, описывающие данное понятие, которые оказались равносильными. Этот факт служит мощным косвенным подтверждением адекватности этих теорий опыту вычислений, справедливости каждого из тезисов Тьюринга, Поста, Черча и Маркова. В самом деле, ведь если бы один из классов оказался шире какого-либо другого класса, то соответствующий тезис Тьюринга, Поста, Черча или Маркова был опровергнут. Например, если бы класс нормально вычислимых функций оказался шире класса рекурсивных функций, то существовала бы нормально вычислимая, но не рекурсивная функция. В силу ее нормальной вычислимости она была бы алгоритмически вычислима в интуитивном понимании алгоритма, и предположение о ее нерекурсивности опровергало бы тезис Черча. Но классы Тьюринга, Поста, Черча и Маркова совпадают, и таких функций не существует. Это служит еще одним косвенным подтверждением тезисов Тьюринга, Поста, Черча и Маркова.

Вопросы к главе 6

1. Кем и когда были разработаны нормальные алгоритмы?
2. Определите сущность понятий «алфавит», «слово», «подслово». Приведите примеры.
3. Дайте определение марковской подстановки. Приведите пример.
4. Опишите действие частных случаев марковских подстановок: *(,Q)*, *(P,)*, *(,)*. Приведите примеры.
5. Что называют формулой подстановки?
6. В чем состоит особенность формулы заключительной подстановки?
7. Что называется схемой нормального алгоритма? Приведите пример.
8. Что называется нормальным алгоритмом Маркова? Приведите пример.
9. Раскройте смысл слов «нормальный алгоритм над алфавитом *A*».
10. Какая функция называется нормально вычислимой? Приведите примеры.
11. Что называется расширением алфавита *A*?
12. Какие способы композиции нормальных алгоритмов существуют? В чем их суть?
13. Какой способ композиции нормальных алгоритмов был использован в примере параграфа 4?
14. Сформулируйте принцип нормализации.
15. Раскройте роль эквивалентности различных формальных моделей алгоритма в теории алгоритмов.

Задания к главе 6

1. Постройте дедуктивную цепочку (последовательность слов), ведущую от слова «мука» к слову «торт», заменяя каждый раз по одной букве так, чтобы каждый раз получалось слово.

2. Пусть $A = \{a,b,c\}$ и задана следующая нормальная схема:

$$S_\alpha: \quad \begin{array}{l} ab \to c \\ cb \to a \\ a \to . \end{array}$$

Выясните, какое слово будет результатом применения этого нормального алгоритма к слову: а) *abbacb*; б) *bc*.

3. Пусть $A = \{a,b\}$. Зададим нормальный алгоритм с помощью алфавита $D = \{a,b,\alpha,\beta,\lambda\}$ и следующей нормальной схемы:

1. $aa\beta \to a\beta a$
2. $ab\beta \to b\beta a$
3. $ba\beta \to a\beta b$
4. $bb\beta \to b\beta b$
5. $\alpha a \to a\beta a\alpha$
6. $\alpha b \to b\beta b\alpha$
7. $\beta \to \gamma$
8. $\gamma \to \wedge$
9. $\alpha \to .$
10. $\wedge \to \alpha$

Найдите результат применения данного алгоритма к словам *ab*, *aba*. Опишите, что делает данный алгоритм.

4. Пусть $A = \{a,b\}$. Зададим нормальный алгоритм с помощью алфавита $D = \{a,b,\alpha,\beta\}$ и следующей нормальной схемы:

1. $\alpha a \to a\beta a\alpha$
2. $\alpha b \to b\beta b\alpha$
3. $\beta aa \to a\beta a$
4. $\beta ab \to b\beta a$
5. $\beta ba \to a\beta b$
6. $\beta bb \to b\beta b$
7. $\beta \to \wedge$
8. $\alpha \to .$
9. $\wedge \to \alpha$

Найдите результат применения данного алгоритма к словам *ab*, *bab*.

5. Постройте нормальный алгоритм побуквенного кодирования, т. е. нормальный алгоритм над алфавитом *A*, выполняющий замену букв *m* алфавита *A* словами K_m в этом алфавите.

6. Постройте нормальный алгоритм, преобразующий любое слово *P* в алфавите *A* в проекцию этого слова на алфавит *B*. (Проекцией слова *P* в алфавите *A* на алфавит *B* называется результат удаления из слова *P* букв алфавита *A\B*.)

7. Постройте нормальный алгоритм правого присоединения слова *aba* в алфавите *A = {a,b}* над этим алфавитом.

8. Постройте нормальный алгоритм:

а) преобразующий каждое слово *P* в алфавите *{a,b,c}* в слово *abcP*;

б) преобразующий каждое слово в алфавите *{a,b,c}* в слово *abc*.

9. Постройте нормальный алгоритм, применение которого к данному слову в алфавите *{a,b,c}* приведет к тому, что будет получено слово, содержащее все буквы «*a*», расположенные в его начале.

10. Постройте нормальный алгоритм, перерабатывающий любое слово в алфавите *{a,b,c}* в слово, в котором все буквы «*a*» (если они имеются) расположены левее остальных, а все буквы «*b*» левее всех букв «*c*».

11. Постройте нормальный алгоритм, преобразующий каждое слово в алфавите *{a,b}* в слово «*baba*».

12. Постройте нормальный алгоритм над алфавитом *{1}*, применение которого к слову, состоящему из четного числа «*1*», приводило бы к слову «*», а применение этого же алгоритма к слову, состоящему из нечетного числа «*1*» к слову «+».

13. Постройте нормальный алгоритм, преобразующий любое слово в алфавите *{a,b}*, содержащее хотя бы две буквы «*a*», в слово «*baabab*», а каждое слово, содержащее не более одной буквы «*a*» в слово, получающееся из исходного заменой букв «*a*» и «*b*» друг на друга.

14. Известно, что *A = {0,1,2,3,4,5,6,7,8,9}*; нормальный алгоритм задан в его расширении *B = A U {a,b}* и определен следующей схемой:

1. $0b \rightarrow .1$
2. $1b \rightarrow .2$
3. $2b \rightarrow .3$
4. $3b \rightarrow .4$
5. $4b \rightarrow .5$
6. $5b \rightarrow .6$
7. $6b \rightarrow .7$
8. $7b \rightarrow .8$
9. $8b \rightarrow .9$
10. $9b \rightarrow b0$
11. $b \rightarrow .1$
12. $a0 \rightarrow 0a$
13. $a1 \rightarrow 1a$
14. $a2 \rightarrow 2a$
15. $a3 \rightarrow 3a$
16. $a4 \rightarrow 4a$
17. $a5 \rightarrow 5a$
18. $a6 \rightarrow 6a$
19. $a7 \rightarrow 7a$
20. $a8 \rightarrow 8a$
21. $a9 \rightarrow 9a$
22. $0a \rightarrow 0b$
23. $1a \rightarrow 1b$
24. $2a \rightarrow 2b$
25. $3a \rightarrow 3b$
26. $4a \rightarrow 4b$
27. $5a \rightarrow 5b$
28. $6a \rightarrow 6b$
29. $7a \rightarrow 7b$
30. $8a \rightarrow 8b$
31. $9a \rightarrow 9b$
32. $\wedge \rightarrow a$

Примените данный алгоритм к словам *499*, *328*, *789*.

Вычисляет ли данный алгоритм какую-либо функцию? Если да, то какую?

15. Известно, что нормальный алгоритм в алфавите $A = \{1\}$ со следующей схемой S_a:

S_a:	$111 \rightarrow$
	$11 \rightarrow .$
	$1 \rightarrow .$
	$\rightarrow .1$

вычисляет некоторую функцию $f(x)$ (x слова в A). Определите формульное выражение функции $f(x)$.

16. Постройте нормальный алгоритм, вычисляющий числовую функцию $f(x) = x+y$ (аргумент x представлен в унарной записи).

17. Пусть *M, N, Q, R* обозначают натуральные числа, представленные в унарной форме. Постройте нормальный алгоритм, который перерабатывает:

а) всякую пару натуральных чисел *N*M* в абсолютную величину их разности;

б) всякое натуральное число в остаток от деления числа на 5;

в) всякое натуральное число *N* в $\left[\frac{N}{5}\right]$;

г) всякое натуральное число *N* в пару *Q*R*, где *Q* частное от деления *N* на *5*, а *R* остаток от этого деления;

д) всякую пару натуральных чисел *N*M* в наибольший общий делитель этих чисел;

е) всякую пару натуральных чисел *N*M* в произведение этих чисел.

18. Замените в десятичном числе все четные цифры на «1», а нечетные на «0».

19. Замените все буквы «а» в слове на «?», если слово начинается на «а».

20. Замените все буквы «я» в слове на «!», если слово заканчивается на «я».

21. Удалите из слова все буквы «б», если их четное число.

22. Удалите из слова все буквы «б», если их нечетное число.

23. Удвойте в слове все одиночные буквы.

24. Если в слове есть одиночные символы, то удалите их.

25. Если в слове есть одинаковые рядом стоящие символы, то удалите их, иначе продублируйте все символы.

26. Поменяйте местами вторую и предпоследнюю буквы в слове.

27. Поменяйте местами буквы в слове попарно.

28. Если в слове четное количество букв, то замените все «а» на «!».

29. Если в слове нечетное количество букв, то после каждой буквы поставьте «!».

30. Если в слове нечетное количество букв «и», то удалите их.

31. Если в слове четное количество букв «а», то удалите первую из них.

32. Если в слове встречаются две подряд буквы «а», то замените каждую пару на «!!», иначе вместо данного слова запишите «*».

33. Удалите в слове все символы, кроме первого и последнего.

34. Если в десятичном числе встречаются последовательности «012», то замените их на «***», иначе добавьте в конце слова знак «–».

35. Добавьте в конце слова столько «+» сколько в слове встречается символ «ы».

Заключение

Рассмотренные нами алгоритмические модели (класс рекурсивных функций, машина Тьюринга, машина Поста, машины произвольного доступа, нормальные алгоритмы Маркова) отождествляют с формальным понятием алгоритма. Моделируя любые другие разумные алгоритмические модели, они универсальны, а поэтому позволяют снять возможное возражение против того, что жесткая фиксация алгоритмической модели приводит к потере общности формализации алгоритма.

Основные алгоритмические модели в теории алгоритмов принято разделять на три типа.

Первый тип трактует алгоритм как некоторое детерминированное устройство, способное выполнять в каждый момент лишь строго фиксированное множество операций. Основной теоретической моделью такого типа является машина Тьюринга, оказавшая существенное влияние на понимание логической природы разрабатываемых ЭВМ. К этому типу относят и такие алгоритмические модели, как машина Поста, машины произвольного доступа.

Второй тип связывает понятие алгоритма с традиционным представлением процедурами вычисления значений числовых функций. Основной теоретической моделью этого типа являются рекурсивные функции исторически первая формализация алгоритма.

Третий тип алгоритмических моделей это преобразования слов в произвольных алфавитах, в которых операциями являются замены частей слов другим словом. Основной теоретической моделью этого типа являются нормальные алгоритмы Маркова.

Все рассмотренные нами алгоритмические модели эквиваленты, и роль их огромна. Они оказали существенное влияние на развитие ЭВМ и практику программирования. Несмотря на то, что все алгоритмические модели эквивалентны, на практике они существенно различаются сложностными эффектами, возникающими при реализации алгоритмов, что способствовало появлению различных направлений в программировании. Так, микропрограммирование строится на идеях машин Тьюринга, структурное программирование заимствовало свои конструкции из теории рекурсивных функций, языки символьной обработки информации (например, ПРОЛОГ) берут начало от нормальных алгоритмов Маркова.

Ответы, указания и решения

Глава 1

Задание 1. *Решение.* Алгоритм сложения в столбик двух натуральных чисел (первое слагаемое содержит m цифр, второе n, считаем, что $m \le n$):

1) запишем исходные числа в столбик так, чтобы разряды единиц были под разрядами единиц, разряды десятков были под разрядами десятков и т. д., т. е. числа выровнены по младшему разряду;

2) положим $i = 1$ (i обрабатываемый разряд, разряды пронумерованы справа налево), $p = 0$ (p значение переноса в старший разряд);

3) сложим цифры i-го разряда обоих слагаемых (если у первого слагаемого уже в данном разряде цифр нет, то считаем, что прибавляем 0), прибавим к результату p;

4) если получившееся число больше либо равно 10, то в i-й разряд результата запишем младшую цифру получившегося числа (остаток от его деления на 10), а p положим равным 1, иначе в i-й разряд результата запишем получившееся число, а p положим равным 0;

5) увеличим i на единицу;

6) если $i \le n$, то переходим на шаг 3;

7) если $p = 0$, то конец алгоритма, иначе в $(n+1)$-й разряд результата запишем 1, конец алгоритма.

При анализе этого алгоритма надо обратить внимание на тот факт, что *в старший разряд может переноситься только единица*.

Задание 2. *Указание.* Для того чтобы приведенный способ стал алгоритмом, надо конкретизировать шаги 1–3, указав точные значения для радиусов окружностей.

Задание 3, 4, 5. *Указание.* Проверьте выполнимость всех основных свойств алгоритма.

Задание 6. *Решение.* Система команд колдуна:

- поставили сосуд с эликсиром на огонь;
- сняли эликсир с огня;
- перевернули трехминутные часы;
- перевернули восьмиминутные часы;
- проверили, что часы пересыпались.

Алгоритм варки эликсира бессмертия:

1) перевернули трехминутные часы; перевернули восьмиминутные часы;

2) если трехминутные часы пересыпались, то переворачиваем их и ставим эликсир на огонь;

3) если трехминутные часы пересыпались, то переворачиваем их (эликсир варится уже 3 минуты);

4) если восьмиминутные часы пересыпались, то переворачиваем часы трехминутные (эликсир варится 5 минут; после переворота трехминутных часов в верхней колбе песка осталось на 2 минуты);

5) если трехминутные часы пересыпались, то снимаем эликсир с огня.

Подчеркнем, что могут быть предложены и другие решения этой задачи.

Задание 7. *Указание.* Составьте алгоритм, который правильно вычисляет и $0!=1$.

Задание 8. *Решение.* Докажем, что в результате работы алгоритма «Решето Эратосфена» в последовательности чисел от 1 до N не будет зачеркнуто ни одно простое число. Действительно, на каждом шаге зачеркиваются только числа, кратные каким-то другим числам. Число $k>1$ из этой последовательности останется незачеркнутым только в том случае, если оно не делится ни на одно из незачеркнутых чисел, не превосходящих $\sqrt{N}$, среди которых содержатся все простые числа, не превосходящие $\sqrt{N}$.

Докажем, что любое составное число m имеет простой делитель, не превосходящий $\sqrt{m}$. Пусть $m = pq$, причем p наименьший множитель $(p < q)$. Тогда $p^2 = p \cdot p < pq = m$, т. е. $p^2 < m$, т. е. $p < \sqrt{m}$. Следовательно, если m составное, то его наименьший делитель меньше $\sqrt{m}$, что и требовалось доказать.

Таким образом, в исходной последовательности остаются незачеркнутыми все простые числа и только они.

Задание 9. *Указание.* Введите вспомогательные обозначения: I вспомогательная переменная, *Slag* очередное слагаемое суммы, S искомая сумма. В качестве условия в блок-схемах попробуйте ввести следующее: $i = n$.

Задание 10. *Решение.* В приведенном алгоритме есть две ошибки. *Первая ошибка* заключается в том, что при нечетном n и для $k = m$ будет неверно выполняться п. 6. Для любого n количество пар, которые надо обработать, равно $\frac{n}{2}$, и в качестве начальных значений min и max надо брать значения соответствующих элементов последней m-й полной пары для четных n или элемент a_n в противном случае.

Вторая ошибка состоит в выборе типа циклической конструкции. В предложенном варианте алгоритма при $n=1$ получим зацикливание, а при $n>1$ алгоритм не будет обрабатывать последнюю пару.

Правильный алгоритм:

1. Разобьем исходную последовательность на пары. Последняя пара может быть неполной, если число n нечетно.

2. Положим $m = \frac{n}{2}$ (m число пар, которые надо обработать).

3. Упорядочим числа в каждой паре по возрастанию. Будем элементы в паре обозначать $P_{k,1}$ и $P_{k,2}$, где k номер пары.

4. Положим $\min = a_n$ и $\max = a_n$ при нечетном n. В противном случае положим: $\min = P_{m,1}$ и $\max = P_{m,2}$.

5. Положим $k = 1$.

6. Если $k = m$, то конец алгоритма.

7. Если $P_{k,2} > \max$, тогда $\max = P_{k,2}$.

8. Если $P_{k,1} < \min$, тогда $\min = P_{k,1}$.

9. Увеличим k на единицу.

10. Переходим на п. 6.

Глава 2

Задание 1. *Указание.* Воспользуйтесь определением вычислимой функции.

Задание 2. *Ответ.* а) x_1 ; б) не определено; в) $x_1 + 2$; г) 0; д) $x_2 + 4$; е) 4.

Задание 3. *Указание.* Можно, причем только из двух простейших функций: O, S .

Задание 4. *Ответ.* а) $vz(u^2 + 2^u)$; б), в) не определено.

Задание 5. *Ответ.* а) $2uv^2z^{2+u} + u + v + z$; б), в) не определено.

Задание 6. *Решение.*

а) Переобозначим: $f(x, y) = h(x, y) = xy$.

Будем искать функции $g^1(x)$ и $f^3(x, y, z)$ из определения примитивной рекурсии:

$$\begin{cases} h(x,0) = x \cdot 0 = 0 = g(x) = O(x), \\ h(x, y+1) = x \cdot (y+1) = \underline{xy + x} = f(x, y, h(x, y)) = f(x, y, \underbrace{xy}_{z}). \end{cases}$$

Следовательно, $g^1(x) = O(x)$, $f^3(x, y, z) = z + x$. Функция же суммы в свою очередь может быть получена из простейших с помощью оператора примитивной рекурсии (доказано ранее в § 2 главы 3). А поэтому исходная функция $f(x, y) = xy$ может быть получена из простейших с помощью оператора примитивной рекурсии.

д) Используйте, что $f(x) = x \dot{-} 1 \overset{\text{определение}}{=} \begin{cases} 0, \text{если } x = 0, \\ x - 1, \text{если } x > 0 \end{cases}$ и $f(x, y) = x \dot{-} y \overset{\text{определение}}{=} \begin{cases} 0, \text{если } x < y, \\ x - y, \text{если } x \geq y. \end{cases}$

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

Задание 7. *Решение.*

1. Найдем аналитическую запись функции $f^2(x, y)$. Из схемы примитивной рекурсии:

$$\begin{cases} f(x,0) = g(x) = I_1^1(x), \\ f(x, y+1) = h(x, y, f(x, y)) = I_3^3(x, y, f(x, y)) + 1 = f(x, y) + 1. \end{cases}$$

Тогда:

$f(x,0) = x$;

$f(x,1) = f(x,0) + 1 = x + 1$;

$f(x,2) = f(x,1) + 1 = (x+1) + 1 = x + 2$;

……..…..

Естественно предположить, что $f(x,y) = x + y, \forall y \in \mathrm{N} \cup \{0\}$. Докажем это методом математической индукции:

1) доказательство базиса очевидно, т .к. $f(x,0) = x + 0$;

2) предположим, что $f(x,y) = x + y$ для некоторого y . Тогда:

$f(x,y+1) = f(x,y) + 1 \overset{\text{предположение}}{=} (x+y) + 1 = x + (y+1)$, что и требовалось доказать.

Таким образом, из функций $h(x,y,z) = I_3^3(x,y,z) + 1$ и $g(x) = I_1^1(x)$ с помощью оператора примитивной рекурсии получается функция $f(x,y) = x + y.$

2. Вычислим $f(2,5)$.

Так как $f(x,y) = x + y$, то $f(2,5) = 2 + 5 = 7$.

Или из схемы примитивной рекурсии:

$f(2,5) = f(2,4+1) = f(2,4) + 1$;

$f(2,4) = f(2,3+1) = f(2,3) + 1$;

$f(2,3) = f(2,2+1) = f(2,2) + 1$;

$f(2,2) = f(2,1+1) = f(2,1) + 1$;

$f(2,1) = f(2,0+1) = f(2,0) + 1$;

Но т. к. $f(2,0) = 2$, то $f(2,1) = 3$, $f(2,2) = 4$, $f(2,3) = 5$, $f(2,4) = 6$, $f(2,5) = 7$.

Задание 8. *Ответ.* $h(u,y) = \begin{cases} 0, \text{если } y \in N \\ 2u, \text{если } y = 0 \end{cases}$

Задание 9. *Ответ.* $h(u,v,y) = 2^u u^2 v^{y+1}$.

Задание 10. *Ответ.* а) $f(x,y) = x^{x^y}$; б) $f(x,y) = x^{\overset{y}{\overbrace{x^{x^{\cdots^x}}}}}$.

Задание 11. *Решение.* б) Попытаемся найти функции h и g, такие, что: $f = R(h,g)$. Ясно, что $g^1(x) = const$, $h^2(x,y)$. Из схемы примитивной рекурсии:

$$\begin{cases} f(0) = const = g(x) = 2\cdot 0 + 1 = 1 = S(O) = G(S,O), \\ f(y+1) = h(y, f(y)) = h(y, \underbrace{2y+1}_{z}) = 2(y+1)+1 = (2y+1)+2 \end{cases}$$

Итак:

1) $g(x) = 1 = G(S,O)$ примитивно рекурсивна;

2) $h(y,z) = z+2$ или $h(x,y) = y+2 = S(S(y)) = G(S,S(y))$ примитивно рекурсивна.

Следовательно, исходная функция $f(x) = 2x+1$ является примитивно рекурсивной.

Задание 12. *Решение.* а) Нужно показать, что $(x-y)$ наименьшее целое неотрицательное число, являющееся решением уравнения $y+z=x$ относительно z. Это действительно так. Поэтому исходное равенство верно.

г) Исходное равенство неверно, потому что значение выражения $\mu_y[y(y-(x+1))=0]$ не определено при $y=0$ (т. к. не определено значение терма $0(0-\underbrace{(x+1)}_{>1})$), а именно не определена разность).

В то же время подчеркнем, что уравнение $y(y-(x+1))=0$ имеет решение $y = x+1$.

Задание 13. *Решение.* а) $\varphi(x) = \mu_y[|x-2y|=0]$. Отметим, что функция φ определена лишь на числах вида $2k, k \in \mathrm{N}$ (поясните почему) и для каждого из них:

$$\varphi(2k) = \mu_y[|2k-2y|=0] = \mu_y[2|k-y|=0] = k \text{, где } k = 0,1,2,3,\ldots .$$

в) $\varphi(x) = \mu_y[8=0]$ не определена на всех x.

Задание 15. *Указание.* Используйте функцию «усеченного вычитания».

Задание 16. *Указание.* Используйте определение оператора минимизации.

Глава 3

Задание 1. *Решение.* Данная машина применима к любым двоичным словам, в том числе она применима и к нулевому входному слову.

Машина Тьюринга выполняет инверсию двоичных слов (заменяет все «1» на «0» и наоборот).

Задание 4. *Решение.*

Понятность. На каждом шаге в ячейку пишется символ из внешнего алфавита, автомат делает одно движение ($Л, П, С$) и машина Тьюринга переходит в одно из допустимых состояний: все эти действия входят в систему команд данного исполнителя.

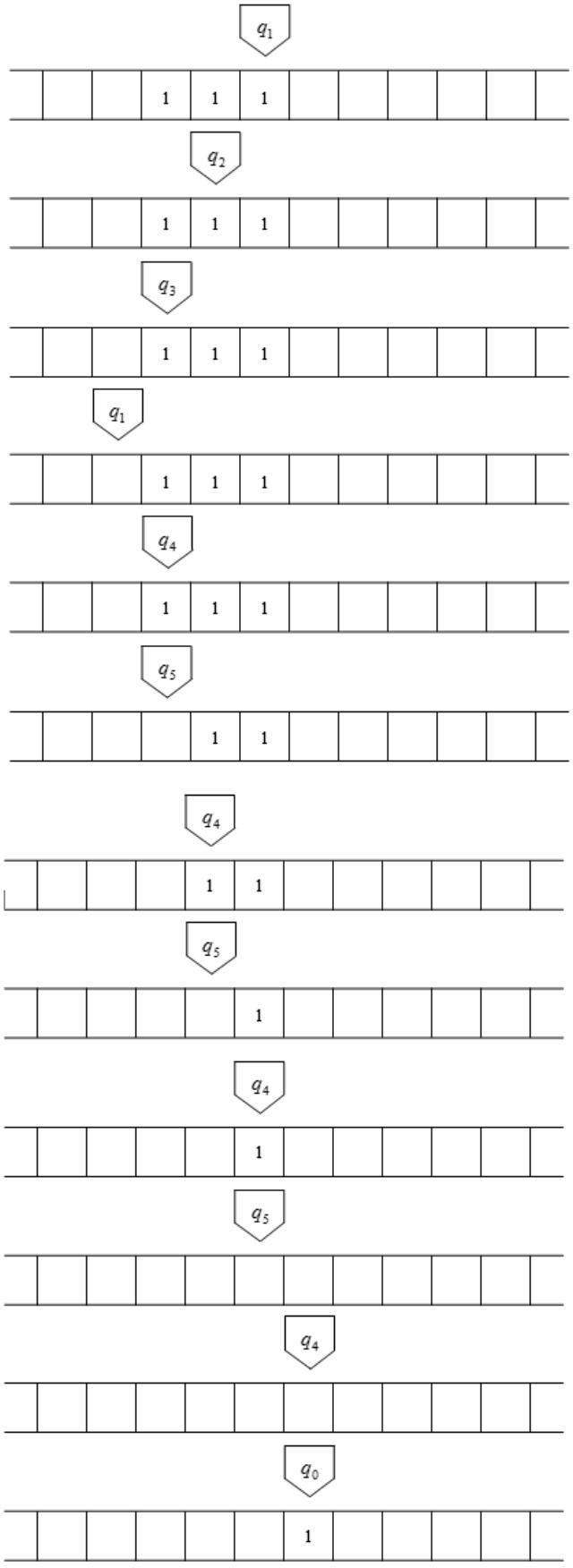
Дискретность. Машина Тьюринга может перейти к $(k+1)$ -у шагу только после выполнения k -го шага, так как именно k -й шаг определяет, каким будет $(k+1)$ -й шаг.

Детерминированность. В каждой клетке таблицы машины Тьюринга записан лишь один вариант действия. На каждом шаге результат определен однозначно, следовательно, последовательность шагов решения задачи определена однозначно, т. е. если машине Тьюринга на вход подают одно и то же входное слово, то выходное слово каждый раз будет одним и тем же.

Результативность. Содержательно результат каждого шага и всей последовательности шагов определен однозначно, следовательно, правильно написанная машина Тьюринга за конечное число шагов перейдет в состояние q_0 , т. е. за конечное число шагов будет получен ответ на вопрос задачи.

Массовость. Каждая машина Тьюринга определена над всеми допустимыми словами из алфавита, в этом и состоит свойство массовости. Каждая машина Тьюринга предназначена для решения одного класса задач, т. е. для каждого класса задач пишется своя (новая) машина Тьюринга.

Задание 5. *Решение.* (д) Выпишем последовательность конфигураций машины при переработке ею слова 111 из начального стандартного положения.



Задание 6. *Указание.* а) Машина останавливается, выдав слово $1\,a_0\,1\,a_0\,a_0\,1$, обозревая в состоянии q_3 четвертую ячейку слева.

б) Машина останавливается, выдав слово $1\,a_0\,1\,a_0\,1$, при этом она обозревает в состоянии q_3 вторую левую пустую ячейку от полученного слова.

в) Машина не меняет данного слова и останавливается в состоянии q_3, обозревая вторую левую пустую ячейку от этого слова.

Задание 9. *Решение.* (д) 11111. Данная машина Тьюринга реализует операцию сложения: в результате ее работы на ленте записано подряд столько единиц, сколько их было всего записано по обе стороны от звездочки перед началом работы машины.

Задание 11. *Ответ.* Машина Тьюринга выполняет сложение данных чисел.

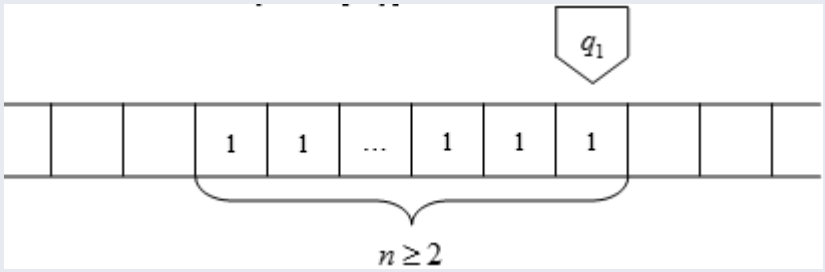
Задание 13. *Решение.* В качестве внешнего алфавита возьмем двухэлементное множество $A=\{a_0,1\}$, где a_0 символ пустой буквы. Количество необходимых внутренних состояний будет определено в процессе составления программы.

Будем считать, что машина начинает работать из стандартного начального положения, т. е. когда в состоянии q_1 обозревается крайняя правая единица из n записанных на ленте.

Рассмотрим три случая: 1) $n\geq 2$, 2) $n=1$, 3) $n=0$.

Случай 1. $n\geq 2$

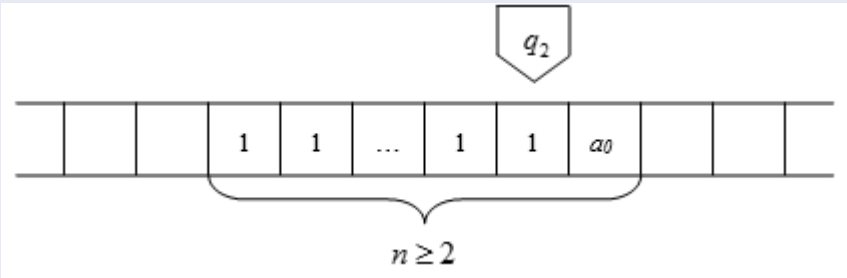
Запишем начальную конфигурацию.

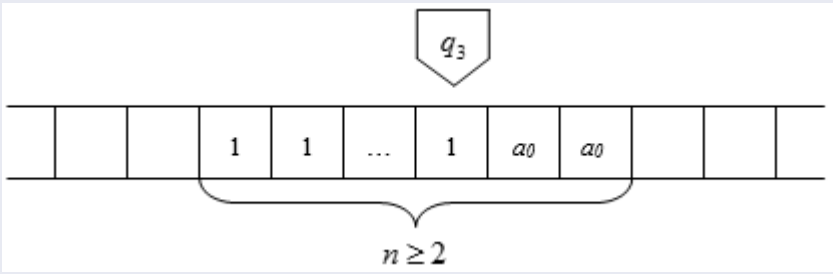


Начнем с того, что сотрем первую единицу, перейдем к обозрению следующей левой ячейки и сотрем там единицу. На каждом таком переходе машина должна переходить в новое внутреннее состояние, ибо в противном случае будут стерты вообще все единицы, записанные подряд. Вот команды, осуществляющие описанные действия:

$$q_1 1 \to q_2 a_0 Л \quad , \quad q_2 1 \to q_3 a_0 Л \quad .$$

Запишем конфигурации машины Тьюринга на данных этапах ее работы.

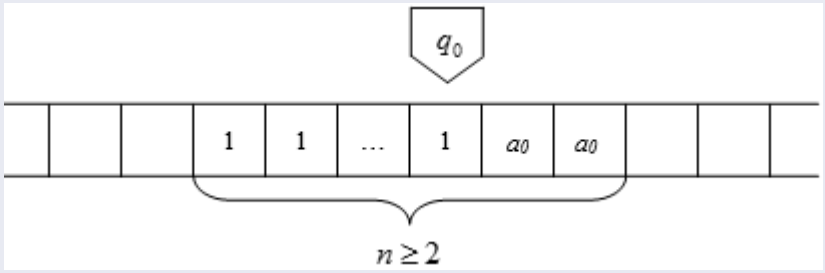




Машина Тьюринга теперь находится в состоянии q_3 и обозревает третью справа ячейку из тех, в которых было записано входное слово (последовательность из $n \geq 2$ единиц). Не меняя содержимого обозреваемой ячейки, машина должна остановиться, т. е. перейти в заключительное состояние q_0, независимо от содержимого ячейки. Вот эти команды:

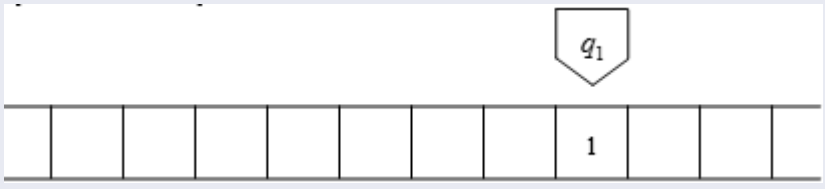
$$q_3 a_0 \to q_0 a_0 C \quad , \quad q_3 1 \to q_0 1 C$$

Соответствующая конфигурация машины будет иметь следующий вид:

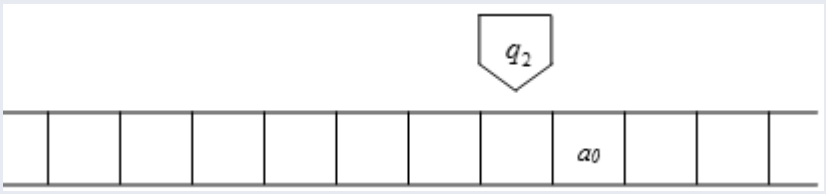


Случай 2. $n = 1$

Если на ленте записана одна единица, то начальная конфигурация будет иметь следующий вид:



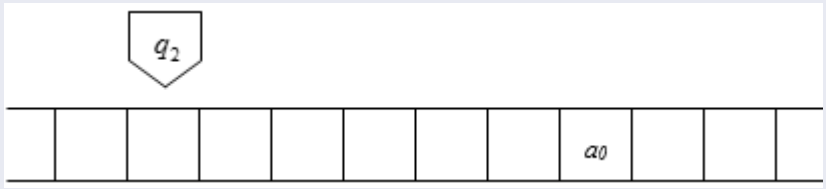
Следующий шаг работы машины Тьюринга будет выполнен в соответствии с командой $q_1 1 \to q_2 a_0 Л$, в результате которого машина будет находиться в состоянии q_2 и обозревать вторую справа ячейку, в которой записан символ пустой буквы a_0 . Запишем соответствующую конфигурацию машины:



По условию задачи необходимо, чтобы в этом случае машина работала вечно, т. е. не приходила в состояние останова q_0. Это можно обеспечить, например, такой командой:

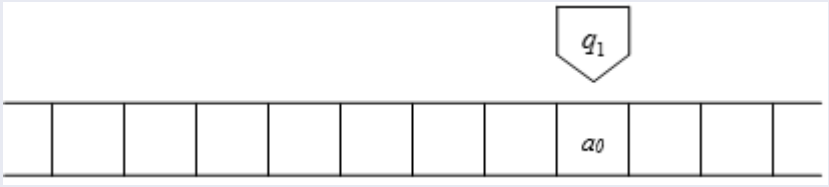
$$q_2 a_0 \to q_2 a_0 Л$$

Выполняя эту команду, шаг за шагом, машина будет двигаться по ленте неограниченно влево:



Случай 3. $n = 0$

Наконец, если на ленте не записано ни одной единицы, то машина, по условию, также должна работать вечно. В этом случае в начальном состоянии q_1 обозревается ячейка с содержимым a_0:



Вечную работу машины можно обеспечить следующей командой:

$$q_1 a_0 \to q_1 a_0 Л$$

Запишем составленную программу (функциональную схему) построенной машины Тьюринга в виде таблицы:

A \ Q	q_1	q_2	q_3
a_0	$q_1 a_0 Л$	$q_2 a_0 Л$	$q_0 a_0 С$
1	$q_2 a_0 Л$	$q_3 a_0 Л$	$q_0 1 С$

Задание 17. *Решение.* Будем формировать исходное число на ленте слева от штрихов (без пустого символа между числом и штрихами). В начальный момент машина Тьюринга обозревает любой из штрихов и находится в состоянии q_1.

Запишем алгоритм решения задачи в текстовой форме:

1) найдем правый конец слова на ленте;

2) если слово будет оканчиваться штрихом, то сотрем этот штрих, иначе остановим машину;

3) прибавим к числу единицу и перейдем к п. 1.

В соответствии с этим алгоритмом каждый раз стирается самый правый штрих и к числу прибавляется единица. Выполнение алгоритма продолжается до тех пор, пока не будет стерт самый последний штрих, после чего, согласно условию из п. 2, машина Тьюринга остановится. Заметим, что каждый из трех пунктов алгоритма может быть реализован одним состоянием машины Тьюринга:

1) состояние q_1 автомат ищет правый конец слова;

2) состояние q_2 автомат стирает штрих;

3) состояние q_3 прибавление к числу единицы.

Ниже приведена программа машины Тьюринга для решения предлагаемой задачи.

Q \ A	a_0	0	1	2	…	8	9	/
q_1	$q_2a_0Л$	$q_10П$	$q_11П$	$q_12П$	…	$q_18П$	$q_19П$	$q_1 / П$
q_2	$q_0a_0С$	$q_00С$	$q_01С$	$q_02С$	…	$q_08С$	$q_09С$	$q_3a_0С$
q_3	$q_11П$	$q_11П$	$q_12П$	$q_13П$	…	$q_19П$	$q_30Л$	$q_3 / Л$

Задание 27. *Ответ.*

A \ Q	q_1	q_2	q_3
0	$q_10П$	$q_20Л$	$q_21Л$
1	$q_11П$	$q_22Л$	$q_23Л$
2	$q_12П$	$q_24Л$	$q_25Л$
3	$q_13П$	$q_26Л$	$q_27Л$
4	$q_14П$	$q_28Л$	$q_29Л$
5	$q_15П$	$q_30Л$	$q_31Л$
6	$q_16П$	$q_32Л$	$q_33Л$
7	$q_17П$	$q_34Л$	$q_35Л$
8	$q_18П$	$q_36Л$	$q_37Л$
9	$q_19П$	$q_38Л$	$q_39Л$
a_0	$q_2a_0Л$	$q_0a_0С$	$q_21Л$

Задание 34. *Ответ.* $f(x) = x + 2$.

Задание 35. *Ответ.* $f(x, y) = x$.

Задание 39. *Решение.* Покажем, что машину Тьюринга можно представить также, как и конечный автомат в следующем виде:

$$A = \{S, X, Y, s0, d, v\}, \text{ где}$$

S – конечное непустое множество (состояний);

X – конечное непустое множество входных сигналов (входной алфавит);

Y – конечное непустое множество выходных сигналов (выходной алфавит);

$s0 \in S$ – начальное состояние;

$d: S \times X \to S$ – функция переходов;

$v: S \times X \to Y$ – функция выходов.

Действительно, в машине Тьюринга $T = \{S,X,Y,s0,d,v\}$:

$S = Q = \{q_0,q_1,q_2,\ldots,q_n\}$ – алфавит внутренних состояний;

$X = A = \{a_0,a_1,a_2,\ldots,a_m\}$ – алфавит входных символов (входной алфавит);

$Y = \{a_0П,a_0Л,a_0С,a_1П,a_1Л,a_1С,\ldots,a_mП,a_mЛ,a_mС\}$ – множество выходных сигналов (выходной алфавит);

$s0 = q1 \in Q$ – начальное состояние;

$d: S \times X \to S$ – функция переходов и $v: S \times X \to Y$ – функция выходов задается в машине Тьюринга одновременно командами вида $q_ja_i \to q_ka_lX$, где $X = \{П,Л,С\}$. В соответствии с указанной командой: $d(q_ja_i) = q_k$, а $v(q_ja_i) = a_lX$, где $X = \{П,Л,С\}$.

Например, пусть дана машина Тьюринга T с внешним алфавитом $A = \{a_0,1\}$ (здесь a_0 – это символ пустой ячейки), алфавитом внутренних состояний $Q = \{q_0,q_1,q_2\}$ и со следующей программой:

$$q_1a_0 \to q_2a_0П,\ q_2a_0 \to q_01С,\ q_11 \to q_11П,\ q_21 \to q_21П.$$

Зададим функции переходов d и выходов v в этой машине Тьюринга T таблично.

Функция переходов машины Тьюринга T

d	a_0	1
q_0	–	–
q_1	q_2	q_1
q_2	q_0	q_2

Функция выходов машины Тьюринга T

v	a_0	1
q_0	–	–
q_1	$a_0П$	$1П$
q_2	$1С$	$1П$

Задание 40. *Решение.* Рассмотрим, например, машину Тьюринга T с внешним алфавитом $A = \{a_0,1\}$ (здесь a_0 – это символ пустой ячейки), алфавитом внутренних состояний $Q = \{q_0,q_1,q_2\}$ и со следующей программой: $q_1a_0 \to q_2a_0П$, $q_2a_0 \to q_01C$, $q_11 \to q_11П$, $q_21 \to q_21П$.

Зададим функции переходов d и выходов v в этой машине Тьюринга T таблично.

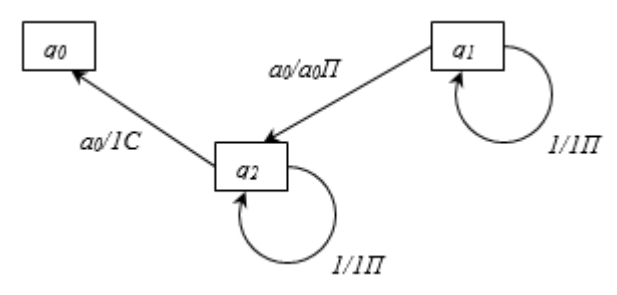
Функция переходов машины Тьюринга T

d	a_0	1
q_0	–	–
q_1	q_2	q_1
q_2	q_0	q_2

Функция выходов машины Тьюринга T

v	a_0	1
q_0	–	–
q_1	$a_0П$	$1П$
q_2	$1C$	$1П$

На основе составленных таблиц представим машину Тьюринга T в виде ориентированного графа.

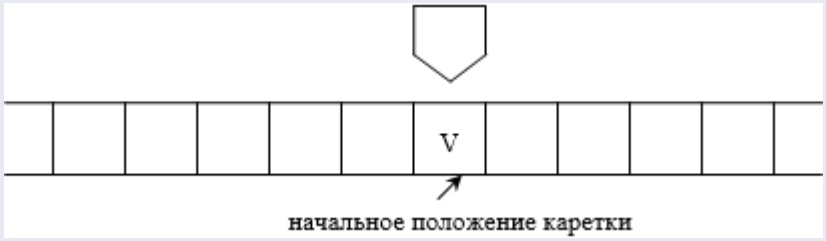


Опишем процесс построения ориентированного графа на основе программы машины Тьюринга.

Каждая команда программы машины Тьюринга $q_ja_i \to q_ka_lX$, где $X = \{П,Л,С\}$ представляется ребром ориентированного графа (направленным отрезком), соединяющим две его вершины. Причем каждой вершине приписывается соответствующее состояние машины Тьюринга (q_j или q_k), а дуге – символ a_i/a_lX.

Глава 4

Задание 1. *Решение.* Программа никогда не остановится, лента машины Поста будет иметь следующий вид:



Задание 2. *Решение.* Мы знаем, что на ленте есть метка, но не знаем, в какую сторону нужно двигаться. Каким же способом можно решить эту задачу? А вот каким. Представим, что мы стоим на бесконечной в обе стороны дороге, а где-то на этой дороге лежит камень, который мы хотим найти. Мы не знаем, где он лежит, знаем только, что где-то он непременно лежит. Как нам поступить?

В какую бы сторону мы ни пошли, может случиться, что волшебный камень лежит в другой стороне. Очевидно, что нам надо двигаться попеременно то в одну сторону, то в другую, постоянно увеличивая размах своих колебаний.

Сначала делаем шаг в одну сторону, потом два шага в другую сторону, потом три шага снова в первую сторону, потом четыре шага во вторую сторону и так далее до тех пор, пока не наткнемся на волшебный камень.

Но возникает одна трудность: как определить момент, когда надо поворачивать, т. е. менять направление? Казалось бы очень просто. Надо вести счет шагов. Но мы имеем дело с бесконечной лентой. Нам, возможно, придется запоминать очень большие числа, которые трудно хранить в памяти, а сколь угодно большие числа хранить в памяти трудно.

Однако выход из положения есть. Будем отмечать пройденный путь камешками. Дойдя по камешкам в одну сторону до конца, будем класть новый камешек, и поворачивать обратно. И так до тех пор, пока не найдем искомый камень.

Недостаток этого метода состоит в том, что надо иметь неограниченный запас камешков. Хотя для машины Поста это не является препятствием. Можно обойтись и всего двумя камешками. Для этого надо вначале нужно положить один камешек слева от себя, другой справа. А затем ходить между ними и передвигать их.

Запишем программу для машины Поста.

$1 V 2$ {положили левый камешек}

$2 \to 3$

$3 ? 5,4$

$4!$ {нашли метку, конец}

$5 V 6$ {положили правый камешек}

$6 \leftarrow 7$ {ищем левый камешек}

$7?6,8$

$8X9$ {стираем левый камешек}

$9 \leftarrow 10$

$10?11,4$

$11V12$ {передвигаем левый камешек}

$12 \rightarrow 13$ {ищем правый камешек}

$13?12,14$

$14X2$ {стираем правый камешек и повторяем действия}

Задание 3. *Решение.* Алгоритм решения этой задачи основан на следующей идее. К отдельно стоящей метке будем постепенно передвигать метки массива, начиная с левого края. То есть крайнюю левую метку массива будем перемещать на правый край массива, уменьшая тем самым расстояние до отдельной метки. Если метка отстоит от массива на одну ячейку, то нам достаточно выполнить только одно передвижение. В общем случае независимо от длины массива n нам придется сделать всего m передвижений.

Запишем программу для машины Поста.

$1X2$

$2 \rightarrow 3$

$3?4,2$

$4V5$

$5 \rightarrow 6$

$6?8,7$

$7!$

$8 \leftarrow 9$

$9?10,8$

$10 \rightarrow 1$

Задание 4. *Решение.* Пусть каретка стоит на левом элементе левого числа. Будем стирать по одной левой метке у каждого числа, пока одно из чисел не кончится. Тогда образовавшееся свободное пространство между числами вновь заполним метками (мы можем это сделать, так как знаем местоположение последней стертой метки), а лишние метки (большего числа) сотрем. Обозначим

левое число a, а правое b.

Запишем программу для машины Поста.

$1X2$ {уменьшили a на 1}

$2 \to 3$

$3?4,11$ {число a закончилось?}

$4V5$ {восстанавливаем метки}

$5 \to 6$

$6?4,7$ {число b началось?}

$7X8$ {стираем лишние метки b}

$8 \to 9$

$9?10,7$ {число b стерто?}

$10!$

$11 \to 12$ {ищем конец a}

$12?13,11$

$13 \to 14$ {ищем начало b}

$14?13,15$

$15X16$ {уменьшили b на 1}

$16 \to 17$

$17?18,26$ {число b закончилось?}

$18 \leftarrow 19$

$19 \leftarrow 20$

$20V21$ {восстанавливаем метки}

$21 \leftarrow 22$

$22?20,23$ {число a началось?}

$23X24$ {стираем лишние метки a}

$24 \leftarrow 25$

$25?10,23$ {число a стерто?}

$26 \leftarrow 27$ {ищем правый конец числа a}

$27?26,28$

$28 \leftarrow 29$ {ищем начало a}

$29?30,27$ {число a прошли?}

$30 \rightarrow 1$ {повторяем действия}

Задание 5. *Указание.* Составим пошаговый алгоритм решения этой задачи. Будем считать, что в начале работы весь массив является нераздвинутым. В результате выполнения очередной итерации алгоритма в нераздвинутой части удаляются две левых метки, а к раздвинутой части справа через пустую ячейку пара меток приписывается. Алгоритм заканчивает работу, если в нераздвинутой части осталась одна или две метки. В начале работы каретка находится на левой метке исходного массива (входного слова).

1. Если в нераздвинутой части осталось не более двух меток, то конец алгоритма.

2. Удаляем в нераздвинутой части две левых метки.

3. Ищем правый конец раздвинутого массива.

4. Ставим через пустую ячейку две метки.

5. Ищем левый конец нераздвинутого массива. Признаком окончания поиска являются две подряд идущие пустых ячейки.

6. Переход на п. 1.

Задание 8. *Указание.* Первая идея решения: оставить в каждом массиве по одной метке, а затем все метки сдвинуть в единый массив.

Вторая идея решения: будем «считать» массивы слева направо, удаляя каждый «посчитанный» массив. При этом слева от последовательности оставшихся массивов будем накапливать массив меток, длина которого соответствует числу «посчитанных» массивов.

Задание 11. *Указание.* Идея решения состоит в следующем: во вторых ячейках от каждого края массива ставим «маячки-пузырьки» (эти ячейки делаем пустыми). Далее последовательно перемещаем к центру левый и правый пузырек. Эти пузырьки встретятся ровно на центральном элементе исходного массива. При реализации программы надо отдельно учесть три случая: $n = 1, n = 3, n \geq 5$.

Задание 12. *Указание.* Идея решения состоит в следующем. Сначала между двумя левыми и двумя правыми метками ставим «маячки» пустые клетки. Первым ставим левый маячок. Затем поочередно сдвигаем эти маячки к центру. Как только маячки сомкнутся, то вместо правого маячка ставим метку, идем к правому краю массива и удаляем самую правую метку. Для простоты считаем, что каретка находится над самой левой меткой.

Глава 5

Задание 1. *Решение.* Условимся, что начальной конфигурацией МПД будет выступать последовательность чисел *(x,0,0,...)*, т. е. в регистре R_1 будет записано число x, а в остальных регистрах R_2, R_3, … – *0*.

Идея составления искомого алгоритма состоит в следующем: будем прибавлять *1* к r_3 и *2* к r_2 до тех пор, пока r_2 не станет равным x; тогда r_3 даст результат, т. е. значение $f(x)$.

Опишем назначение используемых регистров:

R_1 – предназначено для записи числа x, кроме того именно в этот регистр будет помещено искомое значение $f(x)$;

R_2, R_3 – вспомогательные регистры, используемые в ходе работы программы.

Искомая программа будет иметь следующий вид:

1) *J(1,2,6)*,

2) *S(3)*,

3) *S(2)*,

4) *S(2)*,

5) *J(1,1,1)*,

6) *T(3,1)*.

Проиллюстрируем работу МПД по составленной программе на конкретном примере. Пусть $x = 8$, т. е. в регистре R_1 записано число *8*.

Номер команды	*Команда*	*Содержимое регистров*		
		R_1	R_2	R_3
		8	*0*	*0*
1	*J(1,2,6)*	*8*	*0*	*0*
2	*S(3)*	*8*	*0*	*1*
3	*S(2)*	*8*	*1*	*1*
4	*S(2)*	*8*	*2*	*1*
5	*J(1,1,1)*	*8*	*2*	*1*
1	*J(1,2,6)*	*8*	*2*	*1*
2	*S(3)*	*8*	*2*	*2*
3	*S(2)*	*8*	*3*	*2*

4	*S(2)*	*8*	*4*	*2*
5	*J(1,1,1)*	*8*	*4*	*2*
1	*J(1,2,6)*	*8*	*4*	*2*
2	*S(3)*	*8*	*4*	*3*
3	*S(2)*	*8*	*5*	*3*
4	*S(2)*	*8*	*6*	*3*
5	*J(1,1,1)*	*8*	*6*	*3*
1	*J(1,2,6)*	*8*	*6*	*3*
2	*S(3)*	*8*	*6*	*4*
3	*S(2)*	*8*	*7*	*4*
4	*S(2)*	*8*	*8*	*4*
5	*J(1,1,1)*	*8*	*8*	*4*
1	*J(1,2,6)*	*8*	*8*	*4*
6	*T(3,1)*	*4*	*8*	*4*

Остановка МПД будет осуществляться после выполнения последней (шестой) команды – команды переадресации.

Задание 2. *Ответ:* искомая программа может, например, иметь следующий вид:

1) *J(2,3,9)*,

2) *S(3)*,

3) *J(5,4,7)*,

4) *S(1)*,

5) *S(5)*,

6) *J(1,1,3)*,

7) *T(1,4)*,

8) *J(1,1,1)*.

Глава 6

Задание 1. *Ответ:* например,

а) мука – лука – лупа – купа – кипа – кира – кора – корт – торт.

б) мука – рука – река – века – вера – лера – лора – пора – порт – торт.

Задание 2. *Ответ:* а) *acb*; б) неприменим.

Задание 3. *Ответ: abab*, *abaaba*; алгоритм удваивает слова.

Задание 4. *Ответ: abab*, *babbab*; алгоритм удваивает слова.

Задание 5. *Решение.* Поставим в соответствие каждой букве m алфавита A определенное слово K_m в этом алфавите. Теперь, заменяя в произвольном слове P в алфавите A каждую букву m соответствующим словом K_m, получим слово в алфавите A, которое и будет являться результатом замены в слове P букв m словами K_m.

Учитывая сказанное, построим нормальный алгоритм побуквенного кодирования:

S_α:	$\alpha m \to K_m \alpha$
	$\alpha \to .$
	$\wedge \to \alpha$

Здесь буква m пробегает алфавит A.

Например, пусть $A = \{a,b\}$, $K_a = aa$, $K_b = bab$. Перепишем нормальную схему:

S_α:	$\alpha a \to aa\alpha$
	$\alpha b \to bab\alpha$
	$\alpha \to .$
	$\wedge \to \alpha$

Опишем процесс работы нормального алгоритма над словом *bba*:

$bba \to \alpha bba \to bab\alpha ba \to babbab\alpha a \to babbabaa\alpha \to$

$\to .babbabaa$

Задание 6. *Решение.* Соответствующий алгоритм задается следующей нормальной схемой:

S_α:	$\alpha x \to x\alpha$
	$\alpha h \to \alpha$
	$\alpha \to .$
	$\wedge \to \alpha$

Здесь буква x пробегает множество B, h множество $A\backslash B$.

Например, пусть $A = \{a,b,c,d\}$, $B = \{a,c\}$, $P = abcddcba$. Тогда $A\backslash B = \{b,d\}$ и проекцией слова P в алфавите A на алфавит B будет являться слово «*acca*» в алфавите B. Соответствующий нормальный алгоритм задается следующей нормальной схемой:

S_α:	$\alpha a \to a\alpha$
	$\alpha c \to c\alpha$
	$\alpha b \to \alpha$
	$\alpha d \to \alpha$
	$\alpha \to .$
	$\wedge \to \alpha$

Задание 7.

Ответ: $\begin{cases} ca \to ac \\ cb \to bc \\ c \to .aba \\ \wedge \to c \end{cases}$

Задание 14. *Ответ: 500*, *329*, *790*; данный нормальный алгоритм вычисляет функцию $f(x) = x+1$.

Задание 15.

Ответ: $f(x) = \begin{cases} \wedge, & \text{если } x \text{ не делится на } 3, \\ 1, & \text{если } x \text{ делится на } 3. \end{cases}$

Библиографический список

1. Герасимов, А.С. Курс математической логики и теории вычислимости : учеб. пособие / А.С. Герасимов. – Санкт-Петербург : Изд-во «ЛЕМА», 2011. – 284 с.

2. Игошин, В.И. Задачи и упражнения по математической логике и теории алгоритмов : учеб. пособие / В.И. Игошин. Москва : Академия, 2007. 303 с.

3. Игошин, В.И. Математическая логика и теория алгоритмов : учеб. пособие / В.И. Игошин. Москва : Издательский центр «Академия», 2008. 448 с.

4. Крупский, В.Н. Теория алгоритмов : учеб. пособие / В.Н. Крупский, В.Е. Плиско. – Москва : Академия, 2009. – 206 с.

5. Макконнелл, Дж. Основы современных алгоритмов / Дж. Макконнелл ; под ред. С.К. Ландо. Москва : Техносфера, 2006. 368 с.

6. Мальцев, А.И. Алгоритмы и рекурсивные функции / А.И. Мальцев. Москва : Наука, 1986. 368 с.

7. Матрос, Д.Ш. Теория алгоритмов : учебник / Д.Ш. Матрос, Г.Б. Поднебесова. – Москва : БИНОМ, Лаборатория знаний, 2008. – 202 с.

8. Могилев, А.В. Информатика : учеб. пособие / А.В. Могилев, Н.И. Пак, Е.К. Хеннер ; под ред. Е.К. Хеннера. Москва : Издательский центр «Академия», 2007. – 848 с.

9. Могилев, А.В. Практикум по информатике : учеб. пособие / А.В. Могилев, Н.И. Пак, Е.К. Хеннер ; под ред. Е.К. Хеннера. М. : Издательский центр «Академия», 2001. – 608 с.

10.Носов, В.А. Основы теории алгоритмов и анализа их сложности [Электронный ресурс]. – Режим доступа: http://intsys.msu.ru/staff/vnosov/theoralg.htm

11.Поляков, В.И. Основы теории алгоритмов : учеб. пособие / В.И. Поляков, В.И. Скорубский. – Санкт-Петербург : НИУ ИТМО, 2012. – 51 с.

12.Рублев, В.С. Основы теории алгоритмов : учеб. пособие / Р.С. Рублев. – Ярославль : ЯрГУ, 2005. – 143 с.

13.Судоплатов, С.В. Математическая логика и теория алгоритмов : учебник / С.В. Судоплатов, Е.В. Овчинникова. Новосибирск : Изд-во НГТУ, 2010. 256 с.

14.Чеботарев, С.В. Теория алгоритмов : учеб. пособие / С.В. Чеботарев. Барнаул : БГПУ, 2005. 146 с.