Министерство образования и науки Российской Федерации
Ярославский государственный университет им. П. Г. Демидова
Кафедра теоретической информатики

В. С. Рублев

Алгоритмы и машины Тьюринга

(индивидуальная работа № 7 по дисциплине

«Дискретная математика»)

Учебно-методические пособие

Ярославль
ЯрГУ
2019

УДК 514(072)
ББК В126я73
Р82

Рекомендовано
Редакционно-издательским советом университета
в качестве учебного издания. План 2019 года

Рецензент:
кафедра теоретической информатики Ярославского государственного университета им. П. Г. Демидова

Рублев, Вадим Сергеевич.

Р82 Алгоритмы и машины Тьюринга : (индивидуальная работа № 7 по дисциплине «Дискретная математика») : учебно-методическое пособие / В. С. Рублев ; Яросл. гос. ун-т им. П. Г. Демидова. – Ярославль : ЯрГУ, 2019. – 64 с.

Пособие содержит варианты индивидуального задания по теме “Алгоритмы и машины Тьюринга” (дисциплина «Дискретная математика»), а также необходимый материал для ее самостоятельного изучения и выполнения индивидуальных заданий. Для качественного усвоения курса в издании даны подробные определения, примеры, иллюстрации, обоснования, а также методические рекомендации для выполнения индивидуального задания.

Пособие предназначено для студентов, обучающихся по дисциплине «Дискретная математика».

УДК 514(072)
ББК В126я73

Оглавление

1. Проблема определения алгоритма

1.1. Понятие алгоритма

Дискретное преобразование одного дискретного множества в другое может быть выражено алгоритмом.

Под *алгоритмом* решения задачи принято понимать описание вычислительного процесса, приводящего к ее решению. Этот термин обязан имени арабского математика начала IX века Мухамеда бен Мусы, по прозвищу ал-Хорезми (из Хорезма), который в своем трактате «Хисаб ал-джебр вал-мукабала» в словесной форме дал правила решения алгебраических уравнений 1-й и 2-й степени[1].

[1] Термин *алгебра* происходит от второго слова в названии трактата.

С этих пор алгоритм описывается как последовательность вычислительных шагов, каждый из которых определяет элементарные действия над исходными данными задачи и промежуточными величинами, вводимыми в описание алгоритма, а также определяет, какой шаг будет выполняться следующим. Такое определение алгоритма принято называть *неформальным*. До начала XX века казалось, что такого определения вполне достаточно. Но в 1900 году на математическом конгрессе в Париже великий немецкий математик Давид Гильберт в своем докладе о будущем развитии математики определил ряд проблем, которые XIX век оставил XX веку (эти 23 проблемы получили название *проблем Гильберта*). Некоторые из них формулировались как проблемы разработки алгоритмов. Например, десятая проблема Гильберта заключалась в разработке алгоритма решения диофантовых уравнений (алгебраические уравнения или системы уравнений с рациональными коэффициентами, решения которых ищутся в рациональных числах). Долгое время многие из этих проблем не поддавались решению, и к началу 30-х годов XX века возникло сомнение в том, можно ли построить алгоритм для их решения. Но получение математического доказательства невозможности построения алгоритма решения некоторой проблемы требует формализации понятия «алгоритм». Чтобы разобраться в трудностях формализации понятия «алгоритм», прежде всего рассмотрим свойства этого понятия, которые справедливы для указанного неформального определения и должны обязательно быть приняты во внимание при формальном определении.

В первую очередь следует отметить, что каждый алгоритм является способом решения некоторой потенциально *бесконечной совокупности* задач. Действительно, если рассматривать конечную совокупность задач, то, перенумеровав задачи, получив каким-либо образом (не обязательно алгоритмическим) решение каждой из них и перенумеровав эти решения, можно построить способ, с помощью которого каждой задаче из этой совокупности по ее номеру определяется решение с тем же номером. Такой способ выбора решения не следует понимать как алгоритм. Бесконечная совокупность задач, для которой определяется алгоритм, характеризуется выбором исходных данных каждой ее задачи из некоторого бесконечного набора данных. Отмеченное первое

свойство назовем *массовостью* алгоритма.

Второе свойство понятия «алгоритм» характеризует его как совокупность отдельных шагов и называется *дискретностью* алгоритма.

Каждый шаг алгоритма связан с входными данными шага, которыми являются как исходные данные алгоритма, так и выходные данные предыдущих выполненных шагов. Каждый шаг алгоритма связан также с выходными данными шага, которые однозначно образуются из его входных данных элементарным действием. Это характеризует третье и четвертое свойства алгоритма как *детерминированность* и *элементарность* шагов алгоритма.

Результатом выполнения шага алгоритма являются не только выходные данные шага, но и номер следующего выполняемого шага. Во многих случаях следующим при выполнении является шаг, номер которого на 1 больше, чем номер выполняемого шага. Но в некоторых случаях единственное действие алгоритма – определение следующего шага для выполнения. Это пятое свойство алгоритма называется *направленностью* алгоритма.

Число шагов алгоритма при его выполнении должно быть конечным, что составляет его шестое свойство – *конечность* числа шагов[2]. Это не означает, что при выполнении алгоритма каждый шаг должен выполняться только 1 раз. Некоторые шаги могут выполняться лишь при выполнении условия, которое проверяется на предыдущем шаге. Это дополнительное свойство называется *ветвлением* алгоритма. Некоторые шаги алгоритма могут выполняться многократно, если следующим выполняемым шагом становится один из предыдущих шагов. Такое дополнительное свойство называется *цикличностью* алгоритма. Среди шагов алгоритма обязательно должен быть шаг, *прекращающий* выполнение алгоритма. Выходные данные этого шага и являются результатом выполнения алгоритма. Если исходные данные алгоритма таковы, что при его выполнении никогда не выполняется шаг, прекращающий вычисления, то говорят, что алгоритм *зациклился* на этих исходных данных и что алгоритм *недопустим* для этих исходных данных (некорректные данные) [3].

Рассмотрим в качестве примера описание *схемы Горнера* вычисле-

[2] Это свойство иногда называют *результативностью* алгоритма.

[3] Корректный алгоритм должен отвергать некорректные данные.

ния значения полинома $y = a_0 \cdot x^n + a_1 \cdot x^{n-1} + ... + a_{n-1} \cdot x + a_n$. Исходными данными в этом алгоритме являются степень полинома n, вектор коэффициентов $\{a_0, a_1, ..., a_{n-1}, a_n\}$ и значение аргумента x. Описание алгоритма дается следующей последовательностью шагов:

1. Положить $y = a_0,\ k = 1$.
2. Если $k > n$, перейти к шагу 4.
3. Положить $y = y \cdot x + a_k,\ k = k + 1$; перейти к шагу 2.
4. Закончить вычисления с результатом y.

Массовость этого алгоритма демонстрируется его применением к любому полиному и любому аргументу из бесконечной совокупности полиномов и значений аргументов. Дискретность и конечность обеспечиваются выделением 4 шагов алгоритма. Элементарность шагов алгоритма следует из простоты вычислений каждого шага. Направленность обеспечивается указанием следующего шага по умолчанию (для шага 1 и для шага 2, если не выполнено условие), либо указанием следующего шага (для шага 3 и для шага 2, если выполнено условие), либо указанием прекращения вычислений (шаг 4). Дополнительное свойство ветвления алгоритма использовано на шаге 2, а свойство цикличности алгоритма использовано в организации цикла шагов 2 и 3.

С процессом выполнения алгоритма связано понятие *конфигурации* выполнения. Конфигурация описывается номером шага при выполнении алгоритма, значением всех исходных данных и всех промежуточных величин алгоритма и результата, записанных в определенном порядке. Так, для вышеописанного примера определим конфигурацию как следующую совокупность данных:

$$< N, x, n, a_0, a_1, ..., a_n, k, y >,$$

где N – номер шага. Перед выполнением алгоритма $N = 0$. В результате вычисления каждого шага конфигурация изменяется. Выполнению алгоритма соответствует последовательность конфигураций. Например, для вычисления значения $y = 5^2 - 2 \cdot 5 + 1$ следующая последовательность конфигураций описывает выполнение алгоритма:

$$< 0, 5, 2, 1, -2, 1, ?,\ ? >$$

$$< 1, 5, 2, 1, -2, 1, 1,\ 1 >$$
$$< 2, 5, 2, 1, -2, 1, 1,\ 1 >$$
$$< 3, 5, 2, 1, -2, 1, 2,\ 3 >$$
$$< 2, 5, 2, 1, -2, 1, 2,\ 3 >$$
$$< 3, 5, 2, 1, -2, 1, 3, 16 >$$
$$< 2, 5, 2, 1, -2, 1, 3, 16 >$$
$$< 4, 5, 2, 1, -2, 1, 3, 16 > .$$

Подводя итог анализу свойств алгоритма, мы можем уточнить его определение:

> *Алгоритм есть описание вычислительного процесса решения задачи, обладающее свойствами массовости, дискретности, детерминированности, направленности, конечности и элементарности.*

1.2. Неформальное пошаговое описание алгоритма

Для описания алгоритма существует много различных языков. Их можно разделить на 2 группы: неформальные и формальные языки. К первым относятся такие языки, как *язык блок-схем* и *язык пошагового описания алгоритма*. Ко вторым относятся языки программирования процедурного типа и языки логического программирования.

Языки программирования разнообразны, позволяют точно описывать алгоритм для выполнения на компьютере, но их возможности зависят от компьютера, на котором они выполняются. То есть имеются технические ограничения на память и быстродействие компьютера, которые могут затруднить выполнение некоторых алгоритмов. Кроме этого, перенос алгоритма с одного компьютера на другой не является простым. Хотя возможны программы конвертации для некоторых пар языков программирования, во-первых, это требует построения таких программ конвертации, что уже делает описание алгоритма не универсальным, а, во-вторых, иногда зависит от технических характеристик компьютеров.

Неформальные языки являются, как правило, универсальными, и разработка алгоритма на них позволяет затем сравнительно более просто

кодировать программу для того или иного языка программирования. Для кодирования программы на любом языке программирования достаточно лишь среднее образование программиста, так как техническое знание конструкций того или иного языка программирования (особенно процедурного типа) не требует такого уровня логического мышления, который требуется для разработки алгоритма или его тестирования.

Язык блок-схем является наглядным для разработки простых алгоритмов, но с увеличением сложности требуется детализация блоков, что приводит к потере основного преимущества блок-схем — наглядности. При разработке пошагового неформального описания алгоритма используется технология *сверху-вниз*, при которой сначала весь алгоритм разбивается на несколько крупных шагов, а затем каждый из сложных шагов детализируется своим пошаговым описанием, и в этом описании детализации снова может быть повторена идея последующей детализации уже ее сложных шагов и т. д. Это позволяет проектировать алгоритм, постепенно разрабатывая и как бы заменяя каждую из сложных частей на ее детализацию. В этом состоит основная *общая* идея *декомпозиции* сложных задач на более простые. Для связи детализируемого шага с шагами его детализации вводится сквозная нумерация шагов частей алгоритма, при которой номеру каждого шага детализации предшествует номер детализируемого шага и оба номера разделены точкой [1].

Любой алгоритм после разработки своего описания должен быть обоснован. Таким обоснованием является *тестирование алгоритма* на достаточно большом количестве разнообразных исходных данных. Каждая совокупность исходных данных и ожидаемых результатов называется *тестом*, а процесс тестирования состоит из выполнения алгоритма для исходных данных теста и сравнения полученных результатов такого выполнения с ожидаемыми результатами теста. Множество тестов, достаточное для обоснования алгоритма, называется *полным набором тестов*. Полный набор тестов должен содержать достаточное разнообразие тестов по данным, чтобы отразить все возможные случаи данных, на которые рассчитан алгоритм (*полнота по данным*), а также все случаи проверки выполнения всех его ветвей (*полнота по*

[1] Ниже это иллюстрируется примером разработки алгоритма

ветвям). Разработка полного набора тестов полезна уже до разработки описания, так как помогает увидеть разнообразие исходных данных и учесть его при разработке алгоритма. Поэтому разработка полного набора тестов является первым этапом разработки алгоритма, а всего их 3:

1. Разработка полного набора тестов.
2. Разработка описания алгоритма.
3. Тестирование алгоритма на полном наборе тестов.

Следует учесть, что на этапе 2 (разработка описания) может быть выяснено, что набор, разработанный на этапе 1 (набор тестов), не является полным. В этом случае следует его пополнить.

Для лучшего понимания описания этих этапов мы будем иллюстрировать их на примере разработки алгоритма следующей процедуры:

Дана прямоугольная целочисленная матрица. Если сумма элементов всех внутренних столбцов матрицы равна сумме элементов всей матрицы, то часть матрицы, состоящую из элементов внутренних столбцов матрицы, следует повернуть на 180^0.

1.2.1. Разработка полного набора тестов

Начинать разработку полного набора тестов следует с описания условий на тесты: надо выявить все параметры данных алгоритма. Так, например, это размерности матрицы и массив элементов матрицы.

Все тесты можно разбить на *крайние тесты*, для которых те или иные параметры алгоритма вырождены, и *общие тесты*, для которых нет вырожденности параметров и которые в совокупности покрывают при выполнении все ветви алгоритма. По данным общий тест должен иметь различные значения для элементов, которые обмениваются своими значениями. Нулевые значения элементов нежелательны в общем тесте, если эти значения суммируются или входят в произведение. Помимо этого, необходимо следить, чтобы в общих тестах не было особых значений функций, связанных с условиями задачи (например, суммы, произведения и т. д.). Очень важно проверить алгоритм и на общих, и на крайних тестах.

Для рассматриваемого примера размерности вырождены, если имеется всего 1 строка или менее 3 столбцов матрицы (в последнем случае условие задания либо выполняется тривиально, либо не выполняется, но в обоих случаях матрица не преобразуется). В случае 3 столбцов также имеется вырожденность по ветвям алгоритма, так как центральная строка, если она есть, не преобразуется. Центральная строка преобразуется иначе, чем другие строки. Поэтому для общего теста число строк должно быть нечетным и не меньшим 3. В случае общего теста центральный элемент матрицы (в средних строке и столбце) является особым: его значение не изменяется. Поэтому общий тест должен содержать не менее 5 столбцов. Далее, если значения элементов внутренних столбцов одинаково, то ожидаемый результат преобразованной матрицы не отличается от исходной матрицы. Поэтому желательно, чтобы элементы части матрицы из внутренних столбцов были различны (по крайней мере, элементы, меняющиеся своими значениями).

Условия на тесты и их обоснование в обязательном порядке должны быть выписаны перед составлением тестов. Составление тестов необходимо начать с общих тестов. Если один общий тест не может покрыть все ветви алгоритма, то к нему добавляются еще общие тесты. Однако надо стремиться к минимизации количества тестов и объема их данных.

В заключение этого пункта приведем полный набор тестов для указанного примера задания.

Тест 1 (общий). Матрица размерностей 3 на 5. Условие задания выполнено: сумма всех элементов матрицы 45 равна сумме всех элементов внутренних столбцов. Все элементы преобразуемой части различны. В исходной и результирующей матрице элементы соответственно преобразуемых и преобразованных частей матрицы выделены курсивом.

1	*8*	*7*	*6*	4
-4	*5*	*4*	*3*	0
7	*2*	*1*	*9*	-8

$\Rightarrow$

1	*9*	*1*	*2*	4
-4	*3*	*4*	*5*	0
7	*6*	*7*	*8*	-8

Этого теста недостаточно в качестве общего, так как в общем тесте должны быть случаи, не только когда условие задания выполняется, но и когда условие задания не выполняется. Поэтому введем дополнительный общий тест.

Тест 2 (общий). Матрица размерностей 3 на 5. Условие задания не выполнено: сумма всех элементов матрицы 44 не равна сумме всех элементов внутренних столбцов 45. Поэтому результирующая матрица равна исходной.

$$\begin{bmatrix} 1 & 8 & 7 & 6 & 3 \\ -4 & 5 & 4 & 3 & 0 \\ 7 & 2 & 1 & 9 & -8 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 8 & 7 & 6 & 3 \\ -4 & 5 & 4 & 3 & 0 \\ 7 & 2 & 1 & 9 & -8 \end{bmatrix}$$

Перейдем теперь к проектированию крайних тестов. При одной строке преобразование упрощается — переворачивается внутренняя часть строки, если она имеет более одного элемента и условие выполнено. При числе столбцов, меньше 3, матрица не изменяется независимо от выполнения или невыполнения условия.

Тест 3 (крайний). Матрица из 1-й строки. Изменяется при выполнении условия вырожденным случаем: переворачивается внутренняя часть строки.

$$\begin{bmatrix} 1 & 8 & 6 & -1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 6 & 8 & -1 \end{bmatrix}$$

Тест 4 (крайний). Матрица из двух столбцов не изменяется независимо от выполнения условия, так как нет внутренних столбцов.

$$\begin{bmatrix} 1 & -1 \\ -2 & 2 \\ 3 & -4 \\ 4 & -3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & -1 \\ -2 & 2 \\ 3 & -4 \\ 4 & -3 \end{bmatrix}$$

Тест 5 (крайний). Матрица из трех столбцов при выполнении условия изменяется упрощенным способом: переворачивается внутренний столбец.

$$\begin{bmatrix} 1 & 1 & -1 \\ -2 & 2 & 2 \\ 3 & 3 & -4 \\ 4 & 4 & -3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 4 & -1 \\ -2 & 3 & 2 \\ 3 & 2 & -4 \\ 4 & 1 & -3 \end{bmatrix}$$

Сформулируем коротко требования по разработке полного набора тестов:

1. Начинать надо с выписывания условий на полный набор тестов.
2. После этого нужно разработать общий тест, отвечающий условиям по разнообразию данных и по выполнению всех ветвей алгоритма.
3. Если общий тест не покрывает все условия по данным или по ветвям алгоритма, то следует дополнить его еще другими дополнительными общими тестами в минимальном количестве.
4. Сформировать крайние тесты для каждого случая крайних значений параметров данных.

1.2.2. Разработка описания пошагового алгоритма

В описании пошагового алгоритма процедуры нет шагов, связанных с описанием данных (исходных, результатов, рабочих), так как это не является действиями шагов алгоритма. Но их обозначения и смысл должны быть определены перед пошаговым описанием. Поэтому введем преамбулу описания алгоритма (назовем ее ***Идеи алгоритма***), в которой определим обозначения всех данных алгоритма и их смыслы (например, параметры исходных данных и результата, параметры индексов циклов и накапливаемых сумм и т. д.), а также идеи, повышающие эффективность алгоритма. Так, это могут быть особые способы вычисления выражений условий, сложных индексных выражений и т.д. Помимо этого, в идеях алгоритма могут задаваться имена выделенных процедур, осуществляющих вычисления каких-то частей алгоритма.

Рассмотрим вышеуказанный пример задания алгоритма процедуры. В части ***Идеи алгоритма*** исходными данными являются параметры процедуры, но их обозначения и смысл надо определить. Пусть m – число строк, n – число столбцов матрицы и A – массив элементов матрицы, каждый элемент A_{ij} которой имеет индексы i и j, изменяющиеся соответственно от 0 до $m-1$ и от 0 до $n-1$. Результирующая матрица заменяет исходную. Для накопления суммы значений элементов используется переменная S, а для обмена значениями элементов – переменная r. Для проверки условия преобразования матрицы задается процедура-функция ***Condition***, возвращающая значение *истина*, если условие преобразования выполнено, или *ложь* в ином случае. Для

проверки условия преобразования матрицы используем идею, согласно которой сумма значений элементов всех некрайних столбцов матрицы равна сумме значений элементов всей матрицы тогда и только тогда, когда сумма значений элементов двух крайних столбцов матрицы равна 0. Алгоритм вычисления суммы значений элементов двух крайних столбцов проще и имеет на порядок меньшую вычислительную сложность. Поэтому его мы и используем для процедуры-функции ***Condition***.

Перейдем теперь к разработке описания пошагового алгоритма. Каждый шаг описания нумеруется, полностью определяет *действия* этого шага и *номер* выполнения *следующего шага*. По умолчанию следующим является шаг со следующим номером. Если некоторый шаг *сложный* (содержит несколько действий), то он *детализируется* разбиением на несколько шагов. Номера шагов детализации состоят из номера детализируемого шага, за которым после разделяющей точки идет номер шага детализации. *Детализация не является вложенными шагами детализируемого шага*, а как бы заменяет этот шаг, но пишется всегда отдельно, что делает разработку алгоритма более наглядной. Для вызова процедуры может быть разработана детализация алгоритма этой процедуры, но чаще всего делается отдельное описание алгоритма процедуры с отдельной нумерацией шагов. Разработку детализации шагов некоторого алгоритма можно начинать только после окончания разработки предыдущего более высокого уровня описания алгоритма или его части и проверки его правильности. Перед детализацией шага идет заголовок *Детализация такого-то шага такого-то алгоритма*. Некоторые из шагов детализации могут оказаться сложными, и в таком случае они также детализируются.

Таким образом, разработка описания алгоритма начинается с общего алгоритма, в котором шаги нумеруются подряд от 1. Если потребуется детализация шага 2, то нумерация шагов этой детализации идет как 2.1, 2.2,... Если потребуется детализация шага 2.2 этой детализации, то нумерация ее шагов идет как 2.2.1, 2.2.2,..., и т. д. .

Особо следует отметить шаги, связанные с организацией цикла. Язык программирования может иметь сложные конструкции организации цикла с параметром. Такой цикл осуществляет начальное задание параметра, проверку условия его выполнения и изменение параметра

цикла после выполнения тела цикла. Но в пошаговом алгоритме таких конструкций нет, так как такое описание предназначено для кодирования алгоритмов на любых языках. Поэтому организацию такого цикла можно задать при помощи нескольких шагов:

1) начальное значение параметра цикла;

2) шаги выполнения тела цикла;

3) шаг изменения параметра цикла и проверки условия выполнения тела цикла; если оно выполнено, то перейти на первый шаг тела цикла, а иначе перейти к следующему шагу.

Такая организация имеет варианты, когда, например, проверка условия выполнения тела цикла идет сразу после задания начального значения параметра цикла, а переход при невыполнении условия осуществляется на шаг после последнего шага тела цикла и при выполнении – на следующий шаг, т. е. первый шаг тела цикла. При кодировании кодировщик сразу сможет заменить эти шаги на конструкцию с параметром цикла.

Каждая законченная часть описания алгоритма должна быть *проверена прокруткой* на общих тестах алгоритма или на специальных тестах для этого. Эта прокрутка не является этапом 3 разработки алгоритма (тестирование алгоритма), который обосновывает алгоритм. Ее можно не описывать, но обязательно провести прокрутку либо в уме, либо где-то отдельно.

Перейдем теперь к разработке пошагового описания алгоритма вышеуказанного примера.

Общий алгоритм

1. Проверить условия отсутствия внутренних столбцов; если $n < 3$, то закончить выполнение алгоритма.

2. Проверить условия равенства 0 суммы значений элементов в крайних столбцах матрицы вызовом функции ***Condition***. Если функция возвращает значение *ложь*, то закончить выполнение алгоритма.

3. Выполнить преобразования части матрицы из внутренних столбцов поворотом этой части на 180^0. Конец алгоритма.

Алгоритм функции Condition

1. Положить начальное значение суммы $S = 0$.
2. Положить начальное значение номера строки матрицы $i = 0$.
3. Добавить к сумме значения элементов крайних столбцов в i-й строке $S = S + A_{i,1} + A_{i,n-1}$.
4. Изменить номер строки $i = i + 1$ и, если $i < n$, перейти к шагу 3.
5. Если $S = 0$, то возвратить *истину*, иначе возвратить *ложь*.

Детализация шага 3 общего алгоритма

3.1 Положить начальное значение номера строки $i = 0$.

3.2 Если $i \geq (m-1)/2$, перейти к шагу 3.7 (переход к редактированию централной строки)

3.3 Положить начальное значение номера столбца $j = 1$.

3.4 Обменять значения элементов $A_{i,j}$ и $A_{m-1-i,n-1-j}$
(обмен значениями всех элементов внутренних столбцов, кроме центральной строки).

3.5 Увеличить номер столбца $j = j + 1$; если $j < n - 1$, то перейти к шагу 3.4

3.6 Увеличить номера строки $i = i + 1$; перейти к шагу 3.2

3.7 Если m – четное, то закончить алгоритм (нет центральной строки).

3.8 Положить начальное значение номера столбца $j = 1$.

3.9 Обменять значения элементов $A_{i,j}$ и $A_{i,n-1-j}$
(обмен значениями всех элементов центральной строки, кроме центрального столбца).

3.10 Увеличить номера столбца $j = j + 1$;
если $j < (n-1)/2$, то к шагу 3.9.

Для обмена значениями элементов $A_{i,j}$ и $A_{m-1-i,n-1-j}$ для шагов 3.4 и 3.9 рационально ввести процедуру ***ExchangeElem***.

Алгоритм процедуры *ExchangeElem*

1. $r = A_{i,j}$.
2. $A_{i,j} = A_{m-1-i,n-1-j}$.
3. $A_{m-1-i,n-1-j} = r$.

Отметим также, что не следует в некоторых случаях при описании пошагового алгоритма использовать конструкции языков программирования, так как пошаговый алгоритм не зависит от языка программирования, а в том или ином языке программирования использованная конструкция может быть нерациональной. Так, в описанном алгоритме примера используется описание *"m – четное"*. Если его заменить на конструкцию условия $m = 2 * (m/2)$, то она может оказаться неверной и, во всяком случае, не самой эффективной, так как выполняются 2 медленных арифметических операции (умножение и деление – последнее еще медленнее). Эффективной конструкцией в языке C является операция логического умножения $m\&1$, которая возвращает младший бит m (при нечетности m равный 1, а при четности – 0). Но сравнивать с 1 или 0 также не нужно, так как данное выражение автоматически интерпретируется как булево. Поэтому кодировать условие четности m можно, добавив в конструкцию операцию отрицания $!(m\&1)$, и тогда она возвратит значение ***true*** только при четности m. Эти логические операции являются самыми быстрыми в компьютере. Поэтому вопрос эффективного кода следует отнести к стадии кодирования.

1.2.3. Тестирование пошагового алгоритма

Тестирование алгоритма есть формальная прокрутка всех тестов полного набора. Формальная прокрутка означает, что, выполняя каждый шаг алгоритма, мы должны строго следовать описанию действий этого шага, включая и выбор номера следующего шага. Возникшие затруднения означают ошибку в четкости описания действий шага. В этом случае нужно внести исправления, после чего все прокрученные тесты, содержащие этот шаг, прокрутить заново.

Перед началом прокрутки надо вновь выписать для теста его исходные данные и ожидаемые результаты. При выполнении прокрутки шага необходимо выписать действие шага по изменению всех данных, выполняемых на этом шаге. При этом следует писать обозначение данного, согласно действию шага, и в скобках – его новое значение (даже если оно совпадает со старым значением).

После завершения прокрутки выписать по всем измененным данным результат и сравнить его с ожидаемым. Если сравнение даст положительный результат, то написать «тест прошел успешно» и перейти к прокрутке следующего теста.

Для того, чтобы уменьшить объем тестирования, его нужно начинать с алгоритмов самого низкого уровня – уровня детализаций и процедур, выбирая каждый раз разумное количество из имеющегося полного набора тестов или создавая на основе тестов полного набора свой полный набор для этого алгоритма. После этого нужно перейти к тестированию алгоритмов более высокого уровня, для которых все детализации и вызываемые процедуры прокручены. Но при этом каждый шаг, который был детализирован, или вызов процедуры выполняется как один шаг. Это необходимо продолжать, пока не будет прокручен для всего полного набора тестов общий алгоритм.

При положительном результате прокрутки всех алгоритмов описания надо сделать вывод о том, что «прокрутка завершена успешно».

Перейдем к тестированию алгоритма примера.

Тестирование алгоритма процедуры *ExchangeElem*

Тест 1.

1	*8*	*7*	**6**	4		1	*9*	*1*	**2**	4
-4	*5*	*4*	*3*	0	⇒	-4	*3*	*4*	*5*	0
7	**2**	*1*	*9*	-8		7	**6**	*7*	*8*	-8

Для $m = 3, n = 5, i = 0, j = 3$ в исходной матрице и в результирующей матрице выделены полужирным шрифтом меняющиеся элементы $A_{0,3}$ и $A_{2,1}$.

1. $r = A_{i,j}$ $r = A_{0,3}(6)$.
2. $A_{i,j} = A_{m-1-i,n-1-j}$ $A_{0,3} = A_{2,1}(2)$.
3. $A_{m-1-i,n-1-j} = r$ $A_{2,1} = r(6)$.

Результат $A_{0,3} = 2, A_{2,1} = 6$ совпадает с ожидаемым. *Тест прошел успешно.*

Тестирование детализации шага 3 общего алгоритма

Тест 1.

$$\begin{bmatrix} 1 & 8 & 7 & 6 & 4 \\ -4 & 5 & 4 & 3 & 0 \\ 7 & 2 & 1 & 9 & -8 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 9 & 1 & 2 & 4 \\ -4 & 3 & 4 & 5 & 0 \\ 7 & 6 & 7 & 8 & -8 \end{bmatrix}$$

$m = 3, n = 5$. Данные теста для этого алгоритма выделены курсивом в исходной матрице и в ожидаемой результирующей матрице.

3.1 Начальное значение номера строки $i = 0$.

3.2 Если $i \geq (m-1)/2$ $(0 \geq 1)$, то к шагу 3.7 (не выполнено)

3.3 Начальное значение номера столбца $j = 1$.

3.4 Обмен знач. элем. $A_{i,j}$ (8) и $A_{m-1-i,n-1-j}$ (9) $A_{0,1} = 9,\ A_{2,3} = 8$.

3.5 Увелич. ном. столб. $j = j + 1(2)$; если $j < n - 1(2 < 4)$, то к шагу 3.4.

3.4 Обмен знач. элем. $A_{i,j}$ (7) и $A_{m-1-i,n-1-j}$ (1) $A_{0,2} = 1,\ A_{2,2} = 7$.

3.5 Увелич. ном. столб. $j = j + 1(3)$; если $j < n - 1(3 < 4)$, то к шагу 3.4.

3.4 Обмен знач. элем. $A_{i,j}$ (6) и $A_{m-1-i,n-1-j}$ (2) $A_{0,3} = 2,\ A_{2,1} = 6$.

3.5 Увелич. ном. столб. $j = j + 1$ (4); если $j < n - 1$ $(4 < 4)$, то к шагу 3.4 (не выполнено).

3.6 Увелич. ном. стр. $i = i + 1$ (2); переход к шагу 3.2.

3.2 Если $i \geq (m-1)/2$ $(1 \geq 1)$, то к шагу 3.7.

3.7 Если m – четное (3 - неч.), то конец алгоритма (не выполнено).

3.8 Начальное значение номера столбца $j = 1$.

3.9 Обмен знач. элем. $A_{i,j}$ (5) и $A_{i,n-1-j}$ (3) $A_{1,1} = 3,\ A_{1,3} = 5$.

3.10 Увелич. ном. столб. $j = j + 1$ (2); если $j < (n-1)/2$ $(2 < 2)$, то к шагу 3.9 (не выполнено).

В результате прокрутки получена матрица

$$\begin{bmatrix} 1 & 9 & 1 & 2 & 4 \\ -4 & 3 & 4 & 5 & 0 \\ 7 & 6 & 7 & 8 & -8 \end{bmatrix}$$

Она полностью совпадает с ожилаемой. *Тест прошел успешно.*

В прокрутке теста 1 выполняются не все ветви алгоритма детализации шага 2. Надо проверить ветвь и с четным m. Поэтому в качестве

второго теста для этого алгоритма возьмем тест 5 ($m = 4, n = 3$). Данные теста для этого алгоритма выделены курсивом в исходной матрице и в ожидаемой результирующей матрице.

Тест 5.

$$\begin{bmatrix} 1 & \mathit{1} & -1 \\ -2 & \mathit{2} & 2 \\ 3 & \mathit{3} & -4 \\ 4 & \mathit{4} & -3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & \mathit{4} & -1 \\ -2 & \mathit{3} & 2 \\ 3 & \mathit{2} & -4 \\ 4 & \mathit{1} & -3 \end{bmatrix}$$

3.1 Начальное значение номера строки $i = 0$.

3.2 Если $i \geq (m-1)/2$ $(0 \geq 1.5)$, то к шагу 3.7 (не выполнено).

3.3 Начальное значение номера столбца $j = 1$.

3.4 Обмен знач. элем. $A_{i,j}$ (1) и $A_{m-1-i,n-1-j}$ (4) $A_{0,1} = 4,\ A_{3,1} = 1$.

3.5 Увелич. ном. столб. $j = j + 1(2)$; если $j < n - 1(2 < 2)$, то к шагу 3.4 (не выполнено).

3.6 Увелич. ном. стр. $i = i + 1$ (2); переход к шагу 3.2.

3.2 Если $i \geq (m-1)/2$ $(1 \geq 1.5)$, то к шагу 3.7 (не выполнено).

3.3 Начальное значение номера столбца $j = 1$.

3.4 Обмен знач. элем. $A_{i,j}$ (2) и $A_{m-1-i,n-1-j}$ (3) $A_{1,1} = 3,\ A_{2,1} = 2$.

3.5 Увелич. ном. столб. $j = j + 1(2)$; если $j < n - 1(2 < 2)$, то к шагу 3.4 (не выполнено).

3.6 Увелич. ном. стр. $i = i + 1$ (3); переход к шагу 3.2.

3.2 Если $i \geq (m-1)/2$ $(2 \geq 1.5)$. то к шагу 3.7.

3.7 Если m – четное (4 - чет.), то конец алгоритма.

В результате прокрутки получена матрица

$$\begin{bmatrix} 1 & \mathit{4} & -1 \\ -2 & \mathit{3} & 2 \\ 3 & \mathit{2} & -4 \\ 4 & \mathit{1} & -3 \end{bmatrix}$$

Она полностью совпадает с ожидаемой. *Тест прошел успешно.* Поскольку оба теста покрывают все ветви алгоритма, то *прокрутка алгоритма детализации шага 3 общего алгоритма завершена успешно.*

Тестирование алгоритма функции *Condition*

Тест 1.

1	*8*	*7*	*6*	4
-4	*5*	*4*	*3*	0
7	*2*	*1*	*9*	-8

$\Rightarrow$ ***true***

$m = 3, n = 5$. Данные теста для этого алгоритма выделены прямым шрифтом в исходной матрице. Ожидаемым результатом является значение ***true***.

1. Начальное значение суммы $S = 0$.

2. Начальное значение номера строки матрицы $i = 0$.

3. Добавление к сумме значений элементов крайних столбцов в i-й строке $S = S + A_{i,1} + A_{i,n-1} = 0 + 1 + 4 = 5$.

4. Изменение номера строки $i = i + 1$ (1) и, если $i < n$ $(1 < 3)$, переход к шагу 3.

3. Добавление к сумме значений элементов крайних столбцов в i-й строке $S = S + A_{i,1} + A_{i,n-1} = 5 - 4 + 0 = 1$.

4. Изменение номера строки $i = i + 1$ (2) и, если $i < n$ $(2 < 3)$, переход к шагу 3.

3. Добавление к сумме значений элементов крайних столбцов в i-й строке $S = S + A_{i,1} + A_{i,n-1} = 1 + 7 - 8 = 0$.

4. Изменение номера строки $i = i + 1$ (3) и, если $i < n$ $(3 < 3)$, переход к шагу 3 (не выполнено).

5. Если $S = 0$, то возвратить *истину*, иначе возвратить *ложь* ($S = 0$, возвращается ***true***).

Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 2.

$m = 3, n = 5$. Данные теста для этого алгоритма выделены прямым шрифтом в исходной матрице. Ожидаемым результатом является значение ***false***.

1	*8*	*7*	*6*	3
-4	*5*	*4*	*3*	0
7	*2*	*1*	*9*	-8

$\Rightarrow$ ***false***

1. Начальное значение суммы $S = 0$.

2. Начальное значение номера строки матрицы $i = 0$.

3. Добавление к сумме значений элементов крайних столбцов в i-й строке $S = S + A_{i,1} + A_{i,n-1} = 0 + 1 + 3 = 4$.

4. Изменение номера строки $i = i + 1$ (1) и, если $i < n$ $(1 < 3)$, переход к шагу 3.

3. Добавление к сумме значений элементов крайних столбцов в i-й строке $S = S + A_{i,1} + A_{i,n-1}$=4-4+0=0.

4. Изменение номера строки $i = i + 1$ (2) и, если $i < n$ $(2 < 3)$, переход к шагу 3.

3. Добавление к сумме значений элементов крайних столбцов в i-й строке $S = S + A_{i,1} + A_{i,n-1} = 0 + 7 - 8 = -1$.

4. Изменение номера строки $i = i + 1$ (3) и, если $i < n$ $(3 < 3)$, переход к шагу 3 (не выполнено).

5. Если $S = 0$, то возвратить *истину*, иначе возвратить *ложь* ($S = -1$, возвращается ***false***).

Результат совпадает с ожидаемым. *Тест прошел успешно.* Поскольку оба теста покрывают все ветви алгоритма, то *прокрутка алгоритма функции* ***Condition*** *завершена успешно.*

Тестирование общего алгоритма

Тест 1.

1	*8*	*7*	*6*	4
-4	*5*	*4*	*3*	0
7	*2*	*1*	*9*	-8

$\Rightarrow$

1	*9*	*1*	*2*	4
-4	*3*	*4*	*5*	0
7	*6*	*7*	*8*	-8

$m = 3, n = 5$. Данные преобразованной части выделены курсивом в исходной матрице и в ожидаемой результирующей матрице.

1. Проверка условия отсутствия внутренних столбцов;
если $n < 3$ $(5 < 2)$, то закончить выполнение алгоритма (не выполнено).

2. Проверка условия равенства 0 суммы значений элементов в крайних столбцах матрицы вызовом функции ***Condition***. Если функция возвращает значение *ложь*, то закончить выполнение алгоритма (сумма = 0, функция возвращает *истину*).

3. Выполнение преобразования части матрицы из внутренних столбцов поворотом этой части на 180^0. Конец алгоритма.

Результат выполнения алгоритма

1	*9*	*1*	*2*	4
-4	*3*	*4*	*5*	0
7	*6*	*7*	*8*	-8

совпадает с ожидаемым. *Тест прошел успешно.*

Тест 2.

1	*8*	*7*	*6*	3
-4	*5*	*4*	*3*	0
7	*2*	*1*	*9*	-8

$\Rightarrow$

1	*8*	*7*	*6*	3
-4	*5*	*4*	*3*	0
7	*2*	*1*	*9*	-8

$m = 3, n = 5$. Данные преобразованной части выделены курсивом в исходной матрице и в ожидаемой результирующей матрице.

1. Проверка условия отсутствия внутренних столбцов;
если $n < 3$ $(5 < 2)$, то закончить выполнение алгоритма алгоритма (не выполнено).

2. Проверка условия равенства 0 суммы значений элементов в крайних столбцах матрицы вызовом функции ***Condition***. Если функция возвращает значение *ложь*, то закончить выполнение алгоритма
(сумма = -1, функция возвращает *ложь*).

Результат выполнения алгоритма

1	*9*	*1*	*2*	4
-4	*3*	*4*	*5*	0
7	*6*	*7*	*8*	-8

совпадает с ожидаемым. *Тест прошел успешно.*

Тест 3 (крайний). Матрица из 1 строки: $m = 1, n = 4$.

1	*8*	*6*	-1

$\Rightarrow$

1	*6*	*8*	-1

совпадает с ожидаемым. *Тест прошел успешно.*

1. Проверка условия отсутствия внутренних столбцов;
если $n < 3$ $(1 < 2)$, то закончить выполнение алгоритма.

2. Проверка условия равенства 0 суммы значений элементов в крайних столбцах матрицы вызовом функции ***Condition***. Если функция возвращает значение *ложь*, то закончить выполнение алгоритма
(сумма = 0, функция возвращает *истину*).

3. Выполнение преобразования части матрицы из внутренних столбцов поворотом этой части на 180^0. Конец алгоритма.

Результат выполнения алгоритма

1	*6*	*8*	-1

совпадает с ожидаемым. *Тест прошел успешно.*

Тест 4 (крайний). Матрица из двух столбцов не изменяется независимо от выполнения условия, так как нет внутренних столбцов.

$$\begin{bmatrix} 1 & -1 \\ -2 & 2 \\ 3 & -4 \\ 4 & -3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & -1 \\ -2 & 2 \\ 3 & -4 \\ 4 & -3 \end{bmatrix}$$

1. Проверка условия отсутствия внутренних столбцов;
если $n < 3$ $(1 < 2)$, то закончить выполнение алгоритма.

Результат выполнения алгоритма

1	-1
-2	2
3	-4
4	-3

совпадает с ожидаемым. *Тест прошел успешно.*

Тест 5 (крайний). Матрица из трех столбцов $(m = 4, n = 3)$ при выполнении условия изменяется упрощенным способом – переворачивается внутренний столбец.

$$\begin{bmatrix} 1 & 1 & -1 \\ -2 & 2 & 2 \\ 3 & 3 & -4 \\ 4 & 4 & -3 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 4 & -1 \\ -2 & 3 & 2 \\ 3 & 2 & -4 \\ 4 & 1 & -3 \end{bmatrix}$$

1. Проверка условия отсутствия внутренних столбцов;
если $n < 3$ $(3 < 2)$, то закончить выполнение алгоритма (условие не выполнено).

2. Проверка условия равенства 0 суммы значений элементов в крайних столбцах матрицы вызовом функции ***Condition***. Если функция

возвращает значение *ложь*, то закончить выполнение алгоритма (сумма = 0, функция возвращает *истину*).

3. Выполнение преобразования части матрицы из внутренних столбцов поворотом этой части на 180^0. Конец алгоритма.

Результат выполнения алгоритма

1	*4*	-1
-2	*3*	2
3	*2*	-4
4	*1*	-3

совпадает с ожидаемым. *Тест прошел успешно.* Поскольку все тесты покрывают все ветви алгоритма, то *прокрутка алгоритма функции* ***Condition*** *завершена успешно.*
Тестирование алгоритмов завершилось успешно.

1.3. Проблемы универсального формального описания алгоритма

С информационной точки зрения алгоритм есть средство переработки информации. Для каждого набора входной информации, который является корректными исходными данными для алгоритма, он (алгоритм) однозначно определяет выходную информацию. Эта функциональная связь наборов входной информации с наборами выходной информации позволяет нам рассматривать алгоритм как частичную функцию, которую мы будем называть *интуитивно вычислимой* функцией. Основной вопрос теории алгоритмов можно сформулировать следующим образом: *является ли любая заданная функция интуитивно вычислимой*, т. е. для любой ли функции можно построить алгоритм, который осуществляет вычисление значения этой функции? Этот вопрос эквивалентен следующему: *если произвольная задача предполагает однозначное решение для своих исходных данных, то всегда ли существует алгоритм решения этой задачи*? Для ответа на этот вопрос необходимо знать точно, *что такое алгоритм.* Поэтому прежде всего нужно заняться формализацией понятия «алгоритм».

Проанализируем возможность формализации алгоритма, при которой сохраняются свойства неформального определения. Не вызывает

трудностей формализация исходных данных и промежуточных величин алгоритма. Свойства дискретности, направленности и конечности числа шагов алгоритма обеспечиваются последовательной декомпозицией алгоритма, нумерацией его шагов и определением следующего выполняемого шага, что также можно формализовать. Но требование элементарности шага алгоритма вызывает трудность. Не ясно, какие действия следует считать элементарными. Если зафиксировать формально действия, которые мы считаем элементарными, то возникает опасность, не сузили ли мы такой формализацией возможность описывать любые алгоритмы. В этом состоит основная проблема формализации понятия «алгоритм».

Начиная с 30-х г. XX в. было разработано много подходов формализации понятия «алгоритм», такие как частично-рекурсивные функции (С. К. Клини, К. Гёдель, А. Чёрч), машины Тьюринга (А. М. Тьюринг), нормальные алгоритмы Маркова (А. А. Марков), алгоритмы Поста (Э. Л. Пост), лямбда-исчисления (А. Чёрч). Каждый из подходов по-своему определял элементарность шага алгоритма, но замечателен тот факт, что все эти определения оказались эквивалентными, т. е. описывающими одну и ту же совокупность объектов, называемых алгоритмами. Однако формализация, называемая *машинами Тьюринга*, чаще всего используется в теоретических построениях.

2. Машины Тьюринга

2.1. Описание машины Тьюринга

В 1936 г. английский математик Алан Тьюринг описал схему абстрактной машины и предложил назвать алгоритмом то, что умеет делать эта машина. Машины Тьюринга – это не реальные вычислительные машины, а способ описания алгоритма, при помощи которого формализуется понятие алгоритма и его можно исследовать. От реальных машин они отличаются бесконечной памятью. Устройство этих машин намеренно выбрано «бедным», лишь бы оно было универсальным и позволяло доказывать, что для определенных задач не существует такой машины (т. е. нельзя построить алгоритм). Реальные вычислительные машины (компьютеры), наоборот, должны быть «богатыми»

по устройству и уметь хорошо (эффективно) решать задачи.

В основе описания устройства машины Тьюринга (МТ) лежат три ее составные части:

1. *Память* в виде бесконечной в обе стороны ленты, разделенной на ячейки, в каждую из которых может быть записана либо одна из букв *внешнего алфавита* $\{a_1, a_2, ..., a_k\}$, либо пустой символ (будем для единообразия обозначать его греческой буквой Λ и добавим его к алфавиту как символ a_0). Внешний алфавит $A = \{a_0, a_1, ..., a_k\}$ может быть своим для каждой МТ, но всегда *конечным*. На любом шаге работы МТ только конечный участок ленты может содержать буквы внешнего алфавита.

2. *Автомат*, состоящий из *устройства управления* и *считывающей головки*, который на каждом шаге работы МТ передвигается вдоль ленты памяти на 1 ячейку влево, вправо или остается на месте. Движение автомата будем обозначать элементами множества движений $D = \{L, R, N\}$.

3. *Считывающая головка*, которая на любом шаге работы МТ *обозревает* какую-либо ячейку памяти.

4. *Устройство управления*, которое на каждом шаге работы МТ может находиться в одном из конечного числа своих состояний, множество $\{q_1, q_2, ..., q_n\}$ которых называется *внутренним алфавитом*.

Работа МТ может быть описана следующим образом:

1. В начальный момент работы МТ лента памяти находится в начальном состоянии (конечное непустое слово на ленте называется *входным словом*), автомат находится в *начальном* состоянии q_1 и считывающая головка обозревает самый левый непустой символ входного слова.

2. На каждом шаге работы МТ головка считывает символ $a_i \in A$ из обозреваемой ячейки и в зависимости от него и от состояния $q_j \in Q$ устройства управления автомат:

   а) записывает в обозреваемую ячейку символ $a_{i'} \in A$ (может, и не изменяет его),

b) изменяет состояние устройства управления на $q_{j'} \in Q$ (может, и не изменяет его),

c) осуществляет движение в направлении $d \in D$ (может, остается на месте).

3. Если в результате предыдущих шагов автомат переходит в такое состояние, при котором:

   a) состояние автомата не изменяется,

   b) символ в обозреваемой ячейке не изменяется,

   c) движение автомата вдоль ленты памяти не происходит,

   то машина переходит в заключительное состояние *останова*. Для упрощения мы такое состояние обозначим через q_0 и переход к нему будем обозначать как переход к состоянию q_0. Это состояние мы также включим во множество состояний $Q = \{q_0, q_1, ..., q_n\}$. Слово, которое получается при переходе МТ к состоянию останова, будем называть *выходным словом*.

Действия автомата на каждом шаге работы МТ могут быть описаны тремя функциями:

$a_{i'} = a(a_i, q_j)$, возвращающая значение записываемого символа $a_{i'}$, если в состоянии q_j был прочитан символ a_i;

$q_{j'} = q(a_i, q_j)$, возвращающая состояние $q_{j'}$, в которое переходит МТ после того, как в состоянии q_j был прочитан символ a_i;

$d = d(a_i, q_j)$, возвращающая направление движения d после того, как в состоянии q_j был прочитан символ a_i.

Все 3 функции должны быть определены на множестве $A \times (Q - \{q_0\})$. Табличное задание этих функций называется *функциональной схемой* МТ. В этой прямоугольной таблице (см. рис. 1) размера $n \times k$ на пересечении j-й строки ($j = \overline{1, n}$), отвечающей состоянию q_j, и i-го столбца ($i = \overline{0, k}$), отвечающего символу a_i, находится тройка $a_{i'}, q_{j'}, d^{'}$, определяющая изменение символа в обозреваемой ячейке, изменение состояния МТ и движение МТ на шаге работы МТ.

	Λ	a_1	...	a_i	...	a_k
q_1						
...	...	...	...	...	...	...
q_j				$a_{i'}\ q_{j'}\ d$		
...	...	...	...	...	...	...
q_n						

Рис. 1.

С каждым шагом работы МТ свяжем конфигурацию МТ, которая состоит из последовательности символов слова, записанного на ленте МТ перед выполнением шага, и состояния МТ, записанного перед обозреваемым символом на этом шаге. Например, конфигурация abq_2aba означает, что в момент выполнения шага на ленте находится слово $ababa$, МТ обозревает второй символ a и находится в состоянии q_2, а конфигурация $ababaq_3$ означает, что на ленте находится такое же слово, но МТ в состоянии q_3 обозревает пустой символ справа от слова на ленте.

МТ называется *применимой* к входному слову, если, начав работу в начальном состоянии, она через конечное число шагов перейдет в состояние останова, и *неприменимой*, если при ее работе она не переходит в состояние останова ни за какое конечное число шагов. МТ может быть неприменима ни к одному слову. Например, МТ, определенная функциональной таблицей на рис. 2, все время движется вдоль ленты вправо, ничего не меняя.

	Λ	0	1
q_1	$\Lambda\ q_1$ R	$0\ q_1$ R	$1\ q_1$ R

Рис. 2.

МТ может быть применима к любому слову. Например, МТ, заданная функциональной таблицей на рис. 3, стирает все символы входного слова и останавливается.

	Λ	0	1
q_1	$\Lambda\ q_0$ N	$\Lambda\ q_1$ R	$\Lambda\ q_1$ R

Рис. 3.

В дальнейших примерах мы для большей наглядности не будем в клетке таблицы определять
– заменяемый символ, если он не меняется,
– новое состояние, если оно не меняется,
– движение, если оно не происходит.
Введем также столбец для комментариев действий МТ в том или ином состоянии. Рассмотрим в качестве примера разработку МТ, которая увеличивает произвольное десятичное число на 1. Неформальный пошаговый алгоритм можно определить следующим образом:

1. Определить последнюю цифру числа (для этого двигаться вправо до пустого символа и, перейдя к следующему состоянию, вернуться к предыдущей цифре).

2. Увеличить цифру (последнюю) на 1, если она меньше (не равна) 9, и закончить работу МТ; если же цифра равна 9, то заменить ее на 0 и перейти к увеличению предыдущей цифры (единица переноса), для чего повторить действия этого шага.

Функциональная таблица МТ представлена на рис. 4.

	Λ	0	1	...	8	9	
q_1	q_2 L	R	R	...	R	R	поиск последней цифры
q_2	1 q_0	1 q_0	2 q_0	...	9 q_0	0 L	увеличение на 1 и перенос

Рис. 4.

Каждая МТ на множестве входных слов задает частичную функцию f, которая любому применимому входному слову P ставит в соответствие выходное слово $f(P)$. Эта функция называется *вычислимой по Тьюрингу*. Ясно, что каждая вычислимая по Тьюрингу функция является интуитивно вычислимой, т. е. каждая МТ определяет алгоритм. Но, может быть, существуют алгоритмы, для которых построение МТ невозможно? В следующем разделе мы исследуем возможности МТ и покажем, что не видно ограничений для описания алгоритмов при помощи аппарата МТ.

2.2. Тезис Тьюринга

Рассмотрим пример МТ, которая для входного слова, образованного любым натуральным числом палочек, стирает самую правую палочку.

Функциональная таблица этой МТ представлена на рис. 5.

	Λ	|	
q_1	q_2 L	R	поиск последней палочки
q_2	q_0	Λ q_0	стирание палочки и останов

Рис. 5.

Теперь постараемся объединить две последние МТ в одну, которая подсчитывает число палочек во входном слове. Для этого можно число считаемых палочек располагать левее самой левой палочки входного слова и соединить 2 алгоритма стирания правой палочки и увеличения левого числа (палочек) на 1 следующим образом:

1. Выполняем 1-й алгоритм стирания правой палочки, но не останавливаемся, а переходим к следующему шагу.

2. Ищем самую последнюю цифру числа (если ее нет, то 1-й пустой символ справа) и увеличиваем число на 1 при помощи 2-го алгоритма. На этом не останавливаемся, а переходим к следующему шагу.

3. Ищем палочку справа от числа; если ее нет, заканчиваем работу, а иначе переходим к шагу 1.

На рис. 6 представлена функциональная таблица МТ подсчета палочек.

	Λ	0	...	8	9	|	
q_1	q_2 L	R	...	R	R	R	к последней палочке
q_2	q_0	q_0	...	q_0	q_0	Λ q_3 L	стир. пал. или стоп
q_3	1 q_4 R	1 q_4 R	...	9 q_4 R	0 L	L	к посл. циф. и ув. чис.
q_4	q_0	R	...	R	R	q_1 R	нет палочки – стоп

Рис. 6.

Для проверки алгоритма построенной МТ нужно задать достаточный набор тестов и проверить на нем выполнение алгоритма. Например, работу построенной МТ подсчета палочек демонстрирует следующий пример:

$q_1||| \to |q_1|| \to ... \to |||q_1 \to ||q_2| \to |q_3| \to q_3|| \to 1|q_4| \to 1||q_1 \to 1|q_2| \to 1q_3| \to q_31| \to 2q_4| \to 2|q_1 \to 2q_2| \to q_32 \to 3q_4 \to 3q_0$.

Указанный пример построения МТ показывает, что из уже имеющихся МТ можно построить новые МТ, соединяя их различным образом. Так, в примере применено последовательное соединение МТ, стирающей палочку и увеличивающей число на 1, которое затем повторяется в цикле. Неформальное пошаговое определение алгоритма показывает 4 способа соединения алгоритмов:

1. **Последовательное соединение, или суперпозиция**, когда за выполнением шагов первого алгоритма следует выполнение шагов второго алгоритма и входной информацией для второго алгоритма служит выходная информация первого алгоритма.

2. **Композиция**, когда 2 алгоритма параллельно и независимо выполняют работу над входной информацией и результатом их работы является выходная информация обоих алгоритмов.

3. **Ветвление**, когда в зависимости от выполнения условия для входной информации, проверяемого первым алгоритмом, выполняется или не выполняется второй алгоритм.

4. **Цикл**, когда при проверке первым алгоритмом выполнения условия для входной информации соединения алгоритмов или выходной информации предыдущего выполнения второго алгоритма этот второй алгоритм снова выполняется в цикле или происходит выход из цикла с выходной информацией второго алгоритма.

Мы покажем, что МТ также можно соединить подобным образом, приведя лишь идеи построения таких соединений.

Суперпозиция МТ. *Пусть M_1 и M_2 – две машины Тьюринга, вычисляющие функции $f_1(P)$ и $f_2(P)$. Тогда можно построить машину Тьюринга $M = M_2(M_1)$, вычисляющую суперпозицию функций $f_2(f_1(P))$. Она определена тогда и только тогда, когда определены $f_1(P)$ и f_2 на множестве значений $f_1(P)$.*
Доказательство. Не ограничивая общность рассуждений, можно считать, что M_1 заканчивает работу, когда ее головка позиционирует самый левый символ выходного слова, так как в противном случае можно заменить эту МТ на эквивалентную (вычисляющую ту же функцию), удовлетворяющую этому требованию. Сделать это можно, доба-

вив в конце работы переход к дополнительному состоянию, в котором отыскивается пустой символ слева от левого непустого символа выходного слова и производится сдвиг вправо с остановкой.

Пусть M_1 имеет множество состояний $\{q_0, q_1, ..., q_{n_1}\}$, а M_2 имеет множество состояний $\{q_0, q_1, ..., q_{n_2}\}$. Для построения суперпозиции M:

1) переименуем состояния M_2, заменяя q_1 на q_{n_1+1},..., q_{n_2} на $q_{n_1+n_2}$;

2) напишем вторую функциональную таблицу под первой;

3) изменим значения тех ячеек первой функциональной таблицы, в которых осуществляется переход к заключительному состоянию q_0, изменив этот переход на переход к начальному состоянию второй таблицы q_{n_1+1}.

В результате мы получим требующуюся машину M.

Композиция МТ. *Пусть M_1 и M_2 – две машины Тьюринга, вычисляющие функции $f_1(P)$ и $f_2(P)$. Тогда можно построить машину Тьюринга $M = M_1 * M_2$, которая выполняет работу обеих МТ и вычисляет функцию $f(P) = f_1(P) * f_2(P)$, где символ * не встречается в алфавите M_1 и M_2 ($f(P)$ не определена, если не определены $f_1(P)$ или $f_2(P)$).*

Доказательство. Рассмотрим следующую идею образования M:

1) построим МТ M_3, которая копирует входное слово P в виде $P*P$;

2) напишем таблицу M_2 под таблицей M_1;

3) изменим действия M_1 на пустом правом символе на такие же действия на символе *;

4) изменим переход к заключительному состоянию M_1 на переход к начальному состоянию M_2 с движением влево от символа *;

5) изменим действия M_2 на пустом левом символе на такие же действия на символе *.

Описанная идея некорректна в пункте 3, если действия M_1 связаны с одним из следующих:

1) с заменой пустого символа на некоторый непустой (при этом исчезнет символ *, разделяющий левую часть слова, обрабатываемую M_1, и правую часть слова, обрабатываемую M_2, и соединение станет неверным);

2) со стиранием символа слева от пустого (замена на символ * приведет к тому, что обе части слова для обработки M_1 и M_2 будут раз-

делены более чем одним символом *).

Для того чтобы исправить некорректность в первом случае, введем для каждого состояния q_j МТ M_1, где этот случай имеет место, МТ M^r_{jc} со множеством состояний $\{q_{j1}, \ldots, q_{jm}, q_{j0}\}$ (q_{j1} – начальное, а q_{j0} – конечное состояния), которая сдвигает вправо на 1 ячейку правую часть слова, начинающуюся с символа *, записывая в освободившуюся ячейку символ λ (который не встречается в алфавите МТ M_1 и M_2), и останавливается, позиционируя этот символ. Пусть теперь функциональная таблица МТ M_1 в состоянии q_j при чтении пустого символа справа определяется тройкой $(a_i, q_{j'}, d)$. В преобразованной M_1 тройку в строке q_j и столбце, отвечающем символу *, заменим на (λ, q_{j1}, N), где q_{j1} – начальное состояние МТ M^r_{jc}. Добавим в функциональную таблицу M_1 строку q_{j0}, отвечающую конечному состоянию МТ M^r_{jc}, и для дополнительно введенного столбца символа λ этой строки определим тройку $(a_i, q_{j'}, d)$. Тогда действия, отвечающие пункту 3 в этом случае, станут корректны.

Для того чтобы исправить некорректность пункта 3 во втором случае (стирание символа слева от символа *, заменившего пустой символ), введем для каждой пары (j, i), где q_j – состояние, в котором стирается символ a_i, МТ M^r_{jid} со множеством состояний $\{q_{1ji}, \ldots, q_{0ji}\}$ (q_{1ji} – начальное, а q_{0ji} – конечное состояния), которая сдвигает часть слова справа от символа * вместе с ним на 1 ячейку влево, стирая символ слева. Пусть теперь в некотором состоянии M_1 при чтении пустого символа переходит в состояние q_j, сдвигая головку влево, и в состоянии q_j стирает символ a_i, сдвигая после этого головку в направлении d_{ji} и переходя в состояние q_{ji}. В преобразованной МТ M_1 тройку $(\Lambda, q_{ji}, d_{ji})$ заменим на (λ, q_{1ji}, R), где λ – символ, не встречающийся в алфавите МТ M_1 и M_2, который МТ M^r_{jid} заменит на символ *. Добавим в функциональную таблицу M_1 строку q_{0ji}, отвечающую конечному состоянию МТ M^r_{jid}, и для столбца символа * этой строки определим тройку $(*, q_{ji}, d_{ji})$. Тогда действия, отвечающие пункту 3 во втором случае, также станут корректными.

Так же, как для пункта 3 конструирования МТ M, требуются изменения, связанные с действиями МТ M_2 при выполнении пункта 5. Для этого аналогичным образом для каждого состояния q_j МТ M_2, где необходимо добавлять символ слева, вводится МТ M^l_{jc}, которая сдвига-

ет влево на 1 ячейку левую часть слова, заканчивающуюся символом *, записывается в освободившуюся ячейку символ λ и производятся аналогичные изменения функциональной таблицы МТ M_2. Аналогичным образом для каждой пары (j,i), где q_j – состояние M_2, в котором стирается символ a_i, строится МТ M^l_{jid}, которая сдвигает часть слова справа от удаляемого символа на 1 ячейку влево, стирая этот символ. Соответствующие изменения функциональной таблицы МТ M_2 приводят к корректным действиям при выполнении пункта 5.

В результате мы получим машину M, которая сначала перерабатывает в $f_1(P)$ первую копию слова P, а затем перерабатывает в $f_2(P)$ вторую копию слова P, что и требовалось.

Ветвление МТ. *Пусть M_1 и M_2 – две машины Тьюринга, вычисляющие функции $f_1(P)$ и $f_2(P)$, причем множеством значений функции $f_1(P)$ является множество $\{F,T\}$ (F – ложь, T – истина). Тогда можно построить машину Тьюринга $M = M_1{\to}M_2$, которая перерабатывает слово P в $f_2(P)$, если $f_1(P) = T$, и оставляет входное слово без изменений, если $f_1(P) = F$.*

Доказательство. Машину M можно реализовать следующим образом:

1) построим композицию $M_1 * M_2$;

2) изменим в этой композиции переход от работы машины M_1 к работе машины M_2 так, что если после работы M_1 образуется слово $F*P$, то M_1 стирает символы $F*$ и переходит к заключительному состоянию; а если после работы M_1 образуется слово $T*P$, то M_1 стирает символы $T*$ и переходит к начальному состоянию M_2 и продолжает работу, перерабатывая слово P в слово $f_2(P)$.

Цикл МТ. *Пусть M_1 и M_2 – две машины Тьюринга, вычисляющие функции $f_1(P)$ и $f_2(P)$, причем множеством значений функции $f_1(P)$ является множество $\{F,T\}$ (F – ложь, T – истина). Тогда можно построить машину Тьюринга $M = M_1{\circ}M_2$, которая выполняет следующую последовательность действий:*

- вычисляет $f_1(P)$, и если $f_1(P) = T$, то вычисляет новое значение $P := f_2(P)$, а если $f_1(P) = F$, то переходит к заключительному состоянию с выходным словом P (возможно новое значение);

- повторяет действия предыдущего пункта до тех пор, пока для оче-

редного значения слова P не будет выполнено $f_1(P) = F$.

Доказательство. Машину M можно реализовать следующим образом:

1) образуем ветвление машин $M_1 \to M_2$;

2) изменим заключительное состояние M_2, заменив его переходом к копированию получившегося слова $f_2(P)$ и добавив после этого переход к начальному состоянию M_1.

В результате мы получим машину, которая реализует цикл МТ.

Описанные операции над машинами Тьюринга показывают, что можно строить все более сложные МТ. Поскольку удается реализовать арифметические и логические операции, то возникает уверенность в том, что *любая интуитивно вычислимая функция является вычислимой по Тьюрингу*. Именно в этом и состоит тезис Тьюринга. Его не следует рассматривать как некоторое утверждение, требующее доказательства, а следует понимать как определение алгоритма. Другие уточнения понятия «алгоритм» оказались эквивалентными.

3. Индивидуальное задание

3.1. Общее задание

Для функции индивидуального варианта построить машину Тьюринга (МТ):

a) проанализировать функцию задания и построить полный набор тестов;

b) разработать неформальное пошаговое описание алгоритма функции задания;

c) протестировать полученное неформальное пошаговое описание алгоритма;

d) по полученному описанию алгоритма построить функциональную таблицу МТ;

e) прокрутить каждый тест на разработанной МТ, подписывая состояние МТ под рассматриваемым символом строки;

f) провести анализ полученной таблицы МТ и при возможности улучшить ее.

3.2. Варианты задания

1. Увеличение на единицу 8-разрядного целого числа в модифицированном дополнительном коде (для знака отводятся 2 разряда, при неодинаковости которых отмечается переполнение; код отрицательного числа – инвертированный код модуля числа, увеличенный на 1).

2. Уменьшение на единицу 8-разрядного положительного целого числа в прямом коде (выходным является также разряд Z, значение которого 1 при равенстве 0 результата).

3. Разность двух 8-разрядных целых чисел в модифицированном дополнительном коде (для знака отводятся 2 разряда, при неодинаковости которых отмечается переполнение, – разряд $O = 1$; код отрицательного числа – инвертированный код модуля числа, увеличенный на 1).

4. Для входного 9-разрядного двоичного кода выходной код равен 1, если число единиц в коде в 2 раза больше числа нулей; в противном случае выходной код равен 0.

5. Для входного 9-разрядного двоичного кода выходной код равен 1, если на четных местах количество нулей больше, чем количество единиц на нечетных местах; в противном случае выходной код равен 0.

6. Для входного 9-разрядного двоичного кода выходной код равен 1, если в каких-либо трех рядом стоящих разрядах есть 2 нуля; в противном случае выходной код равен 0.

7. Для входного 9-разрядного двоичного кода выходной код равен 1, если в каждых трех подряд идущих разрядах есть единица; в противном случае выходной код равен 0.

8. Сравнение двух целых неотрицательных чисел (по результату разности между первым и вторым числом); возвращает разряд $Z = 1$, отмечающий равенство исходных чисел, и разряд $S = 1$, если первое число меньше второго.

9. Изменение знака числа в дополнительном коде (код, включая знаковый разряд, инвертируется и к полученному числу добавляется 1).

10. Умножение целого неотрицательного числа на 3 (суммируются код числа и код, полученный сдвигом на 1 разряд влево); переполнение отмечается в разряде O.

11. Для входного 9-разрядного двоичного кода выходной код равен 1, если входной код равен коду, полученному циклическим сдвигом на 3 разряда вправо (младшие разряды поступают в старшие); в противном случае выходной код равен 0.

12. Для входного 9-разрядного двоичного кода выходной код равен 1, если входной код не равен коду, полученному циклическим сдвигом на 3 разряда влево (старшие разряды поступают в младшие); в противном случае выходной код равен 0.

13. Для входного 8-разрядного двоичного кода выходной код равен 0, если на нечетных местах количество единиц больше количества нулей на четных местах; в противном случае выходной код равен 1.

14. Для входного 8-разрядного двоичного кода выходной код равен 0, если входной код равен коду, полученному циклическим сдвигом на 2 разряда вправо (младшие разряды поступают в старшие); в противном случае выходной код равен 1.

15. Увеличение на два 8-разрядного целого числа в модифицированном дополнительном коде (для знака отводятся 2 разряда, при разности которых отмечается переполнение; код отрицательного числа – инвертированный код модуля числа, увеличенный на 1).

16. Уменьшение на два 8-разрядного положительного целого числа

в прямом коде (выходным является также разряд Z, значение которого 1 при равенстве 0 результата).

17. Для входного 8-разрядного двоичного кода выходной код равен 0, если в каждых трех подряд идущих разрядах есть нуль; в противном случае выходной код равен 1.

18. Для входного 8-разрядного двоичного кода выходной код равен 0, если входной код как число без знака больше числа, полученного циклическим сдвигом на 2 разряда вправо (младшие разряды поступают в старшие); в противном случае выходной код равен 1.

19. Для входного 8-разрядного двоичного кода выходной код равен 0, если число в первых четырех разрядах кода больше числа в последних четырех разрядах кода; в противном случае выходной код равен 1.

20. Для входного 9-разрядного двоичного кода выходной код равен 0, если число в первых трех разрядах меньше числа в последних трех разрядах и не равно числу в средних трех разрядах; в противном случае выходной код равен 1.

21. Умножение целого неотрицательного числа на 5 (суммируются код числа и код, полученный сдвигом на 2 разряда влево); переполнение отмечается в разряде O.

22. Увеличение на три 8-разрядного целого числа в модифицированном дополнительном коде (для знака отводятся 2 разряда, при разности которых отмечается переполнение; код отрицательного числа – инвертированный код модуля числа, увеличенный на 1).

23. Уменьшение на три 8-разрядного положительного целого числа в прямом коде (выходным является также разряд Z, который равен 1 при равенстве 0 результата).

24. Для входного 9-разрядного двоичного кода выходной код равен 0, если 3 трехразрядных числа, на которые можно разделить входной код, образуют возрастающую последовательность; в противном случае выходной код равен 1.

25. Для входного 9-разрядного двоичного кода выходной код равен 0, если 3 трехразрядных числа, на которые можно разделить входной код, попарно различны; в противном случае выходной код равен 1.

26. Увеличение на единицу 8-разрядного целого числа в модифицированном дополнительном коде (для знака отводятся 2 разряда, при неодинаковости которых отмечается переполнение; код отрицательного числа – инвертированный код модуля числа, увеличенный на 1).

27. Для входного 8-разрядного двоичного кода выходной код равен 1, если входной код как число без знака меньше числа, полученного циклическим сдвигом на 2 разряда вправо (младшие разряды поступают в старшие); в противном случае выходной код равен 0.

28. Умножение целого неотрицательного числа на 3 (суммируются код числа и код, полученный сдвигом на 1 разряд влево); переполнение отмечается в разряде O.

29. Для входного 8-разрядного двоичного кода выходной код равен 1, если число в первых четырех разрядах кода меньше числа в последних четырех разрядах кода; в противном случае выходной код равен 0.

30. Увеличение на три 8-разрядного целого числа в модифицированном дополнительном коде (для знака отводятся 2 разряда, при разности которых отмечается переполнение; код отрицательного числа – инвертированный код модуля числа, увеличенный на 1).

31. Для входного 9-разрядного двоичного кода выходной код равен 1, если число в первых трех разрядах больше числа в последних трех разрядах и не равно числу в средних трех разрядах; в противном случае выходной код равен 0.

32. Увеличение на два 8-разрядного целого числа в модифицированном дополнительном коде (для знака отводятся 2 разряда, при разности которых отмечается переполнение; код отрицательного числа – инвертированный код модуля числа, увеличенный на 1).

33. Для входного 9-разрядного двоичного кода выходной код равен 1, если 3 трехразрядных числа, на которые можно разделить входной код, образуют убывающую последовательность; в противном случае выходной код равен 0.

34. Разность двух 8-разрядных целых чисел в модифицированном дополнительном коде (для знака отводятся 2 разряда, при неодинаковости которых отмечается переполнение – разряд $O = 1$; код отрицательного числа – инвертированный код модуля числа, увеличенный на 1).

35. Для входного 9-разрядного двоичного кода выходной код равен 1, если 3 трехразрядных числа, на которые можно разделить входной код, попарно различны; в противном случае выходной код равен 0.

36. Для входного 9-разрядного двоичного кода выходной код равен 0, если число единиц в коде в 2 раза меньше числа нулей; в противном случае выходной код равен 1.

37. Для входного 9-разрядного двоичного кода выходной код равен 0, если на четных местах количество нулей меньше, чем количество единиц на нечетных местах; в противном случае выходной код равен 1.

38. Для входного 9-разрядного двоичного кода выходной код равен 0, если в каких-либо трех рядом стоящих разрядах есть 2 единицы; в противном случае выходной код равен 1.

39. Уменьшение на два 8-разрядного положительного целого числа в прямом коде (выходным является также разряд Z, значение которого 1 при равенстве 0 результата).

40. Сравнение двух целых неотрицательных чисел (по результату разности между первым и вторым числом); возвращает разряд $Z = 1$, отмечающий равенство исходных чисел, и разряд $S = 1$, если первое число меньше второго.

41. Умножение целого неотрицательного числа на 5 (суммируются код

числа и код, полученный сдвигом на 2 разряда влево); переполнение отмечается в разряде O.

42. Уменьшение на единицу 8-разрядного положительного целого числа в прямом коде (выходным является также разряд Z, значение которого 1 при равенстве 0 результата).

43. Для входного 9-разрядного двоичного кода выходной код равен 0, если в каждых трех подряд идущих разрядах есть нуль; в противном случае выходной код равен 1.

44. Разность двух 8-разрядных целых чисел в модифицированном дополнительном коде (для знака отводятся 2 разряда, при неодинаковости которых отмечается переполнение – разряд $O = 1$; код отрицательного числа – инвертированный код модуля числа, увеличенный на 1).

45. Для входного 9-разрядного двоичного кода выходной код равен 0, если входной код равен коду, полученному циклическим сдвигом на 3 разряда вправо (младшие разряды поступают в старшие); в противном случае выходной код равен 1.

46. Для входного 9-разрядного двоичного кода выходной код равен 0, если входной код не равен коду, полученному циклическим сдвигом на 3 разряда влево (старшие разряды поступают в младшие); в противном случае выходной код равен 1.

47. Уменьшение на три 8-разрядного положительного целого числа в прямом коде (выходным является также разряд Z, который равен 1 при равенстве 0 результата).

48. Для входного 8-разрядного двоичного кода выходной код равен 1, если на нечетных местах количество единиц меньше количества нулей на четных местах; в противном случае выходной код равен 0.

49. Для входного 8-разрядного двоичного кода выходной код равен 1, если входной код не равен коду, полученному циклическим сдвигом на 2 разряда вправо (младшие разряды поступают в старшие); в противном случае выходной код равен 0.

50. Для входного 8-разрядного двоичного кода выходной код равен 0, если каждые 3 подряд идущие разряда содержат нуль; в противном случае выходной код равен 1.

51. Для входного 9-разрядного двоичного кода выходной код равен 1, если во всех трех подряд идущих разрядах есть единица; в противном случае выходной код равен 0.

52. Уменьшение на единицу 8-разрядного положительного целого числа в прямом коде (выходным является также разряд Z, значение которого 1 при равенстве 0 результата).

53. Разность двух 8-разрядных целых чисел в модифицированном дополнительном коде (для знака отводятся 2 разряда, при неодинаковости которых отмечается переполнение – разряд $O = 1$; код отрицательного числа – инвертированный код модуля числа, увеличенный на 1).

54. Для входного 9-разрядного двоичного кода выходной код равен 1, если число единиц в коде в 2 раза больше числа нулей; в противном случае выходной код равен 0.

55. Для входного 9-разрядного двоичного кода выходной код равен 1, если на четных местах количество нулей больше, чем количество единиц на нечетных местах; в противном случае выходной код равен 0.

56. Для входного 9-разрядного двоичного кода выходной код равен 1, если в каких-либо трех рядом стоящих разрядах есть 2 нуля; в противном случае выходной код равен 0.

57. Для входного 9-разрядного двоичного кода выходной код равен 1, если в каждых трех подряд идущих разрядах есть единица; в противном случае выходной код равен 0.

58. Сравнение двух целых неотрицательных чисел (по результату разности между первым и вторым числом); возвращает разряд $Z = 1$, отмечающий равенство исходных чисел, и разряд $S = 1$, если первое число меньше второго.

59. Изменение знака числа в дополнительном коде (код, включая знаковый разряд, инвертируется, и к полученному числу добавляется 1).

60. Умножение целого неотрицательного числа на 3 (суммируются код числа и код, полученный сдвигом на 1 разряд влево); переполнение отмечается в разряде O.

61. Для входного 9-разрядного двоичного кода выходной код равен 1, если входной код равен коду, полученному циклическим сдвигом на 3 разряда вправо (младшие разряды поступают в старшие); в противном случае выходной код равен 0.

62. Для входного 9-разрядного двоичного кода выходной код равен 1, если входной код не равен коду, полученному циклическим сдвигом на 3 разряда влево (старшие разряды поступают в младшие); в противном случае выходной код равен 0.

63. Для входного 8-разрядного двоичного кода выходной код равен 0, если на нечетных местах количество единиц больше количества нулей на четных местах; в противном случае выходной код равен 1.

64. Для входного 8-разрядного двоичного кода выходной код равен 0, если входной код равен коду, полученному циклическим сдвигом на 2 разряда вправо (младшие разряды поступают в старшие); в противном случае выходной код равен 1.

Примечания к условиям некоторых задач

- Обратным для кода называется код, все разряды которого инвертированы (нули заменены на единицы, а единицы – на нули). При этом знаковый разряд также инвертируется.

- Дополнительным для кода называется код, полученный из обратного сложением с единицей в самом младшем разряде с возможным переносом в более старший разряд при сложении. При этом знаковый разряд также участвует в операции сложения. Сумма любого кода с его дополнительным кодом всегда равна 0. Сложение двух чисел в дополнительном коде производится как сложение чисел без знака; при этом знаковый разряд также участвует

в сложении. Для вычитания вычитаемое заменяется на дополнительный код и производится сложение с кодом уменьшаемого.

- Модифицированный дополнительный код отличается от обычного дополнительного кода наличием двух знаковых разрядов. При этом знак плюс кодируется 00, а знак минус – 11. Если при сложении происходит переполнение, то знаковые разряды различны (10 при положительном переполнении и 01 – при отрицательном).

- Результатом выполнения алгоритма для некоторых вариантов является не только преобразуемый исходный код, но и значение одного из специальных двоичных разрядов: N (разряд отрицательности результата – 1, если результат отрицательный); C (разряд переноса – 1, если произошел перенос 1 слева от старшего разряда); O (разряд переполнения – 1, если в результате арифметической операции произошло переполнение кода); Z (разряд нуля – 1, если результатом операции является нулевой код). Этот разряд должен быть отделен от кода результата специальным символом (например,*).

3.3. Примеры задачи и их решения

3.3.1. Пример 1

Разработать МТ 8-разрядного сумматора неотрицательных целых чисел с разрядом переноса.

Решение. Решение разобъем на 3 обязательные для задания части:

1) разработка алгоритма (тесты, идеи алгоритма, описание алгоритма, тестирование алгоритма);

2) разработка таблицы МТ по алгоритму;

3) тестирование МТ.

Разработка тестов

Анализ функции задания показывает, что исходными данными являются два 8-разрядных целых неотрицательных числа, а результатом

вычисления (выполнения функции) – 8-разрядная сумма чисел и значение разряда переноса. Исходные числа можно разделить символом * и таким же образом разделить разряд переноса и сумму чисел.

Примерами входной строки МТ и строки соответствующего результата для нее являются:

$$00000101 * 00001110 \ \rightarrow \ 0 * 00010011 \ (5 + 14 = 19)$$

$$00000101 * 11111101 \ \rightarrow \ 1 * 00000010 \ (5 + 253 = 258 = 256 + 2).$$

Анализ этих примеров показывает, что подобные тесты потребуют очень длинного выполнения. Поэтому при разработке алгоритма будем ориентироваться на исходные данные с любым количеством разрядов (в частности, 8-разрядные), а тесты составим для 2-разрядных чисел, что, с одной стороны, отразит достаточную общность примеров, а, с другой стороны, выполнение тестов окажется короче.

Так как набор тестов должен быть полон (проверять все ветви алгоритма и иметь как данные, являющиеся крайними случаями, так и данные общего случая), то в тесты включим следующий набор:

1. $10 * 01 \ \rightarrow \ 0 * 11$ (сумма допустимая, без переноса – общий случай).
2. $10 * 10 \ \rightarrow \ 1 * 00$ (сумма с переносом – общий случай).
3. $00 * 00 \ \rightarrow \ 0 * 00$ (оба числа наименьшие – крайний случай).
4. $11 * 11 \ \rightarrow \ 1 * 10$ (оба числа наибольшие – крайний случай).

Идеи алгоритма

Алгоритм может быть основан на разных идеях. Наиболее эффективная по трудоемкости (вычислительной сложности) алгоритма идея – поразрядное сложение чисел, но алгоритм в этом случае получается более сложным по числу шагов описания. Более простая для реализации идея поочередного уменьшения на 1 одного из чисел и увеличение другого числа до тех пор, пока уменьшаемое число не обратится в нуль.

Для разработки алгоритма сложения выберем второй способ, при котором первое число (назовем его a) уменьшается на 1, а второе (b) увеличивается на 1, и это повторяется до тех пор, пока первое число не станет равным 0. Этот способ по вычислительной сложности алгоритма

не является лучшим, но является довольно простым для описания, что достаточно для нашей цели. Отметим также, что при проверке числа a на 0 незначащие слева (нулевые разряды) числа a можно удалять (стирать). Символ, считываемый на каждом шаге работы МТ, обозначим через s.

Требуется учесть представление данных в МТ и начальное состояние и положение считывающей головки МТ – в начальном состоянии q_1 под 1-м символом. В конечном состоянии q_0 также головка должна находиться под 1-м символом. Еще одна особенность, которую следует учесть, – это возникновение переполнения – такую особенность мы будем отмечать временной заменой символа * на какой-либо другой символ, отличный от уже имеющихся (например, на символ +). С каждым шагом алгоритма свяжем состояние МТ, соответствующее этому шагу.

Описание алгоритма

Сначала опишем общий алгоритм, который непосредственно реализует идею алгоритма, а затем разработаем детализацию для тех шагов общего алгоритма, которые не смогут быть реализованы одним состоянием МТ. Если же шаг может быть реализован одним состоянием МТ, то описание этих действий дадим в скобках этого же шага.

Общий алгоритм

1. Проверка на 0 первого слагаемого. Если $a = 0$, то переход к шагу 4 (завершение алгоритма); иначе поиск младшего разряда a.

2. Уменьшение a на 1
   (Если $s = 1$, то замена на 0 и переход к шагу 3.1;
   иначе, если $s = 0$ замена на 1, сдвиг влево и повторение шага 2;
   иначе ($s = \lambda$) сдвиг вправо и переход к шагу 3.1).

3. Увеличение b на 1. Переход к шагу 1.

4. Занесение разряда переноса и * перед старшим (левым) разрядом суммы.

Детализация шага 1 общего алгоритма

1.1 Если $s = 0$, то стирание нулевого символа, сдвиг вправо и повторение шага 1.1;

иначе, если $s \in \{*, +\}$, к ш.4 ($a = 0$ – завершение алгоритма);
иначе к шагу 1.2.

1.2 Поиск младшего разряда a
(сдвиг вправо до символа перед * или +).

Детализация шага 3 общего алгоритма

3.1 Поиск последнего разряда b
(сдвиг вправо с повторением этого шага до пустого символа λ и возвращение на символ назад – влево).

3.2 Если $s = 0$, замена на 1 и переход к шагу 3.3;
иначе, если $s = 1$, замена на 0, сдвиг влево и повторение шага 3.2;
иначе $s = *$, замена на символ + и переход к шагу 3.3

3.3 Поиск первого разряда a
(сдвиг влево с повторением ш.3.3 до пустого символа и возвращение на 1 разряд вправо) и переход к шагу 1.1.

Детализация шага 4 общего алгоритма

4.1 Занесение * перед суммой и сдвиг влево к разряду переноса.
(Если $s = +$, замена на *, сдвиг влево и переход к шагу 4.2, иначе переход к шагу 4.3).

4.2 Замена s на 1 и конец алгоритма.

4.3 Замена s на 0 и конец алгоритма.

Тестирование

При тестировании будем в скобках писать номер шага алгоритма перед обозреваемым символом строки. Конец алгоритма будем обозначать шагом 0.

Тестирование детализации шага 1 общего алгоритма

Тест 1 (общий) $(1)10 * 01 \rightarrow 1(2)0 * 01$

$(1.1)10 * 01 \rightarrow (1.2)10 * 01 \rightarrow 1(1.2)0 * 01 \rightarrow 10(1.2) * 01 \rightarrow 1(2)0 * 01$.
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 2 (крайний) $(1)00 * 00 \rightarrow (4) * 00$

$(1.1)00 * 00 \to (1.1)0 * 00 \to (1.1) * 00 \to (4) * 00.$

Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 3 (крайний) $(1)0 + 00 \ \to \ (4) + 00$

$(1.1)0 + 00 \to (1.1) + 00 \to (4) + 00.$

Результат совпадает с ожидаемым. *Тест прошел успешно.*

Все тесты прошли успешно. *Тестирование детализации шага 1 общего алгоритма закончено.*

Тестирование детализации шага 3 общего алгоритма

Тест 1 (общий – сумма без переноса) $(3)01 * 01 \ \to \ (1)01 * 10$

$(3.1)01 * 01 \to 0(3.1)1 * 01 \to 01(3.1) * 01 \to 01 * (3.1)01 \to$
$\to 01 * 0(3.1)1 \to 01 * 01(3.1)\lambda \to 01 * 0(3.2)1 \to 01 * (3.2)00 \to$
$\to 01 * (3.3)10 \to 01(3.3) * 10 \to 0(3.3)1 * 10 \to (3.3)01 * 10 \to$
$\to (3.3)\lambda 01 * 10 \to (1.1)01 * 10.$

Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 2 (общий – сумма с переносом) $(3)0 * 11 \ \to \ (1)0 + 00$

$(3.1)0 * 11 \to 0(3.1) * 11 \to 0 * (3.1)11 \to 0 * 1(3.1)1 \to$
$\to 0 * 11(3.1)\lambda \to 0 * 1(3.2)1 \to 0 * (3.2)10 \to 0(3.2) * 00 \to$
$\to 0(3.3) + 00 \to (3.3)0 + 00 \to (3.3)\lambda 0 + 00 \to (1.1)0 + 00.$

Результат совпадает с ожидаемым. *Тест прошел успешно.*

Все тесты прошли успешно. *Тестирование детализации шага 3 общего алгоритма закончено.*

Тестирование детализации шага 4 общего алгоритма

Тест 1 (общий – сумма без переноса) $(4) * 01 \ \to \ (0)0 * 10$

$(4.1) * 01 \to (4.3)\lambda * 01 \to (0)0 * 01.$

Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 2 (общий – сумма с переносом) $(4) + 00 \ \to \ (0)1 * 00$

$(4.1) + 00 \to (4.2)\lambda * 00 \to (0)1 * 00.$

Результат совпадает с ожидаемым. *Тест прошел успешно.*

Все тесты прошли успешно. *Тестирование детализации шага 4 общего алгоритма закончено.*

Тестирование общего алгоритма

Тест 1 (общий – сумма без переноса) $(1)10 * 01 \ \to \ (0)0 * 11$

$(1)10 * 01 \to 1(2)0 * 01 \to (2)11 * 01 \to (3)01 * 01 \to (1)01 * 10 \to$
$\to (2)1 * 10 \to (3)0 * 10 \to (1)0 * 11 \to (4) * 11 \to (0)0 * 11.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 2 (общий – сумма с переносом) $(1)10 * 10 \to (0)1 * 00$

$(1)10 * 10 \to 1(2)0 * 10 \to (3)01 * 10 \to (1)01 * 11 \to$
$\to (2)1 * 11 \to (3)0 * 11 \to (1)0 + 00 \to (4) + 00 \to (0)1 * 00.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 3 (крайний) $00 * 00 \to 0 * 00$

$(1)00 * 00 \to (4) * 00 \to (0)0 * 00.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 4 (крайний) $11 * 11 \to 1 * 10$

$(1)11 * 11 \to 1(2)1 * 11 \to 1(3)0 * 11 \to (1)10 + 00 \to 1(2)0 + 00 \to$
$\to (3)01 + 00 \to (1)01 + 01 \to (2)1 + 01 \to (3)0 + 01 \to (1)0 + 10 \to$
$\to (4) + 10 \to (0)1 * 10$
Результат совпадает с ожидаемым. *Тест прошел успешно.*
Все тесты прошли успешно. *Тестирование общего алгоритма закончено. Тестирование всех алгоритмов завершено успешно.*

Функциональная таблица МТ

Поскольку каждый шаг алгоритма с детализацией можно реализовать одним состоянием МТ, то будем именовать состояние именем q с индексом, соответствующим номеру шага.

$Q\backslash A$	λ	$*$	0	1	$+$	Комментарий
q_{11}		q_{41}	λR	q_{12}	q_{41}	удаление незначащих нулей до 1
q_{12}		$q_2 L$	R	R	$q_2 L$	поиск младшего разряда a
q_2	$q_{31} R$		$1L$	$0q_{31}$		уменьшение a на 1
q_{31}	$q_{32} L$	R	R	R	R	поиск последнего разряда b
q_{32}		$+q_{33}$	$1q_{33}$	$0L$		увеличение b на 1
q_{33}	$q_{11} R$	L	L	L	L	поиск 1-го разряда a
q_{41}		$q_{42} L$			$*q_{43} L$	организация * перед суммой
q_{42}	$0q_0$					на шаге 4.1 прочитан *
q_{43}	$1q_0$					на шаге 4.1 прочитан символ +

Тестирование МТ

Тест 1 (общий – сумма без переноса) $10 * 01 \ \to \ 0 * 11$

$$10 * 01 \underset{q_{11}}{\to} 10 * 01 \underset{q_{12}}{\to} 10 * 01 \underset{q_{12}}{\to} 10 * 01 \underset{q_{12}}{\to} 10 * 01 \underset{q_2}{\to} 11 * 01 \underset{q_2}{\to} 01 * 01 \underset{q_{31}}{\to}$$

$$\to 01 * 01 \to \cdots \to 01 * 01 \underset{q_{31}}{\to} 01 * 01 \underset{q_{32}}{\to} 01 * 00 \underset{q_{32}}{\to} 01 * 10 \underset{q_{33}}{\to} \cdots \to$$

$$\to 01 * 10 \underset{q_{33}}{\to} 01 * 10 \underset{q_{11}}{\to} 1 * 10 \underset{q_{11}}{\to} 1 * 10 \underset{q_{12}}{\to} 1 * 10 \underset{q_{12}}{\to} 1 * 10 \underset{q_2}{\to} 0 * 10 \underset{q_{31}}{\to}$$

$$\to \cdots \to 0 * 10 \underset{q_{31}}{\to} 0 * 10 \underset{q_{32}}{\to} 0 * 11 \underset{q_{33}}{\to} \cdots \to 0 * 11 \underset{q_{33}}{\to} 0 * 11 \underset{q_{11}}{\to} *11 \underset{q_{11}}{\to}$$

$$\to *11 \underset{q_{41}}{\to} *11 \underset{q_{42}}{\to} 0 * 11. \quad q_0$$

Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 2 (общий – сумма с переносом) $10 * 10 \to 1 * 00$

$$10 * 10 \underset{q_{11}}{\to} 10 * 10 \underset{q_{12}}{\to} 10 * 10 \underset{q_{12}}{\to} 10 * 10 \underset{q_{12}}{\to} 10 * 10 \underset{q_2}{\to} 11 * 10 \underset{q_2}{\to} 01 * 10 \underset{q_{31}}{\to}$$

$$\to 01 * 10 \to \cdots \to 01 * 10 \underset{q_{31}}{\to} 01 * 10 \underset{q_{32}}{\to} 01 * 11 \underset{q_{33}}{\to} \cdots \to 01 * 11 \underset{q_{33}}{\to}$$

$$\to 01 * 11 \underset{q_{11}}{\to} 1 * 11 \underset{q_{11}}{\to} 1 * 11 \underset{q_{12}}{\to} 1 * 11 \underset{q_{12}}{\to} 1 * 11 \underset{q_2}{\to} 0 * 11 \underset{q_{31}}{\to} \cdots \to$$

$$\to 0 * 11 \underset{q_{31}}{\to} 0 * 11 \underset{q_{32}}{\to} 0 * 10 \underset{q_{32}}{\to} 0 * 00 \underset{q_{32}}{\to} 0 + 00 \underset{q_{33}}{\to} \cdots \to 0 + 00 \underset{q_{33}}{\to} 0 + 00 \underset{q_{11}}{\to}$$

$$\to +00 \underset{q_{11}}{\to} +00 \underset{q_{41}}{\to} *00 \underset{q_{43}}{\to} 1 * 00. \quad q_0$$

Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 3 (крайний) $00 * 00 \to 0 * 00$

$$00 * 00 \underset{q_{11}}{\to} 0 * 00 \underset{q_{11}}{\to} *00 \underset{q_{11}}{\to} *00 \underset{q_{41}}{\to} *00 \underset{q_{42}}{\to} 0 * 00. \quad q_0$$

Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 4 (крайний) $11 * 11 \to 1 * 10$

$$11*11 \underset{q_{11}}{\to} 11*11 \underset{q_{12}}{\to} \cdots \underset{q_{12}}{\to} 11*11 \underset{q_{2}}{\to} 11*11 \underset{q_{31}}{\to} 10*11 \to \cdots \to 10*11 \underset{q_{31}}{\to}$$

$$\to 10*11 \underset{q_{32}}{\to} 10*10 \underset{q_{32}}{\to} 10*00 \underset{q_{32}}{\to} 10+00 \underset{q_{33}}{\to} \cdots \to 10+00 \underset{q_{33}}{\to} 10+00 \underset{q_{11}}{\to}$$

$$\to 10+00 \underset{q_{12}}{\to} 10+00 \underset{q_{12}}{\to} 10+00 \underset{q_{12}}{\to} 10+00 \underset{q_{2}}{\to} 11+00 \underset{q_{2}}{\to} 01+00 \underset{q_{31}}{\to} \ldots$$

$$\to 01+00 \underset{q_{31}}{\to} 01+00 \underset{q_{32}}{\to} 01+01 \underset{q_{33}}{\to} \cdots \to 01+01 \underset{q_{33}}{\to} 01+01 \underset{q_{11}}{\to} 1+01 \underset{q_{11}}{\to}$$

$$\to 1+01 \underset{q_{12}}{\to} 1+01 \underset{q_{12}}{\to} 1+01 \underset{q_{2}}{\to} 0+01 \underset{q_{31}}{\to} \cdots \to 0+01 \underset{q_{31}}{\to} 0+01 \underset{q_{32}}{\to}$$

$$\to 0+00 \underset{q_{32}}{\to} 0+10 \underset{q_{33}}{\to} \cdots \to 0+10 \underset{q_{33}}{\to} 0+10 \underset{q_{11}}{\to} +10 \underset{q_{11}}{\to} +10 \underset{q_{41}}{\to} *10 \underset{q_{43}}{\to}$$

$$\to 1*10.$$
$$q_0$$

Результат совпадает с ожидаемым. *Тест прошел успешно.*
Тестирование завершено

Улучшение функциональной таблицы МТ

Полученную функциональную таблицу МТ можно улучшить за счет совмещения некоторых переходов к другому состоянию:
1) в строке состояния q_{11} переход к q_{41} при чтении символов * и $+$ заменим на $q_{42}L$ и $q_{43}L$ соответственно, удалив всю строку состояния q_{41};
2) в строке состояния q_2 при чтении символа 1 переход $0q_{31}$ заменим на $0q_{31}R$;
3) в строке состояния q_{32} переход q_{33} при чтении 0 заменим на $1q_{33}$, а переход $+q_{33}$ при чтении * заменим на $+q_{33}L$.
Эти замены уменьшат число состояний таблицы МТ на 1 и сократят число шагов при выполнении алгоритма МТ.

$Q\backslash A$	λ	$*$	0	1	$+$	Комментарий
q_{11}		$q_{42}L$	λR	q_{12}	$q_{43}L$	удаление незнач. нулей до 1
q_{12}		q_2L	R	R	q_2L	поиск младшего разряда a
q_2	$q_{31}R$		$1L$	$0q_{31}R$		уменьшение a на 1
q_{31}	$q_{32}L$	R	R	R	R	поиск последнего разряда b
q_{32}		$+q_{33}L$	$1q_{33}L$	$0L$		увеличение b на 1
q_{33}	$q_{11}R$	L	L	L	L	поиск 1-го разряда a
q_{42}	$0q_0$					на шаге 1.1 прочитан *
q_{43}	$1q_0$					на шаге 1.1 проч.символ $+$

3.3.2. Пример 2

Для входного 9-разрядного двоичного кода выходной код равен 1, если для 3 трехразрядных чисел, на которые можно разделить входной код, первое число равно третьему и отлично от второго; в противном случае выходной код равен 0.

Решение. Решение разобьем на 3 обязательные для задания части:

1) разработка алгоритма (тесты, идеи алгоритма, описание алгоритма, тестирование алгоритма);

2) разработка таблицы МТ по алгоритму;

3) тестирование МТ.

Разработка тестов

Анализ функции задания показывает, что исходными данными являются 3 3-разрядных целых неотрицательных числа (назовем их A, B, C), а результатом вычисления (выполнения функции) – 1-разрядное число 1 или 0 в зависимости от выполнения условия $A = C \wedge A \neq B$.

Примерами входной строки МТ и строки соответствующего результата для нее являются:
010011010 $\rightarrow$ 1 ($A = 2 \neq B = 3, A = C = 2$)
110110110 $\rightarrow$ 0 ($A = B = 6, A = C = 6$)
001010100 $\rightarrow$ 0 ($A = 1 \neq B = 2, A \neq C = 4$)

$101101011 \rightarrow 0$ ($A = B = 5, A \neq C = 3$).
Эти примеры и возьмем в качестве полного набора тестов.

Идеи алгоритма

Основная часть алгоритма должна проверить выполнение вышеуказанного условия.Для этого нужно сравнение на равенство (неравенство) двух пар чисел, на которые разделен код: A с B и A с C. Идея сравнения пары чисел по их разности является довольно трудоемкой; проще осуществлять поразрядное сравнение таких чисел (1-го разряда 1-го числа с 1-м разрядом второго числа, затем 2-го разряда 1-го числа со 2-м разрядом 2-го числа и т. д.) до тех пор, пока либо разряды не будут различны (тогда числа различны), либо все разряды не окажутся одинаковыми (тогда числа равны). Но для этого нужно знать границы разрядов каждого числа. Это можно сделать либо раздвижением кода со вставкой разделителей между чисел (довольно трудоемкая процедура по числу шагов), либо различным перекодированием разрядов второго в паре числа: 0 кодируется символом “a”, а 1 кодируется символом “b”(такой код мы будем отмечать штрихом у имени числа). Перекодировав число B в B', мы сможем сравнивать разряды A и B', удаляя сравниваемый разряд у первого числа A и помечая для удаления (символом “d”) разряды числа B'. Но в этом случае перед сравнением разрядов первой пары мы скопируем первое число A в конец кода с перекодированием в A', а исходный код A перекодируем в A'' по правилу: $0 \rightarrow c, 1 \rightarrow e$ (это нужно для возврата к 1-му неперекодированному разряду A). Тогда сравнение второй пары C и A' мы сможем провести так же, как сравнение первой пары.

Таким образом, на предварительном этапе мы преобразуем код ABC в код $A''B'CA'$, а затем будем сравнивать пару A'' и B' со стиранием их кодов и пару C и A' со стиранием их кодов. Если при сравнениях условие задачи выполнится, то можно заменить стертый код на 1, а если в процессе проверки условия будет обнаружено его невыполнение, то код стирается и заменяется на 0.

Рассматриваемый на каждом шаге алгоритма символ мы обозначим через s. Его значением могут быть на разных этапах символы множества $\{0, 1, a, b, c, d, e, \lambda\}$.

Описание алгоритма

Сначала опишем общий алгоритм, который непосредственно реализует идею алгоритма, а затем разработаем детализацию для тех шагов общего алгоритма, которые не смогут быть реализованы одним состоянием МТ. Если же шаг может быть реализован одним состоянием МТ, то описание этих действий дадим в скобках этого же шага.

Общий алгоритм

1. Подготовительный этап: перекодирование B в B' и копирование A в конец кода с перекодированием в A' и исходного в A''.
2. Сравнение A'' с B'. Если $A'' = B'$, то переход к шагу 4 (условие не выполнено); иначе переход к шагу 3 (продолжение проверки условия).
3. Сравнение C с A'. Если $C = A'$, то переход к шагу 5 (условие выполнено); иначе переход к шагу 4.
4. Замена кода со стиранием на 0. Конец алгоритма.
5. Замена кода на 1. Конец алгоритма.

Детализация шага 1 общего алгоритма

1.1 Перекодирование B в B': $0 \to a, 1 \to b$.

1.2 Копирование A в конец кода с перекодированием в A'' и A'.

Детализация шага 1.1

1.1.1 Сдвиг вправо к 1-му разряду B.

1.1.2 Перекодирование ($0 \to a, 1 \to b$) и сдвиг вправо.

1.1.3 Перекодирование ($0 \to a, 1 \to b$) и сдвиг вправо.

1.1.4 Перекодирование ($0 \to a, 1 \to b$) и сдвиг влево.

1.1.5 Возврат к 1-му символу кода (сдвиг влево до λ и вправо на 1 разряд).

Детализация шага 1.1.1

1.1.1.1 Сдвиг вправо на 1 разряд.

1.1.1.2 Сдвиг вправо на 1 разряд.

1.1.1.3 Сдвиг вправо на 1 разряд.

Детализация шага 1.2

1.2.1 Если $s \in \{a, b\}$, то переход к шагу 2.1.
Если $s = 0$, то замена символа на "c" и переход к шагу 1.2.2.
Если $s = 1$, то замена символа на "e" и переход к шагу 1.2.4.

1.2.2 Движение вправо в конец кода (до λ) и замена на символ "a".

1.2.3 Возврат к 1-му нескопированному символу кода A (сдвиг влево до $s \in \{c, e\}$ и вправо на 1 разряд) и переход к шагу 1.2.1.

1.2.4 Движение вправо в конец кода (до λ), замена на символ "b" и переход к шагу 1.2.3.

Детализация шага 2

2.1 Движение в начало кода (влево до λ и вправо на 1 разряд).

2.2 Если $s = c$, то удаление символа и переход к шагу 2.3.
Если $s = e$, то удаление символа и переход к шагу 2.5.
Если $s = d$, то переход к шагу 4.1 ($A = B$ – условие задачи не выполнено).

2.3 Движение вправо до символа из $\{a, b\}$.

2.4 Если $s = b$, то переход к шагу 3.1 ($A \neq B$ – 1-я часть условия выполнена).
Если $s = a$, то замена на d и переход к шагу 2.1

2.5 Движение вправо до символа из $\{a, b\}$.

2.6 Если $s = a$, то переход к шагу 3.1 ($A \neq B$ – 1-я часть условия выполнена).
Если $s = b$, то замена на d и переход к шагу 2.1

Детализация шага 3

3.1 Удаление части кода до кода C (стирание символа $s \in \{a, b, c, d, e\}$ с движением вправо на 1 разряд и повторением шага) и переход к шагу 3.3.

3.2 Движение в начало кода (влево до λ и вправо на 1 разряд).

3.3 Если $s = 0$, то удаление символа и переход к шагу 3.4.
Если $s = 1$, то удаление символа на и переход к шагу 3.6.
Если $s = d$, то переход к шагу 5.1 ($C = A$ – условие задачи выполнено).

3.4 Движение вправо до символа из $\{a, b\}$.

3.5 Если $s = b$, то переход к шагу 4.1 ($A \neq B$ – 2-я часть условия не выполнена).
Если $s = a$, то замена на d и переход к шагу 3.2

3.6 Движение вправо до символа из $\{a, b\}$.

3.7 Если $s = a$, то переход к шагу 4.1 ($A \neq B$ – 2-я часть условия не выполнена).
Если $s = b$, то замена на d и переход к шагу 3.2

Детализация шага 3.1

3.1.1 Движение в начало кода.

3.1.2 Удаление части кода до кода C (стирание символа $s \in \{a, b, c, d, e\}$ с движением вправо на 1 разряд и повторением шага) и переход к шагу 3.3.

Детализация шага 4

4.1 Движение в начало кода (влево до λ и вправо на 1 разряд).

4.2 Стирание всего кода (стирание непустого символа с движением вправо на 1 разряд и повторение шага до пустого символа) и записи 0 – конец алгоритма.

Детализация шага 5

5.1 Движение в начало кода (влево до λ и вправо на 1 разряд).

5.2 Стирание всего кода (стирание непустого символа с движением вправо на 1 разряд и повторением шага до пустого символа) и записи 1 – конец алгоритма.

Тестирование

При тестировании будем в скобках писать номер шага алгоритма перед обозреваемым символом строки. Конец алгоритма будем обозначать шагом 0.

Тестирование детализации шага 1.1

Тест 1 $(1.1.1)010011010 \to (1.2)010abb010$

$(1.1.1)010011010 \to 010(1.1.2)011010 \to 010a(1.1.3)11010 \to 010ab(1.1.4)1010 \to 010abb(1.1.5)010 \to (1.1.5)\lambda 010abb010 \to (1.2)010abb010.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тестирование детализации шага 1.2

Тест 1 $(1.2.1)010abb010 \to cec(2.1)abb010aba$

$(1.2.1)010abb010 \to (1.2.2)c10abb010 \to c10abb1010(1.2.2) \to c10abb010(1.2.3)a \to (1.2.3)c10abb010a \to c(1.2.1)10abb010a \to ce(1.2.4)10abb010a \to ce0abb010a(1.2.4) \to ce0abb010a(1.2.3)b \to c(1.2.3)e0abb010ab \to ce(1.2.1)0abb010ab \to ce(1.2.2)cabb010ab \to cecabb010ab(1.2.2) \to cecabb010ab(1.2.3)a \to ce(1.2.3)cabb010aba \to cec(1.2.1)abb010aba \to cec(2.1)abb010aba.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тестирование детализации шага 1

Тест 2 $(1.1)110110110 \to eec(2.1)bba110bba$

$(1.1)110110110 \to (1.2)110bba110 \to eec(2.1)bba110bba.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тестирование детализации шага 2

Тест 1 (вып. 1-й ч. усл.) $cec(2.1)abb010aba \to dd(3.1)b010aba$

$cec(2.1)abb010aba \to (2.2)cecabb010aba \to (2.3)ecabb010aba \to ec(2.4)abb010aba \to ec(2.1)dbb010aba \to (2.2)ecdbb010aba \to$

$(2.5)cdbb010aba \to\ cd(2.6)bb010aba \to\ cd(2.1)db010aba \to$
$(2.2)cddb010aba \to\ (2.3)ddb010aba \to\ dd(2.4)b010aba\ \to$
$dd(3.1)b010aba.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 2 (невып. 1-й ч. усл.) $eec(2.1)bba110bba\ \to\ (4.1)ddd110bba$

$eec(2.1)bba110bba\ \to\ (2.2)eecbba110bba\ \to\ (2.5)ecbba110bba\ \to\ ec(2.6)bba110bba\ \to$
$ec(2.1)dba110bba\ \to\ (2.2)ecdba110bba \to$
$(2.5)cdba110bba \to\ cd(2.6)ba110bba \to\ cd(2.1)da110bba \to$
$(2.2)cdda110bba \to\ (2.3)dda110bba \to\ dd(2.4)a110bba \to$
$dd(2.1)d110bba \to\ (2.2)ddd110bba\ \to\ (4.1)ddd110bba.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тестирование детализации шага 3.1

Тест 1 $dd(3.1.1)b010aba\ \to\ (3.3)010aba$

$dd(3.1.1)b010aba \to (3.1.1)\lambda ddb010aba \to (3.1.2)ddb010aba \to$
$(3.1.2)010aba \to (3.3)010aba.$

Тестирование детализации шага 3

Тест 1 (вып. 2-й ч. усл.) $dd(3.1)b010aba\ \to\ (5.1)ddd$

$dd(3.1)b010aba\ \to\ (3.3)010aba\ \to\ (3.4)10aba\ \to\ 10(3.5)aba\ \to$
$10(3.2)dba\ \to (3.2)\lambda 10dba\ \to\ (3.2)10dba\ \to\ (3.3)10dba\ \to$
$(3.6)0dba\ \to 0d(3.7)ba\ \to\ 0d(3.2)da\ \to\ (3.2)\lambda 0dda\ \to\ (3.2)0dda\ \to$
$(3.3)0dda\ \to\ (3.4)dda\ \to\ dd(3.5)a\ \to\ dd(3.2)d\ \to\ (3.2)\lambda ddd\ \to$
$(3.2)ddd\ \to\ (3.3)ddd\ \to\ (5.1)ddd.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 3 (невып. 2-й ч. усл.) $ced(3.1)ba100aab\ \to\ 00(4.1)aab$

$ced(3.1)ba100aab\ \to\ (3.3)100aab\ \to\ (3.6)00aab\ \to 00(3.7)aab\ \to$
$00(4.1)aab.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тестирование детализации шага 4

Тест 2 (невып. 1-й ч. усл.) $(4.1)ddd110bba\ \to\ (0)0$

$(4.1)ddd110bba\ \to\ (4.2)ddd110bba\ \to\ (4.2)\lambda\ \to\ (0)0.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 3 (невып. 2-й ч. усл.) $00(4.1)aab \rightarrow (0)0$

$00(4.1)aab \rightarrow (4.2)00aab \rightarrow (4.2)\lambda \rightarrow (0)0.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тестирование детализации шага 5

Тест 1 (вып. усл.) $(5.1)ddd \rightarrow (0)1$

$(5.1)ddd \rightarrow (5.2)\lambda \rightarrow (0)1.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тестирование общего алгоритма

Тест 1 (вып. усл.) $(1)010011010 \rightarrow (0)1$

$(1)010011010 \rightarrow cec(2)abb010aba \rightarrow dd(3)b010aba \rightarrow$
$(5)ddd \rightarrow (0)1.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 2 (невып. 1-й ч. усл.) $(1)110110110 \rightarrow (0)0$

$(1)110110110 \rightarrow eec(2)bba110bba \rightarrow (4)ddd110bba \rightarrow (0)0.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 3 (невып. 2-й ч. усл.) $(1)001010100 \rightarrow (0)0$

$(1)001010100 \rightarrow cce(2)aba100aab \rightarrow ed(3)da100aab \rightarrow$
$00(4)aab \rightarrow (0)0.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*

Тест 4 (невып. обоих ч. усл.) $(1)110110110 \rightarrow (0)0$

$(1)101101011 \rightarrow ece(2)bab011bab \rightarrow (4)ddd011bab \rightarrow (0)0.$
Результат совпадает с ожидаемым. *Тест прошел успешно.*
Все тесты прошли успешно. *Тестирование общего алгоритма закончено. Тестирование всех алгоритмов завершено успешно.*

Функциональная таблица МТ

Поскольку каждый шаг алгоритма с детализацией можно реализовать одним состоянием МТ, то будем именовать состояние именем q с индексом, соответствующим номеру шага.

$Q\backslash A$	λ	0	1	a	b	c	e	d	$Comment$
q_{1111}		$q_{1112}R$	$q_{1112}R$						$\rightarrow B$
q_{1112}		$q_{1113}R$	$q_{1113}R$						$\rightarrow B$
q_{1113}		$q_{112}R$	$q_{112}R$						$\rightarrow B$
q_{112}		$aq_{113}R$	$bq_{113}R$						$recode\ B'$
q_{113}		$aq_{114}R$	$bq_{114}R$						$recode\ B'$
q_{114}		$aq_{115}L$	$bq_{115}L$						$recode\ B'$
q_{115}	$q_{121}R$	L	L						$A \leftarrow$
q_{121}		cq_{122}	eq_{124}	q_{21}	q_{21}				$recode\ A''$
q_{122}	aq_{123}	R	R	R	R	R	R		$code\ A'$
q_{123}		L	L	L	L	$q_{121}R$	$q_{121}R$		$A \leftarrow$
q_{124}	bq_{123}	R	R	R	R	R	R		$code\ A'$
q_{21}	$q_{22}R$	L	L	L	L	L	L		$A'' \leftarrow$
q_{22}						λq_{23}	λq_{25}	q_{41}	$delete\ A''$
q_{23}				q_{24}	q_{24}	R	R	R	$\rightarrow B'$
q_{24}				dq_{21}	q_{311}				$A'' = B'\ ?$
q_{25}				q_{26}	q_{26}	R	R	R	$\rightarrow B'$
q_{26}				q_{311}	dq_{21}				$A'' = B'\ ?$
q_{311}	$q_{312}R$			L	L	L	L	L	$first \leftarrow$
q_{312}		q_{33}	q_{33}	λR	λR	λR	λR	λR	$del\ A'', B'$
q_{32}	$q_{33}R$			L	L	L	L	L	$first \leftarrow$
q_{33}		$\lambda q_{34}R$	$\lambda q_{36}R$					q_{51}	$delete\ C$
q_{34}		R	R	q_{35}	q_{35}			R	$\rightarrow A'$
q_{35}				dq_{31}	q_{41}				$C = A'\ ?$
q_{36}		R	R	q_{37}	q_{37}			R	$\rightarrow A'$
q_{37}				q_{41}	dq_{31}				$C = A'\ ?$
q_{41}	$q_{42}R$	L	L	L	L	L	L	L	$first \leftarrow$
q_{42}	$0q_0$	λR	λR	λR	λR	λR	λR	λR	$rezult$
q_{51}	$q_{52}R$	L	L	L	L	L	L	L	$first \leftarrow$
q_{52}	$1q_0$	λR	λR	λR	λR	λR	λR	λR	$rezult$

Тестирование МТ

Тест 1 (вып. усл.) $010011010 \to 1$

$$010011010 \to 010011010 \to 010011010 \to 010011010 \to 010a11010 \to$$

$$\overset{q_{1111}}{010ab1010} \to \overset{q_{1112}}{010abb010} \to \cdots \to \overset{q_{1113}}{010abb010} \to \overset{q_{112}}{010abb010} \to \overset{q_{113}}{}$$

$$\overset{q_{114}}{c10abb010} \to \cdots \to \overset{q_{115}}{c10abb010} \to \overset{q_{115}}{c10abb010a} \to \overset{q_{121}}{} \cdots \to$$

$$\overset{q_{122}}{c10abb010a} \to \overset{q_{122}}{c10abb010a} \to \overset{q_{123}}{ce0abb010a} \to \cdots \to ce0abb010a \to$$

$$\overset{q_{123}}{ce0abb010ab} \to \overset{q_{121}}{} \cdots \to \overset{q_{124}}{ce0abb010ab} \to ce0abb010ab \to \overset{q_{124}}{}$$

$$\overset{q_{123}}{cecabb010ab} \to \cdots \to \overset{q_{123}}{cecabb010ab} \to \overset{q_{121}}{cecabb010aba} \to \cdots \to$$

$$\overset{q_{122}}{cecabb010aba} \to cecabb010aba \to \overset{q_{122}}{cecabb010aba} \to \overset{q_{123}}{} \cdots \to cecabb010aba \to$$

$$\overset{q_{123}}{cecabb010aba} \to \overset{q_{121}}{ecabb010aba} \to \overset{q_{21}}{ecabb010aba} \to ecabb010aba \to \overset{q_{21}}{}$$

$$\overset{q_{22}}{ecabb010aba} \to \overset{q_{23}}{ecdbb010aba} \to \overset{q_{23}}{} \cdots \to ecdbb010aba \to \overset{q_{23}}{ecdbb010aba} \to$$

$$\overset{q_{24}}{cdbb010aba} \to \overset{q_{21}}{} \cdots \to \overset{q_{21}}{cdbb010aba} \to cddb010aba \to \cdots \to \overset{q_{22}}{cddb010aba} \to$$

$$\overset{q_{26}}{cddb010aba} \to \overset{q_{26}}{ddb010aba} \to \overset{q_{21}}{} \cdots \to ddb010aba \to \overset{q_{21}}{ddb010aba} \to$$

$$\overset{q_{22}}{ddb010aba} \to \overset{q_{23}}{ddb010aba} \to \overset{q_{23}}{ddb010aba} \to \cdots \to \overset{q_{24}}{010aba} \to$$

$$\overset{q_{311}}{010aba} \to \overset{q_{311}}{10aba} \to \cdots \to \overset{q_{312}}{10aba} \to 10aba \to \overset{q_{312}}{10dba} \to \cdots \to 10dba \to$$

$$\overset{q_{33}}{10dba} \to \overset{q_{34}}{0dba} \to \cdots \to \overset{q_{34}}{0dba} \to \overset{q_{35}}{0dba} \to \overset{q_{32}}{0dda} \to \cdots \to \overset{q_{32}}{0dda} \to$$

$$\overset{q_{33}}{0dda} \to \overset{q_{36}}{dda} \to \cdots \to \overset{q_{36}}{dda} \to \overset{q_{37}}{dda} \to \overset{q_{32}}{ddd} \to \cdots \to \overset{q_{32}}{ddd} \to ddd \to$$

$$\overset{q_{33}}{ddd} \to \overset{q_{34}}{} \to \overset{q_{34}}{} \to \overset{q_{35}}{1}.$$

$$\overset{q_{51}}{} \quad \overset{q_{51}}{} \overset{q_{52}}{} \overset{q_{0}}{}$$

Тест 2 (невып. 1-й ч. усл.) $110110110 \to 0$

$$110110110 \to 110110110 \to 110110110 \to 110110110 \to 110b10110 \to$$

$$\overset{q_{1111}}{110bb0110} \to \overset{q_{1112}}{110bba110} \to \cdots \to \overset{q_{1113}}{110bba110} \to \overset{q_{112}}{110bba110} \to \overset{q_{113}}{}$$

$$\overset{q_{114}}{e10bba110} \to \cdots \to \overset{q_{115}}{e10bba110} \to \overset{q_{115}}{e10bba110b} \to \overset{q_{121}}{} \cdots \to$$

$$\overset{q_{124}}{e10bba110b} \to e10bba110b \to \overset{q_{124}}{ee0bba110b} \to \overset{q_{123}}{} \cdots \to$$

$$\overset{q_{123}}{ee0bba110b} \to \overset{q_{121}}{ee0bba110bb} \to \overset{q_{124}}{} \cdots \to ee0bba110bb \to$$

$$\overset{q_{124}}{ee0bba110bb} \to \overset{q_{123}}{eecbba110bb} \to \cdots \to \overset{q_{123}}{eecbba110bb} \to$$

$$\overset{q_{121}}{eecbba110bba} \to \overset{q_{122}}{} \cdots \to eecbba110bba \to \overset{q_{122}}{eecbba110bba} \to$$

$$\overset{q_{123}}{eecbba110bba} \to \cdots \to \overset{q_{123}}{eecbba110bba} \to \overset{q_{121}}{eecbba110bba} \to$$

$$\overset{q_{21}}{ecbba110bba} \to \overset{q_{21}}{ecbba110bba} \to \overset{q_{22}}{ecbba110bba} \to ecbba110bba \to$$

$$\overset{q_{25}}{ecdba110bba} \to \overset{q_{25}}{} \cdots \to \overset{q_{25}}{ecdba110bba} \to \overset{q_{26}}{ecdba110bba} \to cdba110bba \to$$

$$\overset{q_{21}}{cdba110bba} \to \cdots \to \overset{q_{21}}{cdba110bba} \to \overset{q_{22}}{cdda110bba} \to \cdots \to \overset{q_{25}}{cdda110bba} \to$$

$$\overset{q_{26}}{cdda110bba} \to \overset{q_{26}}{dda110bba} \to \cdots \to \overset{q_{21}}{dda110bba} \to \overset{q_{21}}{dda110bba} \to$$

$$\overset{q_{22}}{ddd110bba} \to \overset{q_{23}}{ddd110bba} \to \overset{q_{23}}{ddd110bba} \to \overset{q_{24}}{ddd110bba} \to ddd110bba \to$$

$$\overset{q_{21}}{ddd110bba} \to \overset{q_{21}}{} \cdots \to \overset{q_{22}}{} \to \overset{q_{41}}{} \overset{q_{41}}{} \overset{q_{42}}{} \overset{q_{42}}{} \overset{q_0}{0}.$$

Тест 3 (невып. 2-й ч. усл.) $001010100 \to 0$

$$001010100 \to 001010100 \to 001010100 \to 001010100 \to 001a10100 \to$$

$$\overset{q_{1111}}{001ab0100} \to \overset{q_{1112}}{001aba100} \to \cdots \to \overset{q_{1113}}{001aba100} \to \overset{q_{112}}{001aba100} \to \overset{q_{113}}{}$$

$$\overset{q_{114}}{c01aba100} \to \cdots \to \overset{q_{115}}{c01aba100} \to \overset{q_{115}}{c01aba100a} \to \overset{q_{121}}{} \cdots \to$$

$$\overset{q_{122}}{c01aba100a} \to c01aba100a \to \overset{q_{122}}{cc1aba100a} \to \overset{q_{123}}{} \cdots \to$$

$$\overset{q_{123}}{cc1aba100a} \to \overset{q_{121}}{cc1aba100aa} \to \cdots \to \overset{q_{122}}{cc1aba100aa} \to$$

$$\overset{q_{122}}{cc1aba100aa} \to \overset{q_{123}}{cceaba100aa} \to \cdots \to \overset{q_{123}}{cceaba100aa} \to$$

$$\overset{q_{121}}{cceaba100aab} \to \cdots \to \overset{q_{124}}{cceaba100aab} \to cceaba100aab \to \overset{q_{124}}{}$$

$$\overset{q_{123}}{cceaba100aab} \to \cdots \to \overset{q_{123}}{cceaba100aab} \to \overset{q_{121}}{cceaba100aab} \to$$

$$\overset{q_{21}}{ceaba100aab} \to \overset{q_{21}}{ceaba100aab} \to \overset{q_{22}}{ceaba100aab} \to ceaba100aab \to$$

$$\overset{q_{23}}{cedba100aab} \to \cdots \to \overset{q_{23}}{cedba100aab} \to \overset{q_{23}}{cedba100aab} \to \overset{q_{24}}{cedba100aab} \to$$

$$\overset{q_{21}}{edba100aab} \to \cdots \to \overset{q_{21}}{edba100aab} \to \overset{q_{22}}{edba100aab} \to \cdots \to \overset{q_{23}}{edba100aab} \to$$

$$\overset{q_{24}}{edba100aab} \to \cdots \to \overset{q_{24}}{100aab} \to \overset{q_{311}}{100aab} \to 00aab \to \overset{q_{311}}{00aab} \to$$

$$\overset{q_{312}}{00aab} \to \overset{q_{312}}{00aab} \to \overset{q_{33}}{00aab} \to \cdots \to \overset{q_{36}}{} \to \overset{q_{37}}{} \; 0$$

$$\overset{q_{41}}{} \quad \overset{q_{41}}{} \quad \overset{q_{42}}{} \quad \overset{q_{42}}{} \quad \overset{q_0}{}.$$

Учебное издание

Рублев Вадим Сергеевич

АЛГОРИТМЫ И МАШИНЫ ТЬЮРИНГА

(индивидуальная работа № 7 по дисциплине
«Дискретная математика»)

Учебно-методическое пособие

Редактор, корректор Л. Н. Селиванова
Компьютерная верстка В. С. Рублев

Подписано в печать 28.01.2019 Формат 60×84/16.
Усл. печ. л. 3,72. Уч.-изд. л. 3,0.
Тираж 3 экз. Заказ .

Оригинал-макет подготовлен
в редакционно-издательском отделе ЯрГУ.

Ярославский государственный университет им. П. Г. Демидова
150003, Ярославль, ул. Советская, 14.