МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

КУРСОВАЯ РАБОТА

По дисциплине «Фундаментальная информатика»

На тему «Алгоритмические модели Тьюринга и Маркова»

Выполнил:
Студент группы М8О-108Б-23

Власко Михаил Михайлович

Проверил:

Преподаватель

Севастьянов Виктор Сергеевич

Москва 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В рамках предмета фундаментальной информатики важно дать формальное определение алгоритма. Неформальное определение заключается в том, что алгоритм – это точно заданная последовательность правил, указывающая, каким образом можно за конечное число шагов получить выходное сообщение определённого вида, используя заданное входное сообщение, при этом, действия, предписываемые алгоритмом должны быть чисто механическими, всем понятными и легко выполнимыми. Данное определение слишком расплывчато, так как нет чёткого определения слов «всем понятными и легко выполнимыми», порядок действий, кажущийся кому-то элементарным, может описываться весьма сложным алгоритмом. Кроме того, не все математические задачи алгоритмически разрешимы. При этом для установления неразрешимости какой-либо задачи необходимы суждения обо всех возможных алгоритмах, что неизбежно требует чёткого определения того, что является алгоритмом, а что нет. Для этого и необходимо формальное и строгое определение алгоритма.

Формализация понятия алгоритма может быть произведена при помощи построения алгоритмических моделей. В данной работе будут рассмотрены два основных вида алгоритмических моделей: машины Тьюринга и нормальные алгоритмы Маркова. В первом случае алгоритм представляется в качестве процесса работы некоторой машины, способной выполнять небольшое число простых операций, во втором же алгоритм описывается, как набор преобразований слов в произвольных алфавитах.

Целью работы является иллюстрирование определения алгоритма путём построения алгоритмических моделей Тьюринга и Маркова.

В рамках работы будут выполнено составление алгоритмов для решения поставленных задач с помощью машины Тьюринга, эквивалентного ей графического представления – диаграммы Тьюринга, а также нормальных алгоритмов Маркова.

ГЛАВА 1. МАШИНЫ ТЬЮРИНГА

1.1 Вводная часть

Алгоритмическая модель машины Тьюринга была предложена английским математиком Аланом Тьюрингом в 1936 году.

Машина Тьюринга состоит из ограниченной с одного конца бесконечной ленты, разделённой на ячейки, а также комбинированной читающей и пишущей головки, которая может перемещаться вдоль ленты от ячейки к ячейке. В каждой такой ячейке может быть записан один знак некоторого алфавита, называемого рабочим алфавитом МТ, либо несобственный знак (пробел, обозначаемый λ). Головка МТ в каждый момент времени располагается над одной из ячеек ленты, называемой рабочей ячейкой, и воспринимает знак, записанный в этой ячейке. При этом головка находится в одном из конечного множества дискретных состояний, среди которых выделено одно начальное.

Согласно формальному определению, машиной Тьюринга называется упорядоченная четвёрка объектов $T = (A, Q, P, q_0)$, где T – символ МТ, A – конечное множество букв (рабочий алфавит), Q – конечное множество символов (имён состояний), q_0 – имя начального состояния, P – множество упорядоченных четвёрок (q, a, v, q'), $q, q' \in Q, a \in A \cup \{\lambda\}, v \in \{l, r\} \cup A \cup \{\lambda\}$ (программа), определяющее три функции: функцию выхода $F_l: Q \times \bar{A} \to \bar{A}$ $(\bar{A} = A \cup \{\lambda\})$, функцию переходов $F_t: Q \times \bar{A} \to Q$, и функцию движения головки $F_v: Q \times \bar{A} \to \{l, r, s\}$ (символ s означает, что головка неподвижна).

Первоначально программы машин Тьюринга задавались наборами пятёрок, включавших как функцию выхода, так и функцию движения. В пятёрках движение головки и запись буквы дополняют друг друга, а в четвёрках они являются взаимно исключающими действиями.

Также целесообразным будет дать определение состоянию машины Тьюринга. Состоянием (или ситуацией) ленты МТ называется упорядоченная пара объектов $S = (z, k)$, где S – имя ситуации, z – сообщение, записанное на ленте, k – неотрицательно целое число, равное расстоянию (в ячейках) от края ленты до рабочей ячейки. Иными словами, k фиксирует положение рабочей ячейки на ленте. Иными словами, состояние – это всё, что записано на ленте с выделенной рабочей ячейкой. Состояния записываются в наглядном виде, при котором левый край ленты обозначается квадратной

скобкой, правый, уходящий на бесконечность край – треугольной скобкой, а рабочая ячейка выделяется круглыми скобками (рисунок 1).

$$S = [a_{i_1} a_{i_2} \dots a_{i_{k-1}} (a_{i_k}) a_{i_{k+1}} \dots a_{i_n} \lambda\rangle$$

Рисунок 1 – Пример описания состояний МТ

Фактически, программа МТ задаётся упорядоченными наборами (в случае данной задачи - четвёрками) символов (командами) вида *a,b,c,d*, записанными каждая на отдельной строке, где *a* – имя состояния, в котором машина должна находиться в момент выполнения команды, *b* – знак, над которым должна располагаться головка машины в момент выполнения команды, *c* – знак, который нужно поставить на позицию, в которой в данный момент расположена головка, либо символ действия (*l, r, s*), которое нужно выполнить, *d* – имя состояния, в которое машина должна перейти после выполнения команды. Команды выполняются путём сопоставления конечных и начальных состояний, а также знаком, расположенным в текущей ячейки на ленте МТ, и порядок их выполнения не зависит от порядка, в котором они записаны в тексте программы.

В рамках решения поставленной задачи будет использоваться интерпретатор программ машины Тьюринга в четвёрках *jstu4*, реализованный в формате web-страницы (рисунок 2), принимающий на ввод текст программы и входное сообщение и выдающий на выходе преобразованное сообщение.

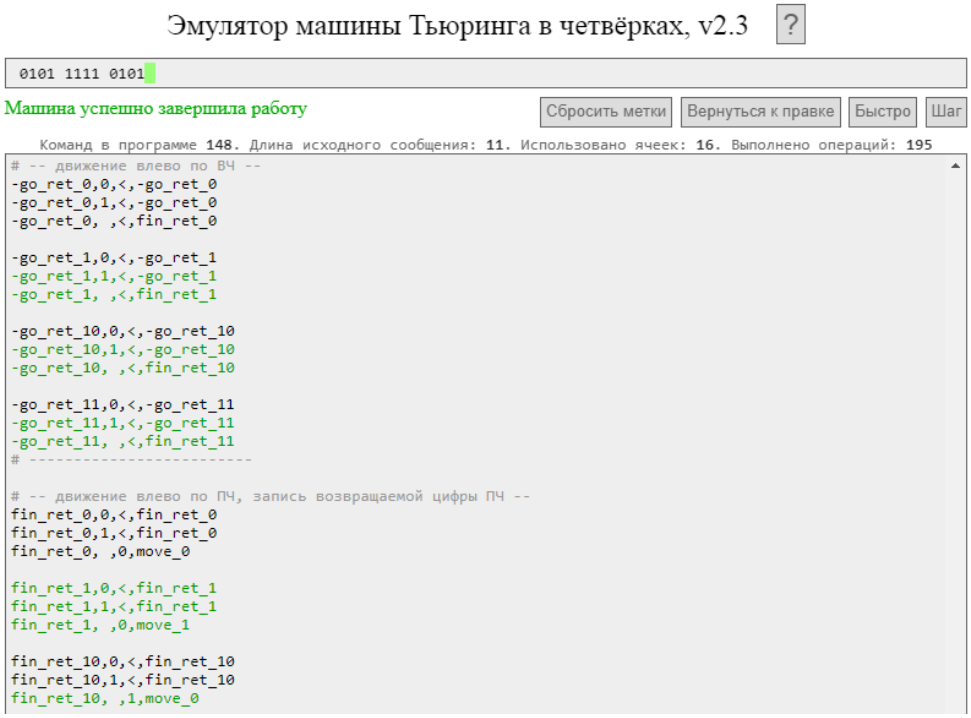

Рисунок 2 – Интерфейс интерпретатора *jstu4*

Интерфейс интерпретатора также допускает использование комментариев.

1.2 Постановка задачи

В рамках задачи, выданной под вариантом №21, необходимо составить алгоритм, выделяющий разряды второго двоичного числа по маске, заданной первым числом. Задание подразумевает, что оба числа одинаковой длины, и для выходного числа выполняется следующее соответствие:

$$a_i = \begin{cases} b_i, & \text{если } c_i = 1, \\ 0, & \text{если } c_i = 0, \end{cases} \qquad (1)$$

где a_i – i-ая цифра выходного числа, b_i – i-ая цифра второго входного числа, c_i – i-ая цифра первого входного числа (маски). Числа на вход интерпретатора поступают, разделённые пробелом, перед первым числом также расположен пробел. Рабочим алфавитом машины будет являться множество $\{0, 1\}$.

1.3 Идея решения задачи

В рамках решения предстоит выполнить две, фактически независимые задачи: копирование второго числа на область ленты, расположенную непосредственно справа от него, и дальнейшее преобразование копии для получения результата. Копирование необходимо для того, чтобы входные данные остались неизменными (обеспечение нормированности).

Далее, при помощи смены состояний в зависимости от значения очередной цифры головка будет обходить все цифры первого числа (далее – маски) и преобразовывать в соответствии с правилом (1) цифры копии.

1.4 Описание алгоритма

В дальнейшем под «запоминанием», «записью в память» или «стиранием из памяти» будет подразумеваться процесс, при котором головка машины, после выполнения определённой команды, начинает выполнять команды и переходить между состояниями из отдельной, независимой и параллельной другим ветки, которая ассоциирована с данной командой (например, восприятием конкретного знака). Некоторые состояния машины,

на которые будет ссылаться описание, приведены на рисунке 3 для конкретного примера входных данных (010 111).

В рамках первого этапа будет производиться копирование второго числа в область, отделённую от его правого края одним пробелом. В начальном состоянии головка находится непосредственно справа от второго числа (S_1). В процессе копирования головка МТ будет пробегать всё второе число до пробела между ним и маской, затем смещаться на ячейку вправо, воспринимать цифру, затирать её (S_2), далее переходить в соответствующее цифре состояние и двигаться к области копирования, в которой копируемая цифра и будет записана, сразу после первого пробела (S_3). Далее головка возвращается к затёртой цифре, восстанавливает её, смещается на ячейку вправо, затирает очередную цифру, перемещается вправо по всему второму числу, переезжает первую цифру копии, и пишет только что затёртую цифру непосредственно справа от неё. Данная процедура повторяется до тех пор, пока не будет скопировано всё второе число. Тот факт, что число полностью скопировано, будет ознаменован обнаружением при сдвиге вправо на ячейку пробела, а не очередной цифры.

После завершения копирования, головка, расположенная справа от копии (S_4), сдвигается к её левому краю, считывает первую слева цифру копии, затирает её, двигается к первой слева цифре маски. Если эта цифра – 0, факт того, какой именно была затёртая цифра копии, «стирается из памяти» программы, так как в любом случае вернётся 0. Первая цифра маски так же запоминается и затирается (S_5), а головка движется к затёртой цифре копии, и пишет туда либо 0 (S_6), если цифра маски была 0, либо цифру, ранее здесь затёртую и хранящуюся в памяти, если первая цифра маски была единица. Затем подобным образом считывается и затирается вторая цифра копии, головка движется к затёртой цифре маски и восстанавливает её, сдвигается на ячейку вправо, считывает очередную цифру маски и затирает её (S_7), возвращается к затёртой цифре копии, пишет туда 0 или затёртую цифру (S_8), и так далее до того момента, пока не будут преобразованы все цифры копии второго числа. Этот процесс преобразования будет завершён, по аналогии с копированием, когда вместо очередной цифры маски справа окажется пробел. Головка остановится непосредственно справа от преобразованной копии (S_9).

$$S_1 = [\lambda 010\lambda 111(\lambda)\lambda >$$

$$S_2 = [\lambda 010\lambda(\lambda)11\lambda\lambda >$$

$$S_3 = [\lambda 010\lambda\lambda 11\lambda(1)\lambda >$$

$$S_4 = [\lambda 010\lambda 111\lambda 111(\lambda)\lambda >$$

$$S_5 = [\lambda(\lambda)10\lambda 111\lambda\lambda 11\lambda\lambda >$$

$$S_6 = [\lambda\lambda 10\lambda 111\lambda(0)11\lambda\lambda >$$

$$S_7 = [\lambda 0(\lambda)0\lambda 111\lambda 0\lambda 1\lambda\lambda >$$

$$S_8 = [\lambda 0\lambda 0\lambda 111\lambda 0(1)1\lambda\lambda >$$

$$S_9 = [\lambda 010\lambda 111\lambda 010(\lambda)\lambda >$$

Рисунок 3 – Некоторые рассмотренные состояния МТ

Исходный код программы с подробными комментариями, поясняющими суть конкретных состояний и систему их взаимодействие, представлен в приложении А.

1.5 Тестирование и оценка сложности

В рамках тестирования работоспособности алгоритма на ввод программы были переданы несколько чисел краевого вида, такие как (0 0), (1 1), (000 111) и другие. Полный список рассмотренных тестовых случаев представлен в таблице 1.

Таблица 1 – Список рассмотренных тестовых случаев

Ввод	Вывод
0 0	0 0 0
1 1	1 1 1
0 1	0 1 0
000 111	000 111 000
111 000	111 000 000
010 101	010 101 000
10110 11011	10110 11011 10010

Сложность алгоритма оценивалась по количеству элементарных операций (команд), выполненных машиной во время выполнения алгоритма. Эта величина зависит только от длины входного сообщения. Результаты тестирования на входных сообщениях разной длины приведены в таблице 2.

Таблица 2 – Зависимость входного сообщения и количества операций

Длина входного сообщения	Количество выполненных операций
2	39
4	79
8	195
16	571
32	1899

На основании вышеприведённых результатов можно отметить, что при увеличении длины входного сообщения в 2 раза количество выполненных операций увеличивается больше чем в 2, но меньше чем в 4 раза, из чего можно сделать вывод, что сложность алгоритма приблизительно равна $O(n\log n)$.

1.6 Выводы по главе

Разработан алгоритм выделения разрядов второго двоичного числа по маске, заданной первым числом, составлена соответствующая программа Машины Тьюринга в четвёрках. Получен практический опыт работы с абстрактным исполнителем Машина Тьюринга, написания алгоритма обработки двоичных чисел на языке предельно низкого уровня.

Получен опыт составления алгоритмов на языке низкого уровня, при котором необходимо многие элементарные действия (перемещение головки по слову, копирование знака и т. д.) задавать большим количеством команд. Вызвано это, прежде всего, тем, что единственным способом «запоминания» какого либо знака или выполнения какого-либо действия является ассоциация его с конкретным состоянием или конкретным знаком. Поэтому, для выполнения каждого элементарного действия нужно задавать столько состояний, над сколькими знаками может оказаться головка машины в момент выполнения этого действия, и для каждого состояния дублировать однотипные команды. Поэтому, при наличии более объёмного алфавита, чем в условии выполненной задачи, код программы был бы гораздо более громоздким.

ГЛАВА 2. ДИАГРАММЫ ТЬЮРИНГА

2.1 Вводная часть

Диаграммы Тьюринга являют собой графическое представление машины Тьюринга, они позволяют избавиться от нагромождения состояний, упомянутого в качестве недостатка в выводах к предыдущей главе. Фактически, диаграммы Тьюринга представляют одни МТ через другие, более простые МТ иным, визуально-топологическим способом. В общем случае, диаграммы инкапсулируют основные действия, такие, как перемещение головки по слову и копирование слов в готовые элементарные машины, которые выполняют эти действия без ручной ассоциации с конкретными знаками, над которыми это действие выполняется. Кроме того, имеется вводить в процессе разработки алгоритма собственные машины, выделяя в них часто повторяющиеся действия, тем самым разделяя тело алгоритма на отдельные структуры и дополнительно сокращая его объём.

Состояния в диаграммах обозначаются точками, символ каждой подмашины в диаграмме ограничен слева и справа точками, обозначающими имя текущего состояния и состояния, в которое переходит машина после выполнения действия подмашины, соответственно. Если после выполнения действия одной подмашины, необходимо выполнить действие второй подмашины, то правая точка первой машины соединяется с левой точки второй машины, а над стрелкой указываются знаки, над которыми должна находиться головка машины, чтобы выполнить действие подмашины, к которой ведёт стрелка. Если выполнение действия подразумевается для всех знаков рабочего алфавита, то над стрелкой ничего не указывается.

Для повторения какого-либо участка диаграммы до тех пор, пока в рабочей ячейке находится одна из букв некоторого фиксированного набора, необходимо замкнуть этот участок диаграммы стрелкой с соответствующей надписью. Прекращение повторения осуществляется по букве, не входящей в этот набор.

Таким образом, диаграмма Тьюринга состоит из символов (имён) машин Тьюринга, точек и стрелок, над которыми написаны знаки рабочего алфавита.

При составлении алгоритмов с помощью диаграмм Тьюринга, диаграмму можно составить, как изобразив на бумаге, так и с помощью специального программного обеспечения, называемого диаграммером,

который обеспечивает интерактивное составление диаграммы и исполнение описываемого ею алгоритма. В рамках выполнения нижеследующего задания будет использоваться диаграммер *Virtual Turing Machine* (рисунок 4).

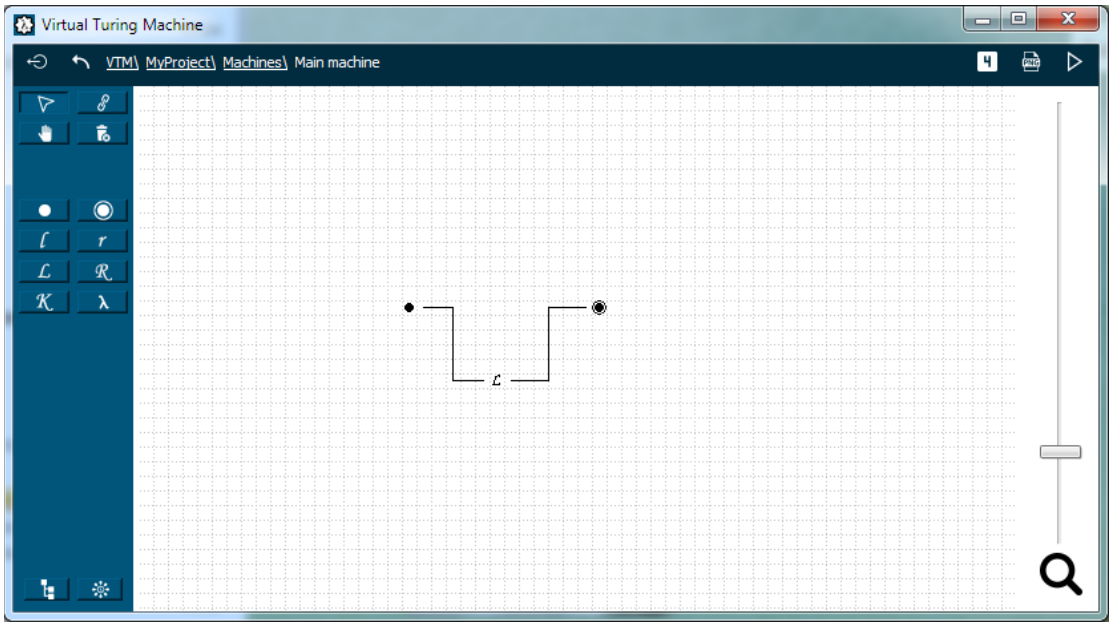

Рисунок 4 – Интерфейс диаграммера *Virtual Turing Machine*

2.2 Постановка задачи

В рамках поставленной задачи, выданной под вариантом №6, необходимо сконструировать диаграмму Тьюринга, реализующую алгоритм перевода числа из четверичной в шестнадцатеричную систему счисления с линейной сложностью.

Перевод числа из четверичной в шестнадцатеричную систему счисления упрощается тем, что каждым двум цифрам числа в четверичной записи соответствует одна цифра в шестнадцатеричной записи, так как основания обеих систем являются степенями числа 2.

Таким образом, суть алгоритма сводится к замене каждой пары цифр исходного числа на одну новую цифру, согласно набору правил (таблица 3).

Таблица 3 – Правила сопоставления четверичных и шестнадцатеричных цифр

Четверичная	00	01	02	03	10	11	12	13	20	21	22	23	30	31	32	33
Шестнадцатеричная	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

2.3 Идея решения задачи

В первую очередь реализуемый диаграммой алгоритм должен скопировать введённое число. Несмотря на то, что имеется встроенная машина копирования *K*, целесообразнее реализовать отдельную машину, которая скопирует число без значащих нулей.

После копирования головка будет проходить копию числа справа налево, зачищая каждые 2 цифры четверичной записи и меняя их на соответствующую цифру шестнадцатеричной записи.

В конечном счёте, от копии числа останутся разделённые пробелами («обособленные») шестнадцатеричные цифры, которые нужно записать подряд, после чего ответ будет получен.

2.4 Описание алгоритма

Диаграмма основной машины представлена на рисунке 5.

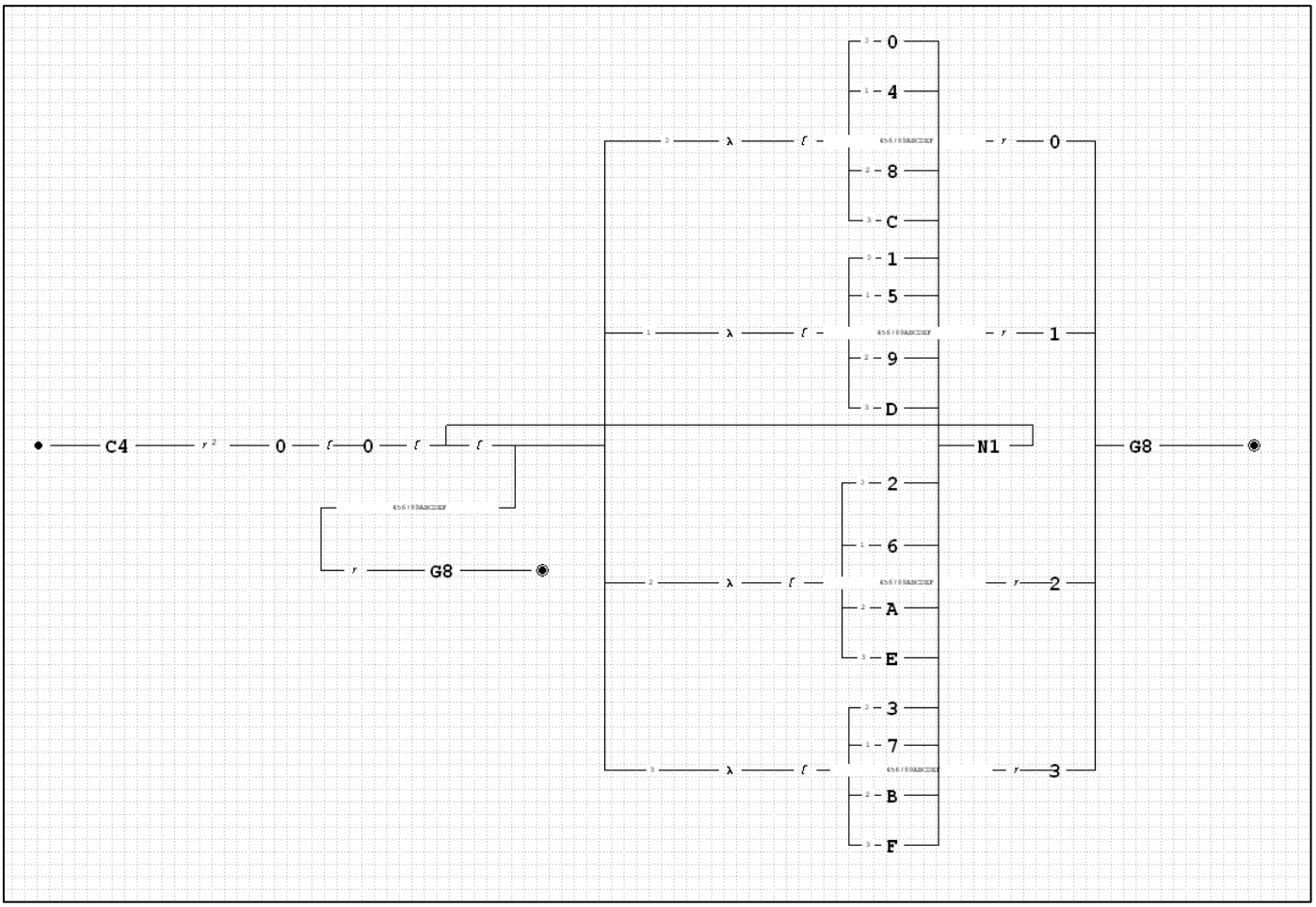

Рисунок 5 – Диаграмма основной машины

Машина копирования *COPY* (C4 на основной диаграмме) (рисунок 6) пройдёт всё число влево до пробела, затем продвинется вправо до первой значащей цифры, пропустив ведущие нули, затрёт первую значащую цифру, продвинется вправо до конца числа, запишет цифру после первого пробела, вернётся на затёртое место, вернёт скопированную цифру, сдвинется вправо, затрёт очередную цифру и продолжит таким образом копирование, пока не дойдёт до конца входного числа. После завершения копирования курсор вернётся к правому краю копии. Если всё число состоит из нулей, из них будет скопирован только один.

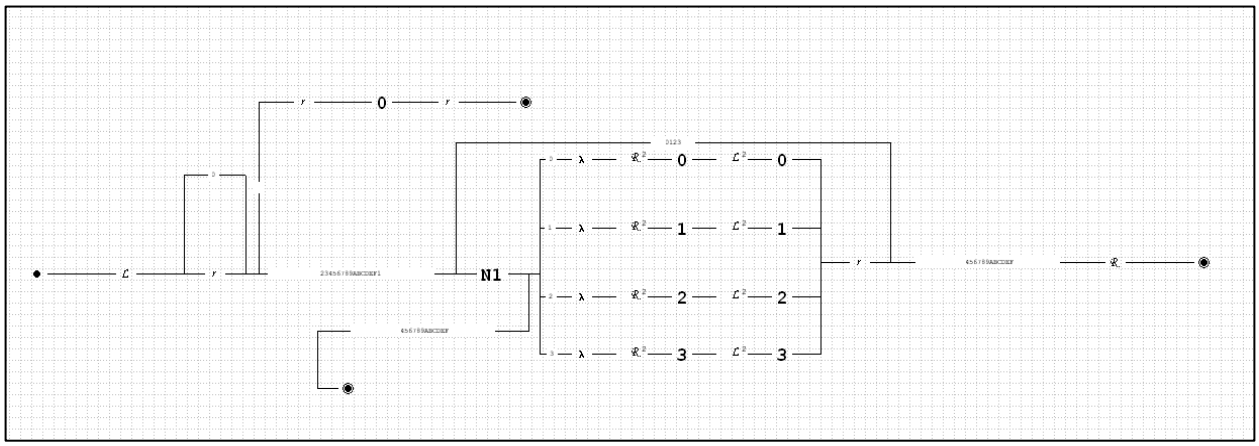

Рисунок 6 – Машина *COPY*

Далее в конец копии через пробел будут установлены два нуля, которые потребуются для остановки алгоритма в дальнейшем. Затем каретка сместится влево до последней цифры копии числа, затрёт её, сместиться ещё на шаг влево и затрёт предпоследнюю цифру копии. Затем, с учётом двух затёртых цифр, на место предпоследней цифры будет установлена соответствующая им шестнадцатеричная цифра. После этого ещё сдвиг влево и аналогичное считывание очередных двух цифр. Все эти действия производится непосредственно в рамках основной машины (ветвление на рис. 5). Преобразование четверичных чисел в шестнадцатеричные производится по приведённым правилам.

Если в числе нечётное количество цифр, первая слева числа цифра в шестнадцатеричной системе равна самой себе и не изменяется.

После завершения описанной выше обработки запускается машина *GROUPING* (G8) (рисунок 7), которая сдвигает каретку с первой слева цифры (двух подряд цифр, если количество цифр в исходном числе было нечётным) вправо до первой "обособленной" цифры, затирает её.

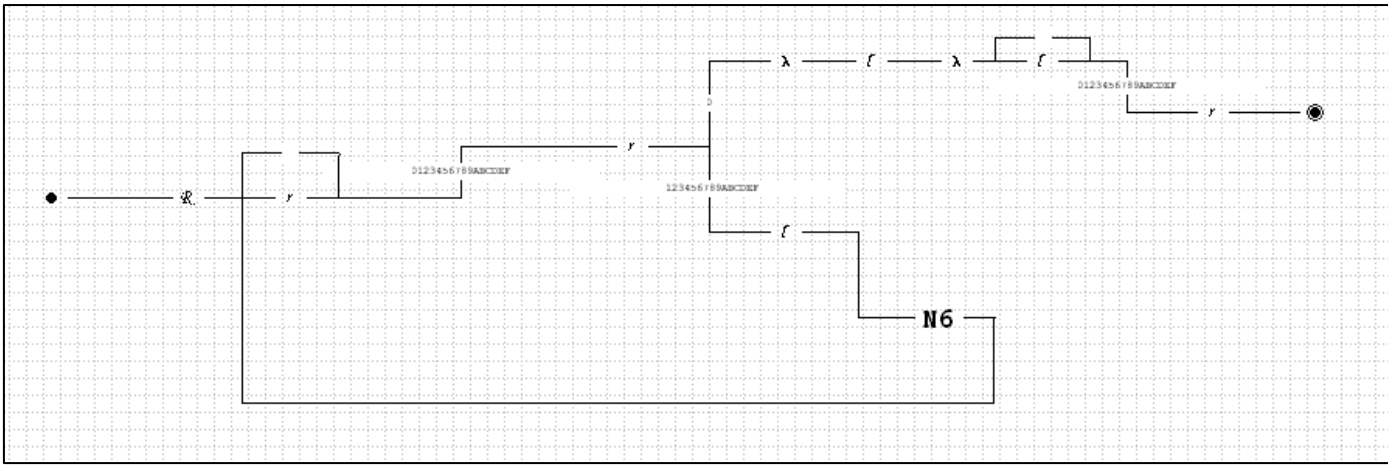

Рисунок 7 – Машина *GROUPING*

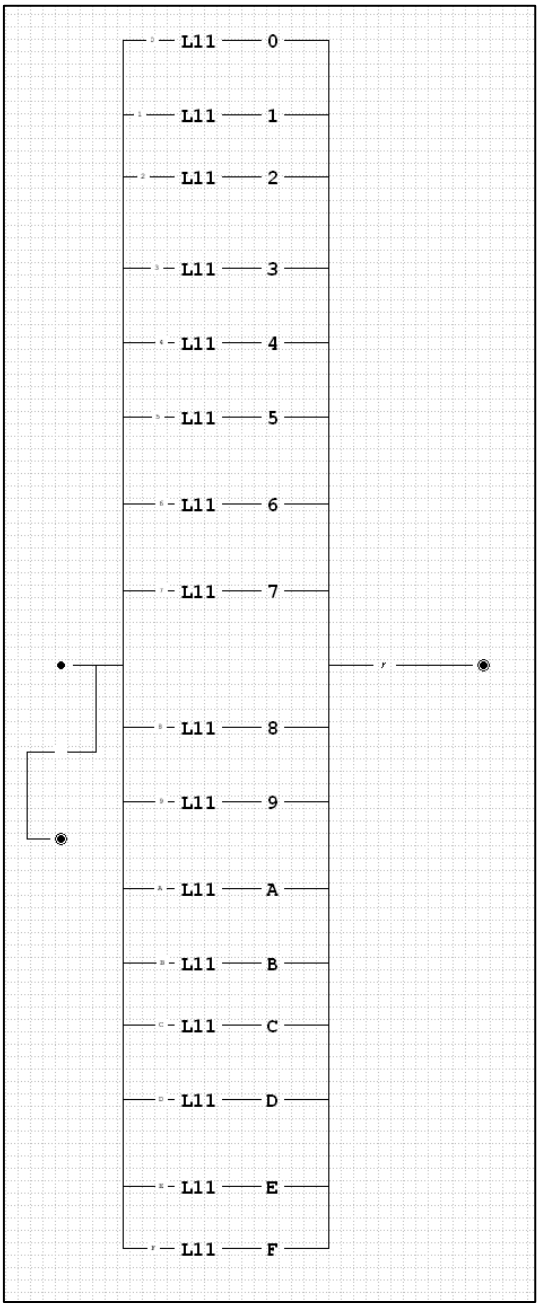

Рисунок 8 – Машина *NORMAL*

После этого головка смещается ещё на ячейку вправо и проверяет, если здесь стояли те два нуля, которые были записаны вначале в качестве границы рабочей зоны, то они затираются, головка возвращается к правому краю преобразованной копии и выполнение алгоритма завершается. Иначе, головка возвращается влево и записывает затёртую цифру вплотную к первой (первым) при помощи вспомогательной машины *NORMAL* (N6) (рисунок 8). В рамках машины *NORMAL* также используется вспомогательная машина *LAMBDA-LEFT* (L11) (рисунок 9), предназначенная для инкапсулирования повторяющегося действия – перемещения по пробелам от места затирания цифра до места её записи. Таким образом, все обособленные цифры записываются вплотную друг к другу.

Рисунок 9 – Машина *LAMBDA-LEFT*

Когда каретка в качестве очередных "обособленных" чисел находит два ранее установленных 0 подряд, они затираются, и каретка возвращается к крайнему правому краю получившегося шестнадцатеричного числа, на чём выполнение алгоритма завершается.

2.5 Тестирование и оценка сложности

В рамках тестирования алгоритма на вход диаграмме были представлены несколько строк краевого вида (одни нули, одна цифра, число с ведущими нулями), а также числа с чётным и нечётным числом цифр. Полный список рассмотренных тестовых случаев представлен в таблице 4.

Таблица 4 – Список рассмотренных тестовых случаев

Входные данные	Выходные данные	Описание тестируемого случая
000	0	Число из одних нулей
1	1	Однозначное число

Продолжение таблицы 4

3213	E7	Число с чётным количеством цифр
31211	365	Число с нечётным количеством цифр
00033231	3ED	Число с ведущими нулями
32333313213	3BFDE7	Случайное число

Так как использованный интерпретатор не приводит данных о количестве выполненных операций, оценка сложности производилось с засечением времени выполнения программы для каждого тестового случая. В таблице 5 приведена зависимость времени выполнения алгоритма от объёма входного сообщения.

Таблица 5 – Зависимость времени выполнения алгоритма от размера сообщения

Длина входного сообщения	Примерное время выполнения, с.
1	3.52
2	5.21
4	9.58
8	21.29

На основании вышеприведённых данных можно утверждать, что при увеличении объёма входного сообщения в 2 раза время выполнения также возрастает приблизительно в 2 раза, значит, в соответствии с заданием, сложность алгоритма линейна и равна O(n).

2.6 Выводы по главе

Разработан и реализован в среде разработки Диаграмм Тьюринга алгоритм перевода чисел из четверичной системы счисления в шестнадцатеричную с линейной сложностью. На примере работы с более высокоуровневой реализацией абстрактного исполнителя Машины Тьюринга дополнен опыт разработки алгоритмов, полученный при решении предыдущей задачи.

По сравнению со средой Машины Тьюринга, при работе с диаграммой удобнее задавать в алгоритме рутинные операции, не задумываясь о наименовании состояний и т. п., а также создавать дополнительные машины,

выделяя в них объёмные, часто повторяющиеся операции. С другой стороны, громоздкий интерфейс диаграммера по сравнению с командной строкой интерпретатора Машины Тьюринга не слишком удобен для быстрой отладки алгоритма.

С другой стороны, диаграммы в некоторой степени удобны для написания их без использования интерпретатора (на бумаге) и последующей отладки путём ручного выполнения (проговаривания) алгоритма.

ГЛАВА 3. НОРМАЛЬНЫЕ АЛГОРИТМЫ МАРКОВА

3.1 Вводная часть

Рассматриваемая алгоритмическая система была предложена академиком А. А. Марковым в 1947-1954 гг. Аналогично машине Тьюринга, в этой модели происходит преобразование текстовых сообщений, основанное на замене подслов исходного сообщения на некоторые другие слова.

Нормальные алгоритмы Маркова по существу являются детерминистическими текстовыми заменами, которые для каждого входного слова однозначно задают вычисления и тем самым в случае их завершения порождают определённый результат. Это может быть обеспечено, например, установлением приоритета применения правил. Такие приоритеты могут быть заданы линейным порядком их записи. В алгоритмической системе Маркова нет понятия ленты, и подразумевается непосредственный доступ к различным частям преобразуемого слова.

В качестве основных принципов выполнения алгоритма в этой системе можно выделить следующие:

1. Если применимо несколько правил, то берётся правило, которое встречается в описании алгоритма первым;
2. Если правило применимо в нескольких местах обрабатываемого слова, то выбирается самое левое из этих мест.

Таким образом, нормальный алгоритм Маркова (НАМ) представляет собой упорядоченный набор правил-продукций – пар слов (цепочек знаков, в том числе пустых цепочек длины 0), соединённых между собой символами →

или ↦ . Каждая продукция представляет собой формулу замены части входного слова, совпадающей с левой частью формулы, на её правую часть.

Процесс выполнения НАМ заканчивается в одном из двух случаев: либо все формулы оказались неприменимыми, то есть в обрабатываемом слове нет вхождений левой части ни одной формулы подстановки; либо только что применилась так называемая терминальная (завершающая) продукция, в которой правую и левую часть разделяет символ ↦. Терминальных продукций в одном НАМ может быть несколько.

В любом из этих случаев НАМ применим к данному входному слову. Если в процессе выполнения НАМ бесконечно долго применяются нетерминальные правила, то алгоритм неприменим к данному входному слову. Существуют следующие достаточные признаки применимости НАМ ко всем входным словам:

1. Левые части всех продукций непустые, а в правых частях нет букв, входящих в левые части;
2. В каждом правиле правая часть короче левой части.

При составлении НАМ для отладки удобно пользоваться различными интерпретаторами. В данной работе использовался интерпретатор *markov* за авторством К. Полякова (рисунок 10).

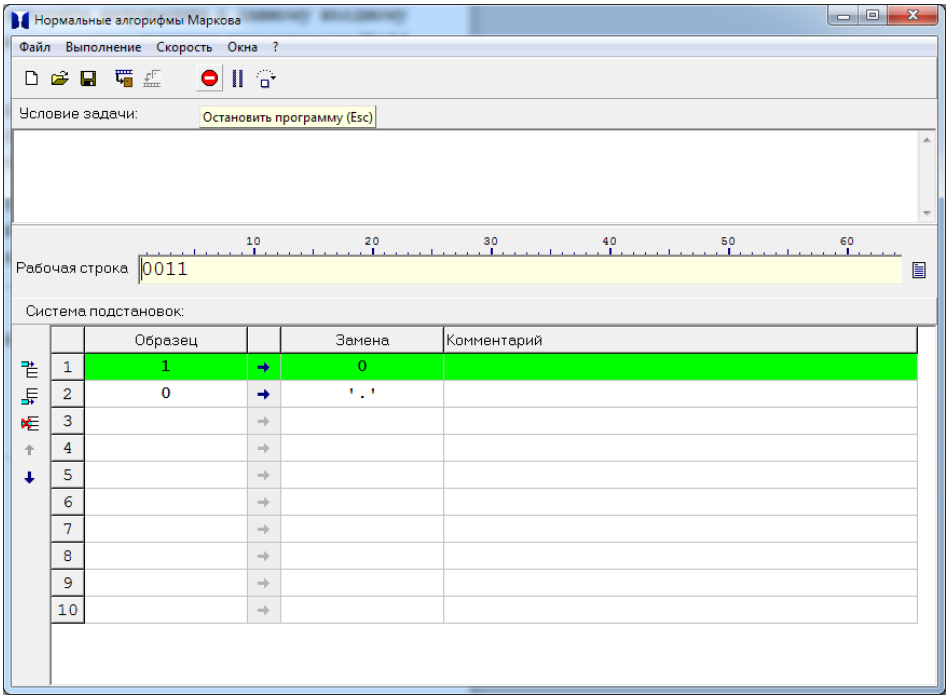

Рисунок 10 – Интерфейс интерпретатора *markov*

В среде данного интерпретатора вместо символа ↦ для обозначения терминальной продукции используется точка, поставленная сразу после правой части формулы. Если точка используется как непосредственно слово для подстановки, то она выделяется одинарными кавычками.

Также в качестве заметной особенности модели НАМ можно отметить, что здесь, в отличие от моделей Тьюринга, вывод не обязан быть нормированным, то есть входные данные не должны оставаться в неизменном виде после завершения программы. Связано это со сложностью организации нормированного вывода, а также с отсутствием перемещающейся линейно рабочей головки.

3.2 Постановка задачи

В рамках поставленной задачи, выданной под вариантом №39 необходимо составить алгоритм перевода десятичных цифровых сообщений в азбуку Морзе.

Для составления такого алгоритма необходимо правила перевода цифровых знаков в элементы азбуки Морзе (таблица 6).

Таблица 6 – Правила перевода цифровых знаков в элементы азбуки Морзе

Цифра	1	2	3	4	5	6	7	8	9	0
Код	.----	..---	...--	....-	.....	-....	--...	---..	----.	-----

Также стоит отметить, что знаки азбуки Морзе разделяются одним, а слова двумя пробелами.

3.3 Идея решения задачи

В рамках решения необходимо составить продукции, заменяющие каждую отдельно взятую цифру на её код в азбуке Морзе, которому предшествует один пробел. Тем самым один пробел между словами в десятичной записи будет преобразован в два пробела в записи Морзе, а между знаками появится один пробел. Алгоритм должен завершиться удалением крайнего левого пробела после преобразования всех цифр.

3.4 Описание алгоритма

В теле алгоритма будут записаны продукции, ставящие каждой десятичной цифре её код в азбуке Морзе, перед которым стоит один пробел. Данный набор из 10 продукций будет выполняться до тех пор, пока остаются непреобразованные цифры. После преобразования всех цифр должна выполниться продукция, записанная после всех правил преобразования цифр и являющаяся терминальной. Данная продукция удалит пробел, поставленный в самое начало обрабатываемой строки в результате преобразования одной из цифр, после чего завершит выполнение алгоритма.

Исходный код алгоритма представлен на рисунке 11.

	Образец		Замена
1	1	→	' .----'
2	2	→	' ..---'
3	3	→	' ...--'
4	4	→	' ....-'
5	5	→	' .....'
6	6	→	' -....'
7	7	→	' --...'
8	8	→	' ---..'
9	9	→	' ----.'
10	0	→	' -----'
11	' '	→	''.

Рисунок 11 – Исходный код алгоритма

3.5 Тестирование

В рамках тестирования алгоритма на вход программе были представлены несколько строк различного вида (один знак, несколько знаков, одно слово, несколько слов). Полный список рассмотренных тестовых случаев представлен в таблице 7.

Таблица 7 – Список рассмотренных тестовых случаев

Входные данные	Выходные данные	Описание тестируемого случая
0	-----	Сообщение из 1 цифры
1	.----	Сообщение из 1 цифры
123	.---- ..--- …--	Сообщение из 1 слова
233 457	..--- …-- …-- ….- ….. --…	Сообщение из 2 слов
345 7677 7	…-- ….- ….. --… -…. --… --… -- …	Сообщение из 3 слов

Для проведения оценки сложности, по аналогии с предыдущим заданием, использовалось засечение времени выполнения алгоритма для входных сообщений различной длины. Результаты наблюдений приведены в таблице 8.

Таблица 8 – Зависимость времени выполнения алгоритма от размера сообщения

Длина входного сообщения	Примерное время выполнения, с.
1	1.49
2	3.04
4	4.28
8	8.76

На основании вышеприведённых данных можно утверждать, что при увеличении объёма входного сообщения в 2 раза время выполнения тоже возрастает примерно в 2 раза, следовательно, сложность алгоритма линейна и равна O(n).

3.6 Выводы по главе

Разработан и реализован в среде разработки алгоритмов алгоритмической модели Маркова алгоритм перевода десятичных цифровых сообщений в азбуку Морзе. Дополнен опыт решения предыдущих задач в области создания алгоритмов в различных алгоритмических моделях.

В отличие от предыдущих моделей, нормальные алгоритмы Маркова сложнее по свой структуре и в процессе реализации, так как отсутствует строгий контроль над текущим состоянием исполнителя (положением

некоторой головки), не зависящий от фактического содержания обрабатываемых данных. Кроме того, во время разработки необходимо отслеживать порядок выполнения правил в зависимости от их местоположения в исходном коде, каковая особенность отсутствовала в моделях Тьюринга. В целом, решение данной задачи не вызвало вышеописанных трудностей, в связи с простотой её условия.

ЗАКЛЮЧЕНИЕ

Итак, в рамках выполнения данной курсовой работы были проиллюстрированы определения алгоритма при помощи алгоритмических моделей Тьюринга и Маркова. С этой целью были выполнены предложенные задания с использованием среды машины Тьюринга, диаграмм Тьюринга и нормальных алгоритмов Маркова. Каждая из рассмотренных алгоритмических моделей имела свои особенности.

Модель машины Тьюринга отличалась особенной низкоуровневостью и строгостью выполнения алгоритма. Было необходимо обеспечить нормированный ввод, при котором входное сообщения должно оставаться в неизменном виде, а каждое действие, совершаемое алгоритмом (рабочей головкой) зависело исключительно от его местоположения на ленте и значения стоящего в этом месте знака. Из-за низкоуровневости модели код её алгоритма был громоздким и весьма объёмным, но при этом подробное описание каждого действия позволяло лучше контролировать совершаемые машиной действия и повышало точность её работы.

С другой стороны, диаграммы Тьюринга полностью эквивалентны машине, и лишь графически представляет её алгоритм, инкапсулируя при этом рутинные и элементарные операции в отдельные подмашины. Таким образом, ускоряется и упрощается разработка и отладка алгоритма. При этом, громоздкий интерфейс интерпретатора, нагромождения соединяющих подмашины стрелок значительно снижают заметность этого преимущества.

Наконец, нормальные алгоритмы Маркова значительно отличались от предыдущих моделей, так как их выполнение зависело не от пространственного положения рабочей головки (состояния машины), а от фактического содержания входного сообщения и взаимного расположения

его знаков. Данная особенность несколько усложняла разработку алгоритма, так как его поведение могло значительно меняться в зависимости от содержания сообщения. Так же усложнялись пространственно зависимые операции, такие как копирование слов. В том числе поэтому, при разработке алгоритма в системе Маркова не нужно было обеспечивать нормированный вывод.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гайсарян С. С., Зайцев В. Е., Курс информатики // Учебное пособие. – Изд-во Вузовская книга. – Москва, 2013. – 474 с.

2. Г.-Д. Эббинхауз, К. Якобе, Ф.-К. Ман, Г. Хермес, Машины Тьюринга и рекурсивные функции. – Мир. – Москва, 1972.

3. Любимский Э. З., Мартынюк В. В., Элементы теории алгоритмов и структур данных. – МГУ. – Москва, 1976.

4. Бауэр Ф., Гооз Т., Информатика. – Мир. – Москва, 1976, 1990.

5. Лекции лауреатов премии Тьюринга. / Пер. с англ. – Мир. – Москва, 1993. – 560 с.

6. Марков А. А., Нагорный Н. М., Теория алгорифмов. – Наука. – Москва, 1984.

7. Зайцев В. Е. и др., Информатика. Практикум. // Учебное пособие. – Москва, 1993.

8. Шеннон К., Универсальная машина Тьюринга с двумя внутренними состояниями. – Работы по теории информации и кибернетие. – ИЛ. – Москва, 1963. – с. 740-750.

ПРИЛОЖЕНИЕ А

Исходный код алгоритма решения задачи для машины Тьюринга

Далее: ВЧ – второе число, ПЧ – первое число, КВЧ – копия второго числа

Копирование второго числа

1. Чтение очередной цифры ВЧ и её затирание

```
read,0, ,mv0
read,1, ,mv1
```

2. Если всё уже скопировано, переход к работе с маской

```
read, ,>,start_mask
```

3. Сдвиг вправо с затёртой позиции

```
mv0, ,>,cp0
mv1, ,>,cp1
```

4. Движение по ВЧ с учётом копируемой цифры

```
cp0,0,>,cp0
cp0,1,>,cp0
cp0, ,>,wr0

cp1,0,>,cp1
cp1,1,>,cp1
cp1, ,>,wr1
```

5. Движение по уже скопированным цифрам и запись копируемой цифры в первой свободной ячейке

```
wr0,0,>,wr0
wr0,1,>,wr0
wr0, ,0,ret0

wr1,0,>,wr1
wr1,1,>,wr1
wr1, ,1,ret1
```

6. Обратное движение по копии

```
ret0,0,<,ret0
ret0,1,<,ret0
ret0, ,<,-ret0

ret1,0,<,ret1
ret1,1,<,ret1
ret1, ,<,-ret1
```

7. Обратное движение по ВЧ и возврат затёртой цифры на своё место

```
-ret0,0,<,-ret0
-ret0,1,<,-ret0
-ret0, ,0,mv

-ret1,0,<,-ret1
-ret1,1,<,-ret1
-ret1, ,1,mv
```

7 Сдвиг вправо на новую цифру и повтор копирования

```
mv,0,>,read
mv,1,>,read
```

Процедура выделения разрядов

Первичные процедуры (повтор 1 раз)

1. Чтение первой цифры КВЧ и её затирание

```
start_mask,1, ,sm1
start_mask,0, ,sm0
```

2. Перемещение по двум пробелам между ВЧ и КВЧ

```
sm1, ,<,-sm1
-sm1, ,<,mv_vch_1
sm0, ,<,-sm0
-sm0, ,<,mv_vch_0
```

3. Перемещение по ВЧ

```
mv_vch_0,0,<,mv_vch_0
mv_vch_0,1,<,mv_vch_0
mv_vch_0, ,<,mv_pch_0

mv_vch_1,0,<,mv_vch_1
```

```
mv_vch_1,1,<,mv_vch_1
mv_vch_1, ,<,mv_pch_1
```

4. Перемещение по ПЧ до первого пробела, смещение вправо и переход к основному циклу

```
mv_pch_0,0,<,mv_pch_0
mv_pch_0,1,<,mv_pch_0
mv_pch_0, ,>,read_mask_0

mv_pch_1,0,<,mv_pch_1
mv_pch_1,1,<,mv_pch_1
mv_pch_1, ,>,read_mask_1
```

Процедуры основного цикла

1. Чтение очередной цифры ПЧ и:

```
// если 0, соответствующая цифра КВЧ в любом случае будет 0
read_mask_0,0, ,mv_null_val

// если 1, соответствующая цифра КВЧ сохраниться, значит, её нужно оставить в памяти
read_mask_0,1, ,mv_0_val

// если пробел, проход ПЧ окончен, переход к процедуре выхода из программы
read_mask_0, ,>,end

// аналогично предыдущему
read_mask_1,0, ,mv_null_val
read_mask_1,1, ,mv_1_val
read_mask_1, ,>,endё затирание
```

2. Сдвиг с затёртой цифры вправо

```
mv_null_val, ,>,null_val
mv_0_val, ,>,0_val
mv_1_val, ,>,1_va
```

3. Движение по ПЧ вправо

```
null_val,0,>,null_val
null_val,1,>,null_val
```

```
null_val, ,>,-null_val

0_val,0,>,0_val
0_val,1,>,0_val
0_val, ,>,-0_val

1_val,0,>,1_val
1_val,1,>,1_val
1_val, ,>,-1_val
```

4. Движение по ВЧ вправо

```
-null_val,0,>,-null_val
-null_val,1,>,-null_val
-null_val, ,>,wr_null_val

-0_val,0,>,-0_val
-0_val,1,>,-0_val
-0_val, ,>,wr_0_val

-1_val,0,>,-1_val
-1_val,1,>,-1_val
-1_val, ,>,wr_1_val
```

5. Движение по КВЧ вправо и запись в первый пробел:

```
// в любом случае 0, если прочитанная цифра ПЧ была 0
wr_null_val,0,>,wr_null_val
wr_null_val,1,>,wr_null_val
wr_null_val, ,0,ret_0

// 0, если проч. Цифра ПЧ была 1 и здесь стоял 0
wr_0_val,0,>,wr_0_val
wr_0_val,1,>,wr_0_val
wr_0_val, ,0,ret_1

// 1, если проч. Цифра ПЧ была 1 и здесь стояла 1
wr_1_val,0,>,wr_1_val
wr_1_val,1,>,wr_1_val
wr_1_val, ,1,ret_1
```

6. Сдвиг вправо на следующую цифру КВЧ

```
ret_0,0,>,-ret_0
ret_1,0,>,-ret_1
```

```
ret_1,1,>,-ret_1
```

7. Чтение очередной цифры КВЧ (аналогично первичной процедуре start_mask) и её затирание

```
// если пробел, значит возвращаем последнюю цифру ПЧ и закругляемся
-ret_0,0, ,mv_ret_0
-ret_0,1, ,mv_ret_1
-ret_0, ,=,mv_ret_0
-ret_1,0, ,mv_ret_10
-ret_1,1, ,mv_ret_11
-ret_1, ,=,mv_ret_10
```

8. Сдвиг влево с затёртой ячейки

```
// с запоминанием конфигурации (ранее затёртая цифра ПЧ, только что прочитанная цифра КВЧ)
mv_ret_0, ,<,go_ret_0
mv_ret_1, ,<,go_ret_1
mv_ret_10, ,<,go_ret_10
mv_ret_11, ,<,go_ret_11
```

9. Движение влево по КВЧ

```
go_ret_0,0,<,go_ret_0
go_ret_0,1,<,go_ret_0
go_ret_0, ,<,-go_ret_0

go_ret_1,0,<,go_ret_1
go_ret_1,1,<,go_ret_1
go_ret_1, ,<,-go_ret_1

go_ret_10,0,<,go_ret_10
go_ret_10,1,<,go_ret_10
go_ret_10, ,<,-go_ret_10

go_ret_11,0,<,go_ret_11
go_ret_11,1,<,go_ret_11
go_ret_11, ,<,-go_ret_11
```

10.Движение влево по ВЧ

```
-go_ret_0,0,<,-go_ret_0
-go_ret_0,1,<,-go_ret_0
```

```
-go_ret_0, ,<,fin_ret_0

-go_ret_1,0,<,-go_ret_1
-go_ret_1,1,<,-go_ret_1
-go_ret_1, ,<,fin_ret_1

-go_ret_10,0,<,-go_ret_10
-go_ret_10,1,<,-go_ret_10
-go_ret_10, ,<,fin_ret_10

-go_ret_11,0,<,-go_ret_11
-go_ret_11,1,<,-go_ret_11
-go_ret_11, ,<,fin_ret_11
```

11.Движение влево по ПЧ, запись возвращаемой цифры ПЧ

```
fin_ret_0,0,<,fin_ret_0
fin_ret_0,1,<,fin_ret_0
fin_ret_0, ,0,move_0

fin_ret_1,0,<,fin_ret_1
fin_ret_1,1,<,fin_ret_1
fin_ret_1, ,0,move_1

fin_ret_10,0,<,fin_ret_10
fin_ret_10,1,<,fin_ret_10
fin_ret_10, ,1,move_0

fin_ret_11,0,<,fin_ret_11
fin_ret_11,1,<,fin_ret_11
fin_ret_11, ,1,move_1
```

12.Сдвиг с возвращённой цифры ПЧ и чтение следующей цифры ПЧ с учётом ранее прочитанной цифры КВЧ

```
move_0,0,>,read_mask_0
move_0,1,>,read_mask_0

move_1,0,>,read_mask_1
move_1,1,>,read_mask_1
```

13.Финальное движение вправо по ВЧ

```
end,0,>,end
```

```
end,1,>,end
end, ,>,-end
```

14.Финальное движение вправо по КВЧ и выход из программы

```
-end,0,>,-end
-end,1,>,-end
-end, ,#,-end
```