МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНТИТУТ
(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
Степанов Никита Евгеньевич

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. МАШИНА ТЬЮРИНГА

1.1 Вводная часть

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 четёрка 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 $F_l : Q \times \overline{A} \to \overline{A}(\overline{A} = A \cup \{\lambda\})$ функцию переходов $F_t : Q \times \overline{A} \to Q$, и функцию движения головки $F_v : Q \times \overline{A} \to \{l, r, s\}$ (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.

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

Пример описания состояний МТ

Фактически, программа МТ задаётся упорядоченными наборами (в случае данной задачи - четвёрками) символов (командами) вида (a, d, 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 (рис.1), принимающий на ввод текст программы и входное сообщение и выдающий на выходе преобразованное сообщение.

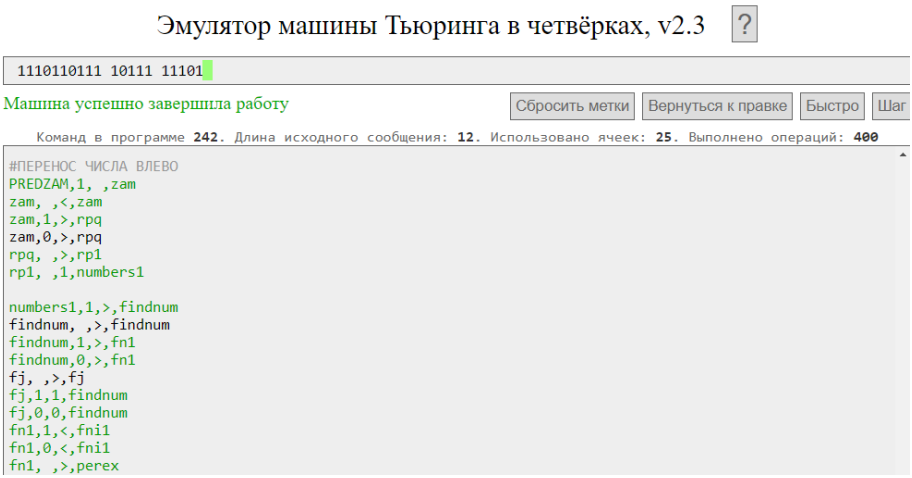

Рисунок 1 - Интерфейс интерпретатора *jstu4*

Интерфейс интерпретатора допускает использование комментариев.

1.2 Постановка задачи

В рамках задачи выданной под вариантом №6 необходимо составить алгоритм, генерирующий два числа из чётных и нечётных разрядов двоичного числа. Задание подразумевает, что для входного числа, мы делаем копирование чётных разрядов через пробел со входным числом, затем возвращаемся в начало числа и берем нечётные разряды и копируем их через 2 пробела от входного числа. Рабочим алфавитом машины будет являться множество $\{0, 1\}$.

1.3 Идея решения задачи

В рамках решения предстоит выполнить две, фактически независимые задачи: копирование цифр для первого числа на область ленты расположенную через пробел после основного числа, а так же копирование второго числа, расположенного уже после числа с нечетными разрядами. Машина Тьюринга будет последовательно считывать каждую цифру числа на ленте и записывать ее в соответствующую переменную в зависимости от того, является ли цифра четной или нечетной.

1.4 Описание алгоритма

Данный алгоритм решения задачи будет проходить в 5 этапов. В начальном состоянии головка находится непосредственно справа от второго числа. В процессе копирования головка МТ будет пробегать по разрядам первого числа.

I этап. Данный этап подразумевает в себе копирование четных чисел в конец входного числа. Головка пробегает справа налево по изначальному числу и если она находит пробел, то переходит в следующее состояние. В данном состоянии, головка пропускает первую цифру и сразу переходит ко второй и заменяет её на пробел, тем самым запоминая её. Далее проходим по входному числу до пробела и как только видим пробел, передвигаем головку вправо и печатаем цифру(если цифра первая, иначе проходим по всему числу), которую запомнили(рис. 2). После того как распечатали цифру, возвращаемся обратно ко второму пробелу, считая справа, и печатаем ту же цифру. Далее пропускаем одну цифру и повторяем данный алгоритм до тех пор, пока не закончатся значения, которые стоят на четных разрядах.

II этап. Данный этап подразумевает в себе копирование четных чисел в конец числа, построенного из чётных разрядов входного числа. Переносим головку МТ в начало входного числа, начинаем уже с первого разря-

да, так как нам нужны цифры на нечётных разрядах. Затираем цифру и запоминаем её, проходим по исходному числу до нахождения пробела. Как только находим пробел, проходим через него и проходимся уже по второму числу до нахождения пробела. Если нашли пробел, передвигаем головку на 1 разряд вправо и ставим цифру (если цифра первая, иначе проходим по всему числу), которую запомнили. Вовзаращаемся обратно, проходя сначала число состоящее из четных разрядов, потом проходим через исходное число до поиска пробела, в котором затирали цифру. Печатаем цифру которую запоминали, затем передвигаем головку на 2 разряда вправо и повторяем данный алгоритм до тех пор, пока не закончатся значения, которые стоят на нечетных разрядах.

III этап. Данный этап заключается в удалении ведущих нулей у числа, сформировавшегося из четных разрядов входного числа. После полного формирования двух чисел, переходим в конец исходного числа и делаем проверку на ведущие нули. Как только нашлась единица, переходим обратно - в начало "четного"числа. Затираем все лишние нули и переносим оставшееся число через 1 пробел от исходного числа.

IV этап. Данный этап заключается в удалении ведущих нулей у числа, сформировавшегося из нечетных разрядов входного числа. После удаления ведущих нулей "четного"числа, начинаем удаление ведущих нулей у "нечетного"числа. Переносим каретку в начало нечетного числа и делаем проверку, как только нашлась единица - переходим в начало нечетного числа и удаляем ведущие нули. Переносим оставшееся число через 1 пробел от исходного числа.

V этап. Заключительный этап в котором мы завершаем программу.

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 (00), (11), (000111) и другие. Полный список рассмотренных тестовых случаев представлен в таблице ниже.

Ввод	Вывод
00	0 0
11	1 1
01	1 0
1111	11 11
000111	11 1
1110110111	10111 11101

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 ниже.

Длина входного сообщения	Количество выполненных операций
2	42
4	91
8	292
16	820
32	2644

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 генерирующий два числа из чётных и нечётных разрядов двоичного числа, 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.

ГЛАВА 2. ДИАГРАММА 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 за-

дания будет использоваться диаграммер $VirtualTuringMachine$ (рис. 2).

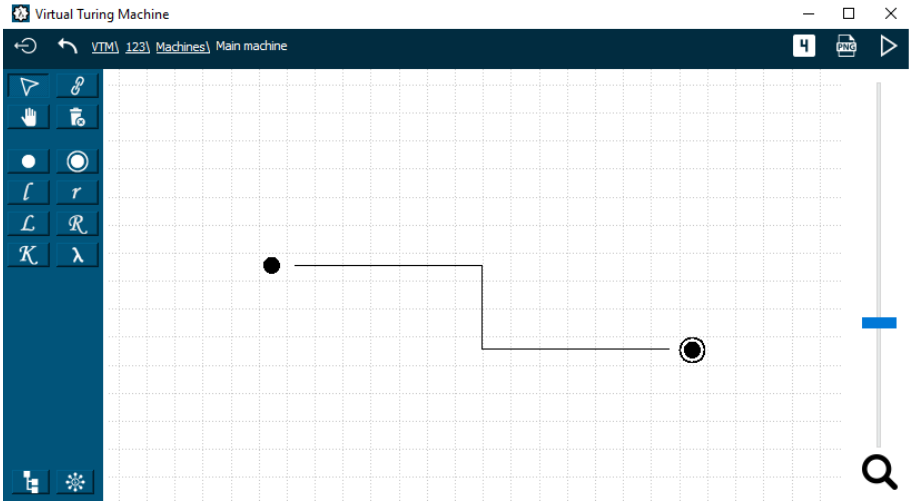

Рисунок 2 - Интерфейс диаграммера $VirtualTuringMachine$

2.2 Постановка задачи

В рамках поставленной задачи, выданной под вариантом №20 необходимо сконструировать диаграмму Тьюринга, реализующую алгоритм вычисления двоичного логического сдвига первого числа вправо на число разрядов, равное второму.

Для решения задачи необходимо определить состояния, символы, правила и переходы, которые будут использоваться в диаграмме Тьюринга. Также необходимо учесть все возможные варианты ввода данных и предусмотреть соответствующие выходы в зависимости от входных параметров.

2.3 Идея решения задачи

В первую очередь реализуемый диаграммой алгоритм должен скопировать введённые числа. Несмотря на то, что имеется встроенная машина копирования K, целесообразнее реализовать отдельные машины, которые скопируют числа без значащих нулей.
В последующем реализовать преобразования над вторым исходным числом, а точнее его вычитание. Данное действие нужно для подсчета логического сдвига право.

После вычитания, согласно заданию, сделать сам сдвиг, который будет меняться в зависимости от последующей цифры в первом исходном числе.

2.4 Описание алгоритма

Диаграмма основной машины представлена на рисунке 5. Основная машина сначала переходит к другим машинам, чтобы выполнить копирования числа, затем делает преобразования над вторым скопированным числом. Данные преобразования заключаются в вычитании из единички из числа. Для того, чтобы реализовать это, нужно найти первую единицу слева и заменить её на 0, а так же заменить все нули, которые стояли после этой единицы на 1.

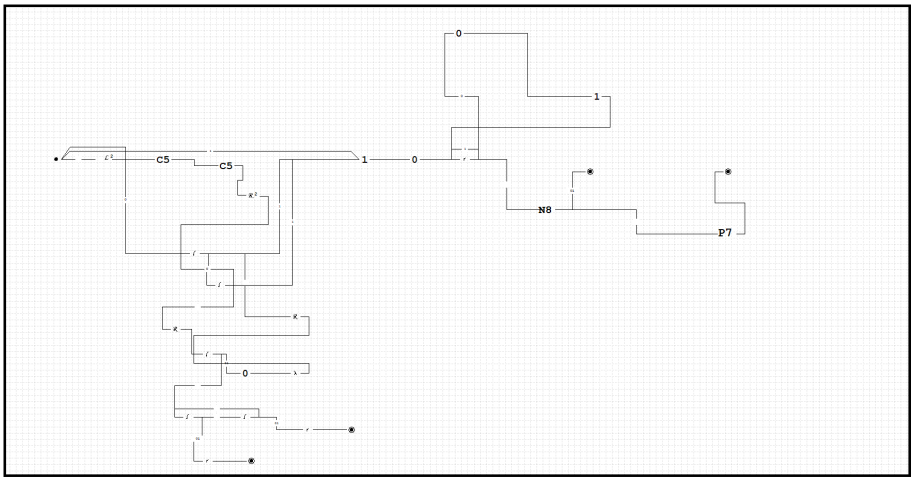

Рисунок 3 – Диаграмма основной машины

Машина копирования COPY (рис. 4) начинает свое движение с пробела перед первым(вторым) числом. При нахождении 1(0), каретка заменяет данное число на лямбду(пробел) и проходит через число(числа), затем через пробел ставит 1(0)(если цифра первая, иначе проходим по всему числу). После печати цифры, возвращаемся на пробел в первом(втором) числе и заменяем его на ту цифру, которую поставили.

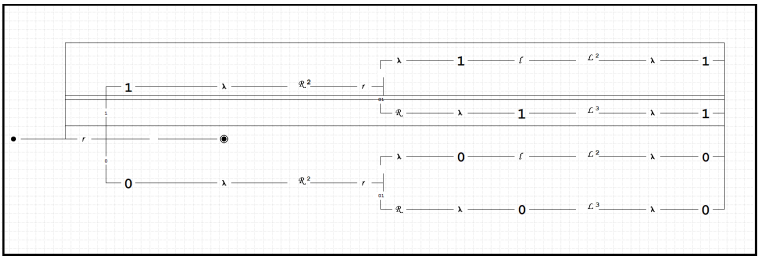

Рисунок 4 – Машина COPY

Машина converting (рис. 5) выполняет главную задачу - логический сдвиг числа. Начинается она с конца первого скопированного числа. Выполнение действий происходит опираясь на условие, если правая цифра равна единице и следующая слева равна нолику, то и правая единица становится ноликом, в данном случае произошли изменения. Если же правая цифра будет равна единице и следующая слева тоже будет равна единице, то в данном случае всё останется без изменений. Следовательно, для числа выполняются следующие действия: если $n_k = 1$, $n_{k-1} = 0 \to n_k = 0$ или же $n_k = 1$, $n_{k-1} = 1 \to n_k = 1$ где n - само число, а k - разряд этого числа. В машине converting это реализовано так: каретка передвигается справа налево и сверяет цифры, если число, на котором каретка останавилась, не равно предыдущему, то переходим к предыдущему и заменяем его на следующее. Данный алгоритм выполняем до тех пор, пока каретка не дойдет до начала первого скопированного числа. Как только каретка доходит до начала, заменяем первую цифру на 0 и проходим через два числа.

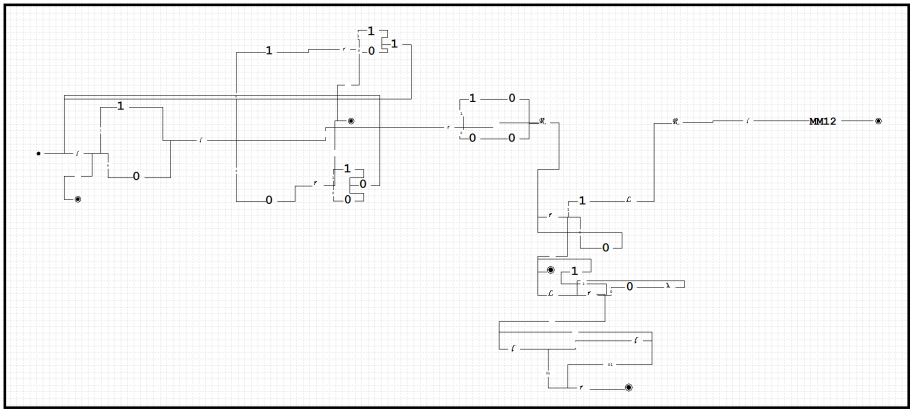

Рисунок 5 – Машина CONVERTING

После всех преобразований, каретка перемещается в конец второго скопированного числа, затирает нули и завершает программу.

2.5 Тестирование и оценка сложности

В рамках тестирования алгоритма на вход диаграмме были представлены несколько строк краевого вида (одни нули, одна цифра, одни единицы и т.Д). Полный список рассмотренных тестовых случаев представлен в таблице ниже.

Входные данные	Выходные данные
000 00	000
1 1	0
1101 11	0001
111 01	011
1000001 101	0000010
11100 0	111000

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 ниже приведена зависимость времени выполнения алгоритма от объёма входного сообщения.

Длина входного сообщения	Примерное время выполнения, с.
2	5.86
4	13.96
8	28.38
16	73.51

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 $\rightarrow$ или $\mapsto$. 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 $\mapsto$. 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 6).

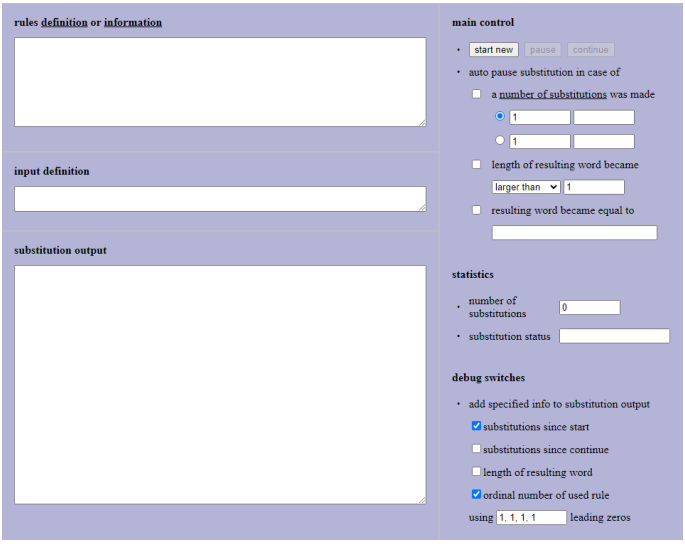

Рисунок 6 – Интерфейс интерпретатора markov

В среде данного интерпретатора вместо символа $\mapsto$ 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 №17 необходимо составить алгоритм вычисления поразрядной конъюкции исходных чисел. Для начала разберемся что такое поразрядная конъюкция.

Поразрядная конъюкция - это операция, выполняемая над двоичными числами, которая соединяет их биты побитно с использованием логической операции "И"(AND). При поразрядной конъюкции каждый бит результирующего числа будет равен 1 только в том случае, если оба соответствующих бита исходных чисел также равны 1. В противном случае, бит результирующего числа будет равен 0. Таким образом, поразрядная конъюкция позволяет комбинировать биты из различных чисел для получения нового числа, в котором каждый бит зависит от соответствующих битов исходных чисел.

3.3 Идея решения задачи

В данной задаче можно использовать метод поразрядной конъюнкции, сравнивая каждый бит первого числа с соответствующим битом второго числа.

Если оба бита равны 1, то устанавливаем соответствующий бит в результирующем числе в 1, в противном случае устанавливаем его в 0.

Полученные значения объединяем в одно число, которое будет представлять результат поразрядной конъюнкции исходных чисел.

3.4 Описание алгоритма

Для того чтобы реализовать данную задачу, для начало нужно поставить определенный символ перед первым числом, чтобы проходиться по первому числу, в данном случае это символ "?".

```
[01|30] *1110101&111
[02|01] ?1110101&111
[03|03] 1?110101&111
[04|03] 11?10101&111
[05|03] 111?0101&111
[06|04] 1110?101&111
[07|03] 11101?01&111
[08|04] 111010?1&111
[09|03] 1110101?&111
```

Рисунок 7 – Проход по первому числу

Затем сравниваем три символа ’1?&’ или же ’0?&’. Если нашлось ’1?&’, то заменяем на ": иначе, если нашлось ’0?&’, то заменяем на ";"и проходимся по второму числу.

```
[10|05] 111010&:111
[11|07] 111010&1:11
[12|07] 111010&11:1
[13|07] 111010&111:
[14|09] 111010&111"
```

Рисунок 8 – Проход по второму числу

После того как прошлись по второму числу используем символ ”‘ для того, чтобы сравнить числа. Теперь реализуем условия: если нашлось ’0“, то заменить на ’=’, иначе, если нашлось ’1“ заменить его на ’+’. Рассмотрим второй случай. Теперь заменим ’+’ на ’ 1’, поставим перед единичной пробел, для того, чтобы не было конфликтов с основным числом. Второй случай будет аналогичным, только вместо ’+’, будет ’=’, а также замена единицы на нолик. После этого начинаем с самого начала.

```
[15|11] 111010&11+
[16|14] 111010&11 1
[17|30] *111010&11 1
[18|01] ?111010&11 1
```

Рисунок 9 – Сравнение чисел

В конце преобразований, как только находится '*&', то заменяем их на '-'. Делаем проверку, если первое число, меньше чем второе, то с помощью знака '-' заменяем все значения второго числа на 0, т.к при поразрядной конъюкции числа, остаток больших из них заменяется на 0.

```
*&1111 0 0 0 0 1
-1111 0 0 0 0 1
0-111 0 0 0 0 1
00-11 0 0 0 0 1
000-1 0 0 0 0 1
0000- 0 0 0 0 1
```

Рисунок 10 – Пример удаление лишних значений

Остается последний шаг - соединить цифры при помощи знака '!' и вывести окончательный результат данной программы.

```
[87|25] 0000! 0 0 0 0 1
[88|26] 0000!0 0 0 0 1
[89|27] 00000! 0 0 0 1
[90|26] 00000!0 0 0 1
[91|27] 000000! 0 0 1
[92|26] 000000!0 0 1
[93|27] 0000000! 0 1
[94|26] 0000000!0 1
[95|27] 00000000! 1
[96|26] 00000000!1
[97|28] 000000001!
[98|29] 000000001
```

Рисунок 11 – Окончатльный вывод

Программные правила, которые выполняют данные преобразования прикреплена на 19 странице

3.5 Тестирование

В рамках тестирования алгоритма на вход программе были представлены несколько строк различного вида(одни нолики, одни единицы, две цифры и т.д). Полный список рассмотренных тестовых случаев представлен в таблице ниже.

Входные данные	Выходные данные
1&1	1
0&0	0
11101&1010	01000
1110101&111	0000101

```
*1->?1
*0->?0

?1->1?
?0->0?
1?&->&:
0?&->&;

// для единицы
:1->1:
:0->0:
:->"
0"->=
1"->+
" -> 0
=-> 0
+-> 1

// для нуля
;1->1;
;0->0;
;->'
0'->(
1'->(
(-> 0
' -> 0

*&->-
-0->0-
-1->0-
-->!
! ->!
!0->0!
!1->1!
!->.

->*
```

Рисунок 12 – Программные правила для реализации задачи

3.6 Выводы по главе

Разработан и реализован в среде разработки алгоритмов алгоритмической модели Маркова алгоритм вычисления поразрядной конъюкции исходных чисел. 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.

Исследование позволило убедиться в широком спектре применения нормальных алгоритмов Маркова и их значимости для решения различных задач. Полученные результаты могут быть использованы в дальнейших исследованиях и разработке новых методов и приложений, основанных на нормальных алгоритмах Маркова.

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 совершае-

мые 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.

Для выполнения поставленных задач мне было необходимо:

1. Эмулятор машины Тьюринга в четвёрках
2. Диаграммер для работы с диаграммами Тьюринга
3. Эмулятор для нормальных алгоритмов Маркова
4. Инстирумент компьютерной вёрстки LaTeX
5. Мой репозиторий на GitHub, куда были выложены все описанные ниже алгоритмы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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

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

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

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

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

7. Лекции лауреатов премии Тьюринга. / Пер. с англ. – Мир. – Москва

8. «Теория вычислений» Декстера К. Козена.

9. «Введение в теорию автоматов, языки и вычисления» Джона Э. Хопкрофта, Раджива Мотвани и Джеффри Д. Уллмана.

10. «Модели вычислений: исследование возможностей вычислений», Джон Э. Сэвидж.

11. «Введение в теорию вычислений» Майкла Сипсера.

12. «Вычислимость и логика» Джорджа С. Булоса, Джона П. Берджесса и Ричарда К. Джеффри.

ПРИЛОЖЕНИЕ А

```
00, ,<,kp1
00,0,<,kp1
00,1,<,kp1
```

#НАХОЖДЕНИЕ ПЕРВОГО ПРОБЕЛА

```
kp1,0,<,kp1
kp1,1,<,kp1
kp1, ,>,kp2

kp2,1,>,kp3
kp2,0,>,kp3
kp3,0, ,num0
kp3,1, ,num1
```

#ПЕРЕХОД К ГЕНЕРАЦИИ ВТОРОГО ЧИСЛА

```
kp3, ,<,rp
rp,1,<,rp
rp,0,<,rp
rp, ,>,pk
```

#КОПИРУЕТ ЦИФРУ 1 В ЧЕТНЫЕ РАЗРЯДЫ

```
num1, ,>,prw
prw,1,>,prw
prw,0,>,prw
prw, ,>,move1v
move1v,1,>,move1v
move1v,0,>,move1v
move1v, ,1,miss1
place1,1,>,place1
place1,0,>,place1
place1, ,1,miss1
miss1,1,<,miss1p
miss1p,1,<,miss1p
miss1p,0,<,miss1p
miss1p, ,<,leftspace1
leftspace1,1,<,leftspace1
leftspace1,0,<,leftspace1
leftspace1, ,1,next1
next1,1,>,kp2
```

#КОПИРУЕТ ЦИФРУ 0 В ЧЕТНЫЕ РАЗРЯДЫ

```
num0, ,>,prw0
prw0,1,>,prw0
prw0,0,>,prw0
prw0, ,>,move0v
move0v,1,>,move0v
move0v,0,>,move0v
move0v, ,0,miss0
place0,1,>,place0
place0,0,>,place0
place0, ,1,miss0
miss0,1,<,miss0p
miss0,0,<,miss0p
miss0p,1,<,miss0p
miss0p,0,<,miss0p
miss0p, ,<,leftspace0
leftspace0,1,<,leftspace0
leftspace0,0,<,leftspace0
leftspace0, ,0,next0
next0,0,>,kp2



kp2, ,<,skip2
skip2,1,<,skip2
skip2,0,<,skip2
skip2, ,>,pk
```

#КОПИРОВАНИЕ ЦИФР В ЧЕТНЫЕ РАЗРЯДЫ

```
pk,1, ,space1
pk,0, ,space0
space1, ,>,spacenum1
space1,1,>,space1
space1,0,>,space1
spacenum1,1,>,spacenum1
spacenum1,0,>,spacenum1
spacenum1, ,>,number1
number1,1,>,number1
number1,0,>,number1
number1, ,>,num21



space0, ,>,spacenum0
```

```
spacenum0,1,>,spacenum0
spacenum0,0,>,spacenum0
spacenum0, ,>,number0
number0,1,>,number0
number0,0,>,number0
number0, ,>,num20
```

#КОПИРУЕТ ЦИФРУ 1 В НЕЧЕТНЫЕ РАЗРЯДЫ

```
num21,1,>,num21
num21,0,>,num21
num21, ,1,skip21
skip21,1,<,skip21
skip21,0,<,skip21
skip21, ,<,nextn1
nextn1,0,<,nextn1
nextn1,1,<,nextn1
nextn1, ,<,nextn2
nextn2,1,<,nextn2
nextn2,0,<,nextn2
nextn2, ,1,next21
next21,1,>,search
search,1,>,pk
search,0,>,pk
search, ,>,proverka1
```

#КОПИРУЕТ ЦИФРУ 0 В НЕЧЕТНЫЕ РАЗРЯДЫ

```
num20,1,>,num20
num20,0,>,num20
num20, ,0,skip20
skip20,1,<,skip20
skip20,0,<,skip20
skip20, ,<,nextn10
nextn10,0,<,nextn10
nextn10,1,<,nextn10
nextn10, ,<,nextn20
nextn20,1,<,nextn20
nextn20,0,<,nextn20
nextn20, ,0,next20
next20,0,>,search0
search0,1,>,pk
search0,0,>,pk
search0, ,>,proverka1
```

```
pk, ,>,proverka1
#БЛОК ПРОВЕРОК
proverka10,0,>,proverka10
proverka10,1,<,obrk
obrk,0,<,obrk
obrk, ,>,delete0
proverka10, , ,final
proverka1,0,>,proverka1
proverka1,1,<,obratno1
obratno1,1,1,promez0
obratno1,0,<,obratno1
obratno1, ,>,delete0
proverka1, ,>,proverka2
proverka2,0,>,proverka2
proverka2,1,1,promez1
proverka2, , ,final


promez1,1, ,promez2
promez2, ,<,promez2
promez2,1,>,promokod
promez2,0,>,promokod
promokod, ,>,promo
promo, ,1,okont
okont,1,>,dolgo1



promez0,1, ,promez01
promez01, ,<,promez01
promez01,1,>,promokod0
promez01,0,>,promokod0
promokod0, ,1,okont0
okont0,1,>,dolgo1
#УДАЛЕНИЕ ВЕДУЩИХ НУЛЕЙ
delete0,0, ,delete0
delete0, ,>,delete0
delete0,1,>,PROVERKA
```

```
PROVERKA,1,<,PREDZAM
PROVERKA,0,<,PREDZAM
PROVERKA, ,<,onenum
onenum,1, ,ostatok1
onenum,0, ,ostatok0



ostatok1, ,<,ostatok1
ostatok1,1,>,post1
ostatok1,0,>,post1
post1, ,>,postav1
postav1, ,1,pspace1
pspace1,1,>,fina



ostatok0, ,<,ostatok0
ostatok0,1,>,post0
ostatok0,0,>,post0
post0, ,>,postav0
postav0, ,0,pspace0
pspace0,0,>,fina



fina, ,>,dolgo
fina,1,1,final
fina,0,>,final
#ПЕРЕНОС ЧИСЛА ВЛЕВО
PREDZAM,1, ,zam
zam, ,<,zam
zam,1,>,rpq
zam,0,>,rpq
rpq, ,>,rp1
rp1, ,1,numbers1

numbers1,1,>,findnum
findnum, ,>,findnum
findnum,1,>,fn1
findnum,0,>,fn1
fj, ,>,fj
```

```
fj ,1 ,1 ,findnum
fj ,0 ,0 ,findnum
fn1 ,1 ,< ,fni1
fn1 ,0 ,< ,fni1
fn1 ,  ,> ,perex
perex ,  ,< ,poxod
perex ,0 ,< ,dryg
perex ,1 ,< ,dryg
fort ,  ,< ,fort
fort ,1 ,  ,fort1
fort ,0 ,  ,fort0

fni1 ,  ,< ,fni1
fni1 ,1 ,  ,perenos1
fni1 ,0 ,  ,perenos0

perenos1 ,  ,< ,findn
findn ,  ,< ,findn
findn ,1 ,> ,out
findn ,0 ,> ,out
out ,  ,1 ,xt1
xt1 ,1 ,> ,fj

perenos0 ,  ,< ,findn0
findn0 ,  ,< ,findn0
findn0 ,1 ,> ,out0
findn0 ,0 ,> ,out0
out0 ,  ,0 ,xt0
xt0 ,0 ,> ,fj

dryg ,  ,< ,dryg
dryg ,1 ,  ,dryg1
dryg ,0 ,  ,dryg0

dryg1 ,  ,< ,dryg1
dryg1 ,1 ,> ,ppi
dryg1 ,0 ,> ,ppi
ppi ,  ,1 ,chikl

dryg0 ,  ,< ,dryg0
dryg0 ,1 ,> ,ppi0
dryg0 ,0 ,> ,ppi0
```

```
ppi0 ,  ,0 ,chikl

poxod ,  ,< ,poxod
poxod ,1 ,  ,poxod1
poxod ,0 ,  ,poxod0

poxod1 ,  ,< ,poxod1
poxod1 ,1 ,> ,perexod1
poxod1 ,0 ,> ,perexod1
perexod1 ,  ,1 ,kones
kones ,1 ,> ,final

poxod0 ,  ,< ,poxod0
poxod0 ,1 ,> ,perexod0
poxod0 ,0 ,> ,perexod0
perexod0 ,  ,0 ,kones0
kones0 ,0 ,> ,final


chikl ,1 ,> ,while
chikl ,0 ,> ,while
while ,  ,> ,while
while ,1 ,1 ,proverka10
while ,0 ,0 ,proverka10


fort1 ,  ,< ,fuil
fuil ,  ,< ,fuil
fuil ,1 ,> ,hor
fuil ,0 ,> ,hor
hor ,  ,1 ,horm
horm ,1 ,> ,proverka1

fort0 ,  ,< ,fuil0
fuil0 ,  ,< ,fuil0
fuil0 ,1 ,> ,hor0
fuil0 ,0 ,> ,hor0
hor0 ,  ,0 ,horm0
horm0 ,0 ,> ,proverka1
#ФИНАЛ
final ,  ,> ,dolgo
dolgo1 ,1 ,> ,dolgo1
```

```
dolgo1,0,>,dolgo1
dolgo1, , ,fin
dolgo, ,<,dolgo
dolgo,1,>,fin
dolgo,0,>,fin

fin, ,#,fin
```