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

Явметдинов Максим Русланович

Проверил:

Преподаватель

Севастьянов Виктор Сергеевич

Москва 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ……..……..4

ГЛАВА 1. МАШИНЫ ТЬЮРИНГА…….……..….5

1.1 Вводная часть…….….5

1.2 Постановка задачи……..8

1.3 Идея решения задачи……..8

1.4 Описание алгоритма……...8

1.5 Тестирование и оценка сложности…….…..9

1.6 Выводы по главе…….……10

ГЛАВА 2. ДИАГРАММЫ ТЬЮРИНГА……..10

2.1 Вводная часть……10

2.2 Постановка задачи……13

2.3 Идея решения задачи……13

2.4 Описание алгоритма…….14

2.5 Тестирование и оценка сложности..……...17

2.6 Выводы по главе……...…18

ГЛАВА 3. НОРМАЛЬНЫЕ АЛГОРИТМЫ МАРКОВА……..…..18

3.1 Вводная часть……18

3.2 Постановка задачи……20

3.3 Идея решения задачи……20

3.4 Описание алгоритма……20

3.5 Тестирование и оценка сложности……21

3.6 Выводы по главе……...…22

ЗАКЛЮЧЕНИЕ…….23

ПРИЛОЖЕНИЕ А…….25

ПРИЛОЖЕНИЕ Б…….30

Введение

В рамках предмета фундаментальной информатики одним из ключевых понятий является алгоритм. Формальное определение алгоритма играет важную роль в компьютерной науке и информатике, так как оно предоставляет точное и строгое описание процесса выполнения задачи. Неформальное определение алгоритма, хоть и дает общее представление о том, что он представляет собой, оставляет много пространства для интерпретации и не учитывает все тонкости и нюансы, связанные с выполнением алгоритма.

В неформальном определении алгоритма указывается, что это последовательность правил, которая позволяет получить выходное сообщение определенного вида из входного сообщения за конечное число шагов. Однако данное определение слишком упрощено и не учитывает различные аспекты, связанные с выполнением алгоритмов. Например, понятия "понятные" и "легко выполнимые" могут быть различно интерпретированы, а также не учитывается, что не все математические задачи могут быть разрешимы алгоритмически.

Для установления неразрешимости какой-либо задачи необходимо четкое и формальное определение алгоритма. Формализация понятия алгоритма может быть проведена с помощью алгоритмических моделей. Две основные алгоритмические модели, которые используются для формализации понятия алгоритма, — это машина Тьюринга и нормальные алгоритмы Маркова.

Машина Тьюринга представляет собой устройство, способное выполнять небольшое количество простых операций, и является одной из самых популярных алгоритмических моделей. Она состоит из бесконечной ленты, разделенной на ячейки, и головки, которая считывает и записывает символы на ленте, и принимает решения на основе прочитанных символов. Машина Тьюринга может быть использована для решения широкого спектра задач, и она имеет математическую эквивалентность с различными моделями вычислений.

Нормальные алгоритмы Маркова представляют собой другую алгоритмическую модель, которая описывает преобразования последовательности символов в произвольных алфавитах. Нормальные алгоритмы Маркова представляют собой набор инструкций для изменения последовательности символов в соответствии с определенными правилами. Эти алгоритмы имеют широкое применение в теории вычислимости и являются важным инструментом для исследования алгоритмов.

Целью данной работы является иллюстрация определения алгоритма путем построения алгоритмических моделей Тьюринга и Маркова. В рамках данной работы будут разработаны алгоритмы для решения различных задач с использованием машины Тьюринга, а также проведено построение эквивалентных графических представлений этих алгоритмов, а также нормальных алгоритмов Маркова.

Глава 1. Машины Тьюринга.

1.1 Вводная часть.

Машина Тьюринга представляет собой абстрактную математическую модель вычислительного устройства, предложенную Аланом Тьюрингом в 1936 году. Основана она на идее обработки данных на бесконечной ленте, которая разделена на ячейки. Эта модель в дальнейшем стала фундаментальной для теории вычислимости и теории алгоритмов, и считается одной из основополагающих для современной информатики.

Машина Тьюринга представляет собой устройство, которое состоит из бесконечной ленты, разделенной на ячейки, и комбинированной головки, способной считывать и писать символы на ленте. Головка может перемещаться вдоль ленты и находится в одном из конечного множества состояний. Модель машины Тьюринга включает в себя рабочий алфавит символов, конечное множество состояний, начальное состояние и программу, определяющую функции переходов, записи и движения головки.

Состояние ленты машины Тьюринга представляет собой упорядоченную пару объектов, где первый объект - сообщение, записанное на ленте, а второй объект - расстояние от края ленты до рабочей ячейки. Интерпретатор программ машины Тьюринга в четвёрках jstu4 представляет собой удобный инструмент для работы с программами машины Тьюринга.

Машина Тьюринга оперирует над входными данными и выполняет преобразования в соответствии с программой, определенной на ней. Основными функциями Машины Тьюринга являются функция выхода, функция переходов и функция движения головки.

Функция выхода определяет, какой символ будет записан на ленту в текущей ячейке после выполнения операции. Согласно формальному определению, функция выхода представляет собой отображение из декартова произведения множества состояний и символов в множество символов.

Функция переходов определяет следующее состояние машины Тьюринга и выполняемую операцию в зависимости от текущего состояния и символа, считываемого с ленты. Эта функция определяет, как машина Тьюринга переходит из одного состояния в другое в зависимости от символа на ленте.

Функция движения головки управляет движением головки машины Тьюринга. Она определяет, в каком направлении и на какое количество ячеек головка должна переместиться после выполнения операции. Функция движения головки может указывать на движение влево, вправо или на неподвижность, если головка должна остаться на месте.

Программирование Машины Тьюринга осуществляется путем определения программы, которая состоит из набора команд, задающих функции переходов, записи и движения головки. Каждая команда представляет собой упорядоченную четвёрку символов, где первый символ - текущее состояние, в котором находится машина, второй символ - символ, считываемый с ленты, третий символ - символ, который необходимо записать в текущей ячейке, четвертый символ - новое состояние, в которое машина должна перейти после выполнения команды.

Программа Машины Тьюринга может быть представлена в виде упорядоченных наборов (в данном случае - четвёрок) символов, которые определяют последовательность выполнения команд. Порядок выполнения команд не зависит от порядка, в котором они записаны в тексте программы.

При программировании Машины Тьюринга важно учитывать, что команды должны быть составлены таким образом, чтобы машина выполняла нужные действия в соответствии с требованиями задачи.

Для наглядного представления состояний Машины Тьюринга используется упорядоченная пара объектов, где первый объект - сообщение, записанное на ленте, а второй объект - расстояние от края ленты до рабочей ячейки. На рисунке 1 показан пример описания состояний Машины Тьюринга, где левый край ленты обозначается квадратной скобкой, правый - треугольной скобкой, а рабочая ячейка выделяется круглыми скобками.

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

Рисунок 1 – Пример описания состояний МТ

Интерпретатор программ Машины Тьюринга в четвёрках jstu4 представляет собой веб-приложение, которое принимает на вход текст

программы и входные данные, а затем выводит результат преобразования в соответствии с программой на Машине Тьюринга. Интерфейс интерпретатора (рисунок 2) также позволяет использовать комментарии, что делает его удобным инструментом для разработки и тестирования программ Машины Тьюринга.

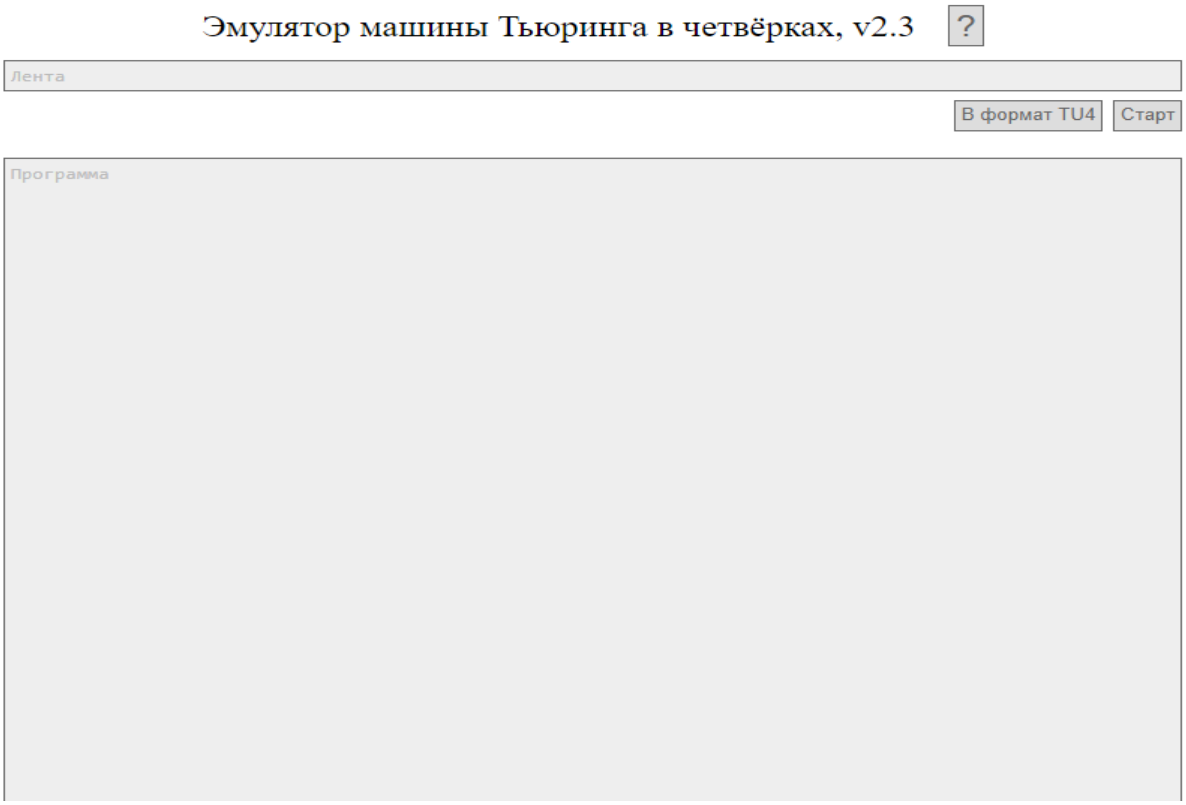

Рисунок 2 – Интерфейс интерпретатора jstu4

Машина Тьюринга имеет важное значение в контексте теории вычислимости. Она является одной из первых абстрактных моделей вычислительного устройства, и её возможности считаются полностью эквивалентными вычислительным возможностям современных компьютеров.

Машина Тьюринга представляет собой универсальную вычислительную модель, которая может выполнять любой вычислимый процесс. Теория вычислимости использует Машину Тьюринга для изучения возможностей и ограничений алгоритмов и вычислений, а также для проверки разрешимости различных задач.

Машина Тьюринга и её модель вычислений нашли применение в различных областях науки и техники. В информатике Машина Тьюринга применяется для моделирования вычислительных процессов, анализа алгоритмов, разработки программного обеспечения и решения различных задач.

Кроме того, Машина Тьюринга используется в теории формальных языков и автоматов для изучения языковых конструкций и алгоритмов обработки строк. Также она имеет важное значение в криптографии, теории сложности вычислений, искусственном интеллекте и многих других областях информатики и математики.

1.2 Постановка задачи.

В рамках задачи №43 необходимо составить алгоритм перевода числа из двоичной системы счисления в восьмеричную. Рабочим алфавитом машины будет являться множество {0, 1, 2, 3, 4, 5, 6, 7}.

1.3 Идея решения задачи.

Первым шагом необходимо провести подсчет количества цифр в двоичном числе. Это позволит определить конечную тройку цифр, с которой надо начать отсчет.

Затем следует собрать каждую тройку цифр, начиная с 000 и заканчивая 111. Каждой тройке будет соответствовать определенная цифра в восьмеричной системе счисления. После этого необходимо приписывать полученную цифру к восьмеричному ответу, который будет формироваться по мере обработки каждой тройки цифр.

1.4 Описание алгоритма

Сначала, для работы с входным двоичным числом, необходимо определить количество цифр в этом числе. Это позволит точно указать длину самой левой тройки и количество цифр в выходном восьмеричном числе. Для этого проходим через все цифры в двоичном числе, сбрасывая счетчик на каждой тройке цифр. Это позволит определить начальную точку для работы с числом. Затем собираем тройку цифр слева направо, учитывая при этом размер самой левой тройки.

Если самая левая “тройка” не состоит из трех цифр, подразумеваем нули слева от цифр и ставим знак *"*"*, чтобы отметить место возврата. После того, как собрали тройку цифр и оставили знак "*" для отметки, сохраняем информацию о том, является ли эта отметка цифрой 0 или 1. Затем, используя эту информацию, программа переходит к соответствующей цепочке состояний, учитывая текущее состояние и входные данные. Это позволяет сохранить место, с которого начали обработку, для последующего возвращения, а также цифру, которая находилась на месте *‘‘*’’*.

Далее продолжаем пробегать число вправо и через пробел добавляем соответствующую цифру справа. Повторяем эту операцию до тех пор, пока не обработаем все цифры во входном числе. Каждый раз возвращаемся к

отметке "*", заменяем ее либо на 0, либо на 1 и повторяем этот процесс для следующей тройки цифр. Этот процесс повторяется, пока входное число полностью не будет обработано.

Исходный код программы с комментариями, поясняющими суть конкретных состояний, представлен в приложении А.

1.5 Тестирование и оценка сложности.

Для проверки алгоритма были использованы различные входные данные. Полный список тестовых случаев приведен в таблице 1.

Таблица 1 – Список рассмотренных тестовых случаев

Ввод	Вывод
0	0
11	3
0000111	7
10000111	207
1000011100	1034
1111111111	1777
10000000000	2000

Сложность алгоритма оценивалась по количеству 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	17
4	37
8	73
16	213
32	617

Основываясь на представленных результатах, можно отметить, что при увеличении длины входного сообщения в 2 раза, количество выполненных операций увеличивается примерно в 3 раза. Из этого можно заключить, что сложность алгоритма примерно соответствует O(n*logn).

1.6 Выводы по главе.

Был разработан алгоритм, который позволяет перевести вводное число из двоичной системы счисления в восьмеричную. Затем была создана программа для Машины Тьюринга, которая реализует этот алгоритм в четвёрках. Этот процесс позволил получить практический опыт работы с Машиной Тьюринга и написания алгоритмов для обработки двоичных чисел на языке низкого уровня.

Полученный опыт показал, что при составлении алгоритмов на языке низкого уровня требуется использовать намного большое количество команд для реализации элементарных действий. Это объясняется тем, что единственный способ "запомнить" знак или выполнить действие заключается в ассоциировании их с определенным состоянием или знаком. Каждое элементарное действие требует задания необходимого количества состояний, в зависимости от того, на каких знаках может находиться головка машины в момент выполнения этого действия. Для каждого состояния также требуется дублирование однотипных команд. Если бы алфавит был более объёмным, чем в условии задачи (например, перевод в шестнадцатиричную), код программы стал бы ещё более громоздким.

ГЛАВА 2. ДИАГРАММЫ ТЬЮРИНГА

2.1 Вводная часть

Диаграммы Тьюринга являются важным инструментом в теоретической информатике и вычислительной математике. Они представляют собой графическое представление машины Тьюринга, который является универсальной моделью вычислений. Машина Тьюринга состоит из бесконечной ленты, разделенной на ячейки, и головки, способной считывать и записывать символы на этой ленте. Диаграммы Тьюринга представляют эту модель в виде графа, где вершины представляют состояния машины, а ребра - переходы между этими состояниями.

Одним из главных преимуществ диаграмм Тьюринга является их способность упрощать сложные алгоритмы. Они позволяют избавиться от

нагромождения состояний, которое может возникать при попытке описать все детали работы машины Тьюринга в текстовой форме. Диаграммы представляют МТ через более простые МТ визуально-топологическим способом, что делает их более удобными для понимания и анализа.

Кроме того, диаграммы Тьюринга позволяют вводить собственные машины в процессе разработки алгоритма. Разработчики могут выделять часто повторяющиеся действия и создавать отдельные структуры для их выполнения, что позволяет сократить объем алгоритма и улучшить его читаемость. Это также упрощает отладку и оптимизацию алгоритмов, поскольку диаграммы Тьюринга позволяют легко выявлять и устранять повторяющиеся участки кода.

Состояния в диаграммах обозначаются точками, и каждая подмашина ограничена слева и справа точками, обозначающими текущее состояние и состояние, в которое машина перейдет после выполнения действия подмашины, соответственно. Это делает структуру алгоритма более ясной и понятной, что упрощает его анализ и понимание.

Еще одним важным аспектом диаграмм Тьюринга является их способность обобщать действия подмашины. Диаграммы Тьюринга позволяют задать действия подмашины без явной ассоциации с конкретными символами, над которыми это действие выполняется. Это делает алгоритм более абстрактным и универсальным, поскольку он может быть применен к различным входным данным.

Важным элементом диаграмм Тьюринга являются стрелки, которые представляют переходы между состояниями машин. Если после выполнения действия одной подмашины необходимо выполнить действие второй подмашины, их точки соединяются стрелкой, а над стрелкой указываются знаки, над которыми должна находиться головка машины, чтобы выполнить действие подмашины, к которой ведет стрелка. Это позволяет явно указать условия перехода между подмашинами и сделать алгоритм более структурированным и понятным.

Еще одним важным аспектом диаграмм Тьюринга является возможность повторения участков алгоритма. Если необходимо повторить какой-либо участок диаграммы до тех пор, пока в рабочей ячейке находится одна из букв некоторого фиксированного набора, этот участок диаграммы замыкается стрелкой с соответствующей надписью. Прекращение повторения осуществляется по букве, не входящей в этот набор. Это делает диаграммы Тьюринга еще более универсальными и мощными инструментами для описания сложных алгоритмов.

Для составления алгоритмов с помощью диаграмм Тьюринга можно использовать специальное программное обеспечение, называемое диаграммером. Диаграммер позволяет разработчикам интерактивно создавать диаграммы и исполнять описываемые ими алгоритмы. Это делает процесс разработки алгоритмов более удобным и эффективным, поскольку разработчики могут визуально представлять работу алгоритма и мгновенно видеть его выполнение.

Одним из популярных инструментов для создания диаграмм Тьюринга является диаграммер jdt (рисунок 3). Он обеспечивает богатые возможности для создания сложных алгоритмов, включая возможность создания разветвлений, циклов и условных переходов. Это делает его полезным инструментом для разработки алгоритмов машины Тьюринга и их анализа.

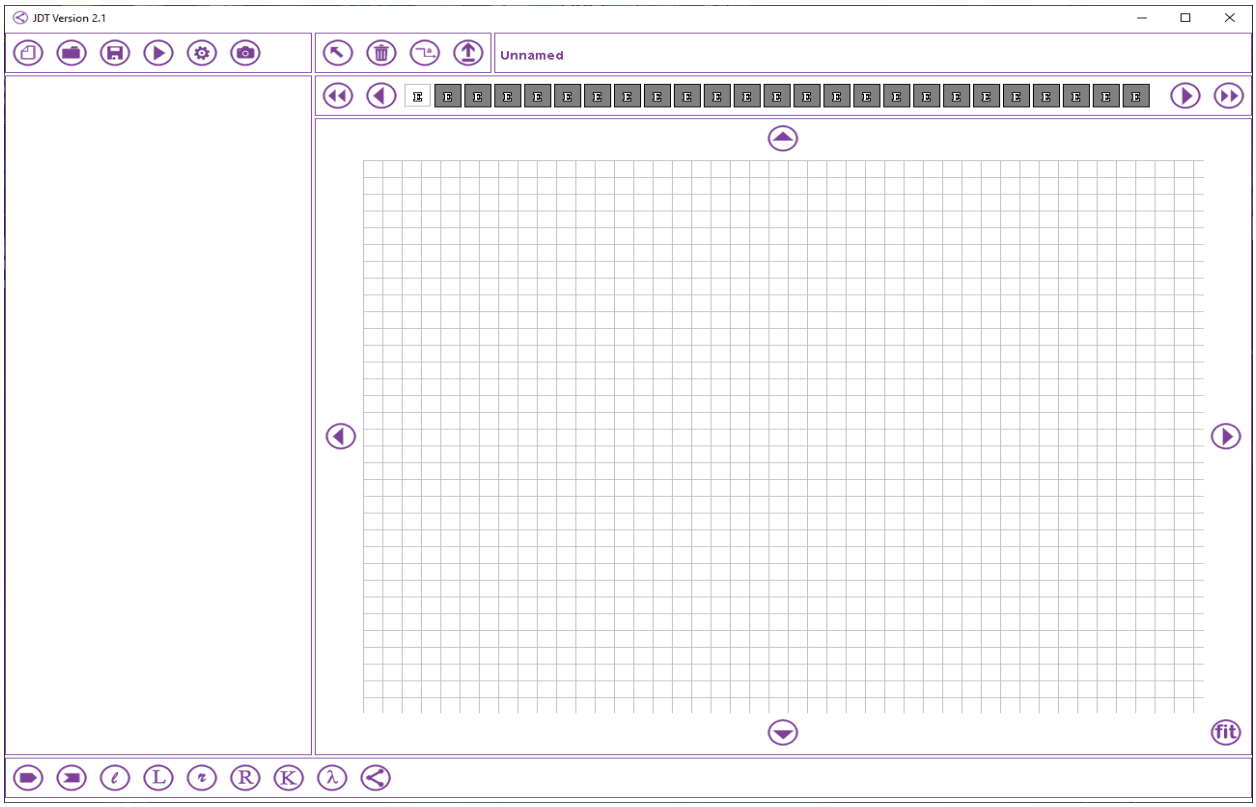

Рисунок 3 – Интерфейс диаграммера jdt

Общий подход к использованию диаграмм Тьюринга заключается в том, что разработчики могут создавать структурированные и понятные алгоритмы, которые легко понимать, анализировать и оптимизировать.

Диаграммы Тьюринга позволяют избежать ненужных деталей и упростить сложные алгоритмы, делая их более понятными и поддающимися анализу. Это позволяет разработчикам создавать более эффективные и надежные программные решения.

Использование диаграмм Тьюринга также позволяет улучшить процесс разработки алгоритмов за счет упрощения структуры и повышения читаемости кода. Разработчики могут легко выделять отдельные структуры и действия в виде подмашин, что делает алгоритм более структурированным и понятным. Это упрощает отладку, тестирование и поддержку алгоритмов, что в конечном итоге приводит к улучшению качества программного обеспечения.

Таким образом, диаграммы Тьюринга представляют собой мощный инструмент для создания, анализа и оптимизации алгоритмов машины Тьюринга. Они позволяют разработчикам представлять сложные алгоритмы в удобной и понятной форме, упрощая их анализ и понимание. Использование диаграмм Тьюринга также способствует улучшению процесса разработки алгоритмов за счет упрощения структуры и улучшения читаемости кода.

2.2 Постановка задачи

В рамках поставленной задачи №33 необходимо сконструировать диаграмму Тьюринга, реализующую алгоритм уменьшения на единицу целого неотрицательного числа в шестнадцатеричной системе счисления.

2.3 Идея решения задачи.

В начале алгоритма используется машина К, которая выполняет копирование исходного числа. Это необходимо для того, чтобы сохранить исходное число и продолжить операции на его копии, т.е. сделать нормированный алгоритм.

Затем сдвигаемся влево на одну ячейку для перехода к последней цифре и отнимаем 1 от текущей цифры. Если эта цифра равна нулю, то заменяем ее символом "f" и продолжаем движение влево. Таким образом, ищем первую ненулевую цифру числа, с которой можно продолжить операции.

Уменьшение на единицу продолжается до тех пор, пока не найдем первую ненулевую цифру. Это гарантирует, что достигнем последней значащей цифры.

После этого производится удаление незначащих нулей, которые могут находиться слева от числа. Это делается путем просмотра каждой цифры числа, начиная с самой старшей, и удалением всех нулей до первой ненулевой цифры.

2.4 Описание алгоритма

Диаграмма основной машины представлена на рисунке 4.

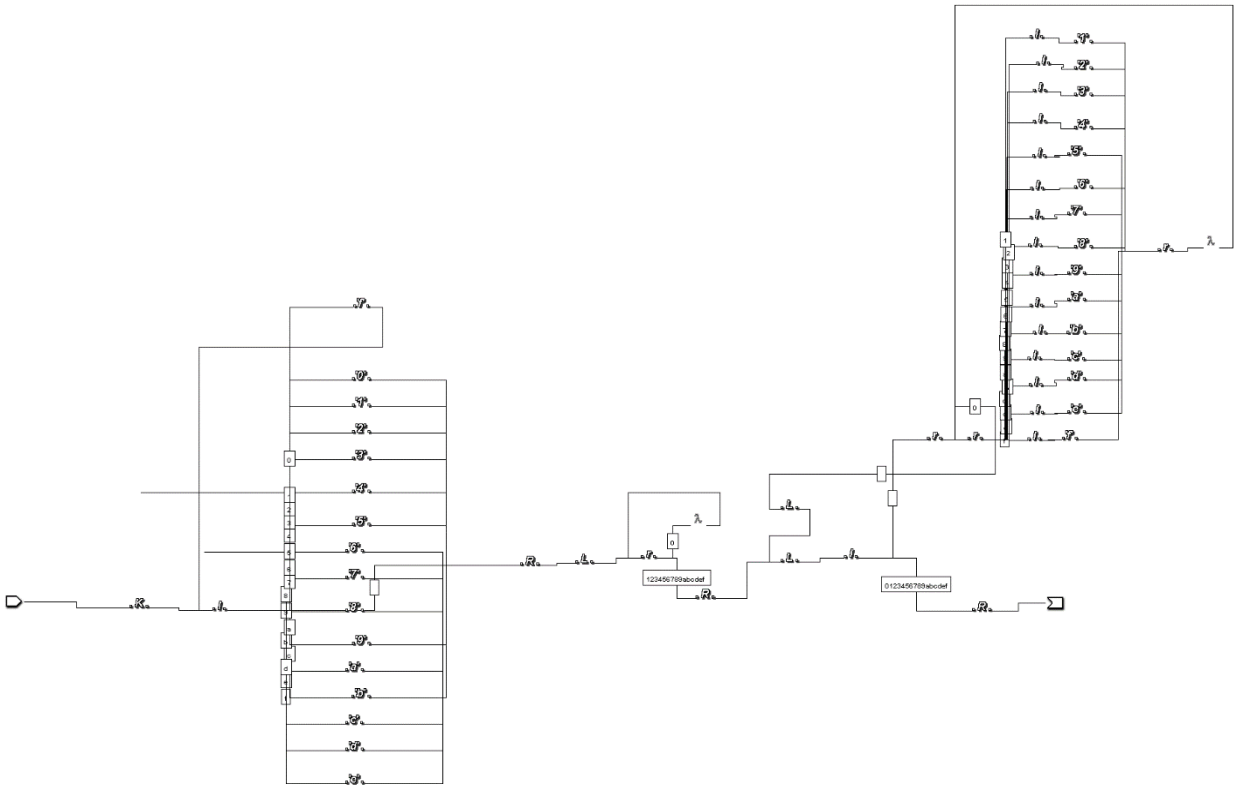

Рисунок 4 – Диаграмма основной машины

Один из первых шагов в алгоритме — это копирование исходного числа с помощью машины К. Это делается для того, чтобы сохранить исходное значение числа и иметь возможность продолжать операции на его копии. Копирование позволяет сохранить исходное число неизменным, что делает алгоритм нормированным.

После копирования числа перемещаемся на одну ячейку влево, чтобы начать операцию вычитания единицы из текущей цифры числа. Это делается путем уменьшения значения текущей ячейки на 1. Если значение текущей цифры числа до вычитания оказывается равным нулю, заменяем эту цифру символом "f". Затем продвигаемся влево на еще одну ячейку и продолжаем операцию вычитания единицы из новой текущей цифры.

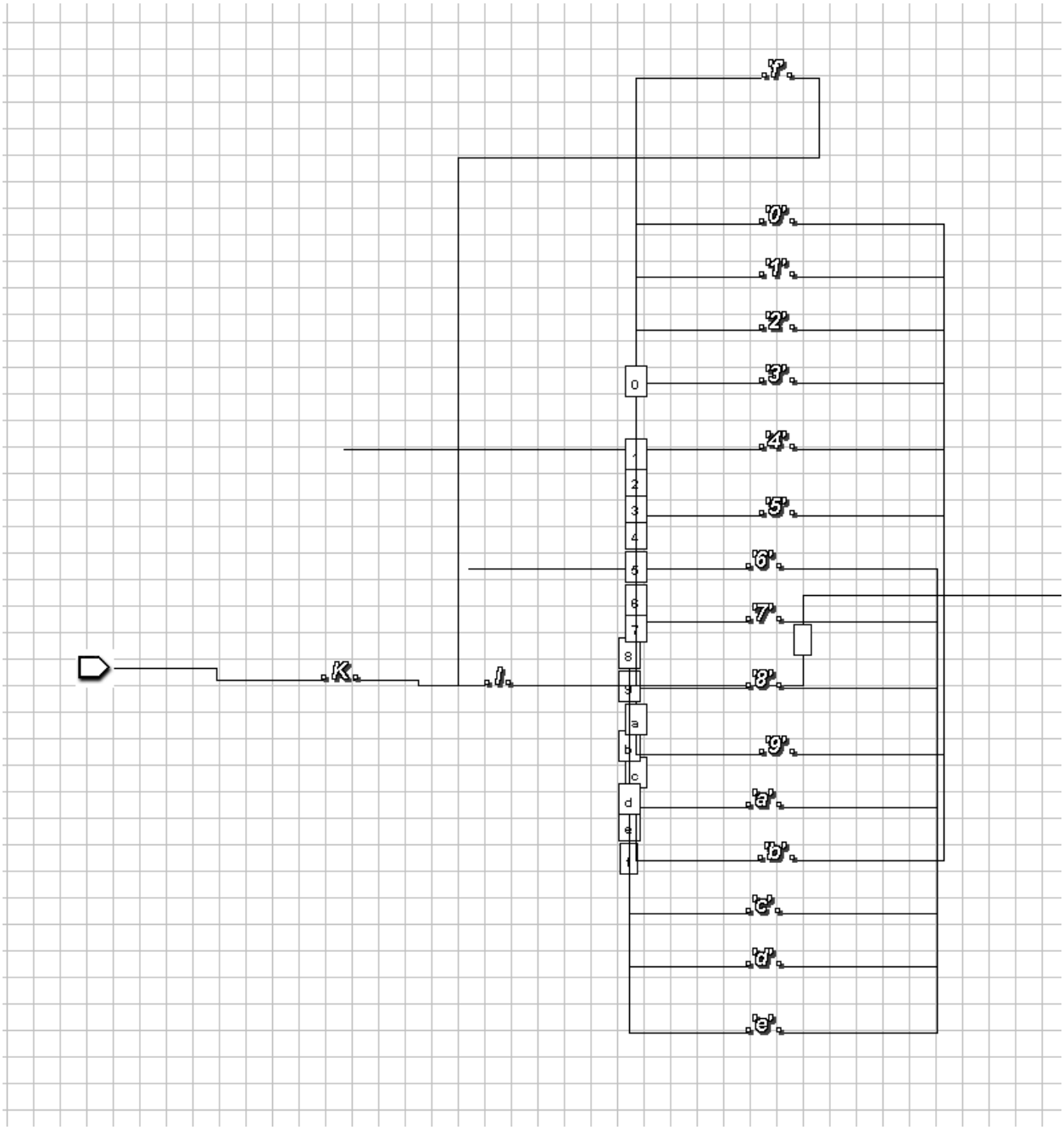

Рисунок 5

Следующий шаг включает проверку на наличие незначащих нулей, которые могут находиться слева от числа.

Для проверки наличия незначащих нулей сравниваем каждую цифру числа, начиная с самой левой, с нулем. Если текущая цифра равна нулю, то удаляем эту цифру и продолжаем проверку со следующей цифры. Если текущая цифра ненулевая, то останавливаем проверку, так как все незначащие нули уже удалены.

Процесс проверки на наличие незначащих нулей происходит итеративно, пока не будет достигнут конец числа или первая ненулевая

цифра. Это гарантирует, что все лишние нули слева от числа будут удалены, оставляя только значимые цифры.

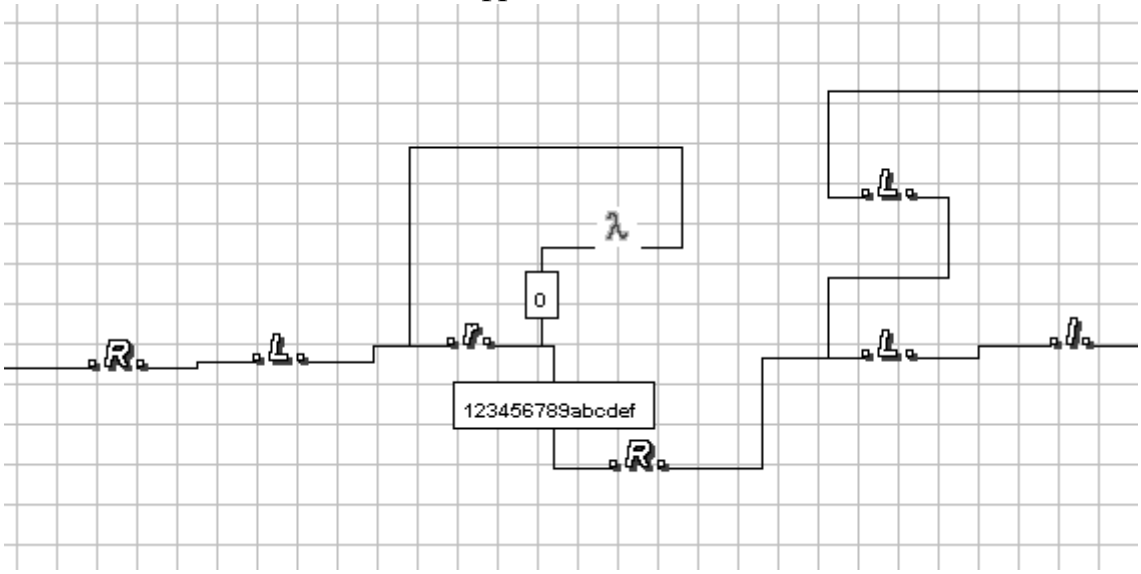

Рисунок 6

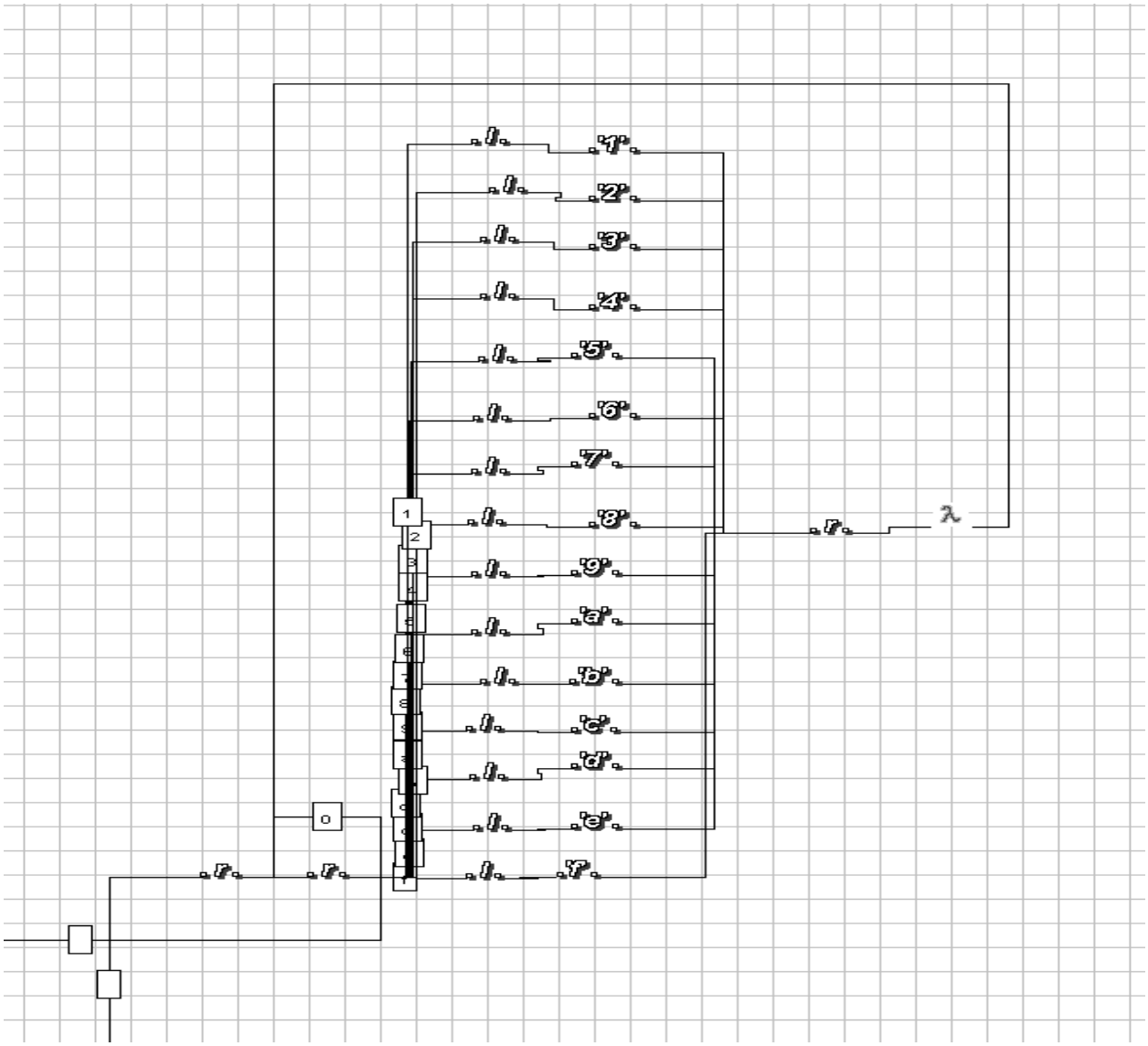

Рисунок 7

Когда завершается перемещение до конца вправо и достигается конечной ячейки, это означает, что произошел выход из машины и можно закончить алгоритм. При этом результатом будет уменьшенное на единицу число в шестнадцатеричной системе счисления.

2.5 Тестирование и оценка сложности

При проведении тестирования алгоритма использовались различные входные данные. В таблице 3 приведен полный список тестовых случаев, которые были рассмотрены.

Таблица 3 – Список рассмотренных тестовых случаев

Входные данные	Выходные данные
10	f
11	10
39390	3938f
800000000	7fffffff
834465210	83446520f

Сложность алгоритма также оценивалась по количеству команд, выполненных машиной во время выполнения алгоритма. Так как использованный интерпретатор не приводит данных о количестве выполненных операций, считать их требовалось вручную. Эта величина зависит только от длины входного сообщения. Результаты тестирования на входных сообщениях разной длины приведены в таблице 4. Для тестирования были взяты числа, состоящие из нулей кроме самого старшего разряда.

Таблица 4 – Зависимость входного сообщения и количества операций

Длина входного сообщения	Количество выполненных операций
1	7
2	13
4	17
8	25
16	41

На основе предоставленных данных о зависимости длины входного сообщения от количества выполненных операций, можно сделать предположение о характере роста временной сложности алгоритма. При

увеличении длины входного сообщения в два раза, количество операций не увеличивается вдвое, что указывает на отсутствие линейной зависимости. Тем не менее, учитывая, что длины входных сообщений являются степенями двойки, можно предположить логарифмический рост числа операций относительно увеличения размера входа. Заметим, что начиная с длины входного сообщения, равной 4, при удвоении длины сообщения количество выполненных операций увеличивается примерно в два раза. Таким образом, временная сложность алгоритма имеет порядок O($n\log n$).

2.6 Выводы по главе

Был разработан и реализован алгоритм уменьшения на единицу целого неотрицательного числа в шестнадцатеричной системе счисления в среде разработки Диаграмм Тьюринга. При работе с более высокоуровневой реализацией абстрактного исполнителя Машины Тьюринга был дополнен опыт разработки алгоритмов, который был использован для решения предыдущей задачи.

При работе с диаграммой в сравнении со средой Машины Тьюринга удобнее задавать в алгоритме рутинные операции без необходимости задумываться о наименовании состояний и других деталях. Однако, интерфейс диаграммы оказался громоздким по сравнению с интерфейсом интерпретатора Машины Тьюринга, что затруднило быструю отладку алгоритма. Также ошибки диаграммера были не очень информативными, что еще больше затрудняло отладку.

С другой стороны, диаграммы оказались удобны для написания без использования интерпретатора, например, на бумаге, и последующей отладки путем ручного выполнения алгоритма.

К сожалению, в программе возникла ошибка, которая привела к тому, что вся диаграмма была удалена, из-за чего мне пришлось делать все сначала, и это затянуло процесс разработки.

ГЛАВА 3. НОРМАЛЬНЫЕ АЛГОРИТМЫ МАРКОВА

3.1 Вводная часть

Исследуемая алгоритмическая система, предложенная академиком А. А. Марковым в период с 1947 по 1954 годы, представляет собой важный инструмент для обработки текстовых сообщений. Она базируется на замене

подслов исходного сообщения на другие слова, аналогично машине Тьюринга. Нормальные алгоритмы Маркова являются детерминистическими текстовыми заменами, что позволяет однозначно задать вычисления для каждого входного слова, обеспечивая тем самым завершение алгоритма и порождение конкретного результата.

Важным аспектом данной модели является определение приоритетов применения правил, которые могут быть установлены линейным порядком их записи. Это позволяет эффективно управлять процессом преобразования текстовых сообщений. В алгоритмической системе Маркова отсутствует понятие ленты, и предполагается непосредственный доступ к различным частям преобразуемого слова.

Основные принципы выполнения алгоритма в данной системе включают выбор первого применимого правила в случае нескольких применимых правил, а также выбор самого левого места применения правила, если правило применимо в нескольких местах обрабатываемого слова.

Нормальный алгоритм Маркова (НАМ) представляет собой упорядоченный набор правил-продукций, где каждая продукция представляет собой формулу замены части входного слова, совпадающей с левой частью формулы, на её правую часть. Процесс выполнения НАМ завершается, когда все формулы оказываются неприменимыми или когда применяется терминальная (завершающая) продукция. Также важно отметить, что если в процессе выполнения НАМ бесконечно долго применяются нетерминальные правила, то алгоритм неприменим к данному входному слову.

Достаточные признаки применимости НАМ ко всем входным словам включают обязательное наличие непустых левых частей всех продукций и отсутствие букв из левых частей в правых частях, а также то, что каждая правая часть правила короче левой части.

Для отладки НАМ удобно использовать различные интерпретаторы. В данной работе использовался интерпретатор nam. В среде данного интерпретатора для обозначения терминальной продукции используется точка, а если точка используется как непосредственно слово для подстановки, то она выделяется одинарными кавычками.

Отличительной особенностью модели НАМ является то, что в отличие от моделей Тьюринга, вывод не обязан быть нормированным, то есть входные данные не должны оставаться в неизменном виде после завершения

программы. Это связано с организацией нормированного вывода и отсутствием перемещающейся линейно рабочей головки.

Изучение и применение алгоритмической системы Маркова открывает новые возможности для обработки и анализа текстовых данных. Предложенная модель позволяет эффективно осуществлять замену подслов в текстовых сообщениях, что имеет применение в таких областях, как обработка естественного языка, компьютерная лингвистика, автоматический перевод, искусственный интеллект и другие.

В целом, алгоритмическая система Маркова представляет собой значимый инструмент, способный обеспечить эффективное преобразование текстовых сообщений на основе замены подслов. Ее уникальные характеристики и принципы работы делают ее важным компонентом в области обработки текстовых данных и языковых анализов.

3.2 Постановка задачи

В рамках поставленной задачи №35 необходимо составить алгоритм двоичного подсчета числа гласных в слове латинского алфавита.

3.3 Идея решения задачи

Исходная строка содержит символы, включая гласные и согласные буквы, знаки препинания и другие символы. Однако согласные буквы не влияют на решение задачи и могут быть удалены.

После удаления согласных букв следует добавить символ-разделитель слева от очищенной строки, например знак "#". Затем поочередно удаляется каждая буква справа от ввода, при этом к двоичному числу слева прибавляется 1. Этот процесс продолжается до тех пор, пока буквы справа от ввода не закончатся, после чего удаляется решетка, обозначая завершение алгоритма.

3.4 Описание алгоритма

Сначала алгоритм удаляет все согласные буквы. Это предварительная обработка данных, необходимая для оптимизации работы алгоритма. Удаляются как строчные, так и заглавные буквы. Удаление согласных букв позволит сократить объем входных данных до минимально необходимого для работы алгоритма, исключая из рассмотрения символы, которые не оказывают влияния на решение поставленной задачи.

После удаления согласных букв следует добавить символ-разделитель слева от очищенной строки. Такой символ-разделитель, например, может быть обозначен знаком "#" и играет важную роль в контексте работы

алгоритма. Он отделяет входные данные, подлежащие обработке, от результата, полученного в результате работы алгоритма.

Далее поочередно удаляется каждая буква справа от символа-разделителя, при этом каждое удаление буквы соответствует прибавлению 1 к двоичному числу, находящемуся слева от решетки. Этот процесс продолжается до тех пор, пока буквы справа от разделителя не закончатся. В этот момент решетка удаляется, что символизирует завершение работы алгоритма.

Полный исходный код алгоритма с комментариями представлен в приложении Б.

3.5 Тестирование

В рамках тестирования алгоритма на вход программе были представлены несколько строк различного вида и длины, включая пустую строку. Полный список рассмотренных тестовых случаев представлен в таблице 5.

Таблица 5 – Список рассмотренных тестовых случаев

Входные данные	Выходные данные
	0
bb	0
aaababababb	110
uuuuuuuuuuuu	1100
AUeuAUeuuJsnnd	1001
BBBAAAAAAAUUUUUUUU	1111

Как и в двух предыдущих алгоритмах, сложность алгоритма оценивалась по количеству 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 6. Для тестирования были взяты строки, состоящие исключительно из гласных букв.

Таблица 6 – Зависимость входного сообщения и количества операций

Длина входного сообщения	Количество выполненных операций
1	3
2	5
4	9

8	19
16	41

Из анализа данных о связи между длиной входного сообщения и количеством операций можно сделать вывод о том, как меняется временная сложность алгоритма. При удвоении длины входных данных количество операций увеличивается более чем в два раза, указывая на временную сложность больше O(n*logn), но меньше O(2^n). Таким образом, сложность алгоритма примерно равна O(n^2).

3.6 Выводы по главе

Алгоритмическая модель Маркова была разработана и реализована в среде разработки для решения задачи двоичного подсчета числа гласных в слове латинского алфавита. Мой опыт в создании алгоритмов в различных алгоритмических моделях позволил мне успешно справиться с этой задачей.

В отличие от предыдущих моделей, нормальные алгоритмы Маркова имеют более сложную структуру и процесс реализации. Это связано с отсутствием строгого контроля над текущим состоянием исполнителя, не зависящим от фактического содержания обрабатываемых данных. Также важно отслеживать порядок выполнения правил в зависимости от их местоположения в исходном коде, что отличает данную модель от моделей Тьюринга.

Однако, решение данной задачи не вызвало значительных трудностей. Условия задачи оказались достаточно простыми, что позволило мне успешно реализовать алгоритм.

В начале разработки алгоритм был неоптимизированным и неэффективным, что представляло вызов для меня. Однако благодаря моему опыту и навыкам в области оптимизации алгоритмов, я смог сделать его более оптимальным и эффективным.

Мне потребовалось провести анализ и оптимизацию каждого шага алгоритма, чтобы достичь более быстрой и эффективной работы. Это требовало от меня глубоких знаний и опыта в работе с алгоритмами, а также умения мыслить логически и креативно. В итоге, благодаря моему опыту, я смог улучшить производительность алгоритма, сделав его более оптимальным и эффективным в решении поставленной задачи.

Таким образом, разработка и реализация алгоритмической модели Маркова для решения задачи двоичного подсчета числа гласных в слове латинского алфавита была успешной. Мой опыт в данной области и навыки

работы с различными алгоритмическими моделями позволили мне эффективно решить поставленную задачу.

Заключение

Во время выполнения данной курсовой работы были применены различные алгоритмические модели, такие как модель Тьюринга и модель Маркова, для иллюстрации определений алгоритма. Для этого были выполнены задания с использованием среды машины Тьюринга, диаграмм Тьюринга и нормальных алгоритмов Маркова. Каждая из этих моделей имела свои особенности.

Модель машины Тьюринга представляет собой низкоуровневую и строгую модель, которая требует нормированного ввода и строго контролирует каждое действие, совершаемое алгоритмом. Машина Тьюринга зависит от своего положения на ленте и значения символов, находящихся в этом месте. Код алгоритма для машины Тьюринга может быть достаточно громоздким и объемным из-за подробного описания каждого действия, но это обеспечивает высокую точность работы машины и более строгий контроль за процессом выполнения алгоритма.

Диаграммы Тьюринга являются графическим представлением машины Тьюринга. Они инкапсулируют рутинные операции в отдельные блоки, что упрощает разработку и отладку алгоритма. Однако, некоторые громоздкие интерфейсы интерпретатора и многочисленные стрелки, соединяющие блоки, могут нивелировать преимущества этой модели, затрудняя четкое понимание алгоритма.

Нормальные алгоритмы Маркова имеют совершенно другие особенности. В данной модели выполнение алгоритма зависит от содержания входного сообщения и расположения его знаков, а не от пространственного положения рабочей головки. Это усложняет разработку алгоритма, поскольку его поведение может изменяться в зависимости от содержания сообщения. Операции, связанные с пространственным расположением, такие как копирование слов, также усложняются в контексте нормальных алгоритмов Маркова. В то же время, в данной модели не требуется строгого контроля за нормированным выводом.

Таким образом, каждая из рассмотренных алгоритмических моделей имеет свои преимущества и недостатки, и их выбор зависит от конкретной задачи. Машина Тьюринга предоставляет высокий контроль над выполнением алгоритма, но может быть громоздкой в реализации. Диаграммы Тьюринга упрощают разработку и отладку, но могут

осложниться из-за сложного интерфейса. Нормальные алгоритмы Маркова ориентированы на содержание входного сообщения, что усложняет разработку и управление алгоритмом, но не требует строгого контроля за нормированным выводом.

ПРИЛОЖЕНИЕ А

Исходный код алгоритма решения задачи для машины Тьюринга

1. Подсчет количества цифр с начала.

```
// считаем количество цифр с начала (01101010 -> >01< 101 010 -> c2s)
c1i,0,<,c2i c1i,1,<,c2i c1i, ,>,c3s c1i,-,<,c1in
c2i,0,<,c3i c2i,1,<,c3i c2i, ,>,c1s c2i,-,<,c2in
c3i,0,<,c1i c3i,1,<,c1i c3i, ,>,c2s c3i,-,<,c3in

c1in,0,<,c2in c1in,1,<,c2in c1in, ,>,c3snp
c2in,0,<,c3in c2in,1,<,c3in c2in, ,>,c1snp
c3in,0,<,c1in c3in,1,<,c1in c3in, ,>,c2snp

c3snp,-,>,c3sn
c1snp,-,>,c1sn
c2snp,-,>,c2sn
```

2. Сбор тройки

```
// собираем двоичное число
// _e = конец
c1s,0,>,b000 c1s,1,>,001

c2s,0,>,c1s_00 c2s,1,>,c1s_01
c1s_00,0,>,b000 c1s_00,1,>,001 c1s_00, ,=,b000_e
c1s_01,0,>,010 c1s_01,1,>,011 c1s_01, ,=,010_e
c1s_10,0,>,100 c1s_10,1,>,101 c1s_10, ,=,100_e
c1s_11,0,>,110 c1s_11,1,>,111 c1s_11, ,=,110_e

c3s,0,>,c2s_0 c3s,1,>,c2s_1
c2s_0,0,>,c1s_00 c2s_0,1,>,c1s_01 c2s_0, ,=,b000_e
c2s_1,0,>,c1s_10 c2s_1,1,>,c1s_11 c2s_1, ,=,100_e

// ставим * для возвращения сюда
```

```
b000,0,*,b000_0 b000,1,*,b000_1 b000,.,=,b000_d b000, ,=,b000_e

001,0,*,001_0 001,1,*,001_1 001,.,=,001_d 001, ,=,001_e

010,0,*,010_0 010,1,*,010_1 010,.,=,010_d 010, ,=,010_e

011,0,*,011_0 011,1,*,011_1 011,.,=,011_d 011, ,=,011_e

100,0,*,100_0 100,1,*,100_1 100,.,=,100_d 100, ,=,100_e

101,0,*,101_0 101,1,*,101_1 101,.,=,101_d 101, ,=,101_e

110,0,*,110_0 110,1,*,110_1 110,.,=,110_d 110, ,=,110_e

111,0,*,111_0 111,1,*,111_1 111,.,=,111_d 111, ,=,111_e
```

3. Добавление соответствующей цифры справа.

```
// едем в конец, ставим цифру, возвращаемся - при * = 0

b000_0,0,>,b000_0 b000_0,1,>,b000_0 b000_0,.,>,b000_0 b000_0,*,>,b000_0 b000_0, ,>,b000_0_c

b000_0_c,0,>,b000_0_ca b000_0_c,1,>,b000_0_ca b000_0_c,2,>,b000_0_ca b000_0_c,3,>,b000_0_ca
b000_0_c,4,>,b000_0_ca b000_0_c,5,>,b000_0_ca b000_0_c,6,>,b000_0_ca b000_0_c,7,>,b000_0_ca
b000_0_c,.,>,b000_0_ca b000_0_c,-,>,b000_0_ca

b000_0_c, ,=,return_0



b000_0_ca,0,>,b000_0_ca b000_0_ca,1,>,b000_0_ca b000_0_ca,2,>,b000_0_ca
b000_0_ca,3,>,b000_0_ca b000_0_ca,4,>,b000_0_ca b000_0_ca,5,>,b000_0_ca
b000_0_ca,6,>,b000_0_ca b000_0_ca,7,>,b000_0_ca b000_0_ca,.,>,b000_0_ca b000_0_ca,-
,>,b000_0_ca

b000_0_ca, ,0,return_0



001_0,0,>,001_0 001_0,1,>,001_0 001_0,.,>,001_0 001_0,*,>,001_0 001_0, ,>,001_0_c

001_0_c,0,>,001_0_c 001_0_c,1,>,001_0_c 001_0_c,2,>,001_0_c 001_0_c,3,>,001_0_c
001_0_c,4,>,001_0_c 001_0_c,5,>,001_0_c 001_0_c,6,>,001_0_c 001_0_c,7,>,001_0_c
001_0_c,.,>,001_0_c 001_0_c,-,>,001_0_c

001_0_c, ,1,return_0

010_0,0,>,010_0 010_0,1,>,010_0 010_0,.,>,010_0 010_0,*,>,010_0 010_0, ,>,010_0_c

010_0_c,0,>,010_0_c 010_0_c,1,>,010_0_c 010_0_c,2,>,010_0_c 010_0_c,3,>,010_0_c
010_0_c,4,>,010_0_c 010_0_c,5,>,010_0_c 010_0_c,6,>,010_0_c 010_0_c,7,>,010_0_c
010_0_c,.,>,010_0_c 010_0_c,-,>,010_0_c

010_0_c, ,2,return_0

011_0,0,>,011_0 011_0,1,>,011_0 011_0,.,>,011_0 011_0,*,>,011_0 011_0, ,>,011_0_c

011_0_c,0,>,011_0_c 011_0_c,1,>,011_0_c 011_0_c,2,>,011_0_c 011_0_c,3,>,011_0_c
011_0_c,4,>,011_0_c 011_0_c,5,>,011_0_c 011_0_c,6,>,011_0_c 011_0_c,7,>,011_0_c
011_0_c,.,>,011_0_c 011_0_c,-,>,011_0_c

011_0_c, ,3,return_0
```

```
100_0,0,>,100_0 100_0,1,>,100_0 100_0,.,>,100_0 100_0,*,>,100_0 100_0, ,>,100_0_c

100_0_c,0,>,100_0_c 100_0_c,1,>,100_0_c 100_0_c,2,>,100_0_c 100_0_c,3,>,100_0_c
100_0_c,4,>,100_0_c 100_0_c,5,>,100_0_c 100_0_c,6,>,100_0_c 100_0_c,7,>,100_0_c
100_0_c,.,>,100_0_c 100_0_c,-,>,100_0_c

100_0_c, ,4,return_0

101_0,0,>,101_0 101_0,1,>,101_0 101_0,.,>,101_0 101_0,*,>,101_0 101_0, ,>,101_0_c

101_0_c,0,>,101_0_c 101_0_c,1,>,101_0_c 101_0_c,2,>,101_0_c 101_0_c,3,>,101_0_c
101_0_c,4,>,101_0_c 101_0_c,5,>,101_0_c 101_0_c,6,>,101_0_c 101_0_c,7,>,101_0_c
101_0_c,.,>,101_0_c 101_0_c,-,>,101_0_c

101_0_c, ,5,return_0

110_0,0,>,110_0 110_0,1,>,110_0 110_0,.,>,110_0 110_0,*,>,110_0 110_0, ,>,110_0_c

110_0_c,0,>,110_0_c 110_0_c,1,>,110_0_c 110_0_c,2,>,110_0_c 110_0_c,3,>,110_0_c
110_0_c,4,>,110_0_c 110_0_c,5,>,110_0_c 110_0_c,6,>,110_0_c 110_0_c,7,>,110_0_c
110_0_c,.,>,110_0_c 110_0_c,-,>,110_0_c

110_0_c, ,6,return_0

111_0,0,>,111_0 111_0,1,>,111_0 111_0,.,>,111_0 111_0,*,>,111_0 111_0, ,>,111_0_c

111_0_c,0,>,111_0_c 111_0_c,1,>,111_0_c 111_0_c,2,>,111_0_c 111_0_c,3,>,111_0_c
111_0_c,4,>,111_0_c 111_0_c,5,>,111_0_c 111_0_c,6,>,111_0_c 111_0_c,7,>,111_0_c
111_0_c,.,>,111_0_c 111_0_c,-,>,111_0_c

111_0_c, ,7,return_0
```

```
// едем в конец, ставим цифру, возвращаемся - при * = 1

b000_1,0,>,b000_1 b000_1,1,>,b000_1 b000_1,.,>,b000_1 b000_1,*,>,b000_1 b000_1, ,>,b000_1_c

b000_1_c,0,>,b000_1_ca b000_1_c,1,>,b000_1_ca b000_1_c,2,>,b000_1_ca b000_1_c,3,>,b000_1_ca
b000_1_c,4,>,b000_1_ca b000_1_c,5,>,b000_1_ca b000_1_c,6,>,b000_1_ca b000_1_c,7,>,b000_1_ca
b000_1_c,.,>,b000_1_ca b000_1_c,-,>,b000_1_ca

b000_1_c, ,=,return_1



b000_1_ca,0,>,b000_1_ca b000_1_ca,1,>,b000_1_ca b000_1_ca,2,>,b000_1_ca
b000_1_ca,3,>,b000_1_ca b000_1_ca,4,>,b000_1_ca b000_1_ca,5,>,b000_1_ca
b000_1_ca,6,>,b000_1_ca b000_1_ca,7,>,b000_1_ca b000_1_ca,.,>,b000_1_ca b000_1_ca,-
,>,b000_1_ca

b000_1_ca, ,0,return_1



001_1,0,>,001_1 001_1,1,>,001_1 001_1,.,>,001_1 001_1,*,>,001_1 001_1, ,>,001_1_c

001_1_c,0,>,001_1_c 001_1_c,1,>,001_1_c 001_1_c,2,>,001_1_c 001_1_c,3,>,001_1_c
001_1_c,4,>,001_1_c 001_1_c,5,>,001_1_c 001_1_c,6,>,001_1_c 001_1_c,7,>,001_1_c
001_1_c,.,>,001_1_c 001_1_c,-,>,001_1_c

001_1_c, ,1,return_1
```

```
010_1,0,>,010_1 010_1,1,>,010_1 010_1,.,>,010_1 010_1,*,>,010_1 010_1, ,>,010_1_c

010_1_c,0,>,010_1_c 010_1_c,1,>,010_1_c 010_1_c,2,>,010_1_c 010_1_c,3,>,010_1_c
010_1_c,4,>,010_1_c 010_1_c,5,>,010_1_c 010_1_c,6,>,010_1_c 010_1_c,7,>,010_1_c
010_1_c,.,>,010_1_c 010_1_c,-,>,010_1_c

010_1_c, ,2,return_1

011_1,0,>,011_1 011_1,1,>,011_1 011_1,.,>,011_1 011_1,*,>,011_1 011_1, ,>,011_1_c

011_1_c,0,>,011_1_c 011_1_c,1,>,011_1_c 011_1_c,2,>,011_1_c 011_1_c,3,>,011_1_c
011_1_c,4,>,011_1_c 011_1_c,5,>,011_1_c 011_1_c,6,>,011_1_c 011_1_c,7,>,011_1_c
011_1_c,.,>,011_1_c 011_1_c,-,>,011_1_c

011_1_c, ,3,return_1

100_1,0,>,100_1 100_1,1,>,100_1 100_1,.,>,100_1 100_1,*,>,100_1 100_1, ,>,100_1_c

100_1_c,0,>,100_1_c 100_1_c,1,>,100_1_c 100_1_c,2,>,100_1_c 100_1_c,3,>,100_1_c
100_1_c,4,>,100_1_c 100_1_c,5,>,100_1_c 100_1_c,6,>,100_1_c 100_1_c,7,>,100_1_c
100_1_c,.,>,100_1_c 100_1_c,-,>,100_1_c

100_1_c, ,4,return_1

101_1,0,>,101_1 101_1,1,>,101_1 101_1,.,>,101_1 101_1,*,>,101_1 101_1, ,>,101_1_c

101_1_c,0,>,101_1_c 101_1_c,1,>,101_1_c 101_1_c,2,>,101_1_c 101_1_c,3,>,101_1_c
101_1_c,4,>,101_1_c 101_1_c,5,>,101_1_c 101_1_c,6,>,101_1_c 101_1_c,7,>,101_1_c
101_1_c,.,>,101_1_c 101_1_c,-,>,101_1_c

101_1_c, ,5,return_1

110_1,0,>,110_1 110_1,1,>,110_1 110_1,.,>,110_1 110_1,*,>,110_1 110_1, ,>,110_1_c

110_1_c,0,>,110_1_c 110_1_c,1,>,110_1_c 110_1_c,2,>,110_1_c 110_1_c,3,>,110_1_c
110_1_c,4,>,110_1_c 110_1_c,5,>,110_1_c 110_1_c,6,>,110_1_c 110_1_c,7,>,110_1_c
110_1_c,.,>,110_1_c 110_1_c,-,>,110_1_c

110_1_c, ,6,return_1

111_1,0,>,111_1 111_1,1,>,111_1 111_1,.,>,111_1 111_1,*,>,111_1 111_1, ,>,111_1_c

111_1_c,0,>,111_1_c 111_1_c,1,>,111_1_c 111_1_c,2,>,111_1_c 111_1_c,3,>,111_1_c
111_1_c,4,>,111_1_c 111_1_c,5,>,111_1_c 111_1_c,6,>,111_1_c 111_1_c,7,>,111_1_c
111_1_c,.,>,111_1_c 111_1_c,-,>,111_1_c

111_1_c, ,7,return_1
```

4. Возврат к знаку. Завершение алгоритма.

```
// блок циклов возврата

return_0,-,<,return_0 return_0, ,<,return_0 return_0,0,<,return_0 return_0,1,<,return_0
return_0,2,<,return_0 return_0,3,<,return_0 return_0,4,<,return_0 return_0,5,<,return_0
return_0,6,<,return_0 return_0,7,<,return_0 return_0,.,<,return_0 return_0,*,0,c3s
```

```
return_1,-,<,return_1 return_1, ,<,return_1 return_1,0,<,return_1 return_1,1,<,return_1
return_1,2,<,return_1 return_1,3,<,return_1 return_1,4,<,return_1 return_1,5,<,return_1
return_1,6,<,return_1 return_1,7,<,return_1 return_1,.,<,return_1 return_1,*,1,c3s


return_d,-,<,return_d return_d, ,<,return_d return_d,0,<,return_d return_d,1,<,return_d
return_d,2,<,return_d return_d,3,<,return_d return_d,4,<,return_d return_d,5,<,return_d
return_d,6,<,return_d return_d,7,<,return_d  return_d,.,>,c3s


return_d0,.,<,return_d


// ставим курсор после результата, выходим

end,0,>,end end,1,>,end end,2,>,end end,3,>,end end,4,>,end end,5,>,end end,6,>,end
end,7,>,end

end, ,#,end
```

ПРИЛОЖЕНИЕ Б

Исходный код алгоритма решения задачи для алгоритма Маркова

1. Удаление согласных букв.

```
B->
b->
C->
c->
D->
d->
F->
f->
G->
g->
H->
h->
J->
j->
K->
k->
L->
l->
M->
m->
N->
n->
P->
p->
Q->
q->
R->
r->
S->
s->
```

```
T->
t->
V->
v->
W->
w->
X->
x->
Y->
y->
Z->
z->
```

2. Подсчет количества гласных.

```
0!0->!10
1!0->!00
1!->!1
!1->1
!->1


0#A->1#
0#a->1#
0#E->1#
0#e->1#
0#I->1#
0#i->1#
0#O->1#
0#o->1#
0#U->1#
0#u->1#


1#A->!0#
1#a->!0#
1#E->!0#
```

```
1#e->!0#
1#I->!0#
1#i->!0#
1#O->!0#
1#o->!0#
1#U->!0#
1#u->!0#

#A->1#
#a->1#
#E->1#
#e->1#
#I->1#
#i->1#
#O->1#
#o->1#
#U->1#
#u->1#
```

3. Завершение алгоритма.

```
1#->.1
0#->.0
#->.0
->#
```