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»

Выполнила:

Студентка группы М8О-108Б-23

Федорова Екатерина Васильевна

Проверил:

Преподаватель каф. 806

Севастьянов Виктор Сергеевич

Москва 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

«Алгоритмические модели Тьюринга и Маркова» – звучит загадочно для человека, не знакомого с темой данной курсовой работы. Что это такое и «с чем его едят»? Чем отличаются друг от друга и где могут быть применимы? Стоит основательно разобраться в данном вопросе, отдав честь тому, что изменило всё человечество.

Всякое дело начинается с теории: как не построить дом без физической базы, так и без знаний в области информатики не понять глубины языков программирования. И это не просто «красное словцо», ведь на рассматриваемых моделях и правда зиждутся языки, и незнакомые, и привычные для обывателя. Но обо всём по порядку.

Для начала стоит понять, что такое «алгоритм». Не зная предмета, я бы ответила, что это – «строгая последовательность действий (команд)», в чём-то даже оказавшись близка к истине. Всё потому, что существует два определения алгоритмов: формальное и неформальное.

Неформальное определение гласит: «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; то есть все исполнители (люди и автоматы) должны понимать и выполнять задачи одинаково». Однако у данного определения есть ряд существенных минусов, главный из которых – расплывчатость определения. Что означает «понимать и выполнять задачи одинаково»? Простой пример: вычисление замечательного предела. Те, кто знаком с математическим анализом, запросто это сделают, но если человек или машина не обучены тому и не знакомы с пределами вовсе? И является ли процедура завязывания шнурков алгоритмом?

Если бы все поставленные математические задачи могли быть алгоритмически решены, можно было бы не уточнять понятие алгоритма: когда для решения какого-нибудь класса задач предлагался конкретный алгоритм, возникало соглашение считать указанный алгоритм действительно алгоритмом. Но, к сожалению, не все математические задачи алгоритмически разрешимы, а доказательство алгоритмической неразрешимости какого-либо класса задач (т.е., того, что не существует алгоритма решения всех подобных задач) неизбежно содержит высказывания обо всех мыслимых алгоритмах. Такие высказывания невозможны без четкого представления о том, что является алгоритмом и что им не является, т.е. они невозможны без строгого формального определения алгоритма.

Итак, стало понятно, что для достижения целей предмета фундаментальной информатики требуется точное определение понятия алгоритма. Формализация понятия алгоритма реализуется с помощью построения алгоритмических моделей. Выделяют три основных типа универсальных алгоритмических моделей: рекурсивные функции, машины Тьюринга и нормальные алгоритмы Маркова. Рекурсивный алгоритм – это алгоритм, решающий задачу путем решения одного или нескольких более узких вариантов той же задачи. Функция называется рекурсивной, если в ее определении содержится вызов этой же функции. Машина Тьюринга – алгоритм, который представляется как описание

процесса работы некоторой машины, способной выполнять лишь небольшое число весьма простых операций. Нормальные алгоритмы Маркова – алгоритмы, которые описываются как преобразования слов в произвольных алфавитах. Весьма интересной представляется также выбор в качестве основной алгоритмической системы клеточных автоматов, однако в рамках данной курсовой работы будут рассматриваться лишь Машины Тьюринга и их графическое представление в виде диаграмм Тьюринга и Нормальные алгоритмы Маркова для решения поставленных задач.

Цель работы: демонстрация определения алгоритма с помощью построения алгоритмических моделей Тьюринга и Маркова.

Задачи:

- изучить выбранные алгоритмы;
- составить программу машины Тьюринга в четвёрках, выполняющую заданное действие над словами, записанными на ленте;
- разработать диаграмму Тьюринга решения задачи с использованием стандартных машин (r, l, R, L, K, a) и вспомогательных машин, определяемых задачей;
- разработать нормальный алгоритм Маркова.

Для этого будут разобраны некоторые лабораторные работы с соответствующими заданиями.

ГЛАВА I. МАШИНА ТЬЮРИНГА

1. Теоретические сведения

1.1. Историческая справка

В 1936 г. аспирант Алан Тьюринг при исследовании алгоритмических проблем разрешимости (одной из знаменитых проблем Гильберта) предложил для уточнения понятия алгоритма использовать абстрактную вычислительную машину с очень простым набором операций. В 1942 г. группа британских математиков и филологов под руководством А. Тьюринга успешно применила теорию алгоритмов к разгадке шифров фашистской Германии. Построенная на базе этой машины алгоритмическая теория Тьюринга оказалась настолько плодотворной, что предвосхитила все последующие идеи универсальных вычислительных машин с программным управлением, включая идеи автоматизации программирования. В знак признания этих заслуг именем Тьюринга названа высшая награда ACM, ежегодно присуждаемая за наиболее выдающиеся результаты в области информатики. Недавно вышедшая книга Шарля Петцольда, посвящённая разбору основополагающего труда Тьюринга, ещё раз подтверждает актуальность Тьюринговской теории алгоритмов.

1.2. Неформальное описание

Машина Тьюринга (далее – МТ) состоит из:

- бесконечной ленты, разделённой на ячейки;
- комбинированной читающей и пишущей головки, которая может перемещаться от ячейки к ячейке;
- дискретных состояний головки, среди которых выделено одно начальное состояние.

В каждой ячейке ленты может быть записан один знак алфавита A, называемого рабочим алфавитом МТ, либо 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 (букву алфавита A или λ). При этом головка находится в одном из конечного множества дискретных состояний. В зависимости от состояния, в котором находится головка, и от буквы, записанной в рабочей ячейке, МТ выполняет одну из команд, составляющих ее программу.

Выполнение команды состоит в выполнении элементарного действия, предписываемого этой командой, и переводе головки в новое состояние (которое, в частности, может совпадать со старым). Определено три вида элементарных действий:

- сдвиг головки на одну ячейку влево (если считать, что лента МТ ограничена слева, то для крайней левой ячейки сдвиг влево не определен);
- сдвиг головки на одну ячейку вправо;

• запись в рабочую ячейку какой-либо буквы рабочего алфавита A либо пробел (при этом буква, которая была записана в рабочей ячейке до выполнения записи, стирается).

Перед началом работы МТ на ее ленту записывается исходное сообщение так, что в каждую ячейку ленты записывается одна буква сообщения, либо пробел. Любое сообщение занимает конечное число ячеек ленты. При этом ячейки, расположенные справа от последней буквы исходного сообщения, заполняются пробелами и считаются пустыми. В начале работы МТ ее головка приводится в начальное состояние и помещается над начальной рабочей ячейкой, которая определенным и фиксированным для каждой конкретной МТ образом расположена относительно исходного сообщения. Обычно в качестве рабочей ячейки берут те, которая расположена вслед за исходным сообщением, то есть ячейку, содержащую символ пробела справа от последней буквы сообщения.

Таким образом, работа МТ полностью определяется ее программой и сообщением, которое было записано на ленте перед началом работы МТ. Ниже (рис. 1, 2 и 3) представлены изображения машин Тьюринга.

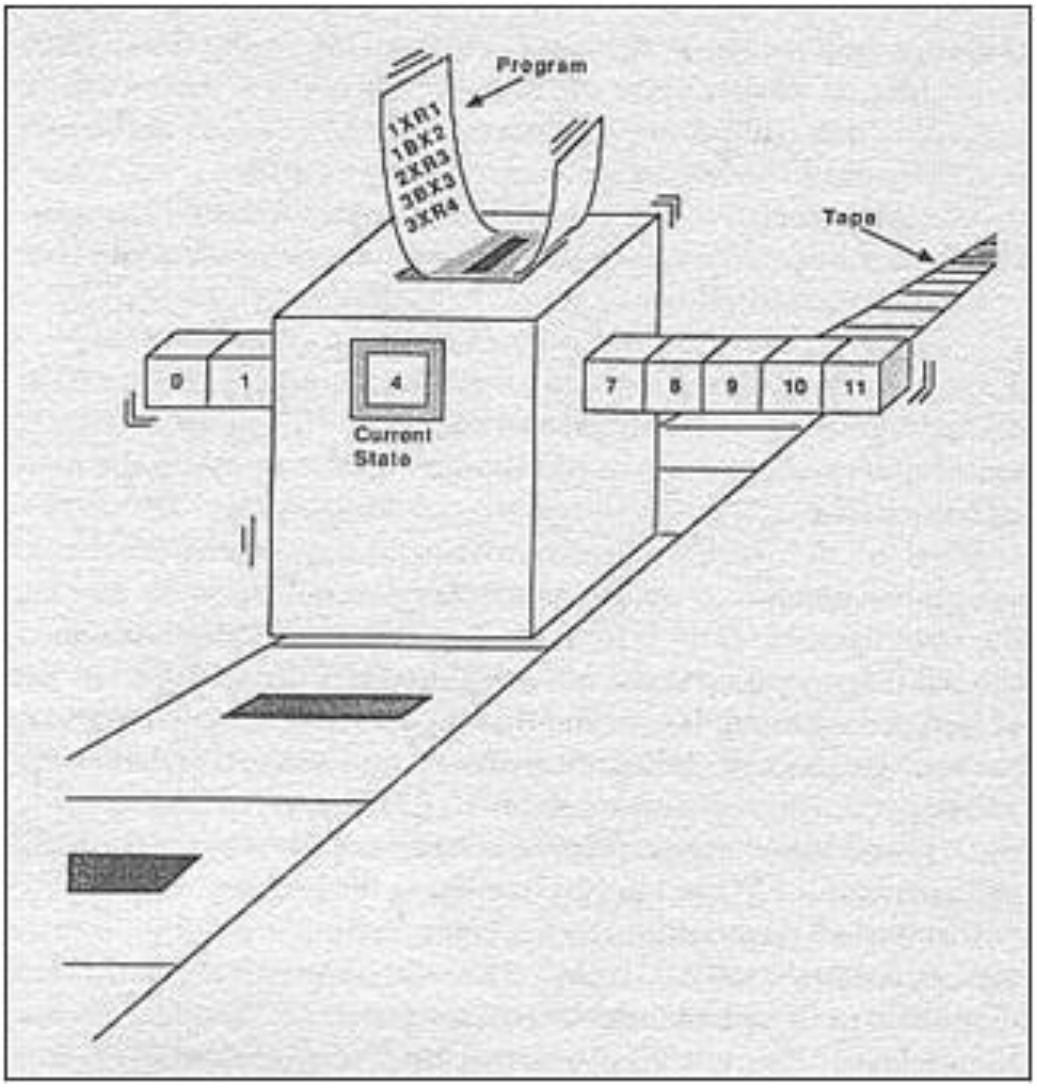

Рисунок 1: Машина Тьюринга наглядно

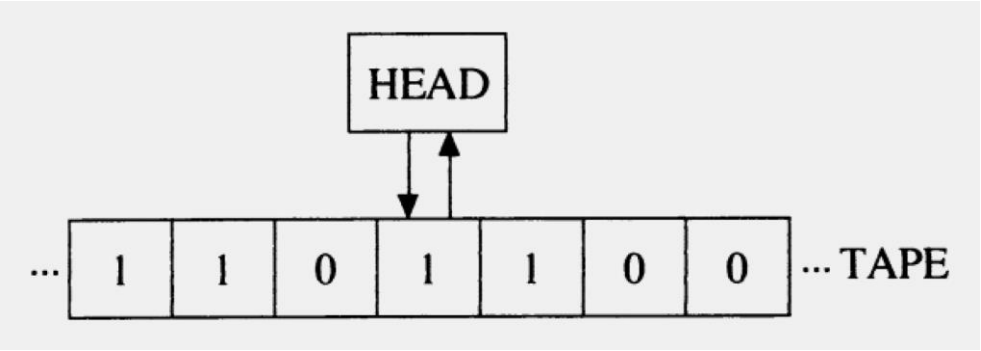

Рисунок 1: Машина Тьюринга

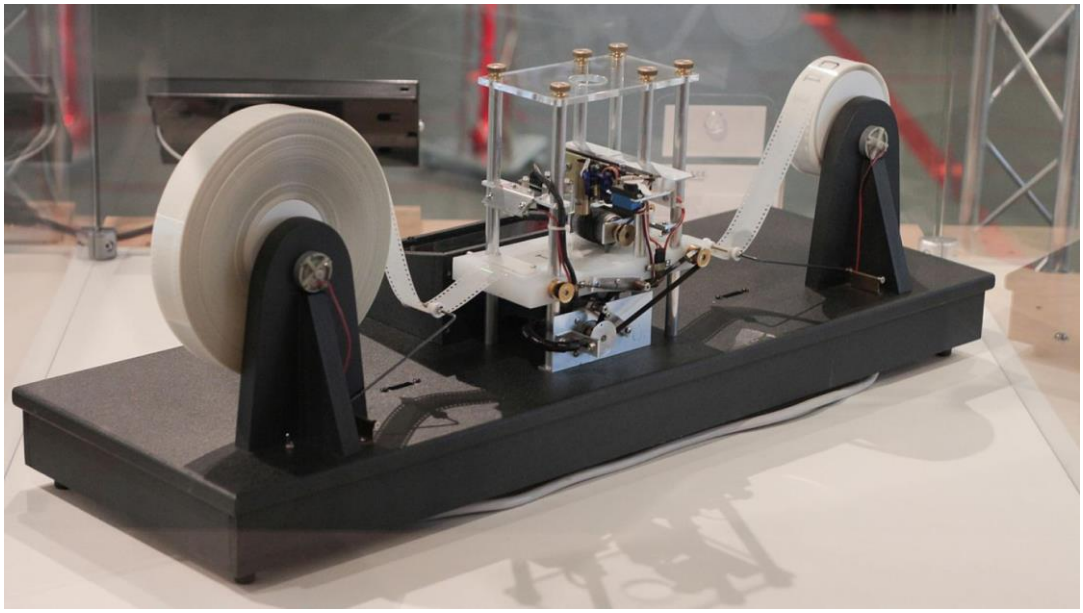

Рисунок 3: Машина Тьюринга

1.3. Строгое математическое описание

Машина Тьюринга – упорядоченная четверка объектов T = (A,Q, P, q_0), где

- T – символ МТ,
- A – конечное множество букв (рабочий алфавит),
- Q – конечное множество символов (имен состояний),
- q_0 – имя начального состояния,
- P – множество упорядоченных четверок (q, a, v, q'), q, q' ∈ Q, a ∈ A∪{λ}, и v ∈ {l, r}∪A∪{λ} (программа), определяющее три функции: функцию выхода F_l: Q * A_λ → A_λ (A_λ = A∪{λ}), функцию переходов F_t: Q * A_λ → Q, и функцию движения головки F_u: Q * A_λ → {1, r, s} (символ s означает, что головка неподвижна).

Состояние (ситуация) ленты МТ – 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. 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.

$$S = [a_{i1}a_{i2}\ldots a_{ik-1}(a_{ik})a_{ik+1}\ldots a_{in}\lambda>.$$

Конфигурация – упорядоченная пара объектов C = (S, q), где S – текущая ситуация, q – текущее состояние головки.

$$C = [a_{i1}a_{i2}\ldots a_{ik-1}(q, a_{ik})a_{ik+1}\ldots a_{in}\lambda>.$$

Каждая конфигурация C однозначно определяет команду МТ, которая должна быть выполнена. После выполнения указанной команды получается новая конфигурация C", про которую говорят, что она непосредственно следует за конфигурацией C; обозначение C → C", где → символ бинарного отношения непосредственного следования. Отношение непосредственного следования выполняется и для ситуаций S и S', соответствующих конфигурациям C и C"; S → S'. Ситуация (конфигурация), соответствующая начальному состоянию МТ, называют начальной ситуацией (конфигурацией).

Историческое дополнение. 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. На данный момент предпочтительнее работать именно в четвёрках.

2. Практическая работа

Цель: составить программу машины Тьюринга в четвёрках, выполняющую заданное действие над словами, записанными на ленте. Отладку и тестирование проводить в среде интерактивного действующего макета jstu4. В начальном состоянии головка МТ находится на пустой ячейке непосредственно справа от записанных на ленте аргументов – слов входного сообщения. В конечном состоянии головка МТ должна находиться на пустой ячейке непосредственно справа от результата (последнего преобразованного или вновь сгенерированного слова результирующего сообщения). Вычисления в программе, как правило, должны быть нормированными (аргументы после работы программы сохраняются на ленте в неизменном виде и не остаётся промежуточных результатов).

Задача: запрограммировать машину Тьюринга для перевода числа из девятеричной системы счисления (далее – 9СС) в троичную систему счисления (далее – 3СС).

Идея, метод, алгоритм решения задачи: копирование числа с «мгновенным переводом»: одна цифра исходного числа записывается парой, при этом игнорируя незначимые нули.

Сценарий выполнения работы:

Как и указано в цели, отладка и тестирование будут проводиться в среде интерактивного действующего макета jstu4 (рис. 4), реализованный в формате web-страницы, принимающий на ввод текст программы и входное сообщение и выдающий на выходе преобразованное сообщение.

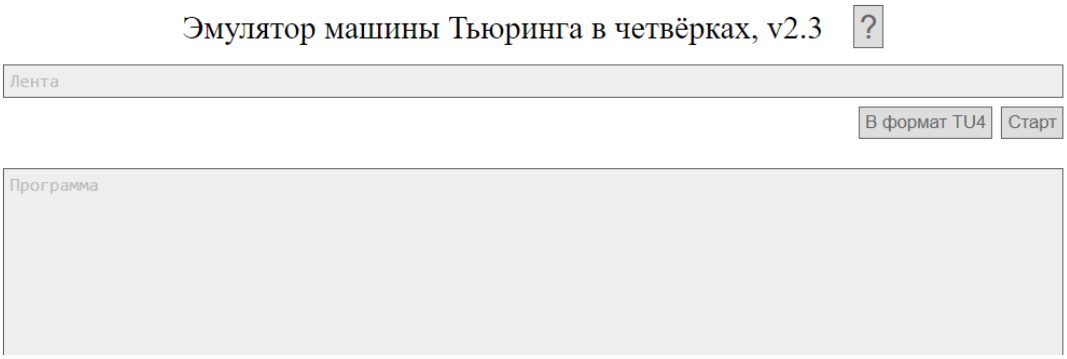

Рисунок 4: Эмулятор МТ

Интерфейс интерпретатора также допускает использование комментариев.

В дальнейшем под «запоминанием», «записью» или «стиранием» 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.

Решение задачи могло быть вполне очевидным: для соблюдения нормированности вычислений – скопировать число, после чего произвести над ним манипуляции перевода. Для этого пришлось бы проходить вспомогательный этап с переводом в десятеричную систему счисления (далее – 10СС). Однако во имя оптимизации будем сразу переводить число из 9СС в 3СС, минуя вспомогательный перевод в 10СС, к тому же исключим этап копирования исходного числа. Для того чтобы данную идею было возможно воплотить программным описанием, каждую цифру 9СС представим парой цифр 3СС следующим образом:

0 в 9СС = 00 в 3СС;

1 в 9СС = 01 в 3СС;

2 в 9СС = 02 в 3СС;

...

7 в 9СС = 21 в 3СС;

8 в 9СС = 22 в 3СС.

Это обеспечивается тем, что $9 = 3^2$.

Приступим к программированию.

Отметим основные простые состояния:

Название состояние	Описание
00	Начальное состояние головки
run	Состояние, позволяющее головке МТ переместиться в начало слова
read	Состояние считывания цифры с ленты для дальнейшего его перевода
runreverse	Состояние, позволяющее головке МТ переместиться в конец числа, записанного на ленте в 3СС, после чего машина завершает свою работу

В самом начале работы головка должна «пробежать» (то есть переместиться в начало или конец числа, остановившись на пробеле – направление зависит от контекста) по исходному слову и вернуться на одну ячейку вправо для считывания первой цифры числа, стерев её (то есть заменив на символ пробела), но запомнив (то есть перейдя в такое состояние, при котором в последующем МТ сможет записать на очищенную ячейку обратно цифру, восстановив данные). После этого головка должна пробежать по числу обратно и, оставив одну пустую ячейку-разделитель, записать уже переведённую в 3СС цифру, занимающую две ячейки. Далее – возвращение в начало первого числа, восстановление данных и переход на соседнюю правую. Алгоритм повторяется до тех пор, пока МТ не обработает всё входное число, после чего её головка перемещается в конец выходного числа. Работа завершается.

Составные состояния:

Начало	Конец	Описание состояния
zero **one** **two** **three** **four** **five** **six** **seven** **eight**	begin	Состояние сразу после прочтения цифры. Начинает её обработку.
	run	Перемещение по исходному слову вправо до его конца
		Перемещение по выходному слову до его конца вправо
	write	Запись цифры, переведённой в 3СС
	reverse	Перемещение по выходному слову до его начала влево
	back	Перемещение по исходному слову до первого пробела
	e	Запись некогда стёртого числа, восстановление даннх

Далее проведём тестирование и отладку. После них добавим алгоритм игнорирования незначащих нулей, чтобы из 10 получалось 100, а не 0100, при этом учитывая особые случаи входного слова типа 0 и 00. Для этого придётся увеличить количество состояний, добавив новые.

Название состояния	Описание
zeroif	Состояние, проверяющее, являются ли первые цифры входного числа нулями. Если да, головка перемещается вправо до первой цифры, отличной от нуля
zerowriteif	Состояние, в которое переходит головка МТ, если входное число состояло только из одних нулей. Тогда на ленту записывается справа 1 ноль, работа МТ завершается.
oneif **twoif**	Состояния, проверяющие, является ли цифра 1 или 2 (в ЗСС соответственно 01 и 02) началом числа. Если да, то на ленту выписывается только 1 или 2 соответственно, не создавая незначимых нулей

Получившуюся программу можно изучить в приложении Б.

Тестирование:

Входные данные	Выходные данные	Описание
0	0 0	Перевод нуля
00	00 0	Перевод нуля с игнорированием незначащих нулей
001	001 1	Перевод числа с игнорированием незначащих нулей
123456780	123456780 10210111220212200	Перевод произвольного числа

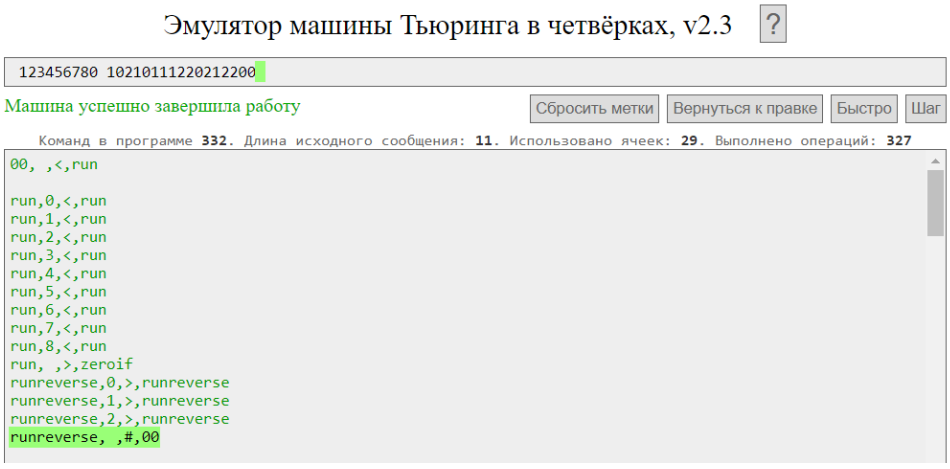

Рисунок 5: Пример запуска МТ

3. Вывод

В ходе выполнения работы разработан алгоритм перевода числа из 9СС в 3СС с оптимизацией и игнорированием незначащих нулей, составлена соответствующая программа Машины Тьюринга в четвёрках. Получен практический опыт работы с абстрактным исполнителем Машина Тьюринга и опыт составления алгоритмов на языке низкого уровня, при котором необходимо многие элементарные действия задавать большим количеством команд. В связи с этим требуется «бережное» отношение к названию состояний головки.

Плюсы Машин Тьюринга:

1. Универсальность: Машина Тьюринга способна эмулировать работу любой другой вычислительной модели, если заданы правильные правила. Это позволяет ей быть универсальным средством моделирования и анализа различных вычислительных процессов.

2. Простота: Модель Машины Тьюринга достаточно проста и понятна для человека. Её работа основывается на наборе простых правил, которые можно легко понять и реализовать.

3. Точность и формальность: Машина Тьюринга основывается на строгих математических основах и формальных правилах, что позволяет точно и детально описывать алгоритмы и вычисления.

Минусы Машины Тьюринга:

1. Теоретическость: Машина Тьюринга является теоретической моделью, и её функционирование не всегда отражает реальное поведение реальных вычислительных систем. Реализация некоторых алгоритмов на реальном компьютере может отличаться от их описания на Машине Тьюринга.

2. Ограниченность: Машина Тьюринга обладает определенными ограничениями в вычислительной мощности. Например, она может быть неэффективна для решения некоторых задач, требующих большой вычислительной мощности или параллельных вычислений.

3. Абстрактность: Машина Тьюринга представляет абстрактный уровень моделирования, что может затруднять понимание и реализацию конкретных алгоритмов на практике. Её формальность и абстрактность могут создавать проблемы при переводе алгоритмов из теории в реальность.

Несомненно, Машина Тьюринга является мощным инструментом для изучения и анализа алгоритмов и вычислительных процессов. Однако следует всегда помнить обо всех её преимуществах и недостатках, чтобы применять на практике с умом.

ГЛАВА II. ДИАГРАММЫ МАШИНЫ ТЬЮРИНГА

1. Теоретические сведения

1.1. Историческая справка

Учитывая перечисленные недостатки Машин Тьюринга, имеет место попробовать от них или хотя бы от их части избавиться путём представлений МТ иным образом.

Как было выяснено на практике, описание Машины Тьюринга в текстовой форме довольно громоздко и неудобно для анализа, поэтому Раймонд Смаллиан, американский математик, предложил в 1951 году использовать диаграммы для представления Машины Тьюринга.

Раймонд Смаллиан разработал нотацию для диаграмм Тьюринга, которая включала в себя круги или овалы для представления состояний и стрелки для обозначения переходов между состояниями в зависимости от символов на ленте. Он также предложил использовать различные символы и стрелки для отображения направления движения на ленте.

Идея диаграмм Тьюринга была быстро принята и стала популярным способом представления Машины Тьюринга.

1.2. Определение

Диаграммы Тьюринга – это графическое представление машины Тьюринга, которое позволяет избавиться от нагромождения состояний. Фактически, диаграммы Тьюринга представляют одни МТ через другие, более простые МТ визуально-топологическим способом. Этот способ так же строг и полон, как и "обычные" МТ. Например, машина, копирующая слово на ленте, может быть представлена через МТ, ищущую начало и конец слова на ленте, выполняющую копирование букв слова и т.д. Более простые МТ, в свою очередь, могут быть представлены еще более простыми МТ, и так далее.

Такой процесс представления МТ через более простые МТ должен обязательно прекратиться, сводя описание каждой МТ к элементарным действиям, определенным при описании МТ. Результатом будет представление МТ через элементарные МТ, которые выполняют только одно элементарное действие и останавливаются.

Диаграмма Тьюринга состоит из:

- символов (имён) машин и подмашин Тьюринга;
- точек, обозначающих состояние МТ;
- стрелок, соединяющих состояние, над которыми написаны знаки рабочего алфавита.

Состояния в диаграммах обозначаются точками, ограничивающими символ каждой подмашины в диаграмме слева и справа точками, обозначающими имя текущего состояния и состояния, в которое переходит машина. Переход из одного состояния в другое обозначается стрелками, соединяющими правую точку начального состояния и

левую точку конечного. Если выполнение действия подразумевается для всех знаков рабочего алфавита, то над стрелкой ничего не указывается, в противном случае над стрелкой размещают алфавит или «условие».

2. Практическая работа

Цель: разработать диаграмму Тьюринга решения задачи в среде интерпретатора jdt или VisualTuring 2.0 с использованием стандартных машин (r, l, R, L, K, λ) и вспомогательных машин, определяемых поставленной задачей. Алфавит диаграммы определяется заданием. В начальном состоянии голова МТ, определяемой диаграммой, находится на пустой ячейке непосредственно справа от записанных на ленте аргументов. В конечном состоянии головка МТ должна находиться на пустой ячейке непосредственно справа от результата (последнего преобразованного или вновь сформированного слова). Определяемые заданием вычисления должны быть нормированными (аргументов после работы программы сохраняются на ленте в неизменном виде, не остаётся промежуточных результатов).

Задача: реверс девятеричного числа без знака (запись цифр в обратном порядке) с избавлением от незначащих нулей.

Идея, метод, алгоритм решения задачи: идея проста: поциферное копирование числа, начиная с его конца. Таким образом сразу обеспечивается его реверс.

Сценарий выполнения работы:

В начале программы головка МТ сдвигается влево и проверяет, являются ли цифры последнего разряда нулями. Если да, то их нужно проигнорировать и не копировать, так как при реверсе числа они станут незначимы. Если при данных сдвигах головка достигнет конца (это означает, что всё число состоит из нулей), то справа, разделив исходное слово-число и новую запись пробелом, будет записано выходное число нуль, программа завершится полностью. Если же встретится иная цифра, то МТ перейдёт к выполнению основной программы.

Для каждой цифры придётся создать собственную подмашину (не элементарную диаграмму) копирования: считываемая цифра меняется на пробел, головка пробегает в конец выходного слова и дописывает скопированную цифру, после чего возвращаетс на пробел и восстанавливает данные. Дойдя до конца числа, головка перемещается на финишную позицию, машина заканчивает работу.

«Главная» диаграмма Тьюринга:

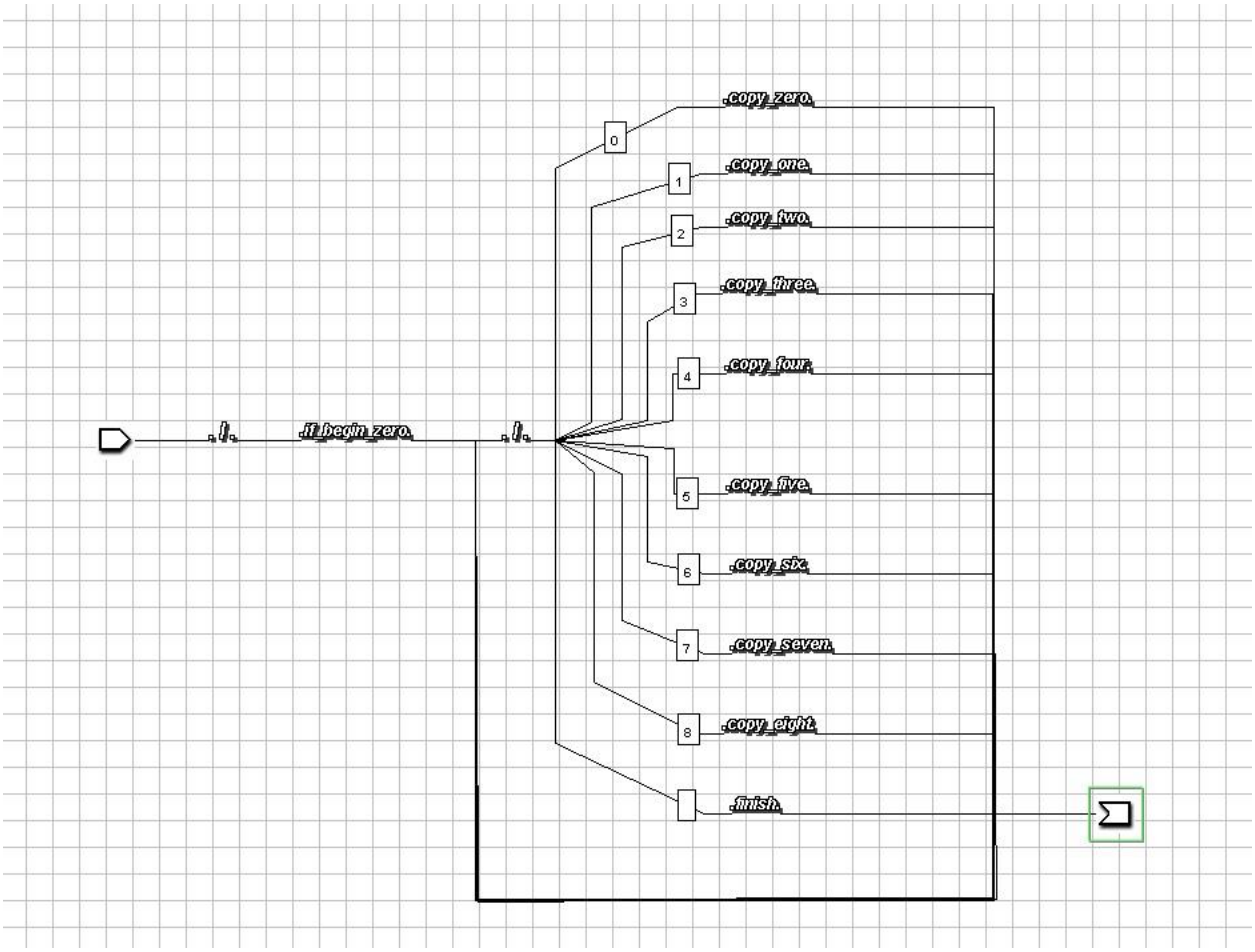

Рисунок 6

Для первичной проверки нулей используется подмашина if_begin_zero.

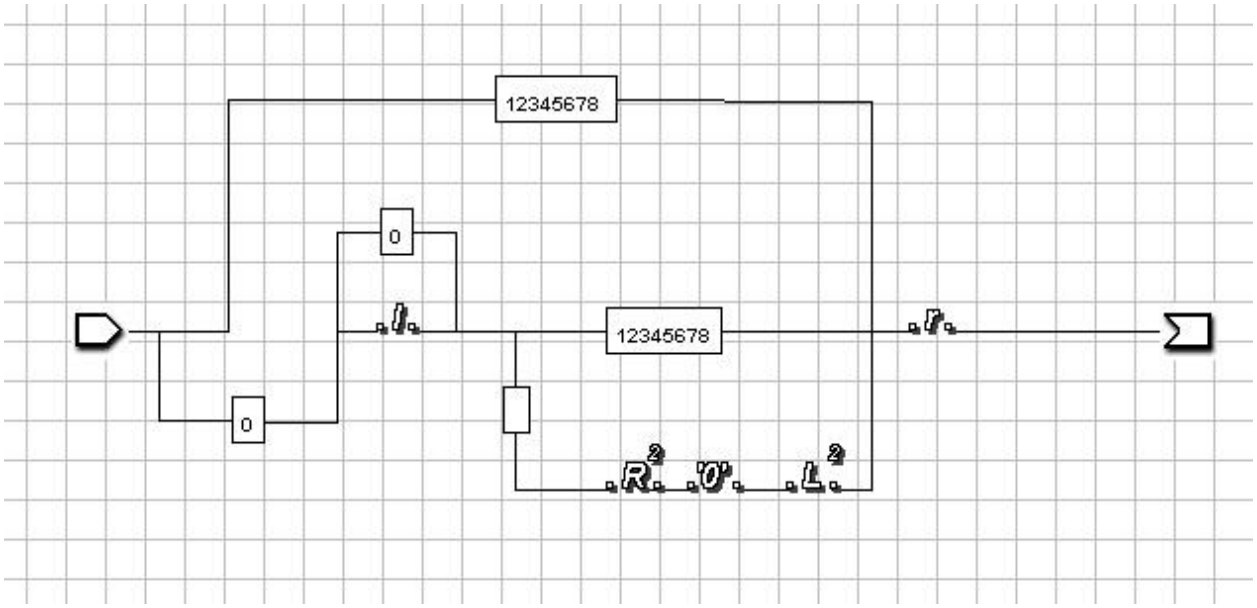

Рисунок 7: if_begin_zero

Для копирования цифры используются подмашины copy_zero, copy_one, copy_two, copy_three, copy_four, copy_five, copy_six, copy_seven, copy_eight.

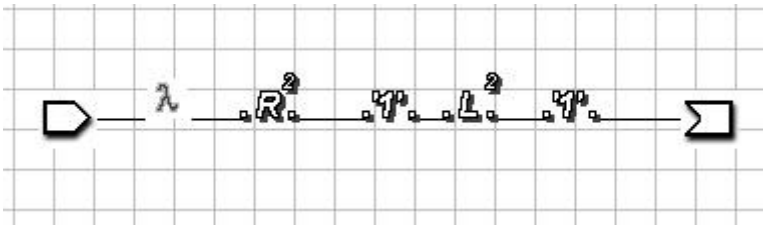

Рисунок 8: copy_one

Подмашина finish перемещает головку МТ в конец выходного числа.

Тестирование:

Входные данные	Выходные данные	Описание
0	0 0	Копирование нуля
000	000 0	Сложное копирование нуля
100	100 1	Удаление незначащих нулей
0123456780	0123456780 876543210	Сложный случай

3. Вывод

Разработан и реализован в среде разработки Диаграмм Тьюринга алгоритм реверса девятеричного числа с избавлением от незначащих нулей. На примере работы с более высокоуровневой реализацией абстрактного исполнителя Машины Тьюринга дополнен опыт разработки алгоритмов, полученный при решении предыдущей задачи.

Плюсы Диаграмм Тьюринга:

1. Визуальная наглядность: Одним из главных преимуществ диаграмм Тьюринга является их визуальная наглядность. За счет графического представления, они позволяют легко понимать и визуализировать, как работает Машина Тьюринга. Диаграммы Тьюринга могут помочь исследователям и студентам лучше понять и запомнить концепции и алгоритмы Машины Тьюринга.

2. Простота чтения и записи: Диаграммы Тьюринга предоставляют более простой способ записи и чтения алгоритмов Машины Тьюринга. Они позволяют сократить объем текстового описания и сделать запись алгоритмов более лаконичной и понятной. Это особенно полезно при обучении, обмене и анализе алгоритмов Машины Тьюринга.

3. Удобство анализа: Диаграммы Тьюринга облегчают анализ работы Машины Тьюринга и позволяют быстро определить, какие переходы происходят в зависимости от

символов на ленте. Благодаря графическому представлению, можно легко отслеживать состояния и переходы, а также анализировать и проверять выполнение алгоритмов.

Минусы Диаграмм Тьюринга:

1. Ограничение масштабирования: При работе с диаграммами Тьюринга важно учесть, что их масштабирование ограничено. С ростом сложности алгоритма может возникнуть необходимость во множестве состояний и переходов, что может привести к созданию громоздких и запутанных диаграмм. В таких случаях может быть трудно осуществить эффективное масштабирование и чтение диаграмм.

2. Ограниченная эффективность: При работе с некоторыми сложными алгоритмами, диаграммы Тьюринга могут оказаться неэффективными для их представления. Возникающая сложность некоторых алгоритмов может привести к слишком крупным и путаным диаграммам, сложностям при анализе и интерпретации. В таких случаях может потребоваться использование других методов визуализации или альтернативных графических моделей.

3. Ограничения языка: Диаграммы Тьюринга могут оказаться недостаточно гибкими для представления некоторых сложных алгоритмов. Они ограничены в своей способности выразить нетривиальные концепции и вычислительные модели. В некоторых случаях может потребоваться использование других формализмов или вычислительных моделей для полного описания и анализа алгоритмов.

Таким образом, диаграммы Тьюринга предоставляют множество плюсов, таких как визуальная наглядность, простота чтения и записи, а также удобство анализа. Однако они также имеют ограничения, связанные с масштабированием, эффективностью и языковыми ограничениями.

ГЛАВА III. НОРМАЛЬНЫЕ АЛГОРИТМЫ МАРКОВА

1. Теоретические сведения

1.1. Историческая справка

История создания модели нормальных алгоритмов Маркова (НАМ) начинается с работ ученого Андрея Андреевича Маркова в области теории вероятностей и математической логики в начале XX века. А.А. Марков был известен своими исследованиями по марковским процессам, а именно, случайным последовательностям с определенными стохастическими свойствами.

Идея использования алгоритмов Маркова в моделировании поведения систем или решении задач компьютерной науки возникла немного позже. В 1961 году исследователи А.А. Маркова, Анатолий Алексеевич Марков и Александр Андреевич Южин предложили новую модель, названную «нормальными алгоритмами Маркова».

Идея модели НАМ получила широкое признание в научном сообществе и стала основой для разработки различных теоретических и практических решений. Она нашла применение в различных областях, включая искусственный интеллект, теорию вычислений, биоинформатику и другие.

С течением времени модель НАМ была дополнена и усовершенствована, и появились новые варианты алгоритмов Маркова, такие как стохастические алгоритмы Маркова и иерархические алгоритмы Маркова.

1.2. Определение

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.

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:

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

Поскольку нет никакой свободы в выборе очередного шага обработки, алгоритмы Маркова являются детерминистическими. Поскольку результат алгоритма Маркова этой стратегией определяется однозначно, то он также является детерминированным.

Итак, нормальный алгоритм Маркова (НАМ) представляет собой упорядоченный набор правил-продукций — пар слов (цепочек знаков, в том числе пустых цепочек длины

0), соединенных между собой символами → или ↦. Каждая продукция представляет собой формулу замены части входного слова, совпадающей с левой частью формулы, на ее правую часть. И левая, и правая части продукций могут быть пустыми: либо выполняется безусловная подстановка правой части, либо удаляется часть исходного слова. Однако поскольку пустое слово присутствует и слева, и справа от каждой буквы преобразуемого слова, то подстановка с пустой левой частью зацикливается, и соответствующий алгоритм неприменим ни к какому входному слову. Если удается применить какую-то форму, подстановки, заменив вхождение ее левой части в исходном слове на правую часть, происходит возврат в начало алгоритма, и снова ищутся вхождения левой части первой продукции в измененное слово. Если же какую-то продукцию не удалось применить, проверяется следующая за ней, и так далее. Процесс выполнения нормального алгоритма. заканчивается в одном из двух случаев: либо все формулы оказались неприменимыми, т.е. 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 нормальный алгоритм применим к данному входному слову. Если же в процессе выполнения нормального алгоритма бесконечно долго применяются нетерминальные правила, то алгоритм неприменим к данному входному слову. Существуют следующие достаточные признаки применимости НАМ ко всем входным словам:

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

2. Практическая работа

Цель: Написать алгоритм Маркова, выполняющий задание, установленное вариантом (ненормированный вывод).

Задача: Входное слово представляет собой произвольную последовательность десятичных чисел без знака, разделённых знаками «#». Составить алгоритм вычисления числа слов в последовательности.

Идея, метод, алгоритм решения задачи: количество чисел напрямую зависит от количества разделителей "#", а именно - ровно на один больше. Это значит, что сами числа нам не важны, а значит можно от них избавиться и посчитать только «#», добавив 1.

Сценарий выполнения работы:

Первым делом следует избавиться от всех чисел, дабы они не мешали последующим вычислениям: для этого воспользуемся вспомогательным маркером «@», а все «#» заменим счётными палочками «|» для корректной работы алгоритмов. Вторым - начнём сами вычисления. Чтобы увеличить разряд не единиц, введём специализированный маркер «+», а также для более быстрых вычислений добавим частное условие подсчёта

сразу 10 «|». Конец программы - удаление маркера «@». Также следует учесть особый случай ввода пустого слова, что означает 0 чисел: для этих целей используем маркер «%» проверки.

Протокол программы НАМ расположен в приложении В.

Тестирование:

Входные данные	Выходные данные	Описание
	0	Пустое слово: чисел нет
1	1	Одно число без разделительного символа "#"
1#2	2	Два числа с разделителем "#"
1#2#3#4#5#6#7#8#9#10	10	Сложная ситуация

3. Вывод

Разработан и реализован в среде разработки алгоритмов алгоритмической модели Маркова алгоритм подсчёта количества чисел, разделённых «#». Дополнен опыт решения предыдущих задач в области создания алгоритмов в различных алгоритмических моделях.

Плюсы Нормальных Алгоритмов Маркова:

1. Простота и понятность: Модель НАМ имеет простую и интуитивно понятную структуру, которая позволяет разработчикам и исследователям легко понять и анализировать процессы, моделируемые этой моделью. Это делает НАМ доступным инструментом как для начинающих, так и для опытных специалистов.

2. Универсальность и гибкость: НАМ можно применять в широком спектре задач и областей. Они могут описывать и моделировать разнообразные процессы, включая алгоритмы, управление системами, протоколы связи, принятие решений и многое другое. Благодаря своей универсальности, НАМ могут быть применимы в различных областях науки и технологии.

3. Генеративность: НАМ могут генерировать последовательности состояний, основываясь на вероятностных правилах перехода между состояниями. Это позволяет создавать разнообразные варианты поведения системы и анализировать их свойства. Генеративная способность НАМ является полезным инструментом в моделировании и предсказании различных процессов.

4. Анализ и оптимизация: НАМ могут быть использованы для анализа и оптимизации систем. Путем изучения вероятностных характеристик переходов между состояниями, можно определить оптимальные стратегии работы системы, оптимальные пути или методы принятия решений. Это позволяет улучшать производительность и эффективность различных систем.

Минусы Нормальных Алгоритмов Маркова:

1. Ограничения в точности предсказания: НАМ основываются на вероятностных правилах, и их предсказания могут быть неточными. Вероятностные компоненты модели могут привести к случайным флуктуациям и неопределенности в результатах. Это может создавать проблемы в задачах, где необходима высокая точность предсказания и надежность.

2. Зависимость от входных данных: НАМ могут быть чувствительны к входным данным и их распределению. Если входные данные не соответствуют предполагаемым распределениям, результаты моделирования НАМ могут быть искаженными или неправильными. Это требует основательного анализа данных и аккуратного выбора параметров модели.

3. Сложность моделей на практике: При моделировании сложных систем часто требуется создание больших и сложных НАМ, что может быть затруднительным и трудоемким заданием. Разработка, настройка и обучение таких моделей могут потребовать значительного количества времени и вычислительных ресурсов.

4. Непригодность для точных алгоритмов и формальных доказательств: В отличие от некоторых других моделей, НАМ не предоставляют формальных гарантий и доказательств для точных алгоритмов и решений. Это связано с их вероятностной природой и случайными переходами. Важные свойства и связности функций могут потребовать дополнительного анализа и проверки.

В целом, НАМ не менее мощный и гибкий инструмент для моделирования и анализа различных процессов и систем. Однако данная модель также имеет ряд существенных недостатков.

ЗАКЛЮЧЕНИЕ

В данной курсовой работе были наглядно показаны формальные определения алгоритмов на основе таких моделей, как Машины Тьюринга, их эквивалентные диаграммы, а также Нормальные Алгоритмы Маркова. Были сформулированы плюсы и минусы для каждой модели, на основе чего можно проследить явные различия между ними.

Так машины Тьюринга используются в различных областях, где требуется моделирование и анализ вычислений:

1. Теоретические науки: Машины Тьюринга широко применяются в теории вычислений, математике и логике. Они служат фундаментальной моделью для исследования различных компьютерных алгоритмов и вычислительных задач. Машины Тьюринга позволяют исследовать основные вопросы, связанные с вычислимостью и сложностью.

2. Компьютерные науки: Машины Тьюринга тесно связаны с компьютерными науками, особенно с теорией языков и компиляции. Они используются для анализа алгоритмов и вычислительной сложности, а также для разработки и анализа языков программирования и компиляторов.

3. Биология и генетика: Машины Тьюринга могут быть использованы для моделирования биологических процессов, таких как взаимодействие генов или функционирование молекулярных механизмов. Они могут помочь в исследовании генетических последовательностей и анализе биологических данных.

4. Искусственный интеллект: Машины Тьюринга играют важную роль в исследованиях и разработке искусственного интеллекта. Они являются основой для моделирования и реализации различных алгоритмов машинного обучения и искусственного интеллекта. Машины Тьюринга используются для анализа и решения различных задач, таких как распознавание образов, обработка естественного языка и принятие решений.

5. Криптография: Машины Тьюринга используются в криптографии для анализа и разработки криптографических алгоритмов. Они позволяют оценить стойкость различных шифров, а также разрабатывать новые криптографические протоколы и системы защиты информации.

Диаграммы Тьюринга во многом используются в тех же сферах, что и МТ, однако всё же имеют и расхождения:

1. Компьютерные науки: Диаграммы Тьюринга используются в компьютерных науках для анализа и проектирования алгоритмов, структур данных и программных систем. Они позволяют лучше понять взаимодействие различных компонентов системы и эффективность выполнения операций.

2. Образование: Диаграммы Тьюринга также широко используются в образовательных процессах для обучения студентов основам теории вычислений и алгоритмам. Они помогают визуализировать абстрактные концепции и упрощают объяснение сложных понятий.

В свою очередь нормальные алгоритмы Маркова серьёзно отличаются от других представленных моделей, поэтому сферы применения их различаются, хоть всё равно и наблюдаются смежные:

1. Языковые модели: Нормальные алгоритмы Маркова используются для построения языковых моделей, которые могут использоваться для предсказания следующего слова в тексте, генерации новых текстов или оценки вероятности последовательности слов. Это находит применение в автоматическом переводе, голосовых ассистентах, чат-ботах и других приложениях в области обработки естественного языка.

2. Анализ временных рядов: Нормальные алгоритмы Маркова широко используются для моделирования и анализа временных рядов, таких как финансовые данные, погодные данные, трафик сети и т.д. Они позволяют предсказывать будущие значения временных рядов на основе предыдущих значений и помогают выявлять тренды, сезонность и аномалии в данных.

3. Искусственный интеллект: Нормальные алгоритмы Маркова используются в области искусственного интеллекта для моделирования и решения задач машинного обучения и обработки данных. Они могут использоваться для классификации, кластеризации, предсказания и других задач, связанных с анализом данных.

4. Генетика: Нормальные алгоритмы Маркова применяются в генетике для моделирования и анализа генетических последовательностей. Они позволяют исследователям оценить вероятность определенных генетических событий, таких как мутации или эволюционные изменения, и помогают понять механизмы наследования и развития организмов.

5. Робототехника: Нормальные алгоритмы Маркова используются в робототехнике для моделирования и управления поведением роботов. Они позволяют роботам адаптироваться к изменяющейся среде, принимать решения на основе входных данных и выполнять задачи в автономном режиме.

Выше перечислены лишь некоторые области применения алгоритмов. Таким образом, можно сделать вывод, что не существует безусловно «хорошей» или «плохой» модели. Программисту предоставляется возможность выбора использования рассматриваемых инструментов для решения конкретно поставленных задач, что может позволить ему оптимизировать задачу и составить эффективную программу.

Используемые ресурсы и источники можете изучить в приложении А.

ПРИЛОЖЕНИЕ А

При выполнении работы использовались:

- Эмулятор машины Тьюринга в четвёрках (jstu4);
- Диаграммер для работы с диаграммами Тьюринга (VirtualTuringMachine);
- Эмулятор для нормальных алгоритмов Маркова (nam);

Вспомогательная литература:

1. С. С. Гайсарян, В. Е. Зайцев «Курс информатики», Москва, Издательство Вузовская книга, 2013 г.

ПРИЛОЖЕНИЕ Б

Протокол программы МТ для решения задачи, приведённой в главе I.

```
00, ,<,run

run,0,<,run
run,1,<,run
run,2,<,run
run,3,<,run
run,4,<,run
run,5,<,run
run,6,<,run
run,7,<,run
run,8,<,run
run, ,>,zeroif
runreverse,0,>,runreverse
runreverse,1,>,runreverse
runreverse,2,>,runreverse
runreverse, ,#,00

read,0, ,zerobegin
read,1, ,onebegin
read,2, ,twobegin
read,3, ,threebegin
read,4, ,fourbegin
read,5, ,fivebegin
read,6, ,sixbegin
read,7, ,sevenbegin
```

read,8, ,eightbegin

read, ,>,runreverse

zeroif,1,=,read

zeroif,2,=,read

zeroif,3,=,read

zeroif,4,=,read

zeroif,5,=,read

zeroif,6,=,read

zeroif,7,=,read

zeroif,8,=,read

zeroif,0,>,zeroif

zeroif, ,>,zerowriteif

zerowriteif, ,0,runreverse

zerobegin, ,>,zerorun

zerorun,0,>,zerorun

zerorun,1,>,zerorun

zerorun,2,>,zerorun

zerorun,3,>,zerorun

zerorun,4,>,zerorun

zerorun,5,>,zerorun

zerorun,6,>,zerorun

zerorun,7,>,zerorun

zerorun,8,>,zerorun

zerorun, ,>,zero

zero,0,>,zero

zero,1,>,zero

zero,2,>,zero

zero, ,0,zerowrite

zerowrite,0,>,zerowrite

zerowrite, ,0,zeroreverse

zeroreverse,0,<,zeroreverse

zeroreverse,1,<,zeroreverse

zeroreverse,2,<,zeroreverse

zeroreverse, ,<,zeroback

zeroback,1,<,zeroback

zeroback,2,<,zeroback

zeroback,3,<,zeroback

zeroback,4,<,zeroback

zeroback,5,<,zeroback

zeroback,6,<,zeroback

zeroback,7,<,zeroback

zeroback,8,<,zeroback

zeroback,0,<,zeroback

zeroback, ,0,zeroe

zeroe,0,>,read

onebegin, ,>,onerun

onerun,0,>,onerun

onerun,1,>,onerun

onerun,2,>,onerun

onerun,3,>,onerun

onerun,4,>,onerun

onerun,5,>,onerun

onerun,6,>,onerun

onerun,7,>,onerun

onerun,8,>,onerun

onerun, ,>,oneif

oneif,0,>,one

oneif,1,>,one

oneif,2,>,one

oneif, ,1,onereverse

one,0,>,one

one,1,>,one

one,2,>,one

one, ,0,onewrite

onewrite,0,>,onewrite

onewrite, ,1,onereverse

onereverse,0,<,onereverse

onereverse,1,<,onereverse

onereverse,2,<,onereverse

onereverse, ,<,oneback

oneback,1,<,oneback

oneback,2,<,oneback

oneback,3,<,oneback

oneback,4,<,oneback

oneback,5,<,oneback

oneback,6,<,oneback

oneback,7,<,oneback

oneback,8,<,oneback

oneback,0,<,oneback

oneback, ,1,onee

onee,1,>,read

twobegin, ,>,tworun

tworun,0,>,tworun

tworun,1,>,tworun

tworun,2,>,tworun

tworun,3,>,tworun

tworun,4,>,tworun

tworun,5,>,tworun

tworun,6,>,tworun

tworun,7,>,tworun

tworun,8,>,tworun

tworun, ,>,twoif

twoif,0,>,two

twoif,1,>,two

twoif,2,>,two

twoif, ,2,tworeverse

two,0,>,two

two,1,>,two

two,2,>,two

two, ,0,twowrite

twowrite,0,>,twowrite

twowrite, ,2,tworeverse

tworeverse,0,<,tworeverse

tworeverse,1,<,tworeverse

tworeverse,2,<,tworeverse

tworeverse, ,<,twoback

twoback,1,<,twoback

twoback,2,<,twoback

twoback,3,<,twoback

twoback,4,<,twoback

twoback,5,<,twoback

twoback,6,<,twoback

twoback,7,<,twoback

twoback,8,<,twoback

twoback,0,<,twoback

twoback, ,2,twoe

twoe,2,>,read

threebegin, ,>,threerun

threerun,0,>,threerun

threerun,1,>,threerun

threerun,2,>,threerun

threerun,3,>,threerun

threerun,4,>,threerun

threerun,5,>,threerun

threerun,6,>,threerun

threerun,7,>,threerun

threerun,8,>,threerun

threerun, ,>,three

three,0,>,three

three,1,>,three

three,2,>,three

three, ,1,threewrite

threewrite,1,>,threewrite

threewrite, ,0,threereverse

threereverse,0,<,threereverse

threereverse,1,<,threereverse

threereverse,2,<,threereverse

threereverse, ,<,threeback

threeback,1,<,threeback

threeback,2,<,threeback

threeback,3,<,threeback

threeback,4,<,threeback

threeback,5,<,threeback

threeback,6,<,threeback

threeback,7,<,threeback

threeback,8,<,threeback

threeback,0,<,threeback

threeback, ,3,threee

threee,3,>,read

fourbegin, ,>,fourrun

fourrun,0,>,fourrun

fourrun,1,>,fourrun

fourrun,2,>,fourrun

fourrun,3,>,fourrun

fourrun,4,>,fourrun

fourrun,5,>,fourrun

fourrun,6,>,fourrun

fourrun,7,>,fourrun

fourrun,8,>,fourrun

fourrun, ,>,four

four,0,>,four

four,1,>,four

four,2,>,four

four, ,1,fourwrite

fourwrite,1,>,fourwrite

fourwrite, ,1,fourreverse

fourreverse,0,<,fourreverse

fourreverse,1,<,fourreverse

fourreverse,2,<,fourreverse

fourreverse, ,<,fourback

fourback,1,<,fourback

fourback,2,<,fourback

fourback,3,<,fourback

fourback,4,<,fourback

fourback,5,<,fourback

fourback,6,<,fourback

fourback,7,<,fourback

fourback,8,<,fourback

fourback,0,<,fourback

fourback, ,4,foure

foure,4,>,read

fivebegin, ,>,fiverun

fiverun,0,>,fiverun

fiverun,1,>,fiverun

fiverun,2,>,fiverun

fiverun,3,>,fiverun

fiverun,4,>,fiverun

fiverun,5,>,fiverun

fiverun,6,>,fiverun

fiverun,7,>,fiverun

fiverun,8,>,fiverun

fiverun, ,>,five

five,0,>,five

five,1,>,five

five,2,>,five

five, ,1,fivewrite

fivewrite,1,>,fivewrite

fivewrite, ,2,fivereverse

fivereverse,0,<,fivereverse

fivereverse,1,<,fivereverse

fivereverse,2,<,fivereverse

fivereverse, ,<,fiveback

fiveback,1,<,fiveback

fiveback,2,<,fiveback

fiveback,3,<,fiveback

fiveback,4,<,fiveback

fiveback,5,<,fiveback

fiveback,6,<,fiveback

fiveback,7,<,fiveback

fiveback,8,<,fiveback

fiveback,0,<,fiveback

fiveback, ,5,fivee

fivee,5,>,read

sixbegin, ,>,sixrun

sixrun,0,>,sixrun

sixrun,1,>,sixrun

sixrun,2,>,sixrun

sixrun,3,>,sixrun

sixrun,4,>,sixrun

sixrun,5,>,sixrun

sixrun,6,>,sixrun

sixrun,7,>,sixrun

sixrun,8,>,sixrun

sixrun, ,>,six

six,0,>,six

six,1,>,six

six,2,>,six

six, ,2,sixwrite

sixwrite,2,>,sixwrite

sixwrite, ,0,sixreverse

sixreverse,0,<,sixreverse

sixreverse,1,<,sixreverse

sixreverse,2,<,sixreverse

sixreverse, ,<,sixback

sixback,1,<,sixback

sixback,2,<,sixback

sixback,3,<,sixback

sixback,4,<,sixback

sixback,5,<,sixback

sixback,6,<,sixback

sixback,7,<,sixback

sixback,8,<,sixback

sixback,0,<,sixback

sixback, ,6,sixe

sixe,6,>,read

sevenbegin, ,>,sevenrun

sevenrun,0,>,sevenrun

sevenrun,1,>,sevenrun

sevenrun,2,>,sevenrun

sevenrun,3,>,sevenrun

sevenrun,4,>,sevenrun

sevenrun,5,>,sevenrun

sevenrun,6,>,sevenrun

sevenrun,7,>,sevenrun

sevenrun,8,>,sevenrun

sevenrun, ,>,seven

seven,0,>,seven

seven,1,>,seven

seven,2,>,seven

seven, ,2,sevenwrite

sevenwrite,2,>,sevenwrite

sevenwrite, ,1,sevenreverse

sevenreverse,0,<,sevenreverse

sevenreverse,1,<,sevenreverse

sevenreverse,2,<,sevenreverse

sevenreverse, ,<,sevenback

sevenback,1,<,sevenback

sevenback,2,<,sevenback

sevenback,3,<,sevenback

sevenback,4,<,sevenback

sevenback,5,<,sevenback

sevenback,6,<,sevenback

sevenback,7,<,sevenback

sevenback,8,<,sevenback

sevenback,0,<,sevenback

sevenback, ,7,sevene

sevene,7,>,read

eightbegin, ,>,eightrun

eightrun,0,>,eightrun

eightrun,1,>,eightrun

eightrun,2,>,eightrun

eightrun,3,>,eightrun

eightrun,4,>,eightrun

eightrun,5,>,eightrun

eightrun,6,>,eightrun

eightrun,7,>,eightrun

eightrun,8,>,eightrun

eightrun, ,>,eight

eight,0,>,eight

eight,1,>,eight

eight,2,>,eight

eight, ,2,eightwrite

eightwrite,2,>,eightwrite

eightwrite, ,2,eightreverse

eightreverse,0,<,eightreverse

eightreverse,1,<,eightreverse

eightreverse,2,<,eightreverse

eightreverse, ,<,eightback

eightback,1,<,eightback

eightback,2,<,eightback

eightback,3,<,eightback

eightback,4,<,eightback

eightback,5,<,eightback

eightback,6,<,eightback

eightback,7,<,eightback

eightback,8,<,eightback

eightback,0,<,eightback

eightback, ,8,eighte

eighte,8,>,read

ПРИЛОЖЕНИЕ В

Протокол программы НАМ для решения задачи, приведённой в главе III.

```
%0->|*
%1->|*
%2->|*
%3->|*
%4->|*
%5->|*
%6->|*
%7->|*
%8->|*
%9->|*
%->.0
*0->*
*1->*
*2->*
*3->*
*4->*
*5->*
*6->*
*7->*
*8->*
*9->*
*#->|*

0+->1
1+->2
```

2+->3

3+->4

4+->5

5+->6

6+->7

7+->8

8+->9

9+->+0

+->1

0|||||||||->+0

0|->1

1|->2

2|->3

3|->4

4|->5

5|->6

6|->7

7|->8

8|->9

9|->+0

|->0|

*->.

->%