МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

Донецкий Национальный Технический Университет

Кафедра программной инженерии

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по дисциплине

«Теория алгоритмов и формальных языков»

на тему: «Машина Тьюринга»

|  |  |
| --- | --- |
| Руководители:  Морозова О.В.  Коломойцева И.А.  Щедрин С.В. | Выполнила: стгр. ПИ-16Б Мамутова В. А. |

Донецк – 2017

РЕФЕРАТ

Отчет по курсовой работе содержит: 35 страниц, 30 рисунков, 3 таблицы, 5 приложений, 5 источников.

Объект исследования — рекурсивные функции, машины Тьюринга, нормальные алгоритмы Маркова.

Цель — сформировать формальное определение алгоритма в виде трех аналитических моделей, написать программную реализацию машины Тьюринга, распознающий язык L = { ww | w ∈ {a, b, c}\*}, построить график временной сложности.

Результат — формальное определение алгоритмов на основе рекурсивных функций, машин Тьюринга и нормальных алгоритмов Маркова, программная реализация машины Тьюринга, распознающей язык L = { ww | w ∈ {a, b, c}\*}, график временной сложности машины Тьюринга, файловый вариант протокола работы машины Тьюринга.

МАШИНА ТЬЮРИНГА, ЯЗЫК, ЛЕНТА, АЛФАВИТ, РАСПОЗНАВАНИЕ, АНАЛИТИЧЕСКАЯ МОДЕЛЬ, ПРОТОКОЛ, ПРИНАДЛЕЖНОСТЬ.

СОДЕРЖАНИЕ

[Введение 4](#_Toc500254426)

[1 Описание аналитической модели алгоритма в виде элементарных машин Тьюринга и композии МТ 5](#_Toc500254427)

[2 Разработка аналитической и программной модели алгоритма для распознающей машины Тьюринга 9](#_Toc500254428)

[2.1 Формальное определение распознающей машин Тьюринга 9](#_Toc500254429)

[2.2 Протоколы работы машины Тьюринга 11](#_Toc500254430)

[2.3 Программная модель машины Тьюринга 12](#_Toc500254431)

[2.4 Протоколы работы машины Тьюринга, построенные программно (на двух словах языка и двух словах, не принадлежащих языку) 13](#_Toc500254432)

[2.5 Расчет временной сложности (график функции временной](#_Toc500254433)

[сложности) 13](#_Toc500254434)

[3 Разработка аналитической модели алгоритма с использованием нормальных алгоритмов Маркова 14](#_Toc500254435)

[4 Описание формальной модели алгоритма на основе рекурсивных функций 17](#_Toc500254436)

[Выводы 19](#_Toc500254437)

[Перечень ссылок 20](#_Toc500254438)

[Приложение А. Техническое задание 21](#_Toc500254439)

[Приложение Б. Руководство пользователя 25](#_Toc500254440)

[Приложение В. Протоколы работы машины Тьюринга, построенные программно 26](#_Toc500254441)

[Приложение Г. Исходный код программы 29](#_Toc500254442)

[Приложение Д. Экранные формы 33](#_Toc500254443)

# Введение

Первым дошедшим до нас алгоритмом в его интуитивном понимании – конечной последовательности элементарных действий, решающих поставленную задачу, считается предложенный Евклидом в Ш веке до нашей эры алгоритм нахождения наибольшего общего делителя двух чисел (алгоритм Евклида). В течение длительного времени, вплоть до начала ХХ века само слово «алгоритм» употреблялось в устойчивом сочетании «алгоритм Евклида». Для описания пошагового решения других математических задач использовалось слово «метод».

Начальной точной отсчета современной теории алгоритмов можно считать работу немецкого математика Курта Гёделя (1931 год - теорема о неполноте символических логик), в котором было показано, что некоторые математические проблемы не могут быть решены алгоритмами из некоторого класса. Общность результата Геделя связана с тем, совпадает ли использованный им класс алгоритмов с классом всех (в интуитивном смысле) алгоритмов. Эта работа дала толчок к поиску и анализу различных формализаций алгоритма.

Первые фундаментальные работы по теории алгоритмов были опубликованы независимо в 1936 году Аланом Тьюрингом. Алонзо Черчем и Эмилем Постом. Предложенные ими машина Тьюринга, машина Поста и лямбда-исчисление Черча были эквивалентными формализации алгоритма. Сформулированные ими тезисы (Поста и Черча-Тьюринга) постулировали эквивалентность предложенных ими формальных систем и интуитивного понятия алгоритма. Важным развитием этих работ стала формулировка и доказательство алгоритмически неразрешимых проблем.

# 1 Описание аналитической модели алгоритма в виде элементарных машин Тьюринга и композии МТ

Для описания аналитической модели алгоритма в виде элементарных машин Тьюринга реализуем алгоритм в алфавите А = { a, b, c }, производящий в слове алфавита замену всех вхождений буквы a на букву b.

Опишем работу машины Тьюринга тремя способами:

1. системой команд;
2. функциональной таблицей;
3. графом (диаграммой) переходов.

На рисунке 1.1 представлено описание машины Тьюринга с помощью системы команд.

|  |  |  |
| --- | --- | --- |
| q0 a | -> | q0 b R |
| q0 b | -> | q0 b R |
| q0 c | -> | q0 c R |
| q0 λ | -> | q0 λ L |
| q1 b | -> | q1 b L |
| q1 c | -> | q1 c L |
| q1 λ | -> | qz λ R |

Рисунок 1.1 – Система команд

В таблице 1.1 представлена функциональная таблица, в которой записаны переходы между состояниями машины Тьюринга, выполняющей поставленную задачу.

Таблица 1.1 – Функциональная таблица

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | b | c | λ |
| q0 | q0 b R | q0 b R | q0 c R | q1 λ L |
| q1 | - | q1 b L | q1 c L | qz λ R |

Граф переходов изображен на рисунке 1.2. Структура данного представления следующая:

- вершины графа – состояния МТ;

- дуги графа и их направление – переходы между состояниями.

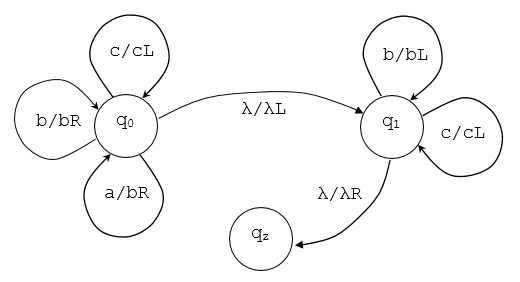


Рисунок 1.2 — Граф переходов

Тестовые примеры работы алгоритма приведены на рисунках 1.3 – 1.5.

|  |
| --- |
| q0baac |
| bq0aac |
| bbq0ac |
| bbbq0c |
| bbbcq0λ |
| bbbq1с |
| bbq1bc |
| bq1bbc |
| q1bbbc |
| q1λbbbc |
| qzbbbc |

Рисунок 1.3 – Тестовый пример на входном слове «baac»

|  |
| --- |
| q0cb |
| cq0b |
| cbq0λ |
| cq1b |
| q1cb |
| q1λcb |
| qzcb |

Рисунок 1.4 – Тестовый пример на входном слове «cb»

|  |
| --- |
| q0λ |
| q0λλ |
| qzλ |

Рисунок 1.5 – Тестовый пример на пустом слове

Для описания аналитической модели алгоритма в виде композиции МТ

реализуем алгоритм для подсчёта количества нечётных цифр в 8-ричной записи числа n.

Элементарные машины Тьюринга, реализующие каждый простой блок композиции, приведены на рисунке 1.6.

|  |  |  |
| --- | --- | --- |
| M0 | – | копирование аргумента |
|  | – | выделение m-го аргумента из n |
| M0 | – | установка нуля (count = 0) |
| M>0 | – | сравнение с нулём:  w = |
| M+ | – | инкремент |
| Мmod | – | нахождение остатка от деления |
| Мdiv2 | – | нахождение целой части от деления на 2 |
| Mdiv8 | – | нахождение целой части от деления на 8 |
| M<>0 | – | проверка на неравенство с нулём |

Рисунок 1.6 – Элементарные машины Тьюринга

Составим блок-схему (рис. 1.7).

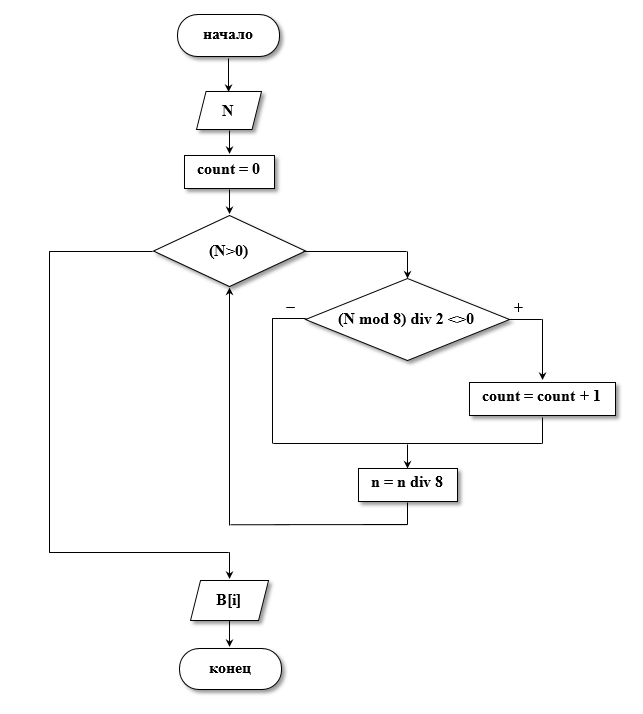


Рисунок 1.7 – Блок-схема алгоритма

Составим композицию машин Тьюринга (рисунке 1.8).

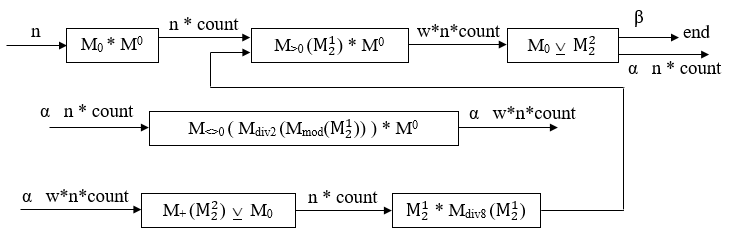


Рисунок 1.8 – Композиция МТ

# 2 Разработка аналитической и программной модели алгоритма для распознающей машины Тьюринга

## 2.1 Формальное определение распознающей машин Тьюринга

Программная модель была написана с целью распознавания языка L = { ww | w ∈ {a, b, c}\*}. Обработка слова требует использования 2 лент для упрощения работы.

Таблица 2.1 – Функциональная таблица

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | λλ | Aλ | Bλ | cλ | aa | ba | Ca | ab | Bb | сb | ac | bc | cc |
| q0 | q3λλLL | q1aλRE | q1bλRE | q1cλRE | - | - | - | - | - | - | - | - | - |
| q1 | q2λλLL | q0aaRR | q0baRR | q0caRR | - | - | - | - | - | - | - | - | - |
| q2 | qz-λEE | q2λλLE | q2λλLE | q2λλLE | q2λλLL | q2λλLL | q2λλLL | - | - | - | - | - | - |
| q3 | qz+λEE | q4aλLE | q4bλLE | q4cλLE | q3\_aLL | q3λbLL | q3λcLL | - | - | - | - | - | - |
| q4 | q5λλRR | q4aλLE | q4bλLE | q4cλLE | - | - | - | - | - | - | - | - | - |
| q5 | qz+λEE | - | - | - | q5λλRR | q6λλRR | q6λλRR | q6λλRR | q5λλRR | q6λλRR | q6λλRR | q6λλRR | q5λλRR |
| q6 | qz-λEE | - | - | - | q6λλRR | q6λλRR | q6λλRR | q6λλRR | q6λλRR | q6λλRR | q6λλRR | q6λλRR | q6λλRR |

## 2.2 Протоколы работы машины Тьюринга

Пример работы алгоритма на слове «abab», которое является словом исходного языка приведен на рисунке 2.1.

λq0ababλ

λq0λ

-------------

λaq1babλ

λq1λ

-------------

λabq0abλ

λaq0λ

-------------

λabaq1bλ

λaq1λ

-------------

λababq0λ

λaaq0λ

-------------

λabaq3bλ

λaq3aλ

-------------

λabq3aλ

λq3abλ

-------------

λaq3bλ

q3λabλ

-------------

λq4abλ

q4λabλ

-------------

q4λabλ

q4λabλ

-------------

λq5abλ

λq5abλ

-------------

λq5bλ

λq5bλ

-------------

λq5λ

λq5λ

-------------

λqz+λ

λqzλ

----------

Рисунок 2.1 – Протокол работы МТ на слове “abab”

На рисунке 2.2 представлен протокол работы машины Тьюринга на пустом слове. Это слово также принадлежит заданному языку.

q0λ

q0λ

--------------

q3λλ

q3λλ

--------------

qz+λλ

qzλ

--------------

Рисунок 2.2 - Протокол работы МТ на пустом слове

На рисунке 2.3 представлен протокол работы распознающей машины

Тьюринга на слове «bcabc», которое не принадлежит языку.

λq0bcabcλ

λq0λ

--------------

λbq1cabcλ

λq1λ

--------------

λbcq0abcλ

λaq0λ

--------------

λbcaq1bcλ

λaq1λ

--------------

λbcabq0cλ

λaaq0λ

--------------

λbcabcq1λ

λaaq1λ

--------------

λbcabq2cλ

λaq2aλ

--------------λbcaq2bλλ

λq2aλλλ

--------------

λbcq2aλλλ

q2λλλλλλ

--------------

λbq2cλλλλ

q2λλλλλλ

--------------

λq2bλλλλλ

q2λλλλλλ

--------------

q2λλλλλλλ

q2λλλλλλ

--------------

qz-λλλλλλ

qzλλλλλλ

--------------

Рисунок 2.3 — Протокол работы МТ на слове “bcabc”

## 2.3 Программная модель машины Тьюринга

Для наглядной демонстрации работы распознающей машины Тьюринга была разработана программа.

Программная модель распознающей машины Тьюринга реализована на языке программирования C#. Исходный код программы представлен в Приложении Г.

В программе формируются два массива, которые в целом имитируют функциональную таблицу. Первый – это n-мерный массив всех допустимых на каждой ленте символов, обозреваемых управляющим устройством на определённом шаге (индекс строки – номер ленты). Строка, полученная путём сложения всех символов в столбце j, будет являться ничем иным как заголовком j-того столбца функциональной таблицы. Второй массив – это двумерный массив команд, в котором индекс строки является порядковым номером состояния МТ, а индекс столбца – это порядковый номер комбинации символов на лентах (заголовок столбца функциональной таблицы). На пересечении i-той строки и j-того столбца записывается правая часть команды, которая выполняется в состоянии i при обозреваемых символах в столбцах j первого массива.

Исходная лента моделируется как переменная типа String, последующие ленты моделируются тем же способом.

В программе предусмотрено пошаговое прохождение слова с выполнением команд, автоматическая обработка слова, построение графика временной сложности алгоритма, запись всех тактов в файл.

## 2.4 Протоколы работы машины Тьюринга, построенные программно (на двух словах языка и двух словах, не принадлежащих языку)

Протоколы работы машины Тьюринга, построенные программно, приведены в приложении В. Символом "\_" обозначен пустой символ.

## 2.5 Расчет временной сложности (график функции временной

## сложности)

График функции временной сложности приведён в приложении Д.

График отображает зависимость между длиной входного слова и числом тактов, которое будет максимальным во всех словах, сгенерированных методом полного перебора на выбранную длину.

Для проверки полученных значений рассчитаем формулу для нахождения максимального числа тактов в разных особых случаях (рис. 2.4).

Рисунок 2.4 – Формула для получения максимального такта в словах

# 3 Разработка аналитической модели алгоритма с использованием нормальных алгоритмов Маркова

С целью разработки аналитической модели алгоритма с использованием нормальных алгоритмов Маркова реализуем алгоритм, выполняющий операцию x mod y над числами в унарном коде.

Марковские подстановки для данного алгоритма приведены на рисунке 3.1.

1\*1 -> 1s\*y

y1 -> yy

1\*yy -> z\*y

1\*y -> z\*0

1z -> z1

z\*0 -> \*0y

\*0 -> \*

1s\* -> s1\*

1sz -> s1z

z\*y -> z\*

z\* -> z

z -> 1

\*y -> \*

s\* -> .

s -> .

1\* -> \*

\*1 -> \*

\* -> .

Рисунок 3.1 – Марковские подстановки для алгоритма x mod y

Примеры работы алгоритма представлены на рисунках 3.2 - 3.8.

111\*11

111s\*y1

111s\*yy

11s1\*yy

11sz\*y

1s1z\*y

1sz1\*y

1szz\*0

1sz\*0y

1s\*0yy

1s\*yy

s1\*yy

sz\*y

sz\*

sz

s1

1

Рисунок 3.2 - Пример работы алгоритма на входном слове «111\*11»

1111\*11

1111s\*y1

1111s\*yy

111s1\*yy

111sz\*y

11s1z\*y

11sz1\*y

11szz\*0

11sz\*0y

11s\*0yy

11s\*yy

1s1\*yy

1sz\*y

s1z\*y

sz1\*y

szz\*0

sz\*0y

s\*0yy

s\*yy

s\*y

s\*

.

Рисунок 3.3 - Пример работы алгоритма на входном слове «1111\*11»

111\*111

111s\*y11

111s\*yy1

111s\*yyy

11s1\*yyy

11sz\*yy

1s1z\*yy

1sz1\*yy

1szz\*y

s1zz\*y

sz1z\*y

szz1\*y

szzz\*0

szz\*0y

sz\*0yy

s\*0yyy

s\*yyy

s\*yy

s\*y

s\*

.

Рисунок 3.4 - Пример работы алгоритма на входном слове «111\*111»

11\*11111

11s\*y1111

11s\*yy111

11s\*yyy11

11s\*yyyy1

11s\*yyyyy

1s1\*yyyyy

1sz\*yyyy

s1z\*yyyy

sz1\*yyyy

szz\*yyy

szz\*yy

szz\*y

szz\*

szz

s1z

sz1

s11

11

Рисунок 3.5 - Пример работы алгоритма на входном слове «11\*111»

11\*

1\*

\*

.

Рисунок 3.6 - Пример работы алгоритма на входном слове «11\*»

\*1

\*

\_\_

Рисунок 3.7 - Пример работы алгоритма на входном слове «\*1»

\*

.

Рисунок 3.8 - Пример работы алгоритма на входном слове «\*»

# 4 Описание формальной модели алгоритма на основе рекурсивных функций

Для описания формальной модели алгоритма на основе рекурсивных функций разработаем алгоритм функции f(x), который вычисляет сумму простых делителей числа . Ниже приведена полученная формула (рис. 4.1).

,

где

Рисунок 4.1 – Формула работы алгоритма

F(11) = sg(11) \* = 1 \* (1 + 3 + 11) = 25

i = 1 :

1 \* = 1 \* 1 \* (0) = 1 \* 1\* 1 = 1

i = 2 :

2 \* = 2 \* 0 \* (0) = 2 \* 0 \* 1 = 0

i = 3 :

3 \* = 3 \* 1 \* (2) = 3 \* 1 \* 0 = 0

i = 4 :

4 \* = 4 \* 0 \* (0) = 4 \* 0 \* 1 = 0

i = 5 :

5 \* = 5 \* 1 \* (1) = 5 \* 1 \* 0 = 0

i = 6 :

6 \* = 6 \* 0 \* (5) = 6 \* 0 \* 0 = 0

i = 7 :

7 \* = 7 \* 1 \* (4) = 7 \* 1 \* 0 = 0

i = 8 :

8 \* = 8 \* 0 \* (3) = 8 \* 0 \* 1 = 0

i = 9 :

9 \* = 9 \* 0 \* (2) = 9 \* 0 \* 1 = 0

i = 10 :

10 \* = 10 \* 0 \* (1) = 10 \* 0 \* 1 = 0

i = 11 :

11 \* = 11 \* 1 \* (0) = 11 \* 1 \* 1 = 11

Рисунок 4.2 — Выполнение алгоритма для числа 11

# Выводы

В ходе выполнения курсовой работы были закреплены навыки работы с рекурсивными функциями, машинами Тьюринга и нормальными алгоритмами Маркова.

Особенно детально были изучены различные многоленточные машины Тьюринга. Также получены навыки построения графика временной сложности алгоритма.

В данной работе разработан программный продукт, демонстрирующий работу распознающей машины Тьюринга для языка L = { ww | w ∈ {a, b, c}\*}. Результатом работы машины является символ «+» или «-» в зависимости от принадлежности или непринадлежности входного слова заданному языку.

Для данной машины Тьюринга был построен график временной сложности для входных слов длиной от 1 до 13.

# Перечень ссылок

Введение в теорию алгоритмов [Электронный ресурс]. – Режим доступа:

http://th-algoritmov.narod.ru/1.htm

Алферова З.В. Теория алгоритмов / З.В. Алферова. - М.: Издательство «Статистика», 1973. – 164 с.

Марков А.А. Теория алгоритмов / А.А. Марков, Н.М. Нагорный. – М.: Наука, 1984. – 217 с.: ил.

Машины Тьюринга [Электронный ресурс]. – Режим доступа: http://www.intuit.ru/studies/courses/1799/146/lecture/4035

Поляков В.И., Скорубский В.И. Основы теории алгоритмов / В.И. Поляков, В.И. Скорубский. – СПб: НИУ ИТМО, 2012. – 51 с.

# Приложение А

# Техническое задание

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра программной инженерии

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ: «Построение аналитических моделей алгоритмов и

оценка их сложности»

ПО КУРСУ: «Теория алгоритмов и формальных языков»

Выдано:

ст. гр. ПИ-16Б

Мамутовой В.А.

Руководители:

Морозова О.В.

Коломойцева И.А.

Щедрин С.В.

Донецк – 2017

1 Основанием для разработки является задание на курсовую работу, выданное кафедрой программной инженерии.

2 Целью разработки является создание программной модели машины Тьюринга, распознающий языка L = { ww | w ∈ {a, b, c}\*}, расчет и экспериментальная проверка расчета временной сложности МТ.

3 Требования к программе:

– при проверке слова на принадлежность языку необходимо запретить ввод с клавиатуры символов не из входного алфавита заданного языка;

– при проверке слова на принадлежность языку выводить на экран каждый шаг работы машины Тьюринга;

– сохранять протокол работы машины Тьюринга в текстовом файле;

– при построении графика временной сложности работы машины Тьюринга, значения для графика получить практически, с помощью созданной программной модели машины Тьюринга; для генерации слов длиной n использовать метод полного перебора;

– построение графика временной сложности работы МТ выполняется в отдельном потоке, значения на график заносятся по мере расчёта; построение графика останавливается по требованию пользователя.

4 Требования к программной документации:

- пояснительная записка;

- руководство пользователя.

5 Этапы разработки

Таблица А.1 – Этапы разработки

|  |  |  |
| --- | --- | --- |
| № этапа | Наименование этапа | Срок выполнения |
| 1 | Выдача задания, составление ТЗ и его утверждение | 2 недели |
| 2 | Построение формальных моделей алгоритмов. | 2-4 недели |
| 3 | Определение структур данных для представления МТ, построение алгоритмов. | 4-5 недели |
| 4 | Написание программы | 5-9 недели |
| 5 | Отладка программы | 9-12 недели |
| 6 | Написание пояснительной записки | 9-13 недели |
| 7 | Защита курсовой работы | 13-14 недели |

# Приложение Б

# Руководство пользователя

Б.1 Назначение программы

Программа предназначена для демонстрации работы многоленточной машины Тьюринга на примере языка L = { ww | w ∈ {a, b, c}\*}. Пользователь может проверить введенное им слово на принадлежность языку, сохранить результаты работы в текстовый файл, построить график сложности алгоритма для выбранной длины.

Б.2 Исполнение программы

Так как программа выполнена на языке C#, то для её выполнения обязательно наличие Microsoft .NET Framework версии, не ниже 4.0. Запуск

приложения осуществляется через файл Курсовая работа по машине Тьюринга.exe.

При запуске приложения открывается основное окно работы программ (примеры экранных форм приведены в программе)‚ где пользователь может с помощью соответствующих кнопок выполнять интересующие его действия.

На главном окне также отображается функциональная таблица МТ, которая считывается из файла при запуске программы.

Для ввода слова, пользователь должен ввести его в однострочную графу и после чего для запуска машины Тьюринга нажать в этом поле "Enter" либо кнопку «Запустить МТ», которая автоматически приведёт в действие алгоритм работы над текущим словом, а после его окончания – запись в файл. В правой области окна отображаются шаги алгоритма (такты для каждой ленты).

# Приложение В

# Протоколы работы машины Тьюринга, построенные программно

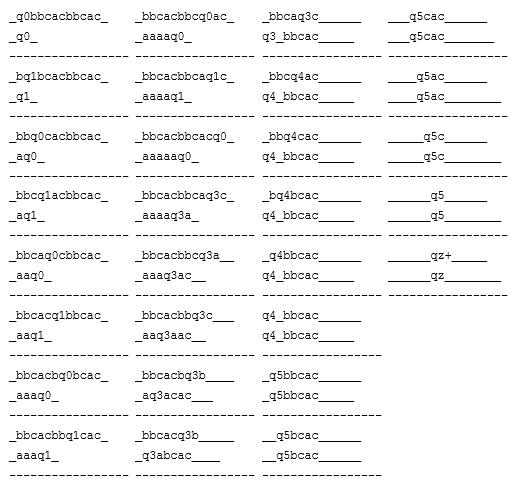


Рисунок В.1 – Тестовый пример на слове «bbcacbbcac», принадлежащем языку

q0aaaaaa\_

\_q0\_

-------------

\_aq1aaaaa\_

\_q1\_

-------------

\_aaq0aaaa\_

\_aq0\_

-------------

\_aaaq1aaa\_

\_aq1\_

-------------

\_aaaaq0aa\_

\_aaq0\_

-------------

\_aaaaaq1a\_

\_aaq1\_

-------------

\_aaaaaaq0\_

\_aaaq0\_

-------------

\_aaaaaq3a\_

\_aaq3a\_

-------------

\_aaaaq3a\_\_

\_aq3aa\_

-------------

\_aaaq3a\_\_\_

\_q3aaa\_

-------------

\_aaq3a\_\_\_\_

q3\_aaa\_

-------------

\_aq4aa\_\_\_\_

q4\_aaa\_

-------------

\_q4aaa\_\_\_\_

q4\_aaa\_

-------------

q4\_aaa\_\_\_\_

q4\_aaa\_

-------------

\_q5aaa\_\_\_\_

\_q5aaa\_

-------------

\_\_q5aa\_\_\_\_

\_\_q5aa\_\_

-------------

\_\_\_q5a\_\_\_\_

\_\_\_q5a\_\_\_

-------------

\_\_\_\_q5\_\_\_\_

\_\_\_\_q5\_\_\_\_

-------------

\_\_\_\_qz+\_\_\_

\_\_\_\_qz\_\_\_\_

-------------

Рисунок В.2 – Тестовый пример на слове «aaaaaa», принадлежащем языку

\_q0aacac\_

\_q0\_

-------------

\_aq1acac\_

\_q1\_

------------

\_aaq0cac\_

\_aq0\_

-------------

\_aacq1ac\_

\_aq1\_

-------------

\_aacaq0c\_

\_aaq0\_

-------------

\_aacacq1\_

\_aaq1\_

-------------

\_aacaq2c\_

\_aq2a\_

-------------

\_aacq2a\_\_

\_q2a\_\_\_

-------------

\_aaq2c\_\_\_

q2\_\_\_\_\_\_

-------------

\_aq2a\_\_\_\_

q2\_\_\_\_\_\_

-------------

\_q2a\_\_\_\_\_

q2\_\_\_\_\_\_

-------------

q2\_\_\_\_\_\_\_

q2\_\_\_\_\_\_

-------------

qz-\_\_\_\_\_\_

qz\_\_\_\_\_\_

-------------

Рисунок В.3 – Тестовый пример на слове «aacac», не принадлежащем языку

\_q0abaccabaca\_

\_q0\_

----------------

\_aq1baccabaca\_

\_q1\_

----------------

\_abq0accabaca\_

\_aq0\_

----------------

\_abaq1ccabaca\_

\_aq1\_

----------------

\_abacq0cabaca\_

\_aaq0\_

----------------

\_abaccq1abaca\_

\_aaq1\_

----------------

\_abaccaq0baca\_

\_aaaq0\_

----------------

\_abaccabq1aca\_

\_aaaq1\_

----------------

\_abaccabaq0ca\_

\_aaaaq0\_

----------------

\_abaccabacq1a\_

\_aaaaq1\_

----------------

\_abaccabacaq0\_

\_aaaaaq0\_

----------------

\_abaccabacq3a\_

\_aaaaq3a\_

----------------

\_abaccabaq3c\_\_

\_aaaq3aa\_

----------------

\_abaccabq3a\_\_\_

\_aaq3aca\_\_

----------------

\_abaccaq3b\_\_\_\_

\_aq3aaca\_\_

----------------

\_abaccq3a\_\_\_\_\_

\_q3abaca\_\_\_

----------------

\_abacq3c\_\_\_\_\_\_

q3\_abaca\_\_\_

----------------

\_abaq4cc\_\_\_\_\_\_

q4\_abaca\_\_\_

----------------

\_abq4acc\_\_\_\_\_\_

q4\_abaca\_\_\_

----------------

\_aq4bacc\_\_\_\_\_\_

q4\_abaca\_\_\_

----------------

\_q4abacc\_\_\_\_\_\_

q4\_abaca\_\_\_

----------------

q4\_abacc\_\_\_\_\_\_

q4\_abaca\_\_\_

----------------

\_q5abacc\_\_\_\_\_\_

\_q5abaca\_\_\_

----------------

\_\_q5bacc\_\_\_\_\_\_

\_\_q5baca\_\_\_\_

----------------

\_\_\_q5acc\_\_\_\_\_\_

\_\_\_q5aca\_\_\_\_\_

----------------

\_\_\_\_q5cc\_\_\_\_\_\_

\_\_\_\_q5ca\_\_\_\_\_\_

----------------

\_\_\_\_\_q5c\_\_\_\_\_\_

\_\_\_\_\_q5a\_\_\_\_\_\_\_

----------------

\_\_\_\_\_\_q6\_\_\_\_\_\_

\_\_\_\_\_\_q6\_\_\_\_\_\_\_\_

----------------

\_\_\_\_\_\_qz-\_\_\_\_\_

\_\_\_\_\_\_qz\_\_\_\_\_\_\_\_

----------------

Рисунок В.4 – Тестовый пример на слове «bbcacbbaca», не принадлежащем языку

# Приложение Г

# Исходный код программы

// Файд Form1.cs.

using System;

using System.IO;

using System.Text;

using System.Windows.Forms;

namespace Курсовая\_работа\_по\_машине\_Тьюринга

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

InitCommandList();

InitFuncTable();

form2 = new Form2(before, after, alphabet);

form2.Name = "Расчёт временной сложности алгоритма";

}

char[] alphabet; // Массив символов алфавита.

char[][] before; // Массив символов для замены (номер строки - номер ленты).

// Массив команд в виде <qr><a1'><a2'><S1><S2, где

// qr - номер состояния,

// a1' и a2' - заменяемыйе символы на 1/2 ленте соответвственно,

// S1 - символ смещения 1-ой ленты,

// S2 - символ смещения 2-ой ленты.

// \*\* Номер строки массива - порядковый номер состояния.

string[][] after;

Form2 form2; // Форма для вывода графика временной сложности.

private void InitCommandList()

{

before = new char[2][];

after = new string[7][];

string Line;

string[] strArr;

char separator = ' ';

int lineCount = 0;

try

{

FileStream file = new FileStream(Application.StartupPath + "\\Команды МТ.txt", FileMode.Open);

StreamReader streamReader = new StreamReader(file, Encoding.UTF8);

if (streamReader.EndOfStream != true)

{

// Считываем символы алфавита.

Line = streamReader.ReadLine();

alphabet = new char[Line.Length];

alphabet = Line.ToCharArray();

// Считываем символы исходной строки для замены.

Line = streamReader.ReadLine();

strArr = Line.Split(separator);

before[0] = new char[strArr.Length];

before[1] = new char[strArr.Length];

for (int i = 0; i < strArr.Length; i++)

{

before[0][i] = strArr[i][0]; // Символ первой ленты.

before[1][i] = strArr[i][1]; // Символ второй ленты.

}

}

while (streamReader.EndOfStream != true)

{

// Считываем команды.

Line = streamReader.ReadLine();

strArr = Line.Split(separator);

after[lineCount] = new string[strArr.Length];

for (int i = 0; i < strArr.Length; i++)

after[lineCount][i] = strArr[i];

lineCount++;

}

streamReader.Close();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при загрузке команд машины Тьюринга из файла\n" + ex.StackTrace + "\n\nПричина: " + ex.Message, "",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void InitFuncTable()

{

// Добавление в функциональную таблицу столбцов с подписью заголовков.

for (int i = 0; i < after[0].Length; i++)

{

FuncTable.Columns.Add(i.ToString(), before[0][i].ToString() + before[1][i].ToString());

FuncTable.Columns[i].SortMode = DataGridViewColumnSortMode.NotSortable;

}

// Заполнение функциональной таблицы.

for (int i = 0; i < after.Length; i++)

{

FuncTable.Rows.Add();

FuncTable.Rows[i].HeaderCell.Value = i.ToString();

for (int j = 0; j < after[i].Length; j++) FuncTable.Rows[i].Cells[j].Value = (after[i][j] == "-" ? "" : "q") + after[i][j];

}

}

private void SourceWord\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar != 8 && e.KeyChar != 13)

{

int i = 0;

// Проверяем, принадлежит ли текущий символ алфавиту.

while (i < alphabet.Length && e.KeyChar != alphabet[i]) i++;

if (i == alphabet.Length)

e.Handled = true;

}

else if (e.KeyChar == 13) Solve\_Click(sender, e); // По нажатию "Enter" запускаем МТ.

}

private void Solve\_Click(object sender, EventArgs e)

{

string sourceWord = "\_" + SourceWord.Text + "\_";

string resultWord = "\_\_";

int currState = 0; // Текушее состояние.

int index1 = 1; // Индекс символа, рассматриваемого на первой ленте.

int index2 = 1; // Индекс символа, рассматриваемого на второй ленте.

int i = 0; // Переменная для прохода по массиву заменяемых симловов.

AlgorithmSteps.Rows.Clear();

AlgorithmSteps.Rows.Add();

AlgorithmSteps.Rows[0].Cells[0].Value = sourceWord.Substring(0, index1) + "q" + currState + sourceWord.Substring(index1);

AlgorithmSteps.Rows[0].Cells[1].Value = resultWord.Substring(0, index2) + "q" + currState + resultWord.Substring(index2);

while (currState != -1)

{

i = 0;

// Ищем неходимую комбинацию символов на двух лентвх.

while (i < 13 && (sourceWord[index1] != before[0][i] || resultWord[index2] != before[1][i]))

i++;

if (i == 13 || after[currState][i] == "-")

{

MessageBox.Show("В алгоритме присутствует ошибка!", "", MessageBoxButtons.OK, MessageBoxIcon.Error);

AlgorithmSteps.Rows.Clear();

return;

}

else

{

if (sourceWord[index1] == before[0][i] && resultWord[index2] == before[1][i])

{

// Если заменяемые символы на лентах отличаются от исходных,

// производим замену.

if (before[0][i] != after[currState][i][1])

{

sourceWord = sourceWord.Remove(index1, 1);

sourceWord = sourceWord.Insert(index1, after[currState][i][1].ToString());

}

if (before[1][i] != after[currState][i][2])

{

resultWord = resultWord.Remove(index2, 1);

resultWord = resultWord.Insert(index2, after[currState][i][2].ToString()) + "\_";

}

// Смещаем управляющее устройство (индексы просматриваемых символов слова на каждой ленте).

switch (after[currState][i][3])

{

case 'R': index1++; break;

case 'L': index1--; break;

}

switch (after[currState][i][4])

{

case 'R': index2++; break;

case 'L': index2--; break;

}

// Изменяем состояние машины Тьюринга.

if (after[currState][i][0] != 'z')

currState = Convert.ToInt32(after[currState][i][0].ToString());

else currState = -1;

// Выводим текущее состояние слова в таблицу шагов.

AlgorithmSteps.Rows.Add();

AlgorithmSteps.Rows[AlgorithmSteps.RowCount - 1].Cells[0].Value = sourceWord.Substring(0, index1) + "q" + (currState == -1 ? "z" : currState.ToString()) + sourceWord.Substring(index1);

AlgorithmSteps.Rows[AlgorithmSteps.RowCount - 1].Cells[1].Value =

resultWord.Substring(0, index2) + "q" + (currState == -1 ? "z" : currState.ToString()) + resultWord.Substring(index2);

}

}

}

// Запись шагов алгоритма в файл.

try

{

FileStream file = new FileStream(Application.StartupPath + "\\" + (SourceWord.Text == "" ? "Пустое слово" : SourceWord.Text) + ".txt", FileMode.Create, FileAccess.Write);

StreamWriter streamWriter = new StreamWriter(file);

for (int j = 0; j < AlgorithmSteps.RowCount; j++)

{

streamWriter.WriteLine(AlgorithmSteps.Rows[j].Cells[0].Value);

streamWriter.WriteLine(AlgorithmSteps.Rows[j].Cells[1].Value);

streamWriter.WriteLine("-----------------------------------");

}

streamWriter.Close();

MessageBox.Show("Шаги алгоритма успешно записаны в файл по следующему пути:\n" + file.Name, "", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show("Обнаружена ошибка при записи шагов алгоритма в файл!\n" + ex.StackTrace + "\n\nПричина: " + ex.Message, "",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void TimeComplexity\_Click(object sender, EventArgs e)

{

form2.ShowDialog();

}

}

}

// Файд Form2.cs.

using System;

using System.ComponentModel;

using System.Text;

using System.Threading;

using System.Windows.Forms;

namespace Курсовая\_работа\_по\_машине\_Тьюринга

{

public partial class Form2 : Form

{

public Form2(char[][] arrBefore, string[][] arrAfter, char[] arrAlphabet)

{

InitializeComponent();

before = arrBefore;

after = arrAfter;

alphabet = arrAlphabet;

x = 0;

Graph.Series[0].Points.AddXY(0, 0);

}

char[] alphabet;

char[][] before;

string[][] after;

int x;

DateTime startTime;

private int TuringMachine(string word)

{

string sourceWord = "\_" + word + "\_";

string resultWord = "\_\_";

int currState = 0; // Текушее состояние.

int index1 = 1; // Индекс символа, рассматриваемого на первой ленте.

int index2 = 1; // Индекс символа, рассматриваемого на второй ленте.

int i; // Переменная для прохода по массиву заменяемых симловов.

int tactNumber = 0;

while (currState != -1)

{

if (backgroundWorker.CancellationPending == true) // Нажата кнопка "Стоп".

return -1;

i = 0;

// Ищем неходимую комбинацию символов на двух лентвх.

while (i < 13 && (sourceWord[index1] != before[0][i] || resultWord[index2] != before[1][i]))

i++;

if (i == 13 || after[currState][i] == "-")

{

MessageBox.Show("В алгоритме присутствует ошибка!", "", MessageBoxButtons.OK, MessageBoxIcon.Error);

return -1;

}

else

{

if (sourceWord[index1] == before[0][i] && resultWord[index2] == before[1][i])

{

// Если заменяемые символы на лентах отличаются от исходных,

// производим замену.

if (before[0][i] != after[currState][i][1])

{

sourceWord = sourceWord.Remove(index1, 1);

sourceWord = sourceWord.Insert(index1, after[currState][i][1].ToString());

}

if (before[1][i] != after[currState][i][2])

{

resultWord = resultWord.Remove(index2, 1);

resultWord = resultWord.Insert(index2, after[currState][i][2].ToString()) + "\_";

}

// Смещаем управляющее устройство (индексы просматриваемых символов слова на каждой ленте).

switch (after[currState][i][3])

{

case 'R': index1++; break;

case 'L': index1--; break;

}

switch (after[currState][i][4])

{

case 'R': index2++; break;

case 'L': index2--; break;

}

// Изменяем состояние машины Тьюринга.

if (after[currState][i][0] != 'z')

currState = Convert.ToInt32(after[currState][i][0].ToString());

else currState = -1;

}

}

tactNumber++; // Увеличиваем значение количества тактов.

}

return tactNumber;

}

private int WordGeneration(int wordLength)

{

uint letterCount = (uint)alphabet.Length; // Количество символов в алфавите.

StringBuilder str = new StringBuilder(new string(alphabet[0], wordLength)); // Сгенерированное слово.

ulong n = 1, temp; // Порядковый номер текущего слова и переменная для генерации слова.

ulong strCount = (ulong)Math.Pow(letterCount, wordLength); // Всего слов перебора.

int maxTacts = 0; // Максимальное число тактов.

int tempTact = 0; // Текущее число тактов.

int progressBarValue = (int)(100 / strCount); // Текущее значение progressBar'a.

int progressIncrease = progressBarValue > 0 ? progressBarValue : 2; // Шаг progressBar'a.

while (n <= strCount)

{

tempTact = TuringMachine(str.ToString());

if (tempTact == -1) return -1; // В алгоритме ошибка или нажата кнопка "Стоп".

if (tempTact > maxTacts)

maxTacts = tempTact;

// Изменяем значение progressBar'a.

if ((int)(n \* 100 / strCount) == progressBarValue)

{

backgroundWorker.ReportProgress(-progressBarValue);

progressBarValue += progressIncrease;

}

// Генерируем новое слово методом полного перебора.

if (n != strCount)

{

temp = n;

for (int i = wordLength - 1; i >= 0; i--)

{

str[i] = alphabet[temp % letterCount];

temp /= letterCount;

}

}

n++;

}

return maxTacts;

}

private void Start\_Click(object sender, EventArgs e)

{

Stop.Enabled = true;

Start.Enabled = false;

Graph.Series[0].Points.Clear(); // Очичаем график.

startTime = DateTime.Now;

backgroundWorker.RunWorkerAsync(); // Запускаем второй поток для отрисовки графика и перебора слов.

}

private void backgroundWorker\_DoWork(object sender, DoWorkEventArgs e)

{

int length = (int)WordLength.Value;

x = 0; // Рисуем график с нуля.

int y = 0; // Максимальное число тактов (значения по оси Оy).

while (x <= length && y != -1)

{

y = WordGeneration(x);

if (y != -1) // y = -1, если в алгоритме ошибка или нажата кнопка "Стоп".

{

if(x > 4) Thread.Sleep(450); // Подождать, чтоб отрисовался progressBar.

backgroundWorker.ReportProgress(y); // Добляем значение на график.

// Подождать, чтоб точка на график добавилась перед тем, как изменится значение x.

if (x < 4) Thread.Sleep(700);

else Thread.Sleep(500);

x++;

}

}

}

private void Stop\_Click(object sender, EventArgs e)

{

backgroundWorker.CancelAsync(); // Отменяем отрисовку графика и перебор слов.

}

private void backgroundWorker\_ProgressChanged(object sender, ProgressChangedEventArgs e)

{

// Положительное значение e.ProgressPercentage - координата y для добаления точки на график,

// отрицательное (показывает, что изменился прогресс генерации слов) - процент progressBar'a.

if (e.ProgressPercentage > 0)

{

if (x < 4) progressBar.Value = 100;

Graph.Series[0].Points.AddXY(x, e.ProgressPercentage);

}

else progressBar.Value = -e.ProgressPercentage;

}

private void backgroundWorker\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)

{

Stop.Enabled = false;

Start.Enabled = true;

progressBar.Value = 100;

TimeSpan resultTime = DateTime.Now - startTime;

MessageBox.Show("График отрисовался! Перебор слов на " + (x - 1) + " символов закончен.\n" +

"Затраченное время: " +

Convert.ToInt32(resultTime.TotalHours).ToString() + " ч " +

resultTime.Minutes.ToString() + " м " +

resultTime.Seconds.ToString() + " c " +

resultTime.Milliseconds.ToString() + " мс");

progressBar.Value = 0;

}

}

}

# Приложение Д

# Экранные формы

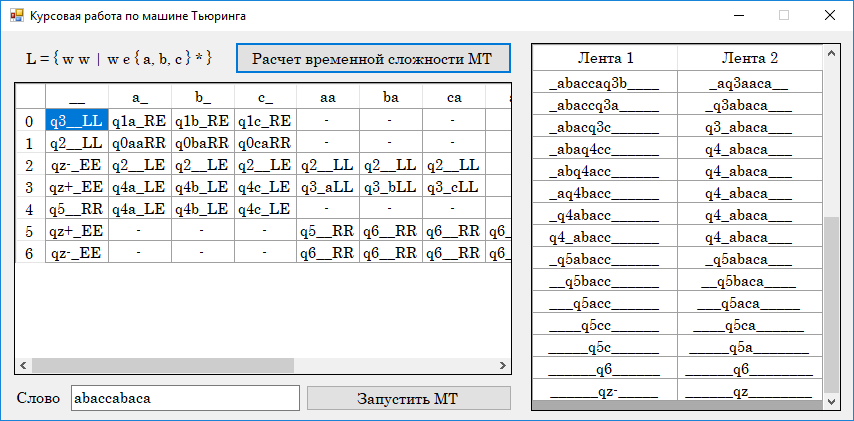


Рисунок Д.1 – Главное окна программы

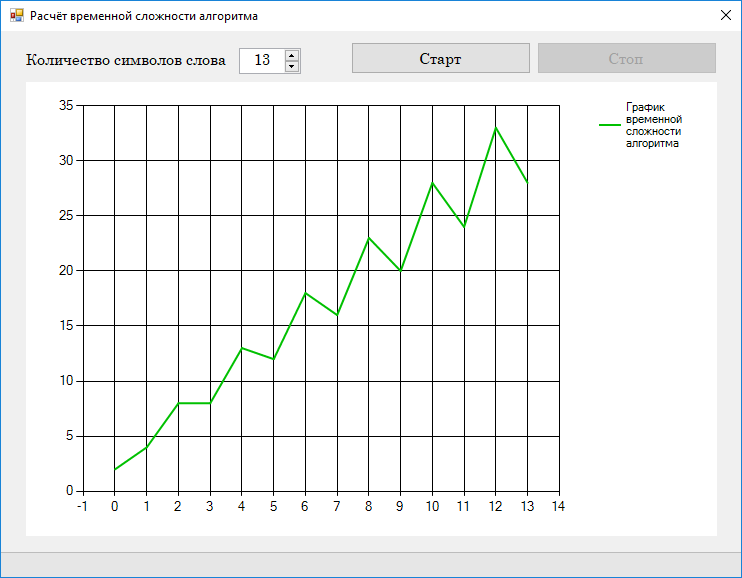


Рисунок Д.2 – График временной сложности алгоритма

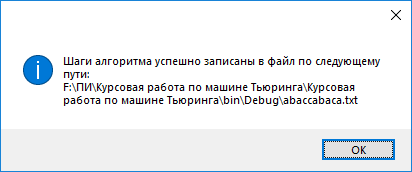


Рисунок Д.3 – Сообщение об успешной записи шагов алгоритма в файл

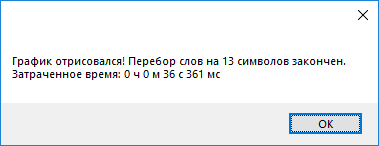


Рисунок Д.4 – Сообщение об окончании построения графика