МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

Донецкий Национальный Технический Университет

КАФЕДРА ПРОГРАМНОЙ ИНЖЕНЕРИИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по дисциплине

«Теория алгоритмов и функциональных языков»

на тему: «Построение аналитических моделей алгоритмов и

оценка их сложности»

|  |  |
| --- | --- |
| Руководитель:  кафедры ПИ | Выполнил:  Студент гр. ПИ18б  Куркурин Н.Л. |

Донецк – 2019

Реферат

Отчет по курсовой работе содержит: страниц, 8 рисунков, 4 таблицы, 4 приложения, 4 источника.

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

Цель – сформировать формальное определение алгоритма в виде трех аналитических моделей, написать программную реализацию машины Тьюринга, распознающей язык L = { wwRw∈{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 = { wwRw∈{a, b, c}\*}, график временной сложности машины Тьюринга, файловый вариант протокола работы машины Тьюринга.

МАШИНА ТЬЮРИНГА, ВРЕМЕННАЯ СЛОЖНОСТЬ, АЛФАВИТ, ЛЕНТА, ЯЗЫК, РАСПОЗНАВАНИЕ, ПРОТОКОЛ, ПРИНАДЛЕЖНОСТЬ

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc26727598)

[1. ОПИСАНИЕ АНАЛИТИЧЕСКОЕ МОДЕЛИ АЛГОРИТМА В ВИДЕ ЭЛЕМЕНТАРНЫХ МАШИН ТЬЮРИНГА И КОМПОЗИЦИЙ МТ 5](#_Toc26727599)

[1.1 Машины Тьюринга 5](#_Toc26727600)

[1.1.1 Поставнока задачи 5](#_Toc26727601)

[1.1.2 Представление машини Тьюринга в разных формах (функциональная таблица, система команд, диаграмма переходов) 5](#_Toc26727602)

[1.1.3 Контрольные примеры: 6](#_Toc26727603)

[1.2 Композиция машин Тьюринга 7](#_Toc26727604)

[1.2.1 Постановка задачи 7](#_Toc26727605)

[1.2.2 Представление композиции машин Тьюринга 7](#_Toc26727606)

[1.2.3 Композиция машин Тьюринга 7](#_Toc26727607)

[2. РАЗРАБОТКА АНАЛИТИЧЕСКОЙ И ПРОГРАММНОЙ МОДЕЛИ АЛГОРИТМА ДЛЯ РАСПОЗНАЮЩЕЙ МАШИНЫ ТЬЮРИНГА 9](#_Toc26727608)

[2.1 Формальное определение машины, распознающей Тьюринга 9](#_Toc26727609)

[2.2 Протоколы работы машины Тьюринга на двух лентах 10](#_Toc26727610)

[2.3 Программная модель машины Тьюринга 12](#_Toc26727611)

[2.4 Протоколы работы машины Тьюринга, построенные программно (на двух словах языка и двух словах, не принадлежащих языку) 13](#_Toc26727612)

[2.5 Расчет временной сложности (график функции временной сложности) 16](#_Toc26727613)

[3. РАЗРАБОТКА АНАЛИТИЧЕСКОЙ МОДЕЛИ АЛГОРИТМА С ИСПОЛЬЗОВАНИЕМ НОРМАЛЬНЫХ АЛГОРИТМОВ МАРКОВА 17](#_Toc26727614)

[3.1 Постановка задачи 17](#_Toc26727615)

[3.2 Алгоритм 17](#_Toc26727616)

[3.3 Контрольные примеры 17](#_Toc26727617)

[ВЫВОДЫ 18](#_Toc26727618)

[ПРИЛОЖЕНИЕ А. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 20](#_Toc26727619)

[ПРИЛОЖЕНИЕ В. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 21](#_Toc26727620)

[ПРИЛОЖЕНИЕ В. ЛИСТИНГ ПРОГРАММЫ 22](#_Toc26727621)

[ПРИЛОЖЕНИЕ Г. ЭКРАННЫЕ ФОРМЫ 30](#_Toc26727622)

ВВЕДЕНИЕ

Теория алгоритмов - предмет, который изучает вопросы существования алгоритмов для решения некоторой задачи и выбора наилучшего из существующих.

Под алгоритмом принято понимать конечную последовательность элементарных операций, исполнение которой приводит к решению любой задачи из заданного множества задач. В это определение входят такие свойства алгоритма, как дискретность, конечность (конечное число выполняемых операций), массовость (решается не единственная задача, а их класс), результативность (в результате получаем решение задачи). Кроме того, должно выполнятся ещё одно необходимое свойство алгоритма – детерминизм, т.е. однозначное понимание каждой операции, или, что то же самое, независимость результата выполнения каждой элементарной операции от того, кто её выполняет.  
 Существуют методы для решения множества универсальных задач, считающиеся универсальными. Они включают: машины Тьюринга, нормальные алгоритмы Маркова, рекурсивные функции.

Актуальность состоит в необходимости решения множества задач.

Задача - написание программы, которая будет реализовать машины Тьюринга определенного языка.  
 Цель работы - написание программы, реализующей работу с данным алфавитом.  
 Область применения программы - программу можно применять в высших учебных заведениях с образовательной целью и для проверки корректности расчетов.

# 1. ОПИСАНИЕ АНАЛИТИЧЕСКОЕ МОДЕЛИ АЛГОРИТМА В ВИДЕ ЭЛЕМЕНТАРНЫХ МАШИН ТЬЮРИНГА И КОМПОЗИЦИЙ МТ

## 1.1 Машины Тьюринга

### 1.1.1 Поставнока задачи

Реализовать функцию  над числами в унарном коде.f

### 1.1.2 Представление машини Тьюринга в разных формах (функциональная таблица, система команд, диаграмма переходов)

Функциональная таблица представлена на рис. 1.1

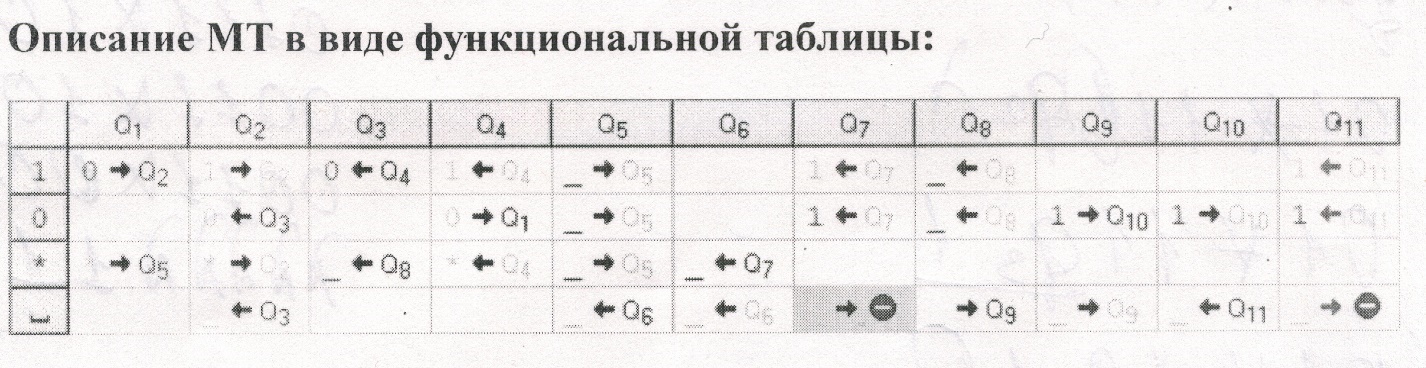


Рисунок 1.1 – Функциональная таблица

Система команд представлена на рис 1.2

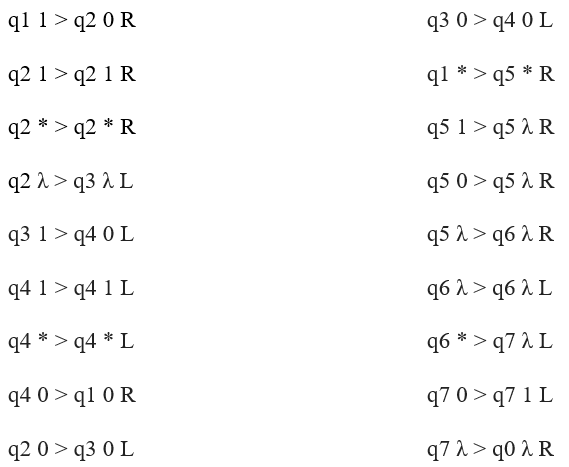


Рисунок 1.2 – Система команд

Диаграмма переходов представлена на рис. 1.3

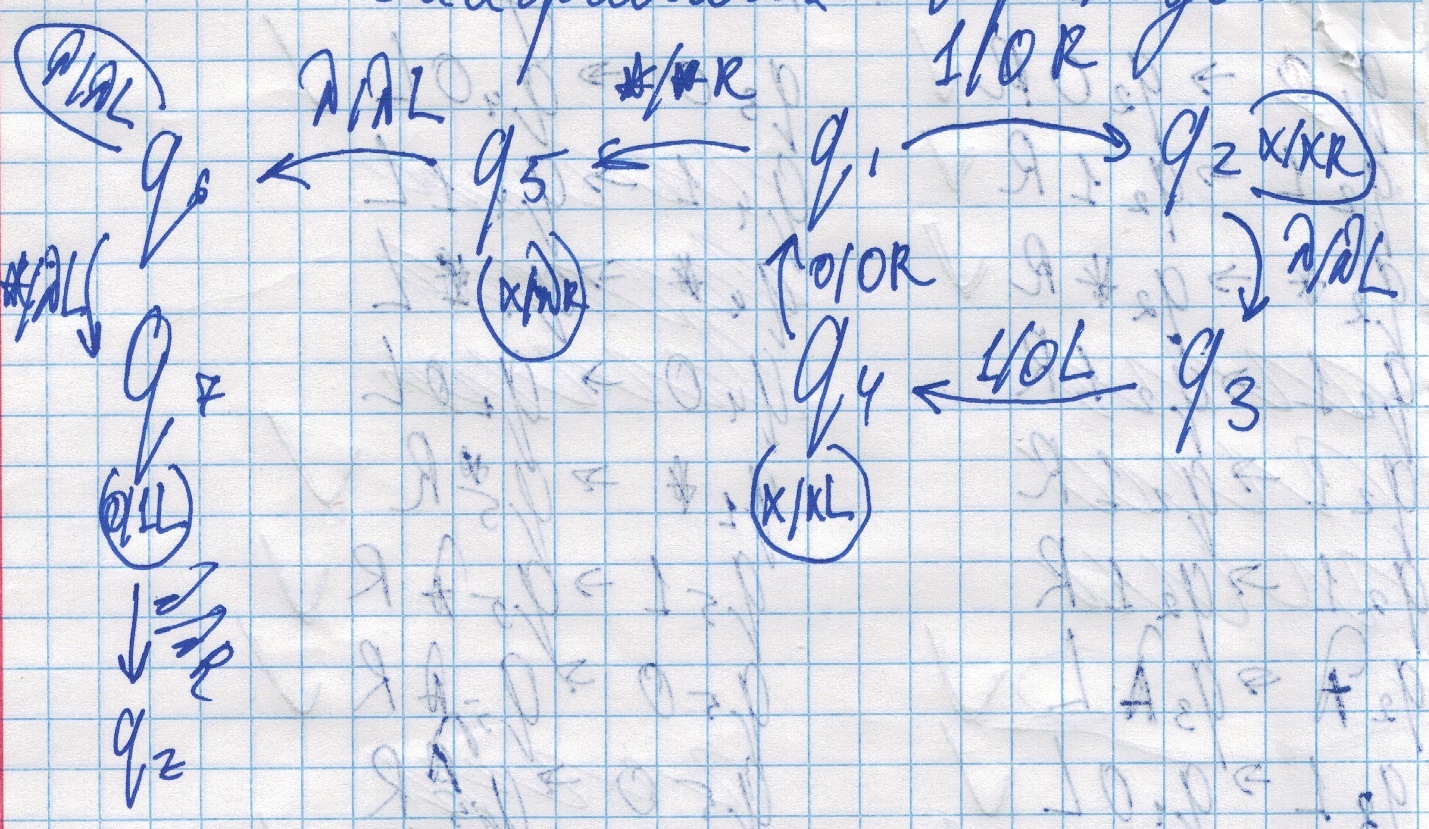


Рисунок 1.3 – Диаграмма переходов

### 1.1.3 Контрольные примеры:

11\*111 > 01\*111 > 01\*110 > 00 \* 100 > 11

1111 \* 11 > 0111 \* 11 > 0111 \* 11 > 0111 \* 10 > 0011 \* 10 > 0011 \* 00 > 11

## 1.2 Композиция машин Тьюринга

### 1.2.1 Постановка задачи

Максимальная цифра в 16-ричной записи числа .

### 1.2.2 Представление композиции машин Тьюринга

Представление композиции в виде блоксхемы на рис. 1.4

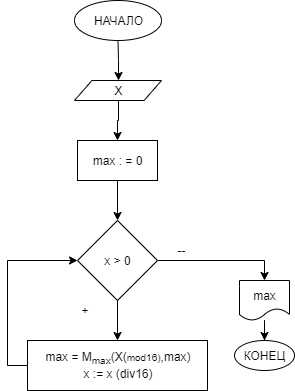


Рисунок 1.4 – Представление композиции в виде блоксхем

Некоторое пояснение:

X – наблюдаемое число (0-f)

max – максимальное число

### 1.2.3 Композиция машин Тьюринга

М0 – мт, реализующая копирование входного слова;

М0 – мт, реализующая функцию установки константы 0;

Мmax – мт, которая из двух выбираем максимальный;

M>0 – мт, вычисляет предикат с восстановлением.

Композиция представлена на рис 1.5

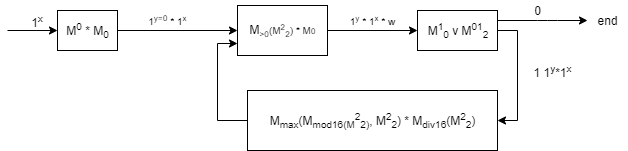


Рисунок 1.5 – Композиция машин Тьюринга

# 2. РАЗРАБОТКА АНАЛИТИЧЕСКОЙ И ПРОГРАММНОЙ МОДЕЛИ АЛГОРИТМА ДЛЯ РАСПОЗНАЮЩЕЙ МАШИНЫ ТЬЮРИНГА

Рассмотрим реализацию машины Тьюринга для языка L = { wwRw∈{a, b, c}\*}, способ реализации в программе, построим график временной сложности. Пустая ячейка на ленте обозначается ‘L’.

# 2.1 Формальное определение машины, распознающей Тьюринга

Суть машины заключается в том, что нужно проверить входное значение на соответствие условию: входное слово, состоящее из алфавита ‘a’, ‘b’, ‘c’ нужно разделить на два равных по длине слова, причем второе слово должно представлять собой реверсию первого. В случае если входные данные удовлетворяют условию задачи программа выводит: ‘1’, иначе ‘0’.

Для реализации данного алгоритма на одноленточной машине Тьюринга составлена система команд, представленная на рисунке 2.1.

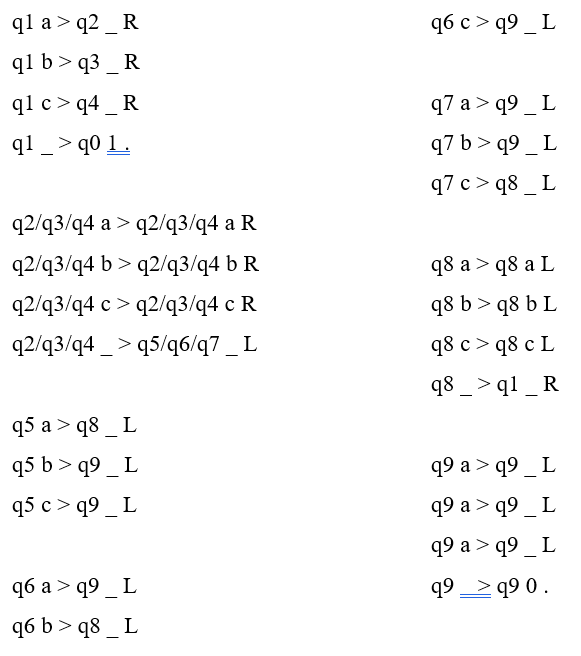


Рисунок 2.1 – Система команд для алгоритма проверки второго слова на реверс первого

Комментарии к таблице:

Для решения поставленной задачи использовано 9 состояний:

* q1 – затираем одну из букв, A B или C и идем в соответствующее состояние: A – q2, B – q3, C – q4; если же не одной из букв не найдено, то ставим один и завершаем работу программы.
* q2, q3, q4 – двигаемся до конца строки вправо до лямбды
* q5, q6, q7 – затираем соответственный символ (A, B, C) и переходим в q8 если затерли соответствующую букву, если нет переходим в q9
* q8 – двигаемся до конца строки влево до лямбды
* q9 – двигаемся до конца строки влево стирая всё на пути, после устанавливая ноль и завершаем работу программы.

Для реализации данного алгоритма на многоленточной машине Тьюринга составлена система команд. (см. рис 2.2)

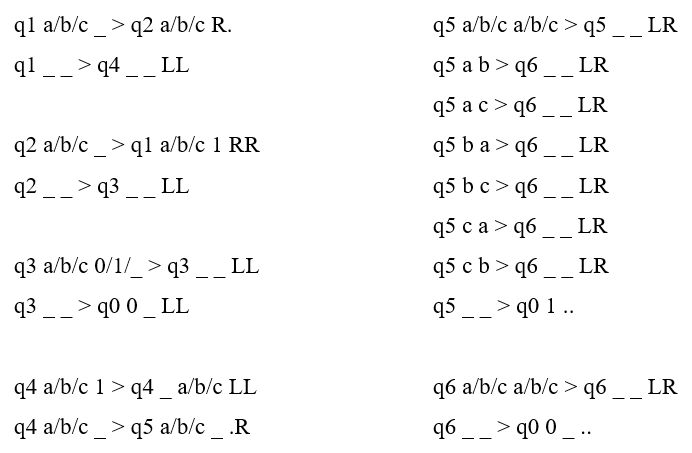


Рисунок 2.2 – Реализация алгоритма на многоленточной машине

# 2.2 Протоколы работы машины Тьюринга на двух лентах

Рассмотрим контрольные примеры для машины Тьюринга, реализовывающей работу с данным алфавитом.

Протоколы работы над словами, принадлежащими языку: “abccba” для одноленточной, и для многоленточной (Табл. 2.2 –Табл. 2.3).

Таблица 2.2 – протокол работы одноленточной машины над словом abccba

|  |  |  |  |
| --- | --- | --- | --- |
| К0= | **\_a**bccba**\_** | К4= | **\_\_\_c**c**\_\_\_** |
| К1= | **\_\_**bccb**a\_** | К5= | **\_\_\_c\_\_\_** |
| К2= | **\_\_b**ccb**\_\_** | К6= | **\_\_\_\_\_\_\_** |
| К3= | **\_\_\_**cc**b\_\_** | К7= | **\_\_1\_\_\_\_** |

Таблица 2.3 – протокол работы многоленточной машины над словом abccba

|  |  |  |  |
| --- | --- | --- | --- |
| К0= | **\_ab**ccba**\_** | К4= | **\_**abcc**b\_** |
| **\_**1**\_** | **\_**1**b**a**\_** |
| К1= | **\_**ab**cc**ba**\_** | К5= | **\_**abc**c\_** |
| **\_**11**\_** | **\_c**ba**\_** |
| К2= | **\_**abcc**ba\_** | К6= | **\_**ab**c\_** |
| **\_**111**\_** | **\_c**ba**\_** |
| К3= | **\_**abccb**a\_** | К7= | **\_**a**b\_** |
| **\_**11**a\_** | **\_b**a**\_** |
|  | | K8= | **\_a\_** |
| **\_a\_** |
| K9= | **\_**1**\_** |
| **\_\_** |

Рассмотрим протоколы работы над словами, не принадлежащими языку: “abbaa” для одноленточной и многоленточной (Табл. 2.4 –Табл. 2.5).

Таблица 2.4 – реализация алгоритма на одноленточной машине

|  |  |  |  |
| --- | --- | --- | --- |
| К0= | **\_a**bbaa**\_** | К4= | **\_b\_** |
| К1= | **\_**bba**a\_** | K5= | **\_\_** |
| К2= | **\_b**ba**\_** | K6= | **\_0\_** |
| К3= | **\_**b**a\_** |  |  |

Таблица 2.5 – реализация алгоритма на многоленточной машине

|  |  |  |  |
| --- | --- | --- | --- |
| К0= | **\_ab**baa**\_** | K3= | **\_**abb**a\_** |
| **\_**1**\_** | **\_\_** |
| К1= | **\_**ab**ba**a**\_** | K4= | **\_**ab**b\_** |
| **\_**11**\_** | **\_\_** |
| К2= | **\_**abba**a\_** | K5= | **\_**a**b\_** |
| \_1**­\_\_** | **\_\_** |
|  | | K6= | **\_a\_** |
| **\_\_** |
| K7= | **\_0\_** |
| **\_\_** |

# 2.3 Программная модель машины Тьюринга

В программе две ленты заданы строками, создан абстрактный класс mt от которого наследуется основная машина. В классе основной машины есть переменные:

* letter, в которую записан элемент, на который указывает головка;
* direction, в каком направлении двигаться машине;
* state, в каком состоянии машина Тьюринга;
* cursor, ссылка на ячейку, на которой стоит головка.

В машине Тьюринга есть метод heart, в котором прописана логика работы машины, а именно, это бесконечный цикл, мы переходим в определенное состояние, где меняем или не меняем будущее состояние, букву на которой находимся, направление движения.

Преимущество данной реализации в красоте и понятности кода, а также в производительности и быстродействии.

Для того, чтобы иметь возможность лицезреть график, он спроектирован в отдельном окне с помощью отдельного потока, генератор также сделан в отдельном потоке, с целью не утруждать основной поток в генерации большого кол-ва тестовых слов.

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

Для вывода листинга пошаговой работы программы был использован файл log.txt. Рассмотрим пример логирования входного слова abbbbbba, удовлетворяющего условию на одноленточной машине. (см. рис. 2.3).

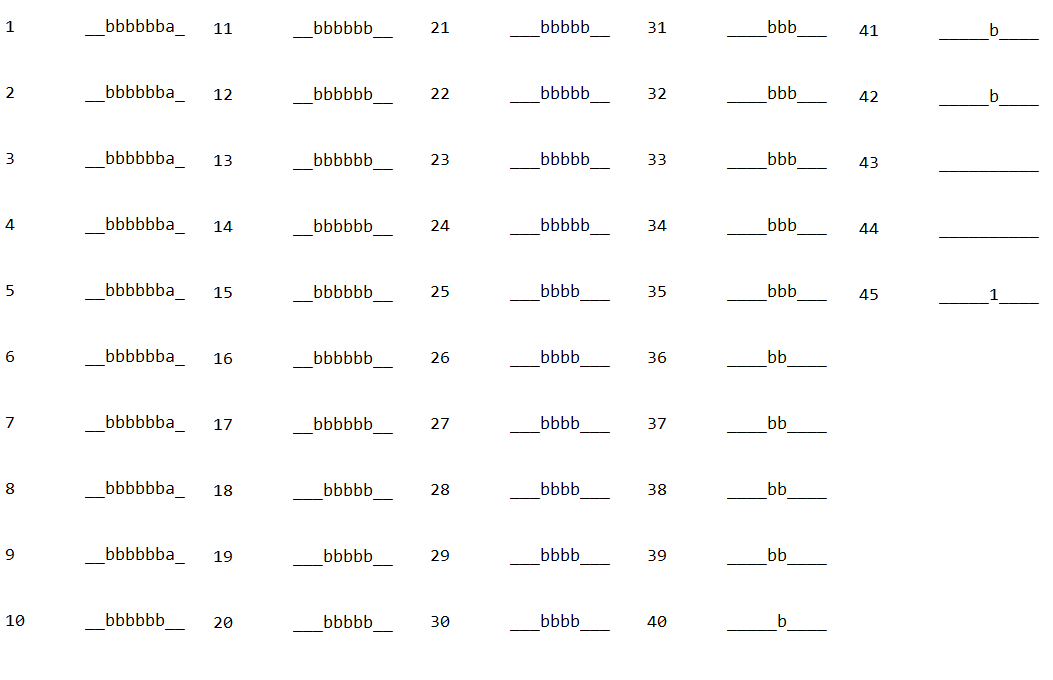


Рисунок 2.3 – Логгинг работы программы над словом “abbbbbba”

Рассмотрим слово, удовлетворяющее условию, слово “bbabbbbabb” на многоленточной машине. (см. рис. 2.4).

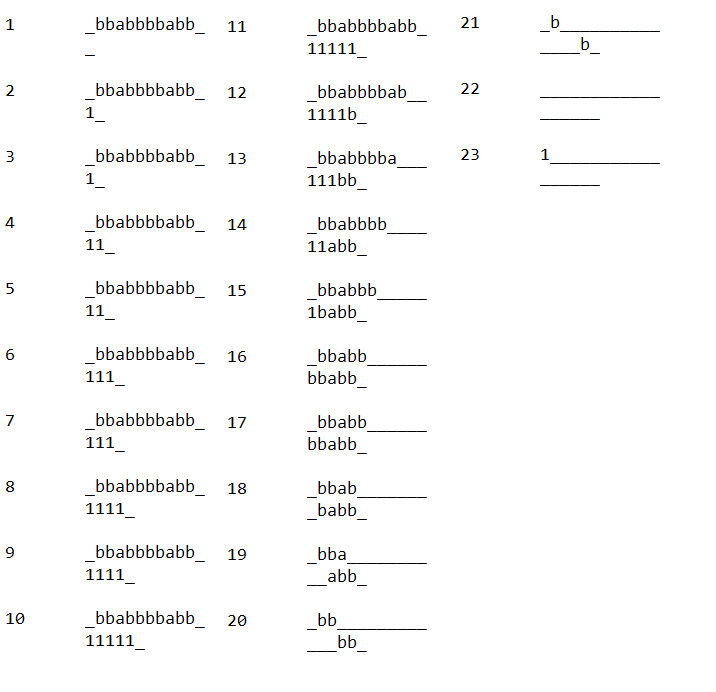


Рисунок 2.4 – Логгинг работы программы над словом “bbabbbbabb”

Рассмотрим слово, не удовлетворяющее условию “abccaaaabaca” на одноленточной машине. (см. рис. 2.5)

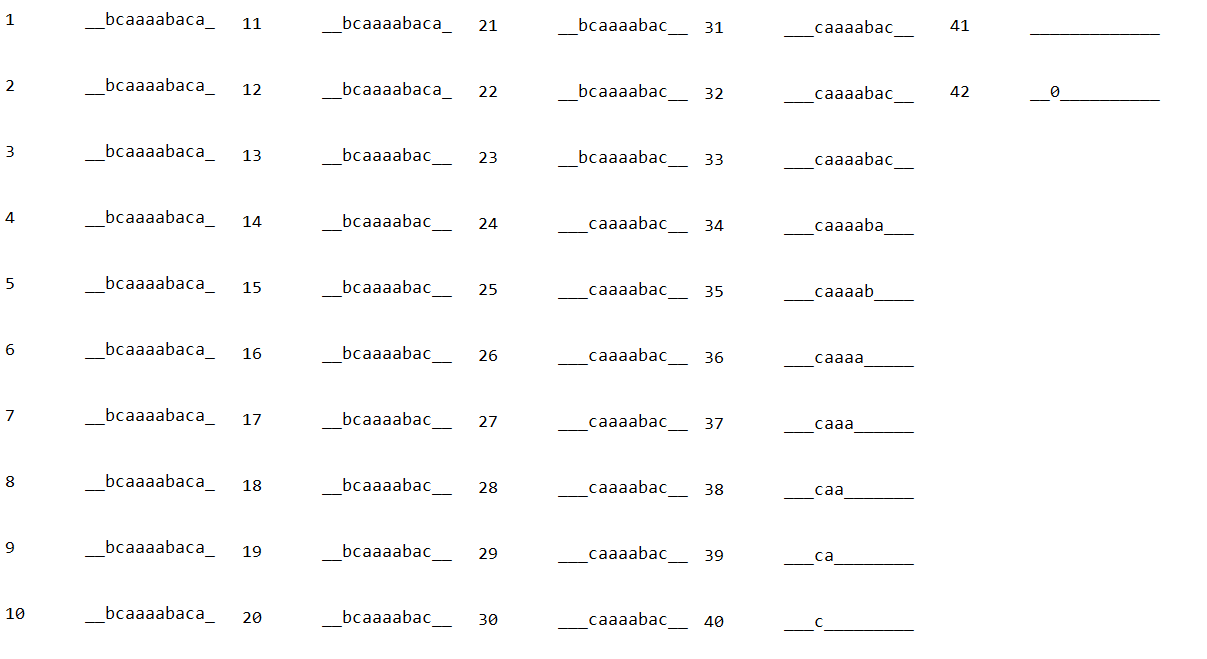


Рисунок 2.5 – Логгинг работы программы над словом “abccaaaabaca”

Рассмотрим слово, не удовлетворяющее условию “abccaaaabaca” на многоленточной машине. (см. рис. 2.6)

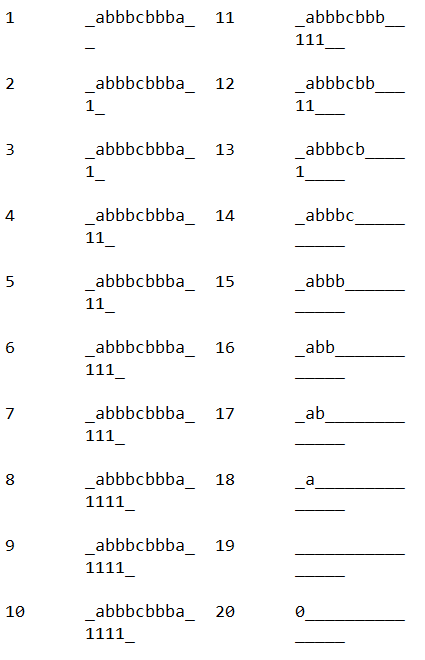


Рисунок 2.6 – Логгинг работы программы над словом “abccaaaabaca”

Как можно увидеть, алгоритм максимально минимизирован, и выдает либо 1 если слово удовлетворяет условию, либо 0 если слово не удовлетворяет условию.

# 2.5 Расчет временной сложности (график функции временной сложности)

Так же при запуске программы создается два дополнительных потока: один поток выделен для построения графика временной сложности, второй – для генерации бесконечного множества значений, удовлетворяющих условию. Работа графика представлена на рис. 2.7. Черным цветом выделена прямая временной сложности многоленточной машины, красным – одноленточной.

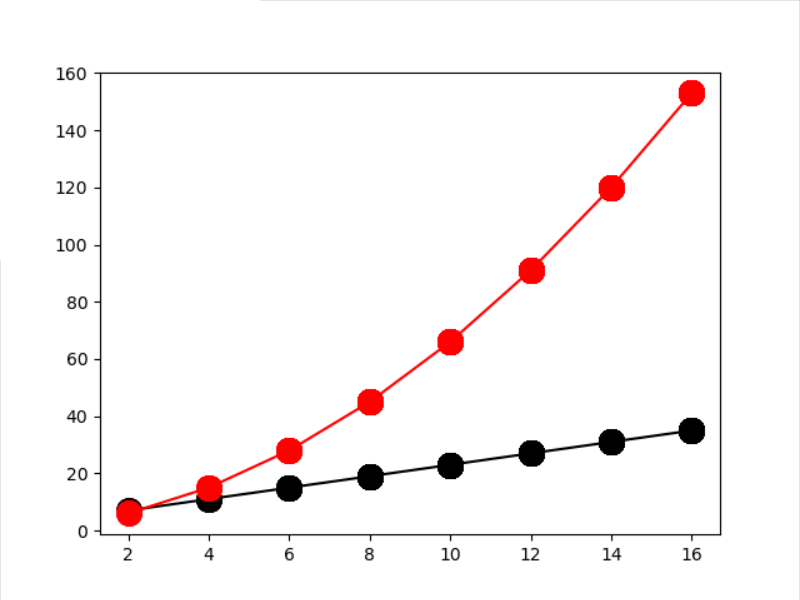


Рисунок 2.7 – График временной сложности

# 3. РАЗРАБОТКА АНАЛИТИЧЕСКОЙ МОДЕЛИ АЛГОРИТМА С ИСПОЛЬЗОВАНИЕМ НОРМАЛЬНЫХ АЛГОРИТМОВ МАРКОВА

## 3.1 Постановка задачи

Составить нормальный алгоритм, преобразующий исходную строку в алфавите  в строку, в которой буквы расположены в алфавитном порядке.

## 3.2 Алгоритм

Замены для данного алгоритма представлены на рис. 3.1

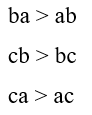


Рисунок 3.1 – Замены для поставленной задачи

## 3.3 Контрольные примеры

Контрольные примеры для данного алгоритма представлены на рис. 3.2

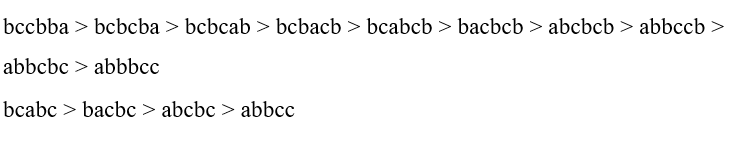


Рисунок 3.2 – Контрольные примеры

# ВЫВОДЫ

Результат работы представляет собой программу, выполняющую алгоритм над словами заданного языка.  
 Достоинствами программы является выполнение задания для заданного слова в двух режимах (на одноленточной и многоленточной машине Тьюринга), а также одновременная генерация слов разной длины, на основании чего строится график временной сложности.  
 Недостатки в данной программе не были замечены. Возможно улучшить программу за счет обновления интерфейса и добавления новых функций.

Список литературы:

1. А. А. Марков, Теория Алгоритмов, Труды матем. ин-та им. Стеклова, т. XLII, Изд-во Академии наук СССР, Москва 1954 г.
2. Р. Петер, Рекурсивные функции, Москва, 1954 г.
3. А. И. Китов, Электронные вычислительные машины, Изд-во Советское радио, Москва, 1956 г.
4. С. Клини, Введение в математику, ИЛ, Москва 1956 г.

# ПРИЛОЖЕНИЕ А. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

# ПРИЛОЖЕНИЕ В. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При входе в программу пользователю представляется два окна: первое окно имеет поле ввода и две кнопки («одноленточная» и «многоленточная»); второе окно является графиком временной сложности, который работает независимо от действий пользователя.

Для того, чтобы ввести данные для проверки, необходимо нажать на поле ввода мышью, и ввести слово. Затем нажать на одну из кнопок ниже. Кнопка «одноленточная» запустит проверку слова на одноленточной машине и выведет результат: слово удовлетворяет условию или нет. Также, после этого будет выведен результат в файлы log.txt (если машина была одноленточная) или multi\_log.txt (если машина была многоленточная). На график будет нанесен маркер соответствующего цвета (если машина одноленточная – красный, если многоленточная – черный).

# ПРИЛОЖЕНИЕ В. ЛИСТИНГ ПРОГРАММЫ

***main.py***

import re  
import threading  
import time  
  
import animation1  
import functional  
from kivy.config import Config  
from kivy.app import App  
from kivy.uix.boxlayout import BoxLayout  
from kivy.uix.button import Button  
from kivy.uix.textinput import TextInput  
Config.set('graphics', 'left', '28')  
Config.set('graphics', 'top', '50')  
Config.set('graphics', 'position', 'custom')  
Config.set('kivy', 'exit\_on\_escape', '1')  
Config.set('graphics', 'height', '517')  
Config.set('graphics', 'width', '300')  
Config.set('graphics', 'resizable', '0')  
  
  
class AlfabInput(TextInput): # переписанный текст инпут, для того чтоб вводили только a b или c  
 pat = re.compile('[^a-c]') # патерн, сюда вводим значения которые нам нужны  
  
 def insert\_text(self, substring, from\_undo=False):  
 pat = self.pat  
 if '.' in self.text:  
 s = re.sub(pat, '', substring)  
 else:  
 s = '.'.join([re.sub(pat, '', s) for s in substring.split('.', 1)])  
 return super(AlfabInput, self).insert\_text(s, from\_undo=from\_undo)  
  
  
class MyApp(App):  
 def build(self):  
 def run\_the\_task(instance):  
 if not text\_input.text or text\_input.text.find('.') > -1: # если введено что-то не то  
 text\_input.text = ''  
 text\_input.hint\_text\_color = [1, 0, 0, 1]  
 text\_input.hint\_text = 'Введите НОРМАЛЬНОЕ слово.'  
 else:  
 for name\_of\_file in ('log.txt', 'multi\_log.txt'): # чистим файлы после прошлого запуска  
 open(name\_of\_file, 'w').close()  
 text\_input.hint\_text, text\_input.hint\_text\_color = \  
 functional.run(text\_input.text, False, True if int(instance.id) == 1 else False)  
 if int(instance.id) != 1:  
 name\_of\_file = 'log.txt'  
 else:  
 name\_of\_file = 'multi\_log.txt'  
 with open(name\_of\_file, 'r') as f:  
 log\_text\_input.text = f.read()  
 text\_input.text = ''  
  
 box\_layout = BoxLayout(orientation='vertical')  
 text\_input = AlfabInput(hint\_text="Введите желаемое слово:",  
 multiline=False,  
 on\_text\_validate=run\_the\_task,  
 id='0',  
 size\_hint\_y=.1)  
 text\_input.focus = True  
 button = Button(text='Одноленточная',  
 on\_release=run\_the\_task,  
 id='0',  
 size\_hint\_y=.1)  
 button\_multi = Button(text='Многоленточная',  
 on\_release=run\_the\_task,  
 id='1',  
 size\_hint\_y=.1)  
 log\_text\_input = TextInput(hint\_text='Тут будут логи машин.',  
 readonly=True)  
 box\_layout.add\_widget(text\_input)  
 box\_layout.add\_widget(button)  
 box\_layout.add\_widget(button\_multi)  
 box\_layout.add\_widget(log\_text\_input)  
 return box\_layout  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 for name\_of\_file in ('time.txt', 'multi\_time.txt'): # чистим файлы с графиками  
 open(name\_of\_file, 'w').close()  
 threading.Thread(target=functional.generate, daemon=True).start() # создаем в отдельном потоке генератор  
 threading.Thread(target=animation1.draw, daemon=True).start() # создаем в отдельном потоке прорисовку временной трудности  
 MyApp().run() # запускаем основное окно с основной задачей

***functional.py***

from for\_one\_tape import mt\_for\_one\_tape  
from for\_multi\_tape import mt\_for\_multi\_tape  
import time  
  
  
def generate():  
 """  
 Генератор работает так, на вход в функцию plus подается слово,  
 и далее как обычная система сложения в столбик, мы смотрим на полсдений  
 элемент входного слова, и увеличиваем его на 1, то есть если была буква  
 a мы меняем её на b и выходим, тоже самое с b, но с c, мы меняем её на a  
 и просматриваем элемент находящий левее, и увеличиваем его.  
 Пример:  
 aba -> abb # т.к. последняя a, мы её меняем на b  
 abb -> abc # т.к. последняя b, мы меняем её на c  
 abc -> aca # т.к. последняя c, мы меняем её на a, и также затрагиваем букву левее  
 ...  
 ccc -> aaaa # т.к. все буквы c, мы меняем каждую на a и потом добавляем a в начало строки  
 """  
 def plus(word): # генератор слов  
 if word[-1] == 'a': # cba -> cbb  
 return word[:-1] + 'b'  
 elif word[-1] == 'b': # cbb -> cbc  
 return word[:-1] + 'c'  
 elif word[-1] == 'c':  
 word = word[:-1] + 'a' # cbc -> cba\*  
 i = 2 # индекс элемента которого меняем  
 while True:  
 try:  
 if word[-i] == 'a':  
 word = list(word)  
 word[-i] = 'b'  
 return ''.join(word)  
 elif word[-i] == 'b':  
 word = list(word)  
 word[-i] = 'c'  
 return ''.join(word)  
 elif word[-i] == 'c':  
 word = list(word)  
 word[-i] = 'a'  
 i += 1 # переходим на след. букву, если ещё раз выпала c: bcfcCa - > bcfCaa  
 except IndexError: # если слово полностью обновляется (ccc) добавляем новый элемент в начало (aaaa)  
 word = ''.join(word)  
 return 'a' + word  
 word = 'a' # начальное слово  
 while True:  
 time.sleep(.001)  
 run(word + ''.join(reversed(word)), True, False) # заносим сгенерированное слово в функцию решения задачи  
 run(word + ''.join(reversed(word)), True, True) # заносим сгенерированное слово в функцию решения задачи  
 word = plus(word) # генерируем новое слово на основе старого  
  
  
def run(word, bot=True, multitape=True): # word - само слово, bot - генератор, multitape - многоленточность  
 """  
 При входе слово обрамляется шаблоном (L по бокам), ставится стартовая ячейка,  
 уведомляется генератор это или нет, и многоленточность или нет.  
 После того как слово прошло через машины, на выходе мы получаем кол-во шагов,  
 потраченное на получение результата, и ответ, да или нет. Записываем результаты  
 а именно кол-во шагов, и длину слова в файл, оттуда график считывает значения для  
 графиков, всего их 3, многоленточный, одноленточный, и пользовательский, и рисует  
 соответственно график зависимости.  
 """  
 if multitape is True:  
 mt = mt\_for\_multi\_tape()  
 namefile = 'multi\_time.txt'  
 else:  
 mt = mt\_for\_one\_tape()  
 namefile = 'time.txt'  
 mt.heart('\_' + word + '\_', bot=bot)  
 if bot is True: # значения сформированные генератором пишем в файл для графика  
 with open(namefile, 'a') as f:  
 f.write('\n' + str(len(word)) + ',' + str(mt.amount\_of\_steps) + '\n')  
 if mt.result\_word == '0':  
 return 'Слово не подходит', [1, 0, 0, 1]  
 else:  
 return 'Слово подходит', [0, .75, 0, 1]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 pass

***animation.py***

import matplotlib.pyplot as plt  
import matplotlib.animation as animation  
  
  
def draw():  
 fig = plt.figure()  
 plt.get\_current\_fig\_manager().window.wm\_geometry('+320+20') # расположение окна  
 ax1 = fig.add\_subplot(1, 1, 1)  
  
 def animate(i):  
 with open('multi\_time.txt', 'r') as f:  
 pullData = f.read()  
 dataArray = pullData.split('\n') # далее достаем данные  
 xar = list()  
 yar = list()  
 for eachLine in dataArray:  
 if len(eachLine) > 2:  
 x, y = eachLine.split(',')  
 xar.append(int(x)) # засовываем в список иксов  
 yar.append(int(y)) # засовываем в список игриков  
 ax1.clear()  
 ax1.plot(xar, yar, '-ok', markersize=14)  
  
 with open('time.txt', 'r') as f:  
 pullData = f.read()  
  
 dataArray = pullData.split('\n') # далее достаем данные  
 xar.clear()  
 yar.clear()  
 for eachLine in dataArray:  
 if len(eachLine) > 2:  
 x, y = eachLine.split(',')  
 xar.append(int(x)) # засовываем в список иксов  
 yar.append(int(y)) # засовываем в список игриков  
 ax1.plot(xar, yar, '-or', markersize=14)  
  
 ani = animation.FuncAnimation(fig, animate, interval=1000) # строим, график. Интервал 5 секунд  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 draw()

***mt\_for\_one\_tape.py***

class mt\_for\_one\_tape:  
 result\_word = str() # результирующее слово  
 amount\_of\_steps = 0 # кол-во шагов для достижения целей  
 tuple\_alfabet = ('a', 'b', 'c') # кортеж с литерами принадлежащим алфавиту  
  
 state = None # состояние, first\_condition , second,  
 # тем самым меняя указатели на функции  
  
 direction = '>' # направление, в которое двигаемся , может быть >, <, stop  
 letter = str() # буква, которую мы сейчас рассматриваем  
 cursor = 1 # курсор , то что бегает по строке  
  
 """  
 Машина тьюринга которая получает результаты второй(выдает первое слово) и третьей(выдает второе слово) мт,  
 сравнивает эти слова, если они реверсивным выводит 1, если нет выводит 0.  
  
 Как работает:  
 Машина начинает свою работу с первой буквы слова, смотрит какая это буква,  
 в зависимости от того какая это буква, переходит в состояние 2 3 или 4,  
 после идет до края и исходя из состояния (2, 3 или 4), переходит в состояния  
 5 6 или 7, соответсятвенно, далее, если буква соответствует состоянию, то есть  
 5-ое состояние это состояние буквы а, если нашли букву а, затираем её, и отправляем в  
 восьмое состояния, которое ведет каретку в начало и всё начинает заного.  
 Если же буква не соответствует, к примеру ожидается a а стоит b, то состояние 5, 6 или 7  
 переходит в состояние 9, затирает всю строчку и ставит 0, если же алгоритм доходит  
 до конца удачно, то ставится 1.  
 """  
  
 def any\_condition(self):  
 """  
 Функция состояния, условие - что делаем с символом на котором стоит,  
 пример :  
 if self.letter == '1': # если заданный символ один (функция получает символ)  
 self.letter = '0' # меняем букву, если прописано, то изменяем если не прописано то остается как было  
  
 self.direction = '>' # изменяем направление движения, если прописано то изменяем, если нет, то остается как и было  
  
 self.number\_of\_state = self.second\_condition # в какое состояние переходим, 1 - функция называет first, 2 - second,  
 # сколько состояний столько и методов состояний  
 # если прописано, то изменяем если не прописано то остается как было  
  
 Порядок именно такой, вначале идут то что меняется 100% и на что, допустим direction = '>' ,  
 и только потом то, что меняется в зависимости от литеры.  
 """  
 pass  
  
 def first\_condition(self):  
 """  
 Смотрим какая буква, затираем, и отправляем в нужное состояние:  
 a -> 2  
 b -> 3  
 c -> 4  
  
 Если же, после сверки обоих слов, когда на ленте осталась  
 ничего, то ставим 1 и останавливаем машину.  
 """  
 self.direction = '>'  
 if self.letter == 'a':  
 self.letter = '\_'  
 self.state = self.second\_condition  
 elif self.letter == 'b':  
 self.letter = '\_'  
 self.state = self.third\_condition  
 elif self.letter == 'c':  
 self.letter = '\_'  
 self.state = self.fourth\_condition  
 elif self.letter == '\_':  
 self.letter = '1'  
 self.direction = 'stop'  
  
 def second\_condition(self):  
 """  
 Двигаем до правого конца, после его достижения переходим в  
 ПЯТОЕ состояние.  
 """  
 self.direction = '>'  
 if self.letter == '\_':  
 self.direction = '<'  
 self.state = self.fifth\_condition  
  
 def third\_condition(self):  
 """  
 Двигаем до правого конца, после его достижения переходим в  
 ШЕСТОЕ состояние.  
 """  
 self.direction = '>'  
 if self.letter == '\_':  
 self.direction = '<'  
 self.state = self.sixth\_condition  
  
 def fourth\_condition(self):  
 """  
 Двигаем до правого конца, после его достижения переходим в  
 СЕДЬМОЕ состояние.  
 """  
 self.direction = '>'  
 if self.letter == '\_':  
 self.direction = '<'  
 self.state = self.seventh\_condition  
  
 def fifth\_condition(self):  
 """  
 В этом состоянии сверяем букву, если это A то переходим в состояние восемь,  
 иначе в состояние девять, если это лябда, ставим 0 и завершаем работу программы.  
 """  
 self.direction = '<'  
 if self.letter == 'a':  
 self.letter = '\_'  
 self.state = self.eighth\_condition  
 elif self.letter in ('b', 'c'):  
 self.letter = '\_'  
 self.state = self.ninth\_condition  
 elif self.letter == '\_':  
 self.letter = '0'  
 self.direction = 'stop'  
  
 def sixth\_condition(self):  
 """  
 В этом состоянии сверяем букву, если это В то переходим в состояние восемь,  
 иначе в состояние девять, если это лябда, ставим 0 и завершаем работу программы.  
 """  
 self.direction = '<'  
 if self.letter == 'b':  
 self.letter = '\_'  
 self.state = self.eighth\_condition  
 elif self.letter in ('a', 'c'):  
 self.letter = '\_'  
 self.state = self.ninth\_condition  
 elif self.letter == '\_':  
 self.letter = '0'  
 self.direction = 'stop'  
  
 def seventh\_condition(self):  
 """  
 В этом состоянии сверяем букву, если это С то переходим в состояние восемь,  
 иначе в состояние девять, если это лябда, ставим 0 и завершаем работу программы.  
 """  
 self.direction = '<'  
 if self.letter == 'c':  
 self.letter = '\_'  
 self.state = self.eighth\_condition  
 elif self.letter in ('a', 'b'):  
 self.letter = '\_'  
 self.state = self.ninth\_condition  
 elif self.letter == '\_':  
 self.letter = '0'  
 self.direction = 'stop'  
  
 def eighth\_condition(self):  
 """  
 Данное состояние возвращает головку к первой букве слова.  
 """  
 self.direction = '<'  
 if self.letter == '\_':  
 self.direction = '>'  
 self.state = self.first\_condition  
  
 def ninth\_condition(self):  
 """  
 Данное состояние затирает все буквы слева, и если в конце выводит 0.  
 """  
 self.direction = '<'  
 if self.letter in self.tuple\_alfabet:  
 self.letter = '\_'  
 elif self.letter == '\_':  
 self.letter = '0'  
 self.direction = 'stop'  
  
 def heart(self, word, bot):  
 """  
 Сердце машины, то есть её работа, всё прописано тут, как она идет по состояниям, что возвращаем и т.д.  
  
 Подробное описание:  
 При передачи слова в данную функцию, мы ставим текущее состояние на первое  
 (меняем указатель переменной на функцию первого состояни), ставим головку в  
 нужное место (курсор), далее преобразуем слово которое ввел пользователь  
 из строки в список, для того чтоб можно было изменять в нем элементы, ибо  
 строки в Python константны.  
 После всех приготовлений, переходим в цикл, который и будет выполнять  
 всю работу по перемещению головки и захода в состояния.  
 Вначале мы берем из списка литеру на которой стоит головка(курсор),  
 если не получается это сделать(ошибка) то это скорее всего из-за того что мы  
 вышли за прописанные пределы (по умолчанию, слово веденное пользователем обрамляеся  
 двумя L слева и справа, иногда требуется дойти до константы дальше, и для этого придумано  
 выкидывание исключения, и если это произошло, просто добавляем константу L в конец списка,  
 ВОЗМОЖНО кастыль:( ). После того как взяли нужную литеру переходи в состояние, на котором  
 стоим, в нем проделываем нужные махинации и возвращаемся сюда. Тут, меняем литеру если её  
 изменяли в состоянии, и с помозью условия и проверки переменной self.direction двигаемся  
 либо, вправо влево, или останавливаемся, движения происходят увеличением или уменьшением  
 переменной self.cursor. Если мт закончила свою работу, удаляем лишние L из списка,  
 преобразовываем список в строку и возвращаем новое слово.  
 """  
 self.state = self.first\_condition # привязываем первое состояние к номеру состояния,  
 # или же на каком состоянии мы стартуем  
 word = list(word) # преобразуем строку в список  
 while True: # бесконечный цикл, это и есть некая головка, которая будет ходить по литерам  
 self.amount\_of\_steps += 1  
 self.letter = word[self.cursor] # получаем литеру на которой стоит головка  
 self.state() # проводим определенные операции с этой литерой  
 word[self.cursor] = self.letter # меняем символ который заменили в состоянии  
  
 # записываем логи в файлы  
 if bot is False: # если пользователь ввёл слово, записываем логи  
 with open('log.txt', 'a') as f:  
 f.write('\n\n' + str(self.amount\_of\_steps) + '\t' + ''.join(word) + '\n')  
  
 # двигаемся в какую-либо сторону  
 if self.direction == '>':  
 self.cursor += 1  
 elif self.direction == '<':  
 self.cursor -= 1  
 else: # если self.direction == 'stop', останавливаем машину  
 self.result\_word = ''.join(word).replace('\_', '') # выходим из машины, соединяем полученное на выходе слово в строку и чистим от L  
 return  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # mt = mt\_for\_one\_tape()  
 # mt.heart('\_abba\_', bot=False, cursor=1)  
 # print(mt.result\_word)  
 pass

***mt\_for\_multi\_tape.py***

class mt\_for\_multi\_tape():  
 result\_word = str() # результирующее слово  
 amount\_of\_steps = 0 # кол-во шагов для достижения целей  
 tuple\_alfabet = ('a', 'b', 'c') # кортеж с литерами принадлежащим алфавиту  
  
 state = None # состояние, first\_condition , second,  
 # тем самым меняя указатели на функции  
  
 direction\_of\_first\_ribbon = '>' # направление, в которое двигаемся на первой ленте, может быть >, <, stop  
 direction\_of\_second\_ribbon = '>' # направление, в которое двигаемся на второй ленте  
 letter\_on\_first\_ribbon, letter\_on\_second\_ribbon = str(), str() # буква, которую мы сейчас рассматриваем  
 cursor\_on\_first\_ribbon, cursor\_on\_second\_ribbon = 1, 0 # курсор , то что бегает по строке  
  
 """  
 Машина тьюринга которая получает результаты второй(выдает первое слово) и третьей(выдает второе слово) мт,  
 сравнивает эти слова, если они реверсивным выводит 1, если нет выводит 0.  
  
 Как работает:  
 Идет по первой ленте вправо, после каждых двух символов записывая единицу на вторую ленту.  
 После, если кол-во нечетное, всё затирает, если чётное, то переводит все единицы снизу на буквы  
 второй половины слова, то есть затирая одну букву на первой ленте, ставим её на вторую.  
 После это манипуляции на первой ленте останется обычное слово, на второй реверсированное,  
 сравниваем побуквенно попутно затирая, если всё прошло удачно ставим 1 выходим, иначе всё затираем  
 ставим 0 и выходим.  
 """  
  
 def first\_condition(self):  
 """  
 Если элемент есть в алфавите (а, b, c) то переходим в состояние 2;  
 Если находим пустоту то переходим в состаяние 4.  
 """  
 if self.letter\_on\_first\_ribbon in self.tuple\_alfabet:  
 self.direction\_of\_first\_ribbon, self.direction\_of\_second\_ribbon = '>', '.'  
 self.state = self.second\_condition  
 elif self.letter\_on\_first\_ribbon == '\_':  
 self.direction\_of\_first\_ribbon, self.direction\_of\_second\_ribbon = '<', '<'  
 self.state = self.fourth\_condition  
  
 def second\_condition(self):  
 """  
 Если не лямбда то добавляем на вторую ленту 1, и идем в первое состояние;  
 Если лямбда то переходим в состояние 3.  
 """  
 if self.letter\_on\_first\_ribbon in self.tuple\_alfabet:  
 self.letter\_on\_second\_ribbon = '1'  
 self.direction\_of\_first\_ribbon, self.direction\_of\_second\_ribbon = '>', '>'  
 self.state = self.first\_condition  
 elif self.letter\_on\_first\_ribbon == '\_':  
 self.letter\_on\_first\_ribbon, self.letter\_on\_second\_ribbon = '\_', '\_'  
 self.direction\_of\_first\_ribbon, self.direction\_of\_second\_ribbon = '<', '<'  
 self.state = self.third\_condition  
  
 def third\_condition(self):  
 """  
 Стираем всё на первой и на второй ленте и выводим 0.  
 """  
 if self.letter\_on\_first\_ribbon != '\_': # если ещё не пуста, делаем её пустой  
 self.letter\_on\_first\_ribbon = '\_'  
 elif self.letter\_on\_first\_ribbon == '\_': # если уже пуста, перестаем её чистить стопаем прогу и выводим 0  
 self.letter\_on\_first\_ribbon = '0'  
 self.direction\_of\_first\_ribbon = 'stop'  
  
 if self.letter\_on\_second\_ribbon != '\_': # если ещё не пуста, делаем её пустой  
 self.letter\_on\_second\_ribbon = '\_'  
 elif self.letter\_on\_second\_ribbon == '\_': # если уже пуста, перестаем её чистить  
 self.direction\_of\_second\_ribbon = '.'  
  
 def fourth\_condition(self):  
 """  
 Двигаемся влево на обеих лентах, пока видим единицы на второй ленте  
 записываем буквы с первой ленты на вторую, попутно на первой ленте затирая  
 записанную только что букву. Когда закончились все единицы на второй ленте  
 переходим в состояние 5.  
 """  
 self.direction\_of\_first\_ribbon, self.direction\_of\_second\_ribbon = '<', '<'  
 if self.letter\_on\_second\_ribbon == '\_':  
 self.direction\_of\_first\_ribbon, self.direction\_of\_second\_ribbon = '.', '>' # точка - стоим на месте  
 self.state = self.fifth\_condition  
 return  
 elif self.letter\_on\_first\_ribbon == 'a':  
 self.letter\_on\_second\_ribbon = 'a'  
 elif self.letter\_on\_first\_ribbon == 'b':  
 self.letter\_on\_second\_ribbon = 'b'  
 elif self.letter\_on\_first\_ribbon == 'c':  
 self.letter\_on\_second\_ribbon = 'c'  
 self.letter\_on\_first\_ribbon = '\_'  
  
 def fifth\_condition(self):  
 """  
 Идем на первой ленте влево, на второй вправо, попутно сравнивая символы, если  
 они равны, двигаем так до лямбд и ставим 1, если нет, то переходим в состояние 6.  
 """  
 self.direction\_of\_first\_ribbon, self.direction\_of\_second\_ribbon = '<', '>'  
 if self.letter\_on\_first\_ribbon == self.letter\_on\_second\_ribbon == '\_':  
 self.letter\_on\_first\_ribbon = '1'  
 self.direction\_of\_first\_ribbon = 'stop'  
 elif self.letter\_on\_first\_ribbon == self.letter\_on\_second\_ribbon:  
 self.letter\_on\_first\_ribbon, self.letter\_on\_second\_ribbon = '\_', '\_'  
 else:  
 self.letter\_on\_first\_ribbon, self.letter\_on\_second\_ribbon = '\_', '\_'  
 self.state = self.sixth\_condition  
  
 def sixth\_condition(self):  
 """  
 Затираем всё на лентах и ставим 0.  
 """  
 if self.letter\_on\_first\_ribbon == '\_':  
 self.letter\_on\_first\_ribbon = '0'  
 self.direction\_of\_first\_ribbon = 'stop'  
 return  
 self.letter\_on\_first\_ribbon, self.letter\_on\_second\_ribbon = '\_', '\_'  
 self.direction\_of\_first\_ribbon, self.direction\_of\_second\_ribbon = '<', '>'  
  
 def heart(self, word\_on\_first\_ribbon, bot):  
 """  
 Логика такова, берем символы с первой и второй ленты, а именно с мест на которых стоят курсоры,  
 после идем в состояние, где происходит замена или нет состояния/ самой буквы/ направления,  
 выходим из состояния, и присваиваем новые значения переменным, а именно, старые буквы  
 на старых местах меняем на новые, смотрим на направление и меняем курсор если оно изменилось.  
 """  
  
 self.state = self.first\_condition # привязываем первое состояние к номеру состояния,  
 # или же на каком состоянии мы стартуем  
 word\_on\_first\_ribbon = list(word\_on\_first\_ribbon)  
 word\_on\_second\_ribbon = ['\_']  
 while True:  
 self.amount\_of\_steps += 1  
  
 self.letter\_on\_first\_ribbon = word\_on\_first\_ribbon[self.cursor\_on\_first\_ribbon]  
 self.letter\_on\_second\_ribbon = word\_on\_second\_ribbon[self.cursor\_on\_second\_ribbon]  
  
 self.state()  
  
 word\_on\_first\_ribbon[self.cursor\_on\_first\_ribbon] = self.letter\_on\_first\_ribbon  
 word\_on\_second\_ribbon[self.cursor\_on\_second\_ribbon] = self.letter\_on\_second\_ribbon  
 if word\_on\_second\_ribbon[self.cursor\_on\_second\_ribbon] == '1':  
 word\_on\_second\_ribbon.append('\_')  
  
 if bot is False:  
 with open('multi\_log.txt', 'a') as f:  
 f.write('\n\n' + str(self.amount\_of\_steps) + '\t' + ''.join(word\_on\_first\_ribbon) + '\n\t' + ''.join(word\_on\_second\_ribbon))  
  
 if self.direction\_of\_second\_ribbon == '>': # двигаемся на второй ленте  
 self.cursor\_on\_second\_ribbon += 1  
 elif self.direction\_of\_second\_ribbon == '<':  
 self.cursor\_on\_second\_ribbon -= 1  
  
 if self.direction\_of\_first\_ribbon == '>': # двигаемся на первой  
 self.cursor\_on\_first\_ribbon += 1  
 elif self.direction\_of\_first\_ribbon == '<':  
 self.cursor\_on\_first\_ribbon -= 1  
 elif self.direction\_of\_first\_ribbon == 'stop':  
 self.result\_word = ''.join(word\_on\_first\_ribbon).replace('\_', '')  
 return  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 mt = mt\_for\_multi\_tape()  
 mt.heart('\_bbbaaaabbb\_', bot=False)  
 print(mt.result\_word)  
 pass

# ПРИЛОЖЕНИЕ Г. ЭКРАННЫЕ ФОРМЫ

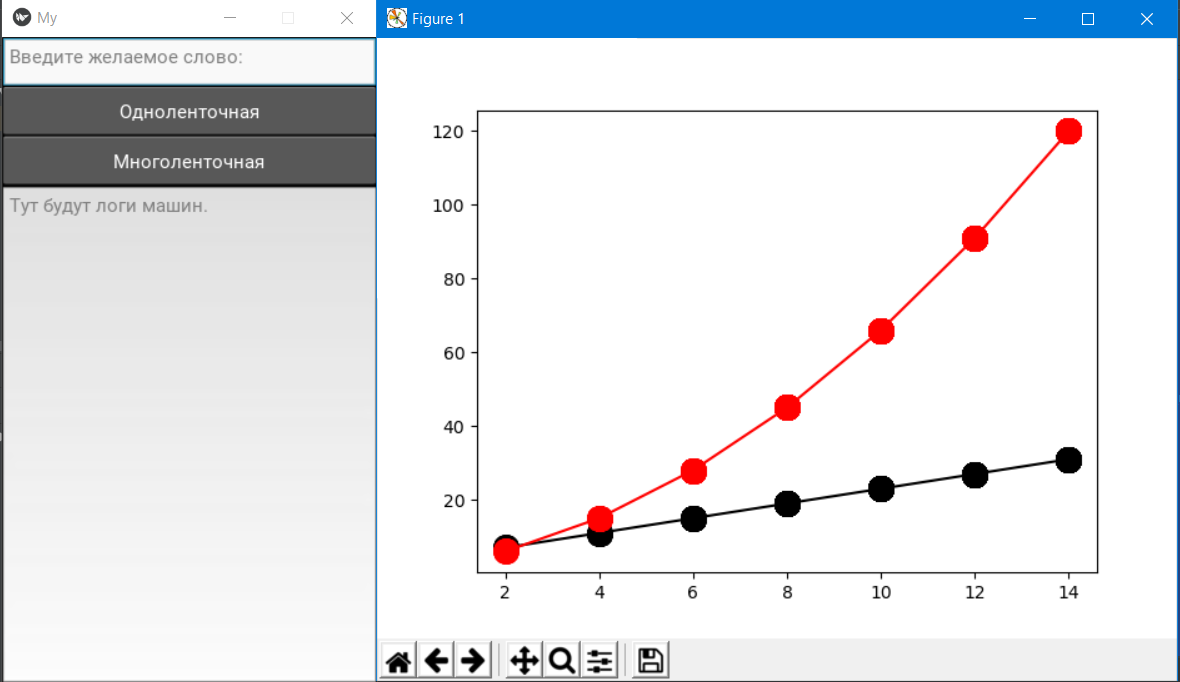


Рисунок Г.1 – Главное меню программы

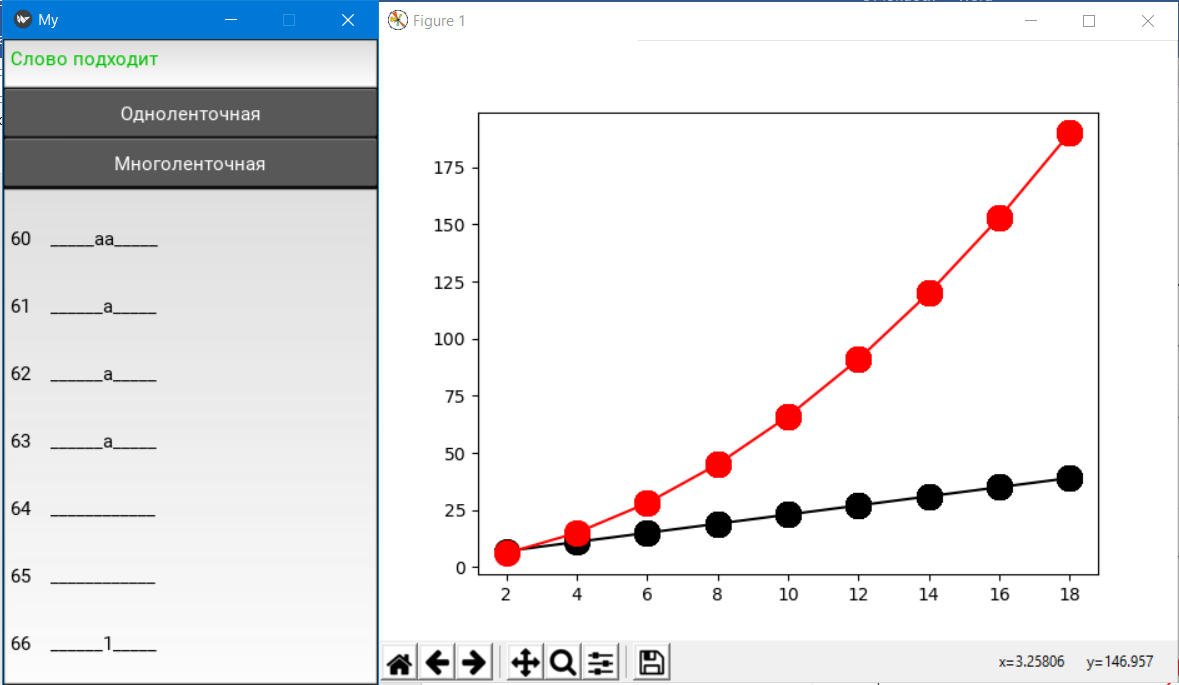


Рисунок Г.2 – Введено слово удовлетворяющее условию на одноленточную машину

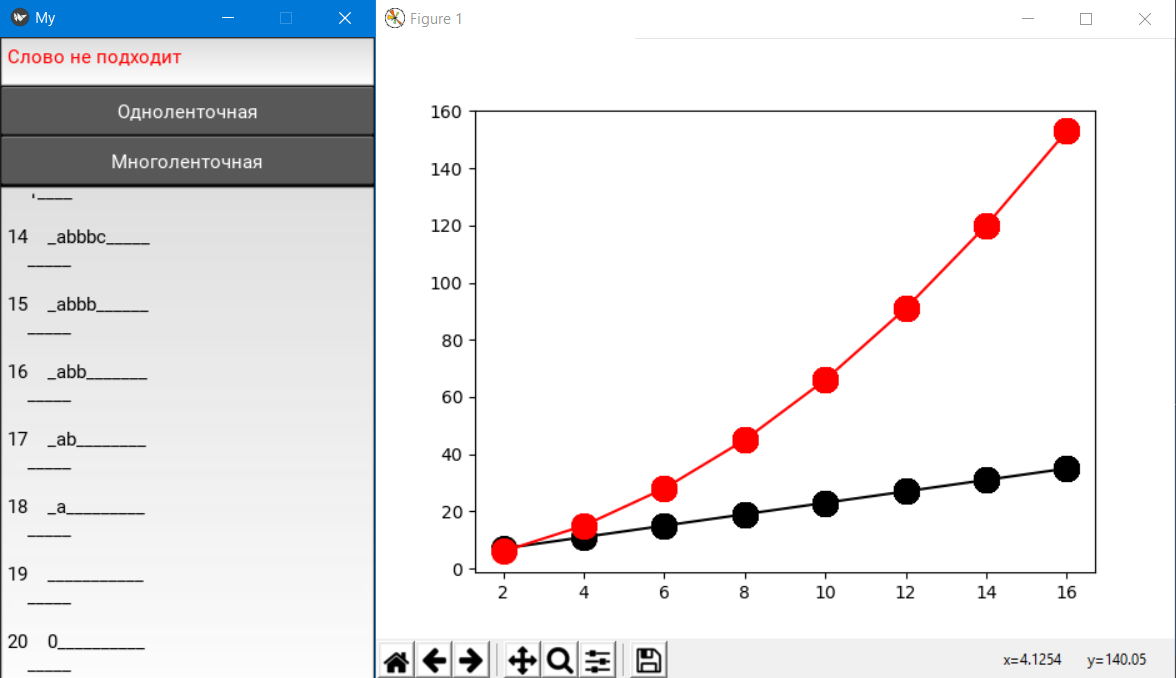


Рисунок Г.3 – Введено слово не удовлетворяющее языку на многоленточную машину