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](#_Toc26222057)

[2. РАЗРАБОТКА АНАЛИТИЧЕСКОЙ И ПРОГРАММНОЙ МОДЕЛИ АЛГОРИТМА ДЛЯ РАСПОЗНАЮЩЕЙ МАШИНЫ ТЬЮРИНГА 5](#_Toc26222058)

[2.1 Формальное определение машины, распознающей Тьюринга 5](#_Toc26222059)

[2.2 Протоколы работы машины Тьюринга на двух лентах 6](#_Toc26222060)

[2.3 Программная модель машины Тьюринга 7](#_Toc26222061)

[2.4 Протоколы работы машины Тьюринга, построенные программно (на двух словах языка и двух словах, не принадлежащих языку) 8](#_Toc26222062)

[2.5 Расчет временной сложности (график функции временной сложности) 10](#_Toc26222063)

[ВЫВОД 11](#_Toc26222064)

[ПРИЛОЖЕНИЕ А. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 13](#_Toc26222065)

[ПРИЛОЖЕНИЕ В. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 14](#_Toc26222066)

[ПРИЛОЖЕНИЕ В. ЛИСТИНГ ПРОГРАММЫ 15](#_Toc26222067)

[ПРИЛОЖЕНИЕ Г. ЭКРАННЫЕ ФОРМЫ 21](#_Toc26222068)

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.

# 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 многоленточной тоже самое только изначальное слово подается на первую ленту, а результат выводится на вторую.

Для реализации данного алгоритма составлена функциональная таблица, представленная на рисунке 2.1.

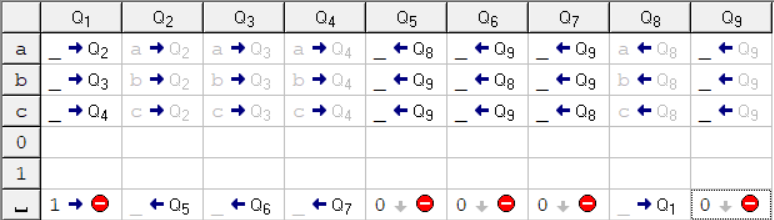


Рисунок 2.1 – Функциональная таблица для алгоритма проверки второго слова на реверс первого

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.

# 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: “aa” для одноленточной, “сbbс” для многоленточной (Табл. 2.2 –Табл. 2.3).

Таблица 2.2

|  |  |
| --- | --- |
| K0 = | LaaL |
| K1 = | LLaL |
| K2 = | LLLL |
| K3 = | LL1L |

Таблица 2.3

|  |  |  |  |
| --- | --- | --- | --- |
| К0= | LcbbcL | K4= | LLLLLL |
| L | L |
| К1= | LLbbcL | K5= | LLL1LL |
| L | 1 |
| К2= | LLbbLL |  | |
| L |

Продолжение таблицы 2.3

|  |  |  |
| --- | --- | --- |
| К3= | LLLbLL |  |
| L |

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

Таблица 2.4

|  |  |
| --- | --- |
| K0 = | LaL |
| K1 = | LLL |
| K2 = | L0L |

Таблица 2.5

|  |  |
| --- | --- |
| K0 = | LabL |
| L |
| K1 = | LLbL |
| L |
| K2 = | LLLL |
| L |
| K3 = | L0LL |
| 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 logging.txt. Рассмотрим пример логирования входного слова abba, удовлетворяющего условию (см. рис. 2.2).

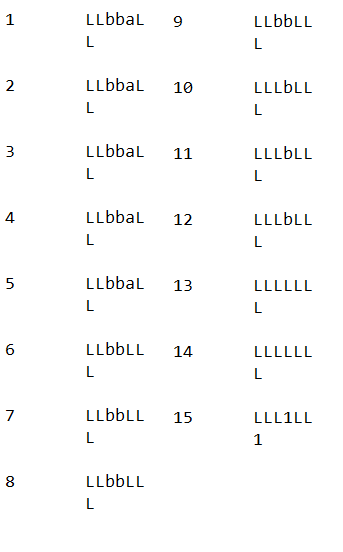


Рисунок 2.2 – Логгинг работы программы над словом “abba”

Рассмотрим слово, не удовлетворяющее условию, слово “cb” (см. рис. 2.3).

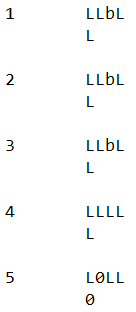


Рисунок 2.3 – Логгинг работы программы над словом “cb”

Рассмотрим слово, удовлетворяющее условию “abccba”. (см. рис. 2.4)

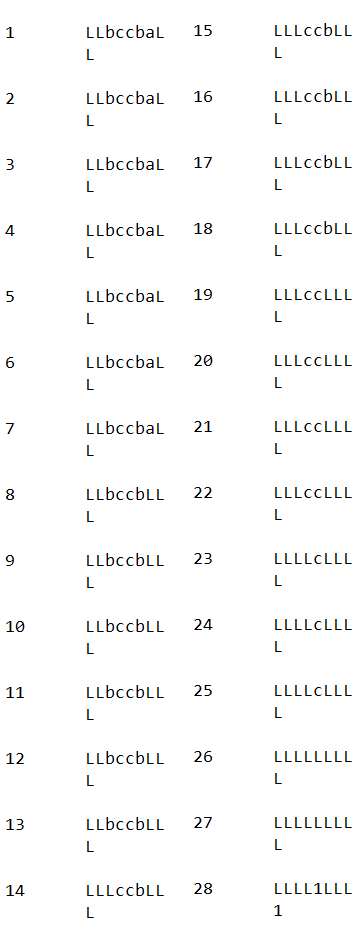


Рисунок 2.4 – Логгинг работы программы над словом “abccba”

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. 2.5.

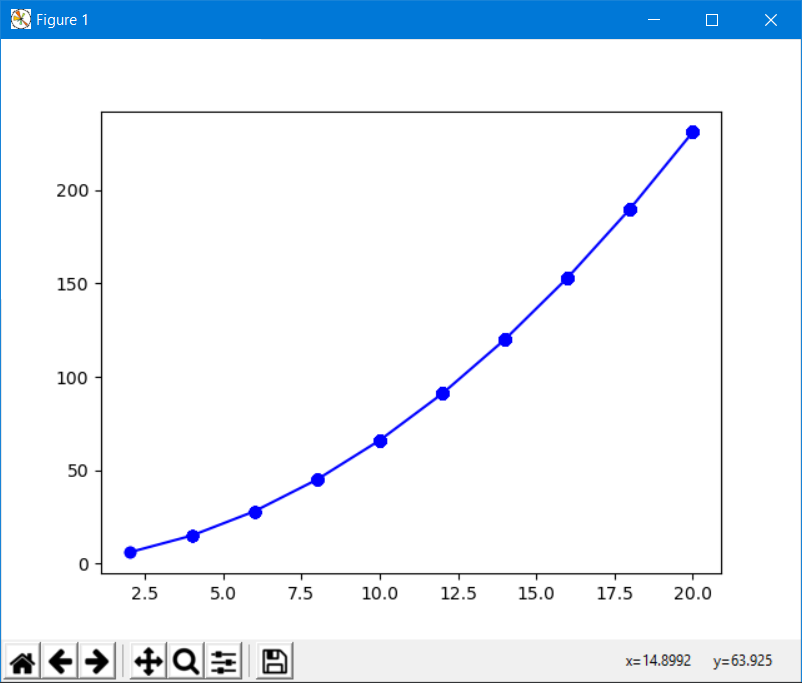


Рисунок 2.5 – График временной сложности

# 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 multitape\_log.txt (если машина была многоленточная). На график будет нанесен маркер соответствующего цвета (если машина одноленточная – черный, если многоленточная – синий).

# ПРИЛОЖЕНИЕ В. ЛИСТИНГ ПРОГРАММЫ

***main.py***

import re  
import threading  
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', '100')  
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', 'multitape\_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)  
 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'  
 )  
 text\_input.focus = True  
 button = Button(text='Одноленточная',  
 on\_release=run\_the\_task,  
 id='0')  
 button\_multi = Button(text='Многоленточная',  
 on\_release=run\_the\_task,  
 id='1')  
 box\_layout.add\_widget(text\_input)  
 box\_layout.add\_widget(button)  
 box\_layout.add\_widget(button\_multi)  
 return box\_layout  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 for name\_of\_file in ('time.txt', 'multi\_time.txt', 'user.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 mt\_for\_check\_words\_plus\_ribbon import mt\_for\_check\_words\_plus\_ribbon  
  
  
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:  
 word = plus(word) # генерируем новое слово на основе старого  
 run(word + ''.join(reversed(word)), True, True) # заносим сгенерированное слово в функцию решения задачи  
  
  
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.  
 """  
 mt = mt\_for\_check\_words\_plus\_ribbon()  
 mt.heart('L' + word + 'L', cursor=1, bot=bot, multitape=multitape)  
 if bot is False: # если не бот, на графике помечаем синим цветом  
 if multitape is True:  
 name\_of\_file = 'user.txt' # если пользователь запустил на многоленточной  
 else:  
 name\_of\_file = 'time.txt' # если пользователь запустил на одноленточной  
 else: # если бот на мульти ленточной  
 name\_of\_file = 'multi\_time.txt'  
 with open(name\_of\_file, '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]

***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):  
 ax1.clear()  
 for name\_of\_file in ( 'multi\_time.txt', 'time.txt', 'user.txt'):  
 with open(name\_of\_file, 'r') as f:  
 pullData = f.read()  
 dataArray = pullData.split('\n') # далее достаем данные  
 xar = list()  
 yar = list()  
 if name\_of\_file == 'user.txt': # одноленточная черная  
 color = '-ob'  
 elif name\_of\_file == 'multi\_time.txt': # многоленточная бот, красная  
 color = '-or'  
 else: # многоленточная синяя  
 color = '-ok'  
 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, color)  
  
 ani = animation.FuncAnimation(fig, animate, interval=1000) # строим, график. Интервал 1 секундa  
 plt.show()

***mt.py***

class mt:  
 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 = 1 # номер состояния, 2, 3 и т.д, позже first\_condition , second,  
 # тем самым меняя указатели на функции  
 direction = '>' # направление, в которое двигаемся , может быть >, <, stop  
 letter = '1' # буква, которую мы сейчас рассматриваем  
 cursor = 1 # курсор , то что бегает по строке  
 second\_ribbon = str()  
  
 all\_new\_letter = ('!', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '\*') # дополнительный алфавит который я добавил  
  
 def first\_condition(self, multitape):  
 """  
 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 second\_condition(self, multitape):  
 pass  
  
 def third\_condition(self, multitape):  
 pass  
  
 def fourth\_condition(self, multitape):  
 pass  
  
 def fifth\_condition(self, multitape):  
 pass  
  
 def sixth\_condition(self, multitape):  
 pass  
  
 def seventh\_condition(self, multitape):  
 pass  
  
 def eighth\_condition(self, multitape):  
 pass  
  
 def ninth\_condition(self, multitape):  
 pass  
  
 def tenth\_condition(self, multitape):  
 pass  
  
 def heart(self, word, cursor, bot, multitape):  
 """  
 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  
 self.cursor = cursor # с какой позиции стартовать, мт не всегда стартует с первой позиции  
 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  
  
 try:  
 self.letter = word[self.cursor] # получаем литеру на которой стоит головка  
 except IndexError: # если произошел выход за строку  
 word.append('L')  
  
 self.state(multitape) # 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  
 if multitape is True: # для многоленточной  
 with open('multitape\_log.txt', 'a') as f:  
 to\_file = '\n\n' + str(self.amount\_of\_steps) + '\t' + ''.join(word) + '\n'  
 if self.second\_ribbon:  
 to\_file += '\t' + self.second\_ribbon  
 else:  
 to\_file += '\tL'  
 f.write(to\_file)  
 else: # для одноленточной  
 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('L', '') # выходим из машины, соединяем полученное на выходе слово в строку и чистим от L  
 return

***mt\_for\_check\_words\_plus\_ribbon.py***

from mt import mt  
  
  
class mt\_for\_check\_words\_plus\_ribbon(mt):  
 """  
 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 first\_condition(self, multitape):  
 """  
 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 = 'L'  
 self.state = self.second\_condition  
 elif self.letter == 'b':  
 self.letter = 'L'  
 self.state = self.third\_condition  
 elif self.letter == 'c':  
 self.letter = 'L'  
 self.state = self.fourth\_condition  
 elif self.letter == 'L':  
 self.letter = '1'  
 if multitape is True:  
 self.second\_ribbon = '1'  
 self.direction = 'stop'  
  
 def second\_condition(self, multitape):  
 """  
 Двигаем до правого конца, после его достижения переходим в  
 ПЯТОЕ состояние.  
 """  
 self.direction = '>'  
 if self.letter == 'L':  
 self.direction = '<'  
 self.state = self.fifth\_condition  
  
 def third\_condition(self, multitape):  
 """  
 Двигаем до правого конца, после его достижения переходим в  
 ШЕСТОЕ состояние.  
 """  
 self.direction = '>'  
 if self.letter == 'L':  
 self.direction = '<'  
 self.state = self.sixth\_condition  
  
 def fourth\_condition(self, multitape):  
 """  
 Двигаем до правого конца, после его достижения переходим в  
 СЕДЬМОЕ состояние.  
 """  
 self.direction = '>'  
 if self.letter == 'L':  
 self.direction = '<'  
 self.state = self.seventh\_condition  
  
 def fifth\_condition(self, multitape):  
 """  
 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 = 'L'  
 self.state = self.eighth\_condition  
 elif self.letter in ('b', 'c'):  
 self.letter = 'L'  
 self.state = self.ninth\_condition  
 elif self.letter == 'L':  
 self.letter = '0'  
 if multitape is True:  
 self.second\_ribbon = '0'  
 self.direction = 'stop'  
  
 def sixth\_condition(self, multitape):  
 """  
 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 = 'L'  
 self.state = self.eighth\_condition  
 elif self.letter in ('a', 'c'):  
 self.letter = 'L'  
 self.state = self.ninth\_condition  
 elif self.letter == 'L':  
 self.letter = '0'  
 if multitape is True:  
 self.second\_ribbon = '0'  
 self.direction = 'stop'  
  
 def seventh\_condition(self, multitape):  
 """  
 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 = 'L'  
 self.state = self.eighth\_condition  
 elif self.letter in ('a', 'b'):  
 self.letter = 'L'  
 self.state = self.ninth\_condition  
 elif self.letter == 'L':  
 self.letter = '0'  
 if multitape is True:  
 self.second\_ribbon = '0'  
 self.direction = 'stop'  
  
 def eighth\_condition(self, multitape):  
 """  
 Данное состояние возвращает головку к первой букве слова.  
 """  
 self.direction = '<'  
 if self.letter == 'L':  
 self.direction = '>'  
 self.state = self.first\_condition  
  
 def ninth\_condition(self, multitape):  
 """  
 Данное состояние затирает все буквы слева, и если в конце выводит 0.  
 """  
 self.direction = '<'  
 if self.letter in self.tuple\_alfabet:  
 self.letter = 'L'  
 elif self.letter == 'L':  
 self.letter = '0'  
 if multitape is True:  
 self.second\_ribbon = '0'  
 self.direction = 'stop'

# ПРИЛОЖЕНИЕ Г. ЭКРАННЫЕ ФОРМЫ

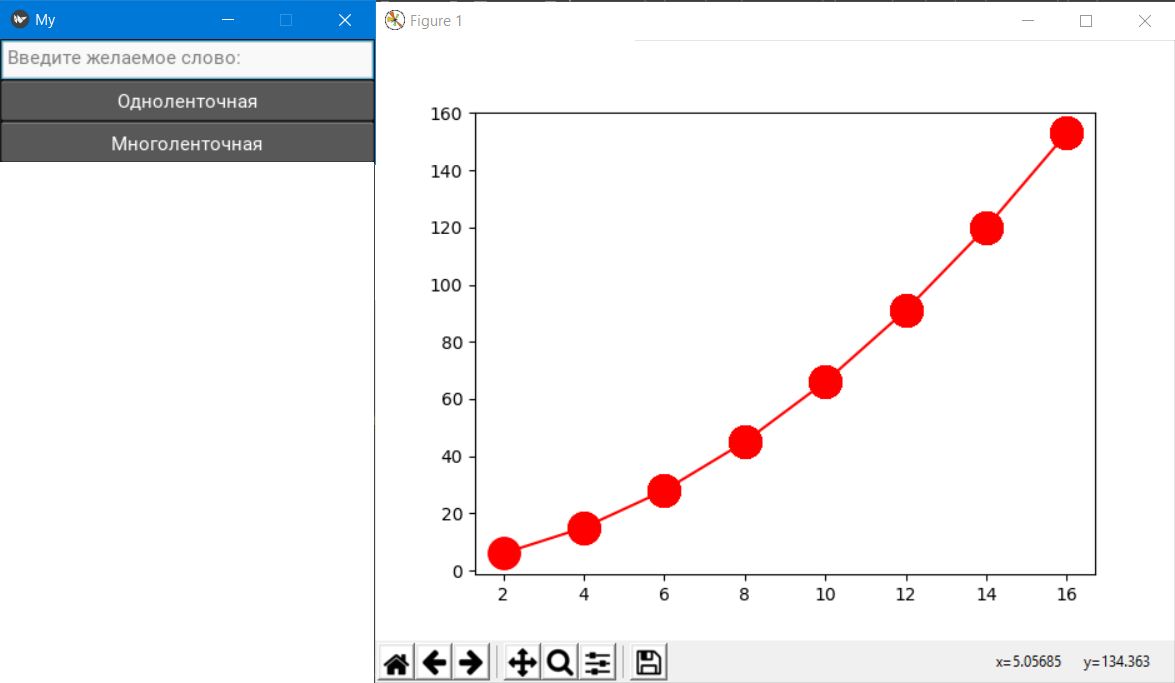


Рисунок Г.1 – Главное меню программы

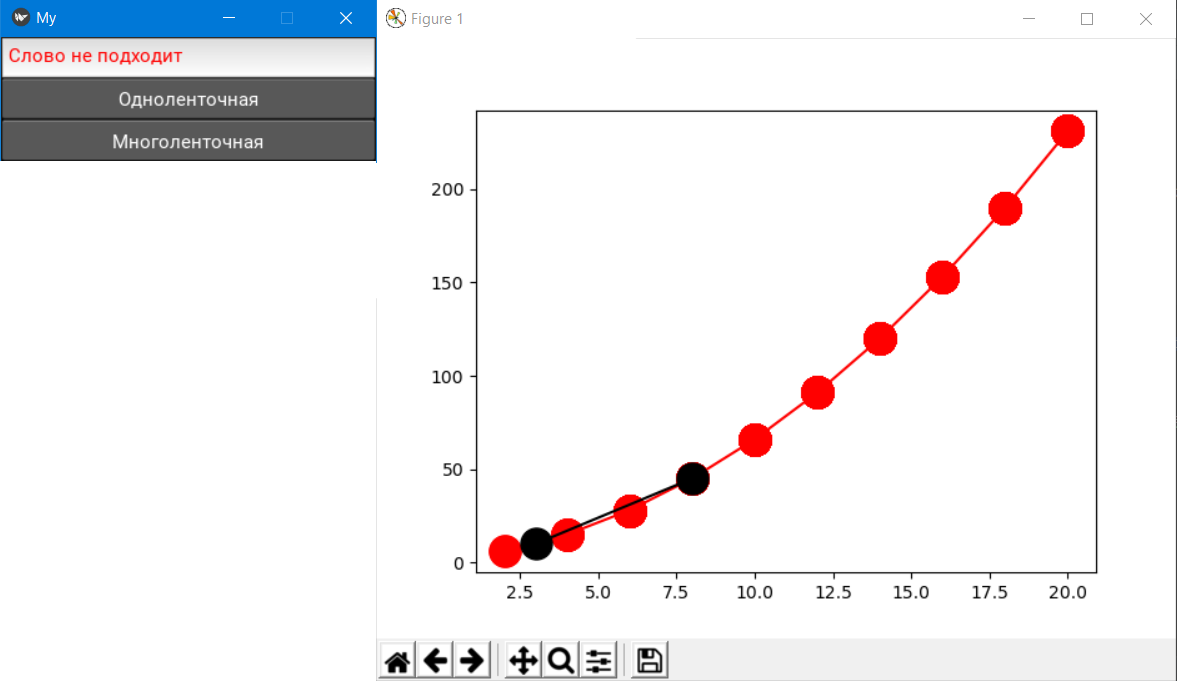


Рисунок Г.2 – Введено 2 слова для проверки на одноленточной машине

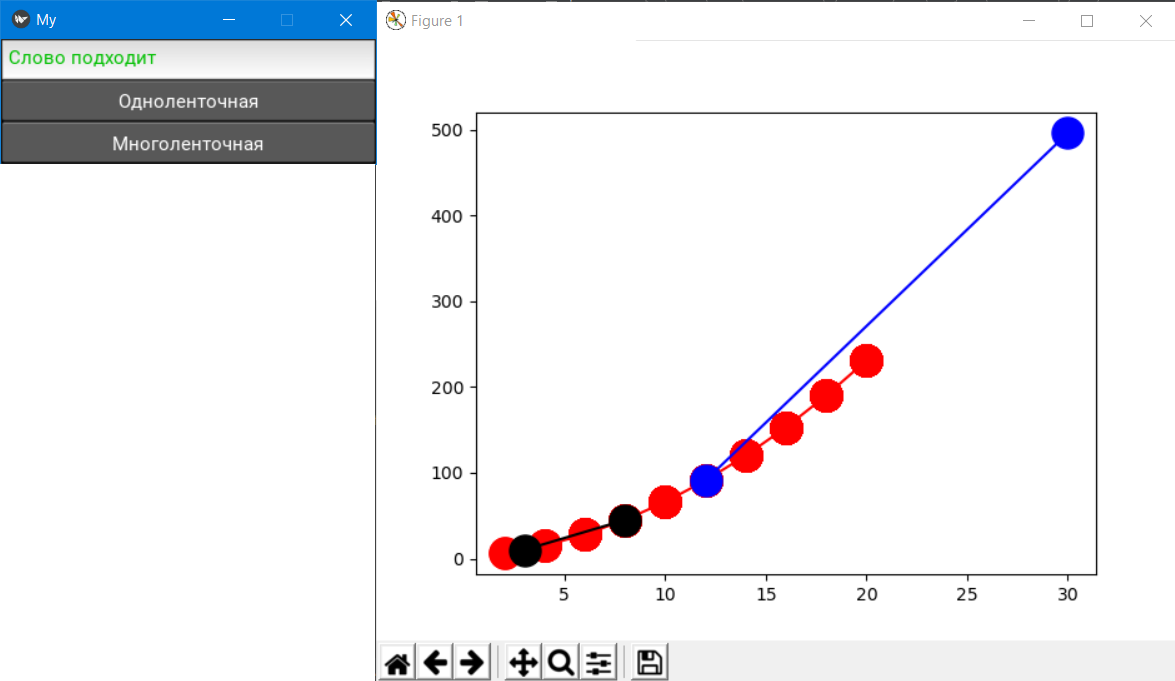


Рисунок Г.3 – Введено 2 слово для проверке на многоленточной машине