МИНОБРНАУКИ РОССИИ

РГУ НЕФТИ И ГАЗА (НИУ) ИМЕНИ И.М. ГУБКИНА

Факультет Автоматики и Вычислительной Техники
Кафедра Автоматизированных систем управления

Оценка комиссии: _____ Рейтинг: _____
Подписи членов комиссии:

_____ Волков Денис Андреевич
(подпись) (фамилия, имя, отчество)

Мухина Анастасия Геннадиевна
_____ _____
(подпись) (фамилия, имя, отчество)

_____
(дата)

КУРСОВАЯ РАБОТА

по дисциплине Математическая логика и теория алгоритмов

на тему Программная реализация алгоритма

«К ЗАЩИТЕ»

к.т.н, доцент, доцент кафедры АСУ
Волков Денис Андреевич
(должность, ученая степень; фамилия, и.о.)

_____
(подпись)

_____
(дата)

ВЫПОЛНИЛ:
Студент группы АС-23-05
(номер группы)

Белкин Арсений Ярославович
(фамилия, имя, отчество)

_____
(подпись)

_____
(дата)

Москва, 2024

МИНОБРНАУКИ РОССИИ

РГУ НЕФТИ И ГАЗА (НИУ) ИМЕНИ И.М. ГУБКИНА

Факультет **Автоматики и вычислительной техники**
Кафедра **Автоматизированных систем управления**

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

по дисциплине Математическая логика и теория алгоритмов

на тему Программная реализация алгоритма

ДАНО студенту Белкину Арсению Ярославовичу (фамилия, имя, отчество в дательном падеже) группы АС-23-05 (номер группы)

Содержание работы:

1. Описание и анализ «Машина Тьюринга».
2. Выбор и обоснование необходимых для разработки средств.
3. Программная реализация.

Исходные данные для выполнения работы:

1. Материалы курса «Математическая логика и теория алгоритмов»
2.
3.

Рекомендуемая литература:

1. Колдаев, В. Д. Основы алгоритмизации и программирования : учебное пособие / В.Д. Колдаев ; под ред. проф. Л.Г. Гагариной. — Москва : ФОРУМ : ИНФРА-М, 2022. — 414 с. — (Среднее профессиональное образование). - ISBN 978-5-8199-0733-7. - Текст : электронный. - URL: https://znanium.com/catalog/product/1735805 (дата обращения: 1.09.2023). – Режим доступа: по подписке.
2. Трофимов, В. В. Алгоритмизация и программирование : учебник для вузов / В. В. Трофимов, Т. А. Павловская ; под редакцией В. В. Трофимова. — 4-е изд. — Москва : Издательство Юрайт, 2023. — 118 с. — (Высшее образование). — ISBN 978-5-534-17497-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/533199 (дата обращения: 1.09.2023).
3. Скорубский, В. И. Математическая логика : учебник и практикум для вузов / В. И. Скорубский, В. И. Поляков, А. Г. Зыков. — Москва : Издательство Юрайт, 2023. — 211 с. — (Высшее образование). — ISBN 978-5-534-01114-2. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/511996 (дата обращения: 14.09.2023).

Графическая часть:

1. _____
2. _____

Требования к представлению результатов:

✓	Электронная версия
	Бумажный вариант и электронный образ документа

Руководитель: к.т.н. доцент _____ Волков Д.А.
(уч.степень) (должность) (подпись) (фамилия, имя, отчество)

Задание принял к исполнению: студент _____ Белкин Арсений Ярославович
(подпись) (фамилия, имя, отчество)

Содержание

Введение

Машина Тьюринга является одной из основополагающих концепций в теории вычислений, предложенной Аланом Тьюрингом в 1936 году. Она служит абстрактной моделью вычислений и позволяет исследовать границы того, что может быть вычислено.

Понимание принципов машины Тьюринга помогает глубже осознать работу алгоритмов.

Глава 1. Формулирование проблемы и возможного её решения

1.1. Задача

Задача звучит так: реализовать машину Тьюринга с графическим интерфейсом, которая состоит из:

- бесконечной в обе стороны ленты;
- головки записи-чтения;
- алфавита;
- таблицы состояний.

В программе должна быть возможность загружать из файла и сохранять в файл.

1.2. План решения поставленной проблемы

Первым делом надо разобраться с основными возможностями машины Тьюринга, эти знания были получены на занятиях по данной дисциплине.

Хранение данных будет в файлах формата .json. А графический интерфейс будет выполнен с помощью библиотеки tkinter[1] языка Python[2].

Создание бесконечной в обе стороны ленты - самый интересный вопрос. Я решил использовать такую структуру данных, как двусвязный список (DoubleLinkedList)[3]. Именно на этой структуре данных основывается моя реализация машины Тьюринга.

После выбора "основы" машины, можно составить план реализации:

1. Определение формата состояний и правил;
2. Определение формата данных, сохраненных в файл;
3. Написание ленты на основе класса DoubleLinkedList;

4. Добавление ленты в класс TuringMachine;
5. Реализация взаимодействия ленты, алфавита, состояний и правил;
6. Создание графического интерфейса.

Глава 2. Программная реализация алгоритма

2.1. Использованные библиотеки

При создании кода я буду использовать следующие библиотеки: tkinter для графического интерфейса, json для работы с json файлами и typing для указания типов переменных в функциях.

2.2. Описание кода

2.2.1. Лента

Лента будет представлена двусвязным списком (DoubleLinkedList), в нем будет подкласс Node, то есть узел - ячейка на ленте, который будет иметь "магический" метод "str", который отвечает за представление ячейки.

```
class DoubleLinkedList:
    class Node:
        def __init__(self, value=None, prev=None, next=None) -> None:
            self.value = value
            self.prev = prev
            self.next = next

        def __str__(self) -> str:
            return str(self.value)

    def __init__(self) -> None:
        self.head = None
        self.tail = None
        self.len = 0
```

Далее будут приведены основные методы, которые используется в реализации машины Тьюринга:

Добавление в начало и конец списка. Эти методы помогают добавить слово на ленту, а так же добавлять символы при выходе за границы текущего слова:

```
def push_front(self, value=None) -> None:
    node = self.Node(value=value)
    node.next = self.head
    if self.head is not None:
        self.head.prev = node
    if self.tail is None:
        self.tail = node
    self.head = node
    self.len += 1

def push_back(self, value=None) -> None:
    node = self.Node(value=value)
    node.prev = self.tail
    if self.head is None:
        self.head = node
    if self.tail is not None:
        self.tail.next = node
    self.tail = node
    self.len += 1
```

Получение какого-либо узла списка. Необходимость данного метода объяснять не приходится, так как получить какой-либо узел требуется очень часто:

```
def get_node(self, index: int) -> Node:
    node = self.head
    n = 0

    while n != index:
        if node is None:
            return node
        node = node.next
        n += 1
```

```
    return node

def __getitem__(self, index: int) -> Node:
    return self.get_node(index)
```

Очищение списка. Очистить сразу весь список нужно в ситуации, когда пользователь загружает новую конфигурацию или просто нажимает кнопку сброса:

```
def clear(self) -> None:
    while self.head is not None:
        self.pop_front()
```

2.2.2. класс TuringMachine

Класс TuringMachine содержит функции, представляющие машину Тьюринга без графического интерфейса.

Конструктор класса. В нем создаются все необходимые поля для работы машины Тьюринга. И загружается слово на ленту:

```
class TuringMachine:
    def __init__(self, path_to_file=None):
        self.tape = DoubleLinkedList()
        self.data = dict()
        self.path_to_file = path_to_file
        self.alphabet = ""
        self.word = ""
        self.rules = dict()
        self.states = set()
        self.currentCondition = None
        self.pointer = None

        self.__load_data(path_to_file=path_to_file)
        self.__set_data()
        self.__set_word()
```

Установка слова на ленту:

```
def __set_word(self) -> None:
    if len(self.word):
        for char in self.word:
            self.tape.push_back(char)
```

Получение данных из файла. Полученные данные записываются в атрибут класса: словарь data:

```
def __load_data(self, path_to_file) -> None:
    if path_to_file is not None:
        with open(path_to_file) as file:
            self.data = json.load(file)
```

Установка текущей конфигурации из словаря data. Данные устанавливаются текущими значениями из словаря data:

```
def __set_data(self) -> None:
    if len(self.data):
        self.alphabet = self.data["alphabet"]
        self.word = self.data["word"]
        self.load_rules()
        self.states = self.get_states()
        self.currentCondition = self.data["start_condition"]
        self.pointer = self.data["start_pointer"]
```

Выполнение шага. С помощью этой функции происходит выполнение одного шага в машине Тьюринга. В ней согласно текущему состоянию и символу на ленте, записывается новый символ на ленту, делается движение, а так же переход в новое состояние:

```
def step(self) -> bool:
    self.currentRule = self.rules.get(
        self.currentCondition, self.rules.get(self.data["start_condition"],
            None)
    )
    symbol = self.tape[self.pointer].value
    command = self.currentRule["commands"][symbol].split()

    new_symbol, move, new_condition = command
```

```
    self.tape[self.pointer].value = new_symbol

    if move == "L":
        self.pointer -= 1
        if self.pointer < 0:
            self.tape.push_front("_")
            self.pointer = 0
    elif move == "R":
        self.pointer += 1
        if self.pointer > self.tape.len - 1:
            self.tape.push_back("_")

    if new_condition != "!":
        self.currentCondition = new_condition
    else:
        self.currentCondition = self.data.get("start_condition", "q0")
        return False

    return True
```

2.2.3. GUI

Код графического интерфейса есть в приложении. В нем присутствуют функции выполняющие загрузку из файла, а так же сохранение текущей конфигурации в файл. Выполнение одного шага или полное выполнение программы доступно только после загрузки конфигурации.

2.2.4. Запуск

Если происходит запуск именно файла app.py (а не импорт), то код исполняется:

```
from tmGUI import *


if __name__ == "__main__":
    root = tk.Tk()
```

```
gui = TuringMachineGUI(root)
root.mainloop()
```

Глава 3. Результат работы программы

Представлено выполнение алгоритма бинарного инкремента:

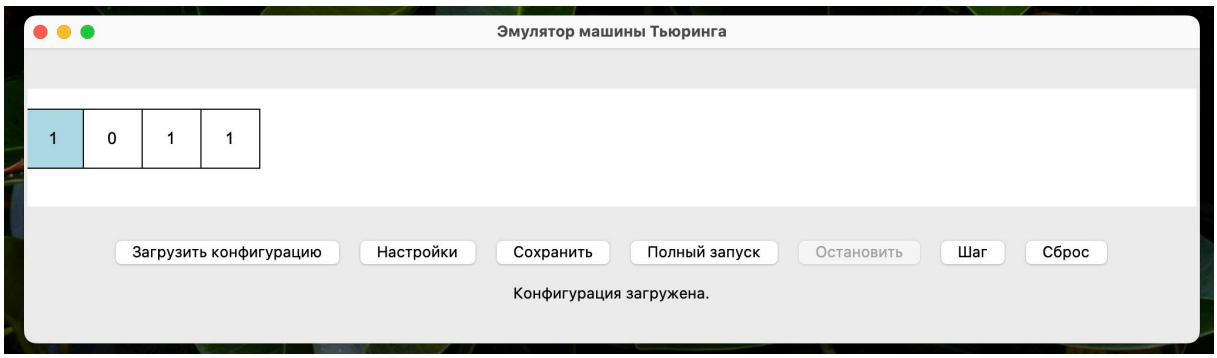

Рис. 1. Старт программы. Загруженное слово на ленту

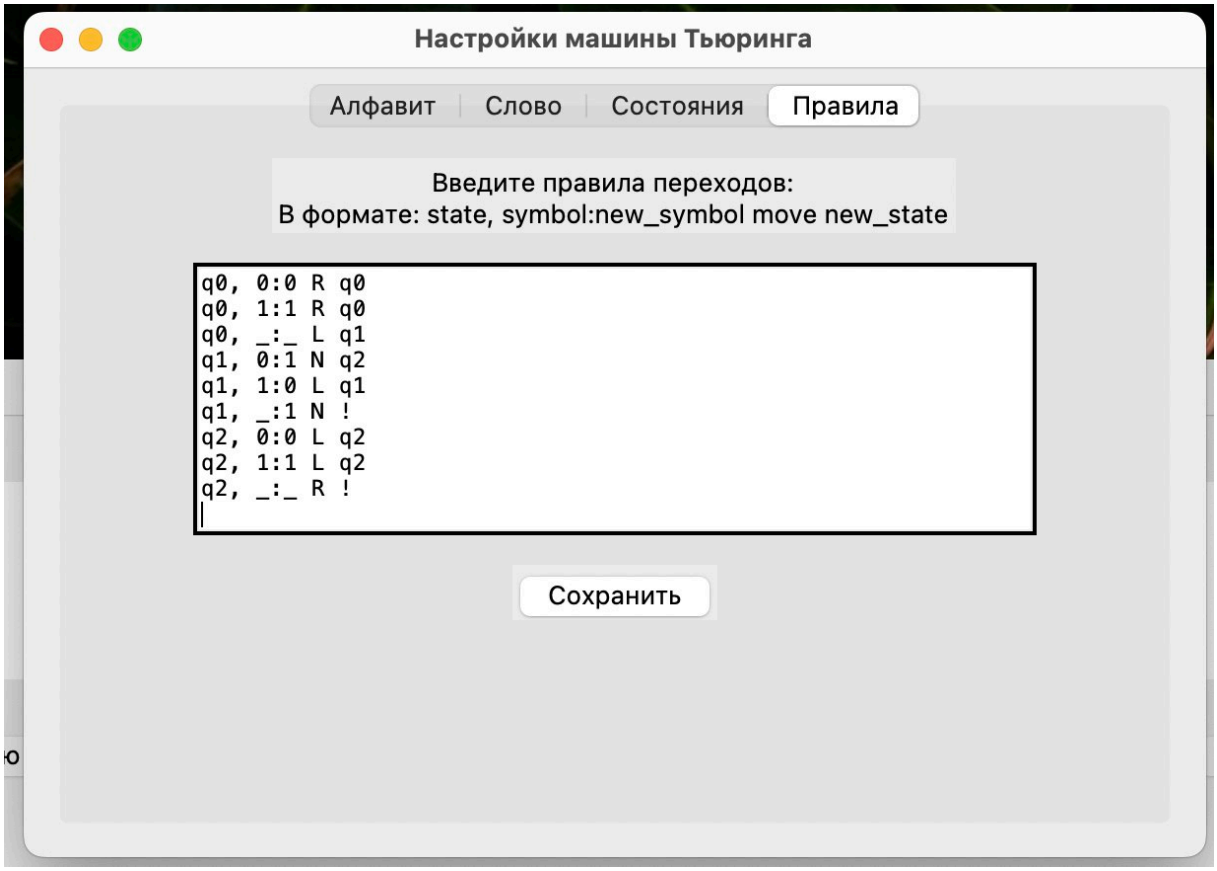

Рис. 2. Список правил

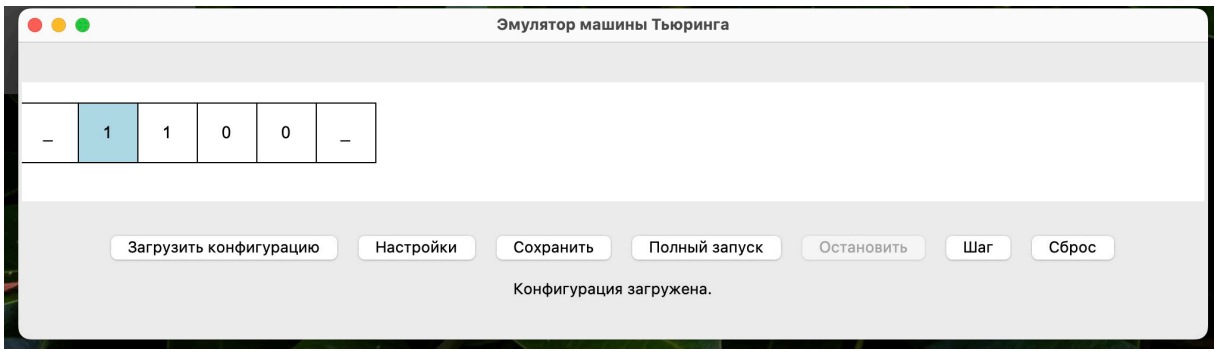

Рис. 3. Результат выполнения программы

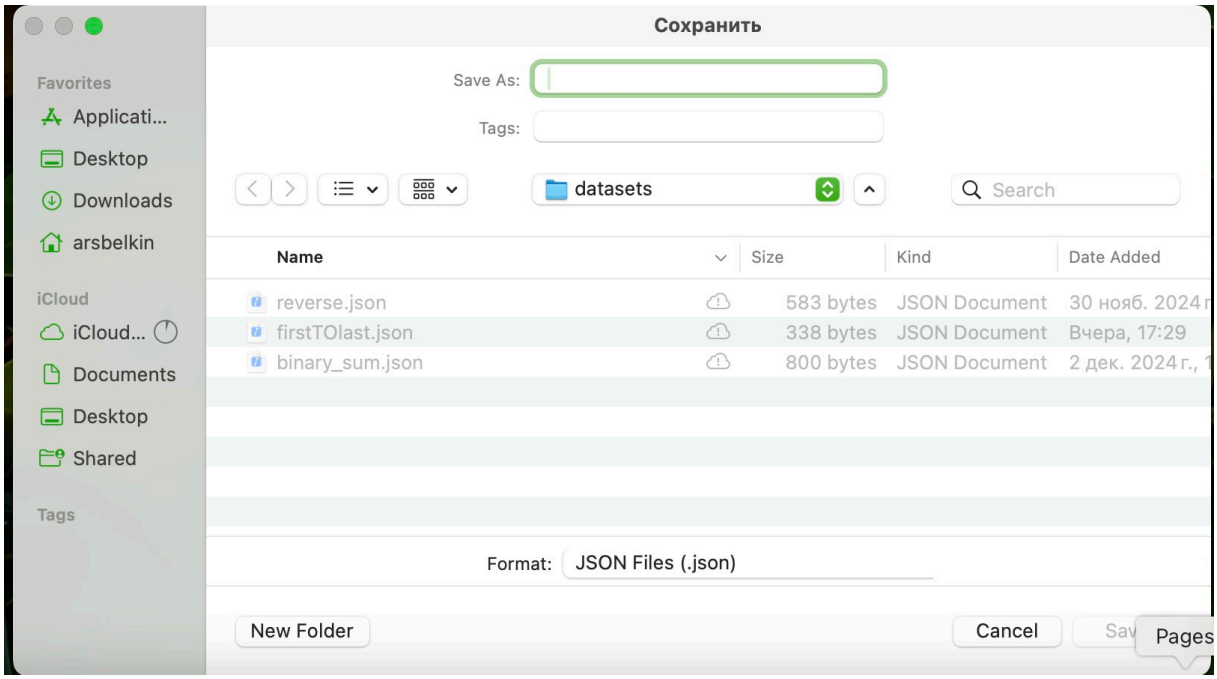

Рис. 4. Сохранение конфигурации

Заключение

Процесс реализации машины Тьюринга помог полностью разобраться с её особенностями. Также применение двусвязного списка для реализации ленты побудило повторить структуры данных.

Конечную программу можно использовать для самостоятельной подготовки к семинарским занятиям по теме "Машина Тьюринга" для моделирования каких-то алгоритмов.

Список литературы

1. Тимур М. Создание настольных Python приложений с графическим интерфейсом пользователя ЛитРес: Самиздат. - 2021.
2. Официальная документация языка Python. — URL: https://docs.python.org/3.12/.
3. ХАБР.Двусвязный список. — URL: https://habr.com/ru/companies/otus/articles/849482/.

Приложение 1

Программный код алгоритма:

tape.py:

```
from typing import Union


class DoubleLinkedList:
    class Node:
        def __init__(self, value=None, prev=None, next=None) -> None:
            self.value = value
            self.prev = prev
            self.next = next

        def __str__(self) -> str:
            return str(self.value)

    def __init__(self) -> None:
        self.head = None
        self.tail = None
        self.len = 0

    def is_empty(self) -> bool:
        return not self.__len

    def push_front(self, value=None) -> None:
        node = self.Node(value=value)
        node.next = self.head
        if self.head is not None:
            self.head.prev = node
        if self.tail is None:
            self.tail = node
        self.head = node
        self.len += 1

    def push_back(self, value=None) -> None:
        node = self.Node(value=value)
        node.prev = self.tail
        if self.head is None:
```

```
            self.head = node
        if self.tail is not None:
            self.tail.next = node
        self.tail = node
        self.len += 1

    def pop_front(self) -> Node:
        if self.head is None:
            return

        node = self.head
        if self.head.next is not None:
            self.head.next.prev = None
        else:
            self.tail = None
        self.head = self.head.next
        self.len -= 1

        return node

    def pop_back(self) -> Union[Node, None]:
        if self.tail is None:
            return

        node = self.tail
        if self.tail.prev is not None:
            self.tail.prev.next = None
        else:
            self.head = None
        self.tail = self.tail.prev
        self.len -= 1

        return node

    def get_node(self, index: int) -> Node:
        node = self.head
        n = 0

        while n != index:
            if node is None:
```

```
                return node
            node = node.next
            n += 1

        return node

    def __getitem__(self, index: int) -> Node:
        return self.get_node(index)

    def insert(self, index: int, value) -> Node:
        right = self.get_node(index)
        if right is None:
            return self.push_back(value=value)

        left = right.prev
        if left is None:
            return self.push_front(value=value)

        node = self.Node(value=value)
        node.prev = left
        node.next = right
        left.next = node
        right.prev = node

        return node

    def erase(self, index: int) -> Union[Node, None]:
        node = self.get_node(index)
        if node is None:
            return

        if node.prev is None:
            return self.pop_front()

        if node.next is None:
            return self.pop_back()

        left = node.prev
        right = node.next
        left.next = right
```

```
        right.prev = left

        return node

    def clear(self) -> None:
        while self.head is not None:
            self.pop_front()

    def print_list(self) -> None:
        current_node = self.head

        while current_node:
            print(current_node.value if current_node.value != "empty" else
                "_", end=" ")
            current_node = current_node.next
        print()

    def __str__(self):
        current_node = self.head
        result = ""
        while current_node:
            result += current_node.value
            current_node = current_node.next
        return result
```

turingMachine.py:

```
from tape import DoubleLinkedList
import json


class TuringMachine:
    def __init__(self, path_to_file=None):
        self.tape = DoubleLinkedList()
        self.data = dict()
        self.path_to_file = path_to_file
        self.alphabet = ""
        self.word = ""
        self.rules = dict()
        self.states = set()
        self.currentCondition = None
```

```
        self.pointer = None

        self.__load_data(path_to_file=path_to_file)
        self.__set_data()
        self.__set_word()

    def __load_data(self, path_to_file) -> None:
        if path_to_file is not None:
            with open(path_to_file) as file:
                self.data = json.load(file)

    def __set_data(self) -> None:
        if len(self.data):
            self.alphabet = self.data["alphabet"]
            self.word = self.data["word"]
            self.load_rules()
            self.states = self.get_states()
            self.currentCondition = self.data["start_condition"]
            self.pointer = self.data["start_pointer"]

    def __set_word(self) -> None:
        if len(self.word):
            for char in self.word:
                self.tape.push_back(char)

    def load_rules(self):
        if len(self.data):
            self.rules = self.data["conditions"]

    def save(self, file):
        self.data["alphabet"] = self.alphabet
        self.data["word"] = str(self.tape)
        json.dump(self.data, file)

    def get_rules(self) -> str:
        rules = ""
        if len(self.rules):
            for cond in self.rules.values():
                for key, value in cond["commands"].items():
                    rules += f"{cond["condition"]}, "
```

```
                rules += f"{key}:"
                rules += f"{value}\n"
    return rules

def get_states(self) -> set:
    if len(self.rules):
        return (cond.get("condition", "") for cond in self.rules.values
            ())
    else:
        return self.states

def set_rules(self, new_rules: str) -> bool:
    self.data["conditions"] = {}

    for line in new_rules.split("\n"):
        parts = line.split(":")
        key = tuple(map(str.strip, parts[0].split(",")))
        value = tuple(map(str.strip, parts[1].split()))

        if "conditions" not in self.data:
            self.data["conditions"] = {}

        if key[0] not in self.data["conditions"]:
            self.data["conditions"][key[0]] = {}
            self.data["conditions"][key[0]]["condition"] = key[0]

        if "commands" not in self.data["conditions"][key[0]]:
            self.data["conditions"][key[0]]["commands"] = {}

        if key[1] not in self.data["conditions"][key[0]]["commands"]:
            self.data["conditions"][key[0]]["commands"][key[1]] = {}
        self.data["conditions"][key[0]]["commands"][
            key[1]
        ] = f"{value[0]} {value[1]} {value[2]}"

def step(self) -> bool:
    self.currentRule = self.rules.get(
        self.currentCondition, self.rules.get(self.data["
            start_condition"], None)
    )
```

```
        symbol = self.tape[self.pointer].value
        command = self.currentRule["commands"][symbol].split()

        new_symbol, move, new_condition = command

        self.tape[self.pointer].value = new_symbol

        if move == "L":
            self.pointer -= 1
            if self.pointer < 0:
                self.tape.push_front("_")
                self.pointer = 0
        elif move == "R":
            self.pointer += 1
            if self.pointer > self.tape.len - 1:
                self.tape.push_back("_")

        if new_condition != "!":
            self.currentCondition = new_condition
        else:
            self.currentCondition = self.data.get("start_condition", "q0")
            return False

        return True

    def run(self) -> None:
        flag = True
        while flag:
            flag = self.step()
```

tmGUI.py:

```
from turing_machine import TuringMachine
import tkinter as tk
from tkinter import messagebox, filedialog, ttk


class TuringMachineGUI:
    CELL_SIZE = 50  # Размерячейкиналенте

    def __init__(self, master: tk.Tk):
```

```
        self.master = master
        self.master.title("Эмулятор машиныТьюринга ")

        self.tm = TuringMachine()
        self.running = False
        self.flag = [False, False, False, False]

        self.create_interface()

    def create_interface(self):
        """Создаёт элементыинтерфейса ."""
        config_frame = tk.Frame(self.master)
        config_frame.pack(pady=10)

        # Лента
        self.canvas = tk.Canvas(self.master, height=100, bg="white")
        self.canvas.pack(fill=tk.X, pady=10)
        self.cells = []

        # Панельуправления
        control_frame = tk.Frame(self.master)
        control_frame.pack(pady=10)

        self.load_button = tk.Button(
            control_frame,
            text="Загрузить конфигурацию",
            command=self.load_configuration,
        )
        self.load_button.pack(side=tk.LEFT, padx=5)

        self.settings_button = tk.Button(
            control_frame, text="Настройки", command=self.open_settings
        )
        self.settings_button.pack(side=tk.LEFT, padx=5)

        self.save_button = tk.Button(
            control_frame, text="Сохранить", state=tk.DISABLED, command=
                self.save
        )
        self.save_button.pack(side=tk.LEFT, padx=5)
```

```
        self.run_button = tk.Button(
            control_frame,
            text="Полный запуск",
            state=tk.DISABLED,
            command=self.run_machine,
        )
        self.run_button.pack(side=tk.LEFT, padx=5)

        self.stop_button = tk.Button(
            control_frame,
            text="Остановить",
            state=tk.DISABLED,
            command=self.stop_machine,
        )
        self.stop_button.pack(side=tk.LEFT, padx=5)

        self.step_button = tk.Button(
            control_frame, text="Шаг", state=tk.DISABLED, command=self.
                perform_step
        )
        self.step_button.pack(side=tk.LEFT, padx=5)

        self.reset_button = tk.Button(
            control_frame, text="Сброс", state=tk.DISABLED, command=self.
                reset_machine
        )
        self.reset_button.pack(side=tk.LEFT, padx=5)

        self.status_label = tk.Label(self.master, text="Статус:
            Ожиданиеконфигурации ")
        self.status_label.pack()

    def update_ui_buttons(self, running=False, loaded=False):
        self.run_button.config(
            state=tk.NORMAL if loaded and not running else tk.DISABLED
        )
        self.step_button.config(
            state=tk.NORMAL if loaded and not running else tk.DISABLED
        )
```

```
        self.reset_button.config(state=tk.NORMAL if loaded else tk.DISABLED
            )
        self.stop_button.config(state=tk.NORMAL if running else tk.DISABLED
            )
        self.load_button.config(state=tk.DISABLED if running else tk.NORMAL
            )
        self.settings_button.config(state=tk.DISABLED if running else tk.
            NORMAL)
        self.save_button.config(state=tk.NORMAL if loaded else tk.DISABLED)

    def save(self):
        filename = filedialog.asksaveasfile(filetypes=[("JSON Files", "*.
            json")])
        if not filename:
            return
        self.tm.save(filename)
        filename.close()

    def open_settings(self):
        if not self.tm:
            self.tm = TuringMachine()

        settings_window = tk.Toplevel(self.master)
        settings_window.title("Настройки машиныТьюринга ")
        settings_window.geometry("600x400")

        tab_control = ttk.Notebook(settings_window)



        alphabet_tab = ttk.Frame(tab_control)
        tab_control.add(alphabet_tab, text="Алфавит")

        tk.Label(alphabet_tab, text="Введите символыалфавитачереззапятую
            :").pack(
            pady=10
        )
        alphabet_entry = tk.Entry(alphabet_tab, width=50)
        alphabet_entry.pack(pady=5)
        alphabet_entry.insert(0, ",".join(self.tm.alphabet))
```

```
def save_alphabet():
    new_alphabet = alphabet_entry.get().split(",")
    if new_alphabet:
        self.tm.alphabet = [s.strip() for s in new_alphabet if s.
            strip()]
        self.tm.data['alphabet'] = self.tm.alphabet
        self.flag[0] = True
        if all(self.flag):
            self.update_ui_buttons(loaded=True)
        messagebox.showinfo("Успех", "Алфавит обновлён!")

tk.Button(alphabet_tab, text="Сохранить", command=save_alphabet).
    pack(pady=10)


word_tab = ttk.Frame(tab_control)
tab_control.add(word_tab, text="Слово")

tk.Label(word_tab, text="Введите начальноеслово :").pack(pady=10)
word_entry = tk.Entry(word_tab, width=50)
word_entry.pack(pady=5)


if self.tm.tape.head:
    word_entry.insert(0, "".join(str(self.tm.tape)))

def save_word():
    new_word = word_entry.get().strip()


    if any((char not in self.tm.alphabet and char != "_") for char
        in new_word):
        messagebox.showerror(
            "Ошибка", "В
                словоможновводитьтолькосимволыизалфавита      !"
        )
        return

    if new_word:
        self.tm.tape.clear()
```

```
            for char in new_word:
                self.tm.tape.push_back(char)
            self.update_tape_display()
            self.flag[1] = True
            if all(self.flag):
                self.update_ui_buttons(loaded=True)
            messagebox.showinfo("Успех", "Слово обновлено!")

    tk.Button(word_tab, text="Сохранить", command=save_word).pack(pady
        =10)


    states_tab = ttk.Frame(tab_control)
    tab_control.add(states_tab, text="Состояния")

    tk.Label(states_tab, text=f"Введите состояниячереззапятую  :").pack
        (pady=10)
    states_entry = tk.Entry(states_tab, width=50)
    states_entry.pack(pady=5)
    states_entry.insert(0, ",".join(self.tm.get_states()))

    tk.Label(states_tab, text="Начальное состояние:").pack(pady=10)
    start_state_entry = tk.Entry(states_tab, width=50)
    start_state_entry.pack(pady=5)
    start_state_entry.insert(0, self.tm.data.get("start_condition", "")
        )

    tk.Label(states_tab, text="Начальный указатель:").pack(pady=10)
    start_pointer_entry = tk.Entry(states_tab, width=50)
    start_pointer_entry.pack(pady=5)
    start_pointer_entry.insert(0, str(self.tm.data.get("start_pointer",
         "")))

    def save_states():
        new_states = states_entry.get().split(",")
        new_start_state = start_state_entry.get().strip()
        try:
            new_pointer_state = int(start_pointer_entry.get().strip())
            if self.tm.tape.len > new_pointer_state < 0:
                raise ValueError
```

```
    except ValueError:
        messagebox.showinfo(
            "Ошибка", "Начальный
                указательдолженбытьцелочисленным   !"
        )

    if new_states and new_start_state and str(new_pointer_state):
        self.tm.states = [s.strip() for s in new_states if s.strip
            ()]
        self.tm.data["start_pointer"] = new_pointer_state
        self.tm.pointer = new_pointer_state
        if new_start_state in self.tm.states:
            self.tm.data["start_condition"] = new_start_state
            self.flag[2] = True
            if all(self.flag):
                self.update_ui_buttons(loaded=True)
            messagebox.showinfo("Успех", "Состояния обновлены!")
        else:
            messagebox.showinfo(
                "Ошибка", "Начальное
                    состояниедолжнобытьспискасостояний    !"
            )
        self.update_tape_display()

tk.Button(states_tab, text="Сохранить", command=save_states).pack(
    pady=10)

# ВкладкаПравила ""
rules_tab = ttk.Frame(tab_control)
tab_control.add(rules_tab, text="Правила")

tk.Label(
    rules_tab,
    text="Введите правилапереходов :\Bn формате: state, symbol:
        new_symbol move new_state",
).pack(pady=10)
rules_text = tk.Text(rules_tab, height=10, width=60)
rules_text.pack(pady=5)

# Отображаемтекущиеправила
```

```
        rules_text.insert(1.0, self.tm.get_rules())

        def save_rules():
            new_rules = rules_text.get(1.0, tk.END).strip()
            try:
                self.tm.set_rules(new_rules=new_rules)
                self.tm.load_rules()
                self.flag[3] = True
                if all(self.flag):
                    self.update_ui_buttons(loaded=True)
                messagebox.showinfo("Успех", "Правила обновлены!")
            except Exception as e:
                messagebox.showerror("Ошибка", f"Некорректный форматправил
                    : {str(e)}")

        tk.Button(rules_tab, text="Сохранить", command=save_rules).pack(
            pady=10)

        tab_control.pack(expand=1, fill="both")

    def load_configuration(self):
        """Загрузка конфигурацииизфайла   JSON."""
        filename = filedialog.askopenfilename(filetypes=[("JSON Files", "*.
            json")])
        if not filename:
            return
        try:
            # Загружаемновуюконфигурацию
            self.tm = TuringMachine(path_to_file=filename)
            self.flag = [True, True, True, True]
            self.update_tape_display()

            # Обновляемсостояниеинтерфейса
            self.update_ui_buttons(loaded=True)
            self.status_label.config(text="Конфигурация загружена.")
            self.master.geometry(
                f"{max(self.tm.tape.len * self.CELL_SIZE + 100, 1000)}x250"
            )
        except Exception as e:
            messagebox.showerror(
```

```
                "Ошибка", f"Не удалосьзагрузитьконфигурацию  : {str(e)}"
            )

    def update_tape_display(self):
        """Обновление отображениялентынаэкране   ."""
        if not self.tm:
            return
        self.canvas.delete("all")
        self.cells = []

        current_node = self.tm.tape.head
        index = 0
        while current_node:
            x_start = index * self.CELL_SIZE
            x_end = x_start + self.CELL_SIZE
            rect = self.canvas.create_rectangle(
                x_start, 20, x_end, 70, fill="white", outline="black"
            )
            text = self.canvas.create_text(
                (x_start + x_end) // 2, 45, text=current_node.value, font=(
                    "Arial", 14)
            )
            self.cells.append((rect, text))
            if index == self.tm.pointer:
                self.canvas.itemconfig(rect, fill="lightblue")
            current_node = current_node.next
            index += 1

    def perform_step(self):
        """Выполнение одногошага ."""
        if not self.tm:
            return
        try:
            result = self.tm.step()
            self.update_tape_display()
            if not result:
                self.update_ui_buttons(loaded=True)
                messagebox.showinfo("Финал", "Работа завершена.")
        except Exception as e:
            messagebox.showerror("Ошибка", f"Произошла ошибка: {str(e)}")
```

```
def run_machine(self):
    """Полный запускмашины ."""
    if not self.tm:
        return
    self.running = True
    self.update_ui_buttons(running=True)

    def step_through():
        if not self.running:
            self.update_ui_buttons(loaded=True)
            return
        try:
            result = self.tm.step()
            self.update_tape_display()
            if not result:
                self.running = False
                self.update_ui_buttons(loaded=True)
                messagebox.showinfo("Финал", "Работа завершена.")
            else:
                self.master.after(500, step_through)
        except Exception as e:
            self.running = False
            self.update_ui_buttons(loaded=True)
            messagebox.showerror("Ошибка", f"Произошла ошибка: {str(e)}
                ")

    step_through()

def stop_machine(self):
    """Останавливает выполнениемашины ."""
    self.running = False
    self.status_label.config(text="Выполнение остановлено.")
    self.update_ui_buttons(loaded=True)

def reset_machine(self):
    """Сброс машиныкначальномусостоянию   ."""
    if not self.tm:
        return
    try:
```

```
        self.tm = TuringMachine()
        self.update_tape_display()
        self.update_ui_buttons(loaded=False)
        self.status_label.config(text="Машина сброшена.")
        self.flag = [False, False, False, False]
    except Exception as e:
        messagebox.showerror("Ошибка", f"Не удалосьсброситьмашину  : {
            str(e)}")
```