САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасик. Вариант 7.

Студент гр. 3343 _____ Иванов П. Д.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2025

Цель работы

Изучить принцип работы алгоритма Ахо-Корасик. Написать программы, которые реализуют поиск нескольких подстрок в тексте.

Задание

Задание №1.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст (T,1≤|T|≤100000T,1≤|T|≤100000).

Вторая - число nn (1≤n≤30001≤n≤3000), каждая следующая из nn строк содержит шаблон из набора P={p1,…,pn}1≤|pi|≤75P={p1,…,pn}1≤|pi|≤75 Все строки содержат символы из алфавита {A,C,G,T,N}

Выход:

Все вхождения образцов из PP в TT. Каждое вхождение образца в текст представить в виде двух чисел - i p. Где i - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером p (нумерация образцов начинается с 1). Строки выхода должны быть отсортированы по возрастанию, сначала номера позиции, затем номера шаблона.

Sample Input:

NTAG

3

TAGT

TAG

T

Sample Output:

2 2

2 3

Задание №2.

Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с джокером. В шаблоне встречается специальный символ, именуемый джокером (wild card), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу PP необходимо найти все вхождения P в текст T. Например, образец ab??c?ab??c? с джокером ? встречается дважды в тексте xabvccbababcax. Символ джокер не входит в алфавит, символы которого используются в TT. Каждый джокер соответствует одному символу, а не подстроке неопределённой длины. В шаблон входит хотя бы один символ не джокер, т.е. шаблоны вида ??? недопустимы.

Все строки содержат символы из алфавита {A,C,G,T,N}

Вход:

Текст (T,1≤|T|≤100000T,1≤|T|≤100000)

Шаблон (P,1≤|P|≤40P,1≤|P|≤40)

Символ джокера

Выход:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер). Номера должны выводиться в порядке возрастания.

Sample Input:

ACTANCA

A$$A$

$

Sample Output:

1

Вариант 7. Вывод графического представления автомата.

Выполнение работы

Для выполнения задачи №1 и задачи №2 был реализован алгоритм Ахо-Корасика с возможностью визуализации построенного автомата. Ниже приводится подробное описание всех функций, реализованных в рамках проекта.

Задача №1: Поиск точных вхождений шаблонов

Класс *AhoCorasickAutomaton*

- *__init__(self)*

Инициализирует автомат, создавая корневой узел.

Определяет алфавит: {A, C, G, T, N} и строит словарь отображения символов в индексы.

- *_create_node(self)*

Создает новый узел автомата, добавляя его в список узлов.

Возвращает индекс нового узла.

- *_char_index(self, char)*

Возвращает числовой индекс символа в алфавите.

Используется для доступа к нужному элементу массива переходов.

- *add_pattern(self, pattern, pattern_index)*

Добавляет шаблон pattern с номером pattern_index в автомат.

Постепенно проходит по символам шаблона и создает недостающие переходы.

Финальный узел получает шаблон в список выходов.

- *build(self)*

Строит суффиксные и терминальные ссылки с использованием BFS.

Все переходы из корня инициализируются либо как прямые, либо как переходы в корень.

Для каждого узла определяются:

failure_link — fallback при несовпадении символа.

term_link — ускоряет обход к узлам с выходами.

- *search(self, text)*

Ищет вхождения всех добавленных шаблонов в тексте.

Для каждого символа текста осуществляется переход по автомату, при необходимости через failure_link.

При нахождении выхода добавляется результат: (позиция, индекс шаблона).

Возвращает список найденных вхождений.

Вспомогательные функции:

- *debug_print(*args, **kwargs)*

Если включен режим отладки, выводит сообщения в консоль с префиксом [DEBUG].

- *get_result(text: str, patterns: list[str])*

Главная функция логики поиска:

  - Создает автомат.
  - Добавляет шаблоны.
  - Строит автомат.
  - Запускает поиск.

Преобразует позиции и возвращает отсортированный список результатов и сам автомат.

- *run_console()*

Читает входные данные из stdin, вызывает get_result.

Печатает все найденные вхождения в формате позиция шаблон.

- *parse_args()*

Обрабатывает аргументы командной строки: --debug, --output.

Позволяет выбрать между режимом GUI и консоли.

Задача №2: Поиск шаблона с джокерами

- *wildcard_search(text: str, pattern: str, joker: str)*

Основная функция для задачи с джокерами.

Шаг 1: Инициализация

Выводит отладочную информацию о тексте, шаблоне и символе джокера.

Шаг 2: Разбиение шаблона

Разбивает шаблон на непустые сегменты (подстроки между джокерами).

Сохраняет пары (сегмент, смещение от начала шаблона).

Шаг 3: Построение автомата

Создает автомат AhoCorasickAutomaton.

Добавляет каждый сегмент как отдельный шаблон в автомат с уникальным ID.

Строит автомат.

Визуализирует автомат с помощью AutomatonVisualizer.render_png().

Шаг 4: Поиск совпадений

Запускает поиск по тексту.

Для каждого совпадения сохраняется позиция и ID сегмента.

Шаг 5: Подсчет совпадений в возможных позициях

Для каждой позиции потенциального вхождения полного шаблона (с учетом смещения сегмента) увеличивает счётчик.

Шаг 6: Финальные совпадения

Выводит только те позиции, где совпали все сегменты шаблона.

Индивидуализация GUI: AhoGUI и визуализация автомата

- *AhoGUI.__init__(self, root)*

Создает окно GUI с вводом текста, шаблонов, кнопкой запуска и визуализацией автомата.

- *AhoGUI.run(self)*

Считывает данные из GUI, запускает get_result().

Показывает совпадения в текстовом поле.

Отображает граф автомата с помощью библиотеки PIL и Graphviz.

- *AutomatonVisualizer.render_png(automaton, filename_base="tree")*

Строит граф автомата:

Узлы отображают номер и выходные шаблоны.

Прямые переходы — обычные стрелки.

Суффиксные ссылки — пунктирные стрелки.

Терминальные ссылки — точечные стрелки.

Сохраняет изображение автомата в PNG.

Оценка сложности алгоритма

Временная сложность:

Добавление шаблонов (add_pattern):

- Для каждого шаблона длины p проход по всем символам: O(p).
- Для n шаблонов общей длины M=∑(p) суммарно — O(M).

Построение суффиксных ссылок (build):

- Узлов в автомате получается не более M+1.
- Для каждого узла и каждого символа алфавита (σ = |{A,C,G,T,N}| = 5) одно константное действие:(σ×(M+1))=O(M)
- Очередь BFS тоже обрабатывает каждый узел ровно один раз.

Поиск в тексте (search):

- Для текста длины N по каждому символу происходит:
  - переход по предвычисленным ссылкам — O(1),
  - обход терминальных ссылок при каждом совпадении.
- В худшем случае, если всего Z совпадений, суммарно — O(N+Z).

Итог для get_result(text, patterns):

Добавить шаблоны: O(M)

Построить ссылки: O(M)

Поиск: O(N + Z)

Всего по времени: O(M + N + Z)

Память:

Узлы автомата:

- Количество узлов ≤ M + 1.

Переходы:

- Каждый узел хранит массив длины σ = 5: $(M+1)\times\sigma=O(M)$

Выходы (output):

- В сумме хранятся все индексы шаблонов: O(M).

Суффиксные и терминальные ссылки

- Два целых числа на узел: O(M).

Всего по памяти: O(M)

Сложность для функции *wildcard_search(text, pattern, joker):*

Пусть |P| — длина шаблона, |T| = N, и в шаблоне K сегментов общей длины $S \leq |P|$.

1. Разбиение шаблона на сегменты — O(|P|).

2. Построение автомата для сегментов:

- Добавление сегментов: O(S).
- Построение ссылок: O(S)

3. Поиск сегментов в тексте: O(N + Z′) (Z′ — общее число вхождений сегментов).

4. Подсчёт полных совпадений (пробег по всем возможным позициям текста) — O(N).

Итоговое время: $O(|P|+S+N+Z') = O(N+|P|+Z')$.

Память: O(|P|).

Тестирование

Результаты тестирования представлены в таблице 1.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	NTAG 3 TAGT TAG T	2 2 2 3	Результат вычислен верно.
2.	ACCGTACA 2 AC GT	1 1 4 2 6 1	Результат вычислен верно.
3.	ACGT 3 ACGT CG GT	1 1 2 2 3 3	Результат вычислен верно.
4.	ACTANCA A$$A$ $	1	Результат вычислен верно.

Табл. 1. – Результаты тестирования

Выводы

Изучен принцип работы алгоритма Ахо-Корасик. Написаны программы, корректно решающие задачу поиска набора подстрок в строке, в также программа поиска подстроки с джокером.

ПРИЛОЖЕНИЕ А

1 – Файл main.py

```
import sys
from collections import deque
import argparse

DEBUG_MODE = False


def debug_print(*args, **kwargs):
    if DEBUG_MODE:
        print("[DEBUG]", *args, **kwargs)


class AhoCorasickNode:
    def __init__(self, alphabet_size):
        self.transitions = [-1] * alphabet_size
        self.output = []
        self.failure_link = -1
        self.term_link = -1

    def __repr__(self):
        return (f"Node(trans={self.transitions}, out={self.output}, "
                f"fail={self.failure_link}, term={self.term_link})")


class AhoCorasickAutomaton:
    ALPHABET = ['A', 'C', 'G', 'T', 'N']
    ALPHABET_MAP = {char: idx for idx, char in enumerate(ALPHABET)}

    def __init__(self):
        self.nodes = [AhoCorasickNode(len(self.ALPHABET))]
        debug_print("Инициализирован корень узла:", self.nodes[0])

    def _create_node(self):
        node_id = len(self.nodes)
        self.nodes.append(AhoCorasickNode(len(self.ALPHABET)))
        debug_print(f"Создан новый узел {node_id}")
        return node_id

    def _char_index(self, char):
        return self.ALPHABET_MAP[char]
```

```
        def add_pattern(self, pattern, pattern_index):
            node = 0
            debug_print(f"=== Вставка паттерна [{pattern_index}] '{pattern}'
===")
            for pos, char in enumerate(pattern):
                debug_print(f"Текущий узел: {node}, символ[{pos}]='{char}'")
                if char not in self.ALPHABET_MAP:
                    raise ValueError(f"Недопустимый символ '{char}' в паттерне
'{pattern}'")
                idx = self._char_index(char)
                next_node = self.nodes[node].transitions[idx]
                debug_print(f"  Индекс символа: {idx}, переход из {node} ->
{next_node}")
                if next_node == -1:
                    next_node = self._create_node()
                    self.nodes[node].transitions[idx] = next_node
                    debug_print(f"   Установлен переход: {node} --{char}-->
{next_node}")
                node = next_node
                debug_print(f"  Переходим в узел {node}")
            self.nodes[node].output.append(pattern_index)
            debug_print(f"Узел {node} помечен выходом для паттерна
{pattern_index}")
            debug_print("Текущее состояние узлов после вставки:")
            for i, n in enumerate(self.nodes): debug_print(f"  {i}: {n}")

        def build(self):
            queue = deque()
            root = self.nodes[0]
            root.failure_link = 0
            root.term_link = -1
            debug_print("=== Начало построения суффиксных ссылок ===")
            # Инициализация первого уровня
            for idx in range(len(self.ALPHABET)):
                child = root.transitions[idx]
                if child != -1:
                    self.nodes[child].failure_link = 0
                    self.nodes[child].term_link = -1
                    queue.append(child)
                    debug_print(f"Корневой переход по '{self.ALPHABET[idx]}' ->
узел {child}")
                else:
```

```
                    root.transitions[idx] = 0

            while queue:
                debug_print("Очередь для BFS:", list(queue))
                current = queue.popleft()
                debug_print(f"Взят из очереди узел {current}")
                for idx in range(len(self.ALPHABET)):
                    child = self.nodes[current].transitions[idx]
                    if child != -1:
                        fallback = self.nodes[current].failure_link
                        debug_print(f"   По   символу   '{self.ALPHABET[idx]}'
сначала fallback={fallback}")
                        while self.nodes[fallback].transitions[idx] == -1:
                            fallback = self.nodes[fallback].failure_link
                            debug_print(f"  Шаг назад по failure: {fallback}")
                        failure = self.nodes[fallback].transitions[idx]
                        self.nodes[child].failure_link = failure
                        if self.nodes[failure].output:
                            self.nodes[child].term_link = failure
                        else:
                            self.nodes[child].term_link                    =
self.nodes[failure].term_link
                        debug_print(f"  Для узла {child}: failure -> {failure},
term -> {self.nodes[child].term_link}")
                        queue.append(child)
                    else:
                        self.nodes[current].transitions[idx] = self.nodes[
                            self.nodes[current].failure_link
                        ].transitions[idx]
                        debug_print(f"   Доработан переход из {current} по
'{self.ALPHABET[idx]}' на {self.nodes[current].transitions[idx]}")
            debug_print("=== Завершено построение. Итоговое состояние узлов:
===")
            for i, n in enumerate(self.nodes): debug_print(f"  {i}: {n}")

        def search(self, text):
            matches = []
            node = 0
            debug_print("=== Начало поиска в тексте ===")
            for i, char in enumerate(text):
                if char not in self.ALPHABET_MAP:
                    debug_print(f"Символ   '{char}'   пропускается   (не   в
алфавите)")
```

```
                    node = 0
                    continue
                idx = self._char_index(char)
                prev_node = node
                node = self.nodes[node].transitions[idx]
                debug_print(f"Символ[{i}]='{char}',     idx={idx},     переход
{prev_node}->{node}")
                check = node
                while check != -1:
                    if self.nodes[check].output:
                        for pattern_index in self.nodes[check].output:
                            debug_print(f"  Найден паттерн {pattern_index} на
позиции {i}")
                            matches.append((i, pattern_index))
                    check = self.nodes[check].term_link
                    if check != -1:
                        debug_print(f"  Переход по term_link к узлу {check}")
            debug_print("=== Поиск завершён ===")
            return matches


    def get_result(text: str, patterns: list[str]):
        automaton = AhoCorasickAutomaton()
        lengths = [len(p) for p in patterns]
        for i, p in enumerate(patterns):
            automaton.add_pattern(p, i)
        automaton.build()
        raw = automaton.search(text)
        res = []
        for end, pid in raw:
            start = end - lengths[pid] + 1
            res.append((start+1, pid+1))
        res.sort()
        return res, automaton


    def run_console():
        data = sys.stdin.read().split()
        if len(data) < 2:
            return
        text = data[0]
        n = int(data[1])
        patterns = data[2:2+n]
```

```
        matches, _ = get_result(text, patterns)
        w = sys.stdout.write
        for pos, idx in matches:
            w(f"{pos} {idx}\n")


    def parse_args():
        parser = argparse.ArgumentParser()
        parser.add_argument("--output",             choices=["gui","console"],
default="console")
        parser.add_argument("--debug", action="store_true")
        return parser.parse_args()


    def main():
        global DEBUG_MODE
        args = parse_args()
        DEBUG_MODE = args.debug
        if args.output == "console":
            run_console()
        else:
            import main as _m
            _m.DEBUG_MODE = args.debug
            from AhoGUI import AhoGUI, Tk
            root = Tk()
            AhoGUI(root)
            root.mainloop()


    if __name__ == "__main__":
        main()
```

2 – Файл joker.py

```
import sys
from collections import deque
from AhoGUI import AutomatonVisualizer


DEBUG_MODE = True
```

```
    def debug_print(*args, **kwargs):
        if DEBUG_MODE:
            print("[DEBUG]", *args, **kwargs)


    class AhoCorasickNode:
        def __init__(self, alphabet_size):
            self.transitions = [-1] * alphabet_size
            self.output = []
            self.failure_link = -1
            self.term_link = -1

        def __repr__(self):
            return (f"Node(trans={self.transitions}, out={self.output}, "
                    f"fail={self.failure_link}, term={self.term_link})")


    class AhoCorasickAutomaton:
        ALPHABET = ['A', 'C', 'G', 'T', 'N']
        ALPHABET_MAP = {char: idx for idx, char in enumerate(ALPHABET)}

        def __init__(self):
            self.nodes = [AhoCorasickNode(len(self.ALPHABET))]
            debug_print("Инициализирован корень узла:", self.nodes[0])

        def _create_node(self):
            node_id = len(self.nodes)
            self.nodes.append(AhoCorasickNode(len(self.ALPHABET)))
            debug_print(f"Создан новый узел {node_id}")
            return node_id

        def _char_index(self, char):
            return self.ALPHABET_MAP[char]

        def add_pattern(self, pattern, pattern_index):
            node = 0
            debug_print(f"=== Вставка паттерна [{pattern_index}] '{pattern}'
===")
            for pos, char in enumerate(pattern):
                debug_print(f"Текущий узел: {node}, символ[{pos}]='{char}'")
                if char not in self.ALPHABET_MAP:
                    raise ValueError(f"Недопустимый символ '{char}' в паттерне
'{pattern}'")
```

```
                idx = self._char_index(char)
                next_node = self.nodes[node].transitions[idx]
                debug_print(f"  Индекс символа: {idx}, переход из {node} ->
{next_node}")
                if next_node == -1:
                    next_node = self._create_node()
                    self.nodes[node].transitions[idx] = next_node
                    debug_print(f"   Установлен переход: {node} --{char}-->
{next_node}")
                node = next_node
                debug_print(f"  Переходим в узел {node}")
            self.nodes[node].output.append(pattern_index)
            debug_print(f"Узел   {node}   помечен   выходом   для   паттерна
{pattern_index}")
            debug_print("Текущее состояние узлов после вставки:")
            for i, n in enumerate(self.nodes): debug_print(f"  {i}: {n}")

        def build(self):
            queue = deque()
            root = self.nodes[0]
            root.failure_link = 0
            root.term_link = -1
            debug_print("=== Начало построения суффиксных ссылок ===")
            # Инициализация первого уровня
            for idx in range(len(self.ALPHABET)):
                child = root.transitions[idx]
                if child != -1:
                    self.nodes[child].failure_link = 0
                    self.nodes[child].term_link = -1
                    queue.append(child)
                    debug_print(f"Корневой переход по '{self.ALPHABET[idx]}' ->
узел {child}")
                else:
                    root.transitions[idx] = 0

            while queue:
                debug_print("Очередь для BFS:", list(queue))
                current = queue.popleft()
                debug_print(f"Взят из очереди узел {current}")
                for idx in range(len(self.ALPHABET)):
                    child = self.nodes[current].transitions[idx]
                    if child != -1:
                        fallback = self.nodes[current].failure_link
```

```
                        debug_print(f"   По   символу   '{self.ALPHABET[idx]}'
сначала fallback={fallback}")
                        while self.nodes[fallback].transitions[idx] == -1:
                            fallback = self.nodes[fallback].failure_link
                            debug_print(f"  Шаг назад по failure: {fallback}")
                        failure = self.nodes[fallback].transitions[idx]
                        self.nodes[child].failure_link = failure
                        if self.nodes[failure].output:
                            self.nodes[child].term_link = failure
                        else:
                            self.nodes[child].term_link                        =
self.nodes[failure].term_link
                        debug_print(f"  Для узла {child}: failure -> {failure},
term -> {self.nodes[child].term_link}")
                        queue.append(child)
                    else:
                        self.nodes[current].transitions[idx] = self.nodes[
                            self.nodes[current].failure_link
                        ].transitions[idx]
                        debug_print(f"   Доработан переход из {current} по
'{self.ALPHABET[idx]}' на {self.nodes[current].transitions[idx]}")
            debug_print("=== Завершено построение. Итоговое состояние узлов:
===")
            for i, n in enumerate(self.nodes): debug_print(f"  {i}: {n}")

        def search(self, text):
            matches = []
            node = 0
            debug_print("=== Начало поиска в тексте ===")
            for i, char in enumerate(text):
                if char not in self.ALPHABET_MAP:
                    debug_print(f"Символ   '{char}'   пропускается   (не   в
алфавите)")
                    node = 0
                    continue
                idx = self._char_index(char)
                prev_node = node
                node = self.nodes[node].transitions[idx]
                debug_print(f"Символ[{i}]='{char}',     idx={idx},     переход
{prev_node}->{node}")
                check = node
                while check != -1:
                    if self.nodes[check].output:
```

```
                        for pattern_index in self.nodes[check].output:
                            debug_print(f"  Найден паттерн {pattern_index} на
позиции {i}")
                            matches.append((i, pattern_index))
                    check = self.nodes[check].term_link
                    if check != -1:
                        debug_print(f"  Переход по term_link к узлу {check}")
            debug_print("=== Поиск завершён ===")
            return matches


    def wildcard_search(text: str, pattern: str, joker: str):
        debug_print("ШАГ 1: Инициализация")
        debug_print(f"Текст: {text}")
        debug_print(f"Шаблон: {pattern}")
        debug_print(f"Джокер: '{joker}'")

        n, m = len(text), len(pattern)

        segments = []
        i = 0
        debug_print("\nШАГ 2: Разбиение шаблона на подстроки (сегменты)")
        while i < m:
            if pattern[i] == joker:
                i += 1
                continue
            j = i
            while j < m and pattern[j] != joker:
                j += 1
            segment = pattern[i:j]
            segments.append((segment, i))
            debug_print(f"  -> Сегмент '{segment}' с позицией в шаблоне {i}")
            i = j

        debug_print("\nШАГ 3: Добавление сегментов в автомат")
        aho = AhoCorasickAutomaton()
        for idx, (seg, _) in enumerate(segments):
            aho.add_pattern(seg, idx)
            debug_print(f"  -> Добавлен сегмент '{seg}' как шаблон с ID {idx}")

        aho.build()
        AutomatonVisualizer.render_png(aho)
```

```
        debug_print("\nШАГ 4: Поиск совпадений сегментов в тексте")
        raw = aho.search(text)
        if DEBUG_MODE:
            debug_print("  Результаты поиска:")
            for end_pos, seg_id in raw:
                seg, seg_off = segments[seg_id]
                debug_print(f"      -> Найден сегмент '{seg}' (ID={seg_id})
заканчивается на позиции {end_pos}")

        debug_print("\nШАГ 5: Проверка согласованности позиций")
        count = [0] * (n - m + 1)
        for end_pos, seg_id in raw:
            seg, seg_off = segments[seg_id]
            start_of_match = end_pos - len(seg) + 1
            top = start_of_match - seg_off
            debug_print(f"   -> Сегмент '{seg}' (offset {seg_off}) найден с
{start_of_match} по {end_pos} => потенциальная позиция шаблона: {top}")
            if 0 <= top <= n - m:
                count[top] += 1

        debug_print("\nШАГ 6: Финальные совпадения")
        for i in range(n - m + 1):
            if count[i] == len(segments):
                debug_print(f"  >> Шаблон совпадает на позиции {i + 1}")
                print(i + 1)
            else:
                debug_print(f"   .. Позиция {i + 1} отклонена (совпадений:
{count[i]}/{len(segments)})")


    def main():
        data = sys.stdin.read().split()
        if len(data) < 3:
            return
        T, P, W = data[0], data[1], data[2]

        wildcard_search(T, P, W)


    if __name__ == '__main__':
        main()
```

3 – Файл AhoGUI.py

```
from main import AhoCorasickAutomaton, get_result
from tkinter import Tk, Label, Button, Text, Scrollbar, filedialog, END, Frame, BOTH
from PIL import Image, ImageTk
import graphviz


class AutomatonVisualizer:
    @staticmethod
    def render_png(automaton: AhoCorasickAutomaton, filename_base: str = "tree"):
        dot = graphviz.Digraph('AhoCorasick', format='png')
        for i, node in enumerate(automaton.nodes):
            label = f"{i}\n{node.output}"
            dot.node(str(i), label)

        for u, node in enumerate(automaton.nodes):
            for idx, v in enumerate(node.transitions):
                if v != -1 and v != 0:
                    dot.edge(str(u), str(v), label=AhoCorasickAutomaton.ALPHABET[idx])

        for v, node in enumerate(automaton.nodes):
            if v != 0:
                dot.edge(str(v), str(node.failure_link), style='dashed', label='fail')
            if node.term_link != -1:
                dot.edge(str(v), str(node.term_link), style='dotted', label='term')

        output_path = dot.render(filename=filename_base, cleanup=True)
        return output_path


class AhoGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Aho-Corasick Visualizer")
        self.root.geometry("1000x700")
        self.root.grid_rowconfigure(0, weight=1)
        self.root.grid_columnconfigure(0, weight=1)
        self.root.grid_columnconfigure(1, weight=2)
```

```
            left_frame = Frame(root)
            left_frame.grid(row=0, column=0, sticky="nsew", padx=10, pady=10)
            left_frame.grid_rowconfigure(5, weight=1)

            Label(left_frame, text="Text to search in:").pack(anchor="w")
            self.text_input = Text(left_frame, height=5, wrap="word")
            self.text_input.pack(fill=BOTH, expand=False)

            Label(left_frame, text="Patterns (one per line):").pack(anchor="w",
pady=(10, 0))
            self.pattern_input = Text(left_frame, height=5, wrap="word")
            self.pattern_input.pack(fill=BOTH, expand=False)

            self.run_button = Button(left_frame, text="Build Automaton &
Visualize", command=self.run)
            self.run_button.pack(pady=10)

            Label(left_frame, text="Matches:").pack(anchor="w")
            self.result_output = Text(left_frame, height=10, wrap="word")
            self.result_output.pack(fill=BOTH, expand=True)

            right_frame = Frame(root)
            right_frame.grid(row=0, column=1, sticky="nsew", padx=10, pady=10)
            right_frame.grid_rowconfigure(0, weight=1)
            right_frame.grid_columnconfigure(0, weight=1)

            self.image_label = Label(right_frame)
            self.image_label.grid(row=0, column=0, sticky="nsew")

        def run(self):
            text = self.text_input.get("1.0", END).strip()
            patterns = self.pattern_input.get("1.0", END).strip().splitlines()
            self.result_output.delete("1.0", END)

            if not text or not patterns:
                self.result_output.insert(END, "Введите текст и хотя бы один
паттерн.\n")
                return

            try:
                matches, automaton = get_result(text, patterns)
```

```
            self.result_output.insert(END, "\n".join(map(str, matches)))

            image_path = AutomatonVisualizer.render_png(automaton)
            image = Image.open(image_path)
            image.thumbnail((800, 600), Image.LANCZOS)
            photo = ImageTk.PhotoImage(image)
            self.image_label.configure(image=photo)
            self.image_label.image = photo

        except ValueError as e:
            self.result_output.insert(END, f"Ошибка: {e}\n")
```