**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 Р в текст Т. Например, образец аb??с?аb??с? 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 — О(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)×σ=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 ≤ |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")