МИНОБРНАУКИ РОССИИ
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: Ахо-Корасик

Студент гр. 3343 _____ Гребнев Е.Д,

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2025

Цель работы.

Изучить принцип работы алгоритма Кнута-Морриса_Пратта. Написать функцию, вычисляющую для каждого элемента строки максимальное значение длины префикса и с помощью данной функции решить поставленные задачи. А именно написать программу, осуществляющую поиск вхождений подстроки в строку, а также программу, определяющую, являются ли строки циклическим сдвигом друг друга, найти индекс начала вхождения второй строки в первую.

Задание №1.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

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

Вторая - число n*n* (1≤n≤30001≤*n*≤3000), каждая следующая из n*n* строк содержит шаблон из набора P={p1,…,pn}1≤|pi|≤75*P*={*p*1,…,*pn*}1≤|*pi*|≤75

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

Выход:

Все вхождения образцов из P*P* в T*T*.

Каждое вхождение образца в текст представить в виде двух чисел - i *i* p*p*

Где i*i* - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером p*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 P*P* необходимо найти все вхождения PP в текст TT.

Например, образец ab??c?a*b*??c? с джокером ?? встречается дважды в тексте *xabvccbababcax*.

Символ джокер не входит в алфавит, символы которого используются в T*T*. 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}{*A*,*C*,*G*,*T*,*N*}

Вход:

Текст (T,1≤|T|≤100000*T*,1≤|*T*|≤100000)

Шаблон (P,1≤|P|≤40*P*,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 1. На месте джокера может быть любой символ, за исключением заданного.

Описание алгоритмов.

Описание алгоритма Ахо-Корасик.

Алгоритм создает префиксное дерево из букв искомых подстрок. Затем в полученном дереве ищутся суффиксные ссылки. Суффиксная ссылка вершины u – это вершина v, такая что строка v является максимальным суффиксом строки u. Для корня и вершин, исходящих из корня, суффиксной ссылкой является корень. Для остальных вершин осуществляется переход по суффиксной ссылке родителя и, если оттуда есть ребро с заданным символом, суффиксная ссылка назначается в вершину, куда это ребро ведет. Далее создаются терминальные ссылки – такие суффиксные ссылки, которые ведут в вершину, которая является терминальной.

Текст, в котором нужно найти подстроки побуквенно передается в автомат. Начиная из корня, автомат переходит по ребру, соответствующему переданному символу. Если нужного ребра нет, переходит по ссылке. Если встреченная вершина является терминальной, значит была встречена подстрока. Если

найдено совпадение нужно пройти по терминальным ссылкам, если они не None, чтобы вывести все шаблоны заканчивающиеся на этом месте. Номер подстроки (подстрок) хранится в поле *terminate* вершины. В ответ сохранятся индекс, на котором началась эта подстрока в тексте и сам номер подстроки.
алгоритма для нахождения шаблонов с маской.

Описание модифицированного алгоритма.

Алгоритм тот же, но в качестве подстрок берутся кусочки шаблона, разделенные джокером, запоминаются позиции полученных подстрок в исходном шаблоне. Создается массив C длины s, где s – длина текста, где ищется шаблон. При нахождении подстроки, в массиве C увеличивается на единицу число по индексу, соответствующему возможному началу шаблона. Индекс высчитывается по формуле: текущий индекс - (длина найденной подстроки - 1) - (позиция подстроки в шаблоне -1). Затем проходим по полученному массиву, каждый i для которого C[i] = количеству подстрок, является вероятным началом шаблона. В соответствии с индивидуализацией, для каждого найденного шаблона проверяются буквы, стоящие на месте джокера. Если не было встречено запрещенного символа, найденный шаблон добавляется в ответ.

1. **Класс** `Node`

Представляет узел в префиксном дереве (trie). Содержит:

- `parent`: ссылка на родительский узел.
- `children`: словарь дочерних узлов (ключ — символ, значение — узел).
- `suffix_link`: суффиксная ссылка для оптимизации переходов при несовпадении символа.
- `terminal_link`: ссылка на ближайший терминальный узел.
- `terminate`: идентификатор шаблона, если узел терминальный.
- `name`: символьное имя узла (для отладки).
- Логирование состояния через `verbose`.

2. **Класс** `AhoCorasickAutomaton`

Реализует автомат для поиска множества шаблонов в тексте. Основные этапы:

- Инициализация дерева (`_build_automaton`).
- Построение trie (`build_trie`).
- Создание суффиксных (`build_suffix_links`) и терминальных ссылок (`build_terminal_links`).
- Метод `search_patterns` для поиска вхождений в текст.

3. **Метод** `build_trie`

Строит trie из переданных шаблонов. Для каждого символа шаблона:

- Создает дочерние узлы, если их нет.
- Помечает терминальные узлы идентификаторами шаблонов.
- Логирует процесс при `verbose=True`.

4. **Метод** `_build_suffix_links`

Создает суффиксные ссылки через обход в ширину (BFS):

- Для корневых детей ссылки ведут в корень.
- Для остальных узлов: переход по суффиксным ссылкам родителя до нахождения совпадения символа.

5. **Метод** `build_terminal_links`

Создает терминальные ссылки через BFS:

- Для каждого узла проверяет суффиксные ссылки до первого терминального узла.
- Устанавливает `terminal_link` на найденный терминальный узел.

6. **Метод** `search_patterns`

Алгоритм поиска:

- Посимвольный обход текста с переходами по суффиксным ссылкам при несовпадении.

- Проверка терминальных узлов через `terminal_link` для обнаружения вхождений.
- Возвращает позиции найденных шаблонов в формате `[start_pos, pattern_id]`.

7. **Функция** `read_input`

Читает шаблоны из ввода, присваивая уникальные идентификаторы.
Формат ввода:

- Первая строка — число шаблонов `n`.
- Следующие `n` строк — шаблоны.

8. **Функция** `main`

Координирует процесс:

- Чтение текста и шаблонов.
- Построение автомата.
- Запуск поиска и вывод результатов.

Описание расширенной версии с джокерами (task2.py)

1. **Класс** `TrieNode`

Расширенная версия узла:

- `depth`: глубина узла в дереве (используется для определения позиции в тексте).
- `terminate`: список смещений для шаблонов, заканчивающихся в этом узле.
- Остальные поля аналогичны `Node` из task1.

2. **Класс** `Trie`

Инкапсулирует логику построения trie:

- `add_pattern`: добавляет шаблон в дерево с указанием смещений.
- Логирует создание узлов при `verbose=True`.

3. **Класс** `AhoCorasick`

Расширенный автомат для работы с джокерами:

- Принимает шаблоны с указанием смещений (например, части шаблона между джокерами).
- Методы `_build_suffix_links` и `_build_terminal_links` аналогичны task1, но с поддержкой `depth`.

4. **Метод** `search`

Поиск с учетом джокеров:

- Подсчитывает количество совпадений подшаблонов для каждой позиции текста.
- Проверяет, чтобы совпали все части исходного шаблона (игнорируя джокеры).
- Возвращает позиции, где исходный шаблон полностью совпадает с текстом.

5. **Функция** `get_pattern_parts`

Разделяет шаблон с джокерами на подшаблоны:

- Например, шаблон `ab*c?d` разделяется на `ab, c` с соответствующими смещениями.
- Возвращает словарь вида `{подшаблон: [список_смещений]}`.

6. **Функция** `main`

Логика для задачи с джокерами:

- Ввод текста, шаблона с джокерами и символа-джокера.
- Разделение шаблона на части.
- Построение автомата и поиск.
- Вывод позиций, где шаблон полностью совпадает с текстом.

task1.py (Базовый алгоритм Ахо-Корасик)

Временная сложность:

1. **Построение trie:**
   - Зависит от суммы длин всех шаблонов: **$O(\Sigma|P_i|)$**, где $|P_i|$ — длина i-го шаблона.
2. **Суффиксные ссылки:**
   - Обработка всех узлов через BFS: **$O(\Sigma|P_i|)$**.
3. **Терминальные ссылки:**
   - Обход узлов и переходы по суффиксным ссылкам: **$O(\Sigma|P_i|)$**.
4. **Поиск в тексте:**
   - Линейный проход по тексту: **$O(|T|)$**, где $|T|$ — длина текста.
   - Сбор результатов: **$O(Z)$**, где Z — количество найденных вхождений.

Итог по времени:

$O(\Sigma|P_i| + |T| + Z)$.

Пространственная сложность:

- Хранение trie: **$O(\Sigma|P_i|)$**.
- Дополнительные структуры для ссылок: **$O(\Sigma|P_i|)$**.

Итог по памяти:

$O(\Sigma|P_i|)$.

---

task2.py (Расширение с джокерами)

Временная сложность:

1. **Разделение шаблона на подшаблоны (**get_pattern_parts**):**
   - Линейный проход по исходному шаблону: **$O(L)$**, где L — длина шаблона с джокерами.
2. **Построение trie:**

- Сумма длин всех подшаблонов: **O(Σ|S_j|)**, где |S_j| — длина j-го подшаблона.

3. **Суффиксные и терминальные ссылки:**
   - Аналогично task1.py: **O(Σ|S_j|)**.
4. **Поиск в тексте (**search**):**
   - Линейный проход текста: **O(|T|)**.
   - Проверка совпадений всех подшаблонов: **O(|T| · L)** (из-за цикла по длине исходного шаблона).
   - Сбор результатов: **O(Z')**, где Z' — количество совпадений.

Итог по времени:

O(Σ|S_j| + |T| · L + Z').

Пространственная сложность:

- Хранение trie: **O(Σ|S_j|)**.
- Массив для подсчёта совпадений: **O(|T|)**.

Итог по памяти:

O(Σ|S_j| + |T|).

Тестирование.

Входные данные	Ответ	Комментарий
NTAG 3 TAGT TAG T	2 2 2 3	Верно
ACCGTACA 2 AC GT	1 1 4 2 6 1	Верно
ACGT 3 ACGT CG GT	1 1 2 2 3 3	Верно

Таблица 1 – Тестирование алгоритма Ахо-Корасик

Входные данные	Ответ	Комментарий
ACTANCA A$$A$ $ G	1	Верно
ACACAA ACXA X	3	Верно
ACGANGAAAT A$G $	1 4	Верно

Таблица 1 – Тестирование алгоритма поиска с джокером

Результат работы программы с отладочным выводом для первого задания (см. рис 1, 2, 3).

```
D:\Documents\Educating\PIAA\lb5>python ./task1.py < data_task1
      • Добавлен шаблон №1: 'TAGT'
      • Добавлен шаблон №2: 'TAG'
      • Добавлен шаблон №3: 'T'
✓ Дерево шаблонов инициализировано

========== ПОСТРОЕНИЕ ДЕРЕВА ШАБЛОНОВ ==========
→ Добавляем шаблон 'TAGT' (№1)
  └─ Создан узел 'T'
  └─ Создан узел 'A'
  └─ Создан узел 'G'
  └─ Создан узел 'T'
⇒ Шаблон 'TAGT' помечен в узле T
→ Добавляем шаблон 'TAG' (№2)
      • Узел для 'T' уже существует
      • Узел для 'A' уже существует
      • Узел для 'G' уже существует
⇒ Шаблон 'TAG' помечен в узле G
→ Добавляем шаблон 'T' (№3)
      • Узел для 'T' уже существует
⇒ Шаблон 'T' помечен в узле T
```

Рисунок 1

```
========== ПОСТРОЕНИЕ СУФФИКСНЫХ ССЫЛОК ==========
      • Суфф. ссылка для 'T' → корень
→ Обработка узла 'T'
  └─ Добавлен в очередь 'A'
⇒ Суфф. ссылка для 'A' → 'root'
→ Обработка узла 'A'
  └─ Добавлен в очередь 'G'
⇒ Суфф. ссылка для 'G' → 'root'
→ Обработка узла 'G'
  └─ Добавлен в очередь 'T'
⇒ Суфф. ссылка для 'T' → 'T'
→ Обработка узла 'T'

========== ПОСТРОЕНИЕ ТЕРМИНАЛЬНЫХ ССЫЛОК ==========
→ Узел 'T'
→ Узел 'A'
→ Узел 'G'
→ Узел 'T'
⇒ Термин. ссылка для 'T' → 'T'
✓ Автомат готов к поиску
```

Рисунок 2

```
========== ПОИСК ПО ТЕКСТУ ==========
→ Символ 'N' (i=0)
    • 'N' нет в 'root', следуем по суфф. ссылке
→ Символ 'T' (i=1)
✓ Найден шаблон 'T' в позиции 2
→ Символ 'A' (i=2)
→ Символ 'G' (i=3)
✓ Найден шаблон 'TAG' в позиции 2
⇒ Всего совпадений: 2

========== РЕЗУЛЬТАТЫ ПОИСКА ==========
2 2
2 3
```

Рисунок 3

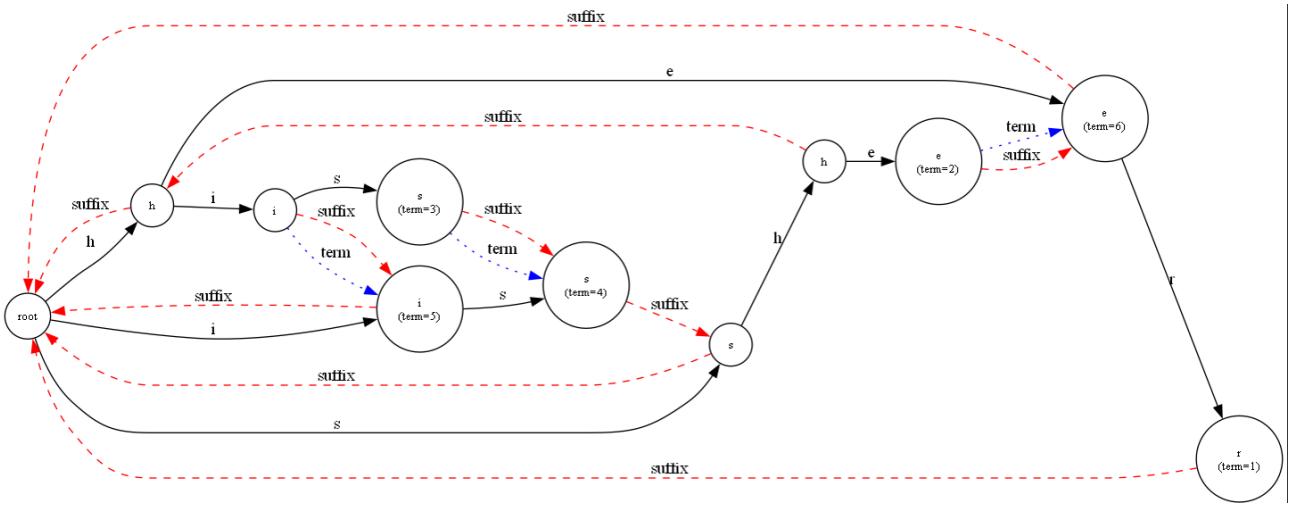

Рис. 4 – Визуализация для {'her':1, 'she':2, 'his':3, 'is':4, 'i':5, 'he':6}

Исследование.

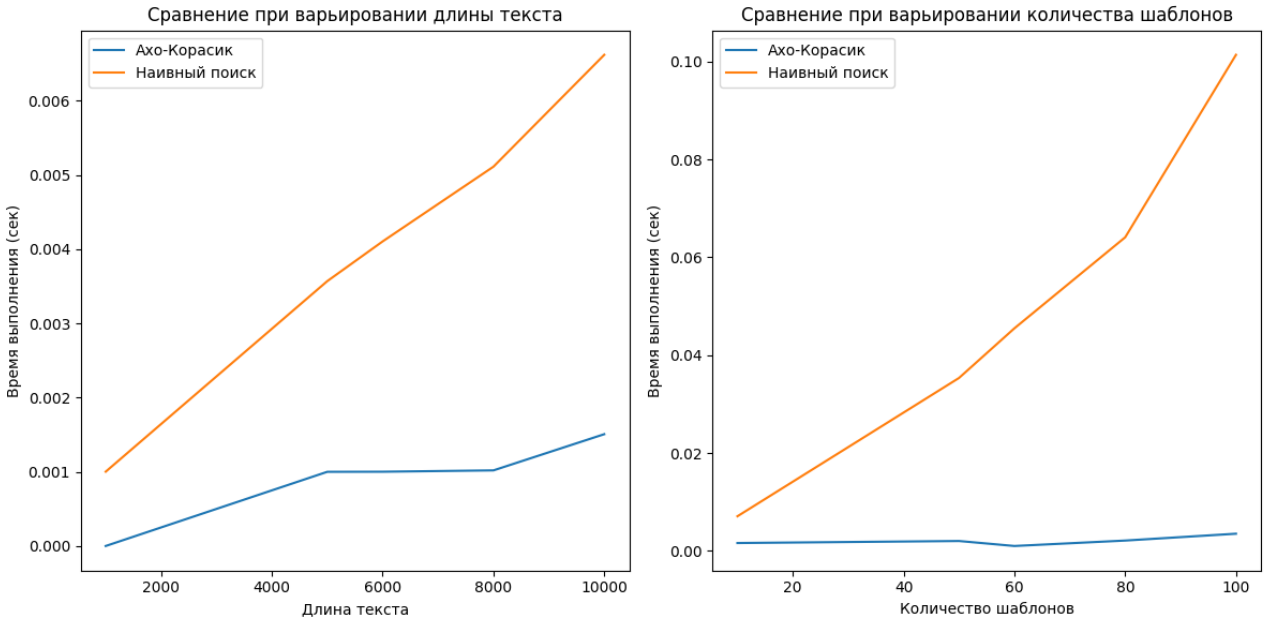

Рисунок 8 – Сравнение алгоритма Ахо-Корасик и наивного поиска

Можно сделать вывод, что Ахо-Корасик выполняется значительно быстрее, чем наивный алгоритм.

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
ИСХОДНЫЙ КОД ПРОГРАММЫ

task1.py

```
from operator import itemgetter
from collections import deque
from utils import *

class Node:
    def __init__(self, link=None, name="root", verbose=False):
        self.parent = None
        self.children = {}
        self.suffix_link = link
        self.terminal_link = None
        self.terminate = 0  # Номер шаблона, если узел терминальный
        self.name = name    # Имя узла (символ или "root")
        self.verbose = verbose

    def __str__(self):
        parent_name = self.parent.name if self.parent else None
        suffix_name = self.suffix_link.name if self.suffix_link else
None
        children_keys = list(self.children.keys()) if self.children
else None

        return (
            f"Node(name={self.name}, "
            f"parent={parent_name}, "
            f"children={children_keys}, "
            f"suffix_link={suffix_name}, "
            f"terminate={self.terminate})"
        )

class AhoCorasickAutomaton:
    def __init__(self, patterns: dict, verbose=False):
        self.verbose = verbose
        if self.verbose:
            log_success("Дерево шаблонов инициализировано")
        self.root = Node(verbose=verbose)
        self.patterns = patterns
        self.patterns_inverse = {v: k for k, v in patterns.items()}
```

```
            self._build_automaton()
            if self.verbose:
                log_success("Автомат готов к поиску")

        def _build_automaton(self):
            """Построение автомата Ахо-Корасик в три этапа"""
            self._build_trie()
            self._build_suffix_links()
            self._build_terminal_links()

        def _build_trie(self):
            """Построение исходного бора"""
            if self.verbose:
                log_section("Построение дерева шаблонов")
            for pattern, pattern_id in self.patterns.items():
                if self.verbose:
                            log_substep(f"Добавляем  шаблон  '{pattern}'
(№{pattern_id})")
                current_node = self.root
                for symbol in pattern:
                    if symbol not in current_node.children:
                           new_node = Node(name=symbol, link=self.root,
verbose=self.verbose)
                        new_node.parent = current_node
                        current_node.children[symbol] = new_node
                        if self.verbose:
                            log_action(f"Создан узел '{symbol}'")
                        current_node = new_node
                    else:
                        current_node = current_node.children[symbol]
                        if self.verbose:
                          log_info(f"Узел для '{symbol}' уже существует")
                current_node.terminate = pattern_id
                if self.verbose:
                        log_result(f"Шаблон  '{pattern}'  помечен  в  узле
{current_node.name}")

        def _build_suffix_links(self):
            """Построение суффиксных ссылок в ширину (BFS)"""
            if self.verbose:
                log_section("Построение суффиксных ссылок")
            queue = deque(self.root.children.values())

            # Для детей корня суффиксная ссылка ведет в корень
```

```
            for child in queue:
                child.suffix_link = self.root
                if self.verbose:
                   log_info(f"Суфф. ссылка для '{child.name}' → корень")

            while queue:
                current_node = queue.popleft()
                if self.verbose:
                    log_substep(f"Обработка узла '{current_node.name}'")

                for child in current_node.children.values():
                    queue.append(child)
                    if self.verbose:
                        log_action(f"Добавлен в очередь '{child.name}'")
                    self._set_suffix_link_for_child(child)
                    if self.verbose:
                         log_result(f"Суфф. ссылка для '{child.name}' →
'{child.suffix_link.name}'")

        def _set_suffix_link_for_child(self, node):
            """Установка суффиксной ссылки для конкретного узла"""
            link = node.parent.suffix_link

            while link and (node.name not in link.children):
                link = link.suffix_link

            if link:
                      node.suffix_link  =  link.children.get(node.name,
self.root)
            else:
                node.suffix_link = self.root

        def _build_terminal_links(self):
            """Построение терминальных ссылок в ширину (BFS)"""
            if self.verbose:
                log_section("Построение терминальных ссылок")
            queue = deque(self.root.children.values())

            while queue:
                current_node = queue.popleft()
                if self.verbose:
                    log_substep(f"Узел '{current_node.name}'")

                # Добавляем детей текущего узла в очередь
```

```
                queue.extend(current_node.children.values())

                  # Ищем ближайшую терминальную вершину по суффиксным
ссылкам
                temp = current_node.suffix_link
                while temp != self.root:
                    if temp.terminate:
                        current_node.terminal_link = temp
                        if self.verbose:
                                        log_result(f"Термин. ссылка для
'{current_node.name}' → '{temp.name}'")
                        break
                    temp = temp.suffix_link

        def search_patterns(self, text: str) -> list[str]:
            """Поиск всех вхождений шаблонов в тексте"""
            if self.verbose:
                log_section("Поиск по тексту")
            results = []
            current_node = self.root

            for position, symbol in enumerate(text):
                if self.verbose:
                    log_substep(f"Символ '{symbol}' (i={position})")

                  # Переходим по суффиксным ссылкам, пока не найдем
подходящего ребенка
                        while current_node and symbol not in
current_node.children:
                    if self.verbose:
                      log_info(f"'{symbol}' нет в '{current_node.name}',
следуем по суфф. ссылке")
                    current_node = current_node.suffix_link

                if current_node:
                       current_node = current_node.children.get(symbol,
self.root)
                    temp_node = current_node

                  # Проверяем терминальные узлы по терминальным ссылкам
                    while temp_node:
                        if temp_node.terminate:
                                                        pattern =
self.patterns_inverse[temp_node.terminate]
```

```
                                start_pos = position - len(pattern) + 2  #
+2 для 1-based индексации
                                                    results.append([start_pos,
temp_node.terminate])
                                if self.verbose:
                                    log_success(f"Найден шаблон '{pattern}'
в позиции {start_pos}")
                            temp_node = temp_node.terminal_link
                    else:
                        current_node = self.root

                # Сортируем результаты и форматируем вывод
                results.sort(key=itemgetter(0, 1))
                formatted = [' '.join(map(str, item)) for item in results]
                if self.verbose:
                    log_result(f"Всего совпадений: {len(formatted)}")
                return formatted

        def read_input(verbose=False):
            """Чтение входных данных"""
            n = int(input())
            patterns = {}

            for pattern_id in range(1, n+1):
                pattern = input().strip()
                patterns[pattern] = pattern_id
                if verbose:
                    log_info(f"Добавлен шаблон №{pattern_id}: '{pattern}'")

            return patterns

        def main():
            verbose = True

            text = input().strip()
            patterns = read_input(verbose)
            automaton = AhoCorasickAutomaton(patterns, verbose)
            results = automaton.search_patterns(text)

            if results:
                if verbose:
                    log_section("Результаты поиска")
                print('\n'.join(results))
            else:
```

```
            log_warning("Совпадения не найдены")

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

task2.py

```
from collections import deque
from utils import *

# === Структуры ===
class TrieNode:
    def __init__(self, parent=None, name="root", verbose=False):
        self.parent = parent
        self.children = {}
        self.suffix_link = None
        self.terminal_link = None
        self.terminate = []
        self.name = name
        self.depth = parent.depth + 1 if parent else 0
        self.verbose = verbose

    def __repr__(self):
          return f"TrieNode(name='{self.name}', depth={self.depth},
terminate={self.terminate})"

class Trie:
    def __init__(self, verbose=False):
        self.root = TrieNode(verbose=verbose)
        self.verbose = verbose
        if self.verbose:
            log_success("Дерево шаблонов инициализировано")

    def add_pattern(self, pattern, offsets):
        if self.verbose:
           log_substep(f"Добавление шаблона: '{pattern}' с оффсетами:
{offsets}")
        node = self.root
        for symbol in pattern:
            if symbol not in node.children:
                if self.verbose:
                     log_action(f"Создание нового узла для символа:
'{symbol}'")
                     node.children[symbol] = TrieNode(parent=node,
name=symbol, verbose=self.verbose)
            else:
                if self.verbose:
                    log_action(f"Используется существующий узел для
символа: '{symbol}'")
            node = node.children[symbol]
```

```
            node.terminate = offsets
            if self.verbose:
                    log_result(f"Шаблон '{pattern}' успешно добавлен.
Терминальный узел: {node}")

    class AhoCorasick:
        def __init__(self, patterns, verbose=False):
            self.verbose = verbose
            if self.verbose:
                log_section("Инициализация автомата Ахо-Корасика")
            self.trie = Trie(verbose=verbose)
            self.patterns = patterns
            self._build_trie()
            self._build_suffix_links()
            self._build_terminal_links()
            if self.verbose:
                log_success("Построение автомата завершено")

        def _build_trie(self):
            if self.verbose:
                log_section("Построение дерева шаблонов")
            for pattern, offsets in self.patterns.items():
                self.trie.add_pattern(pattern, offsets)

        def _build_suffix_links(self):
            if self.verbose:
                log_section("Построение суффиксных ссылок")
            root = self.trie.root
            queue = deque(root.children.values())

            for child in queue:
                child.suffix_link = root
                if self.verbose:
                        log_info(f"Суффиксная ссылка для '{child.name}'
установлена на корень")

            while queue:
                current = queue.popleft()
                if self.verbose:
                        log_substep(f"Обработка узла: '{current.name}'
(глубина: {current.depth})")

                for symbol, child in current.children.items():
                    queue.append(child)
```

```
                    if self.verbose:
                        log_action(f"Потомок '{child.name}' добавлен в
очередь")

                    link = current.suffix_link
                    while link is not None and symbol not in link.children:
                        if self.verbose:
                            log_info(f"Символ '{symbol}' не найден в
'{link.name}', переход по ссылке")
                        link = link.suffix_link

                    if link and symbol in link.children:
                        child.suffix_link = link.children[symbol]
                        if self.verbose:
                            log_result(f"Суффиксная ссылка для
'{child.name}' установлена на '{child.suffix_link.name}'")
                    else:
                        child.suffix_link = root
                        if self.verbose:
                            log_result(f"Суффиксная ссылка для
'{child.name}' установлена на корень")

        def _build_terminal_links(self):
            if self.verbose:
                log_section("Построение терминальных ссылок")
            root = self.trie.root
            queue = deque(root.children.values())

            while queue:
                current = queue.popleft()
                if self.verbose:
                    log_substep(f"Обработка узла: '{current.name}'")

                queue.extend(current.children.values())

                temp = current.suffix_link
                while temp and temp != root:
                    if temp.terminate:
                        current.terminal_link = temp
                        if self.verbose:
                            log_result(f"Терминальная ссылка для
'{current.name}' установлена на '{temp.name}'")
                        break
                    temp = temp.suffix_link
```

```
        def search(self, text, pattern_input, joker):
            if self.verbose:
                log_section("Поиск по тексту")
            result = [0] * len(text)
            node = self.trie.root

            for index, symbol in enumerate(text):
                if self.verbose:
                    log_substep(f"Символ '{symbol}' на позиции {index +
1}")
                while node and symbol not in node.children:
                    node = node.suffix_link

                if node:
                    node = node.children.get(symbol, self.trie.root)
                else:
                    node = self.trie.root

                temp = node
                while temp:
                    if temp.terminate:
                        matched = text[index - temp.depth + 1: index +
1]
                        for offset in self.patterns[matched]:
                            pos = index - temp.depth - offset + 1
                            if pos >= 0:
                                result[pos] += 1
                    temp = temp.terminal_link

            output = []
            total_patterns = sum(len(v) for v in self.patterns.values())
            for i in range(len(result) - len(pattern_input) + 1):
                if result[i] == total_patterns:
                    valid = True
                    for j in range(len(pattern_input)):
                        if pattern_input[j] == joker and text[i + j] ==
joker:
                            valid = False
                            break
                    if valid:
                        output.append(str(i + 1))

            return output
```

```
    def get_pattern_parts(pattern, joker, verbose=False):
        if verbose:
            log_section("Разделение шаблона по джокеру")
        parts = {}
        last_j = -1

        for i, char in enumerate(pattern):
            if char == joker:
                if last_j < i - 1:
                    sub = pattern[last_j + 1: i]
                    parts.setdefault(sub, []).append(last_j + 1)
                    if verbose:
                        log_action(f"Найден подшаблон: '{sub}' (оффсет:
{last_j + 1})")
                last_j = i

        if last_j != len(pattern) - 1:
            sub = pattern[last_j + 1:]
            parts.setdefault(sub, []).append(last_j + 1)
            if verbose:
                log_action(f"Найден подшаблон: '{sub}' (оффсет: {last_j
+ 1})")

        if verbose:
            log_success(f"Итоговые подшаблоны: {parts}")
        return parts

    def main():
        verbose = False
        text = input().strip()
        pattern_input = input().strip()
        joker = input().strip()

        patterns = get_pattern_parts(pattern_input, joker, verbose)

        if not patterns:
            log_warning("Не найдено ни одного подшаблона")
            return

        automaton = AhoCorasick(patterns, verbose)
        result = automaton.search(text, pattern_input, joker)

        if result:
```

```
            if verbose:
                log_section("Результаты")
            print('\n'.join(result))
        else:
            log_warning("Шаблон не найден в тексте")

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

visualization.py

```
from graphviz import Digraph

class Visualizer:
    def __init__(self, automaton):
        self.automaton = automaton
        self.graph = Digraph(format='png')
        self.graph.attr('node', shape='circle', fontsize='10')
        self.graph.attr(rankdir='LR')

    def _add_nodes(self, node, visited):
        # Label includes name and terminate flag
        label = f"{node.name}"
        if node.terminate:
            label += f"\\n(term={node.terminate})"
        self.graph.node(str(id(node)), label)
        visited.add(node)
        for child in node.children.values():
            if child not in visited:
                self._add_nodes(child, visited)

    def _add_trie_edges(self, node, visited):
        visited.add(node)
        for symbol, child in node.children.items():
            self.graph.edge(str(id(node)),          str(id(child)),
label=symbol)
            if child not in visited:
                self._add_trie_edges(child, visited)

    def _add_suffix_edges(self, node, visited):
        visited.add(node)
        if node is not self.automaton.root:
            target = node.suffix_link
            self.graph.edge(
```

```
                str(id(node)), str(id(target)),
                label='suffix', style='dashed', color='red'
            )
        for child in node.children.values():
            if child not in visited:
                self._add_suffix_edges(child, visited)

    def _add_terminal_edges(self, node, visited):
        visited.add(node)
        if node.terminal_link:
            target = node.terminal_link
            self.graph.edge(
                str(id(node)), str(id(target)),
                label='term', style='dotted', color='blue'
            )
        for child in node.children.values():
            if child not in visited:
                self._add_terminal_edges(child, visited)

    def render(self, filename='aho_automaton'):
        # Build graph
        self._add_nodes(self.automaton.root, set())
        self._add_trie_edges(self.automaton.root, set())
        self._add_suffix_edges(self.automaton.root, set())
        self._add_terminal_edges(self.automaton.root, set())

        # Render to file
        self.graph.render(filename, view=True)


from task1 import AhoCorasickAutomaton
patterns = {'her':1, 'she':2, 'his':3, 'is':4, 'i':5, 'he':6}
automaton = AhoCorasickAutomaton(patterns, verbose=False)
vis = Visualizer(automaton)
vis.render('aho_visualization')
```