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

ОТЧЕТ

по лабораторной работе 6
по дисциплине «Построение и анализ алгоритмов»
Тема: Ахо-Корасик

Студент гр. 3384 _____ Козьмин Н.В.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург
2025

Цель работы.

Разработать программы, которые обрабатывают строки с помощью алгоритма Ахо-Корасика.

Задание.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст (T, $1 \leq |T| \leq 100000$).

Вторая - число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P = \{p_1,...,p_n\}$, $1 \leq |p_i| \leq 75$

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

Выход:

Все вхождения образцов из P в T. Каждое вхождение представить в виде двух чисел - i p, где i - позиция в тексте (нумерация с 1), с которой начинается вхождение образца с номером p (нумерация с 1). Вывод отсортировать по возрастанию: сначала по позиции, затем по номеру шаблона.

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 необходимо найти все вхождения P в текст T.

Например, образец "ab??c?" с джокером "?" встречается дважды в тексте "xabvccbababcax".

Символ джокер не входит в алфавит, символы которого используются в 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}

Вход:

Текст (T, $1 \leq |T| \leq 100000$)

Шаблон (P, $1 \leq |P| \leq 40$)

Символ джокера

Выход:

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

Описание работы.

Был выбран Python, так как не требуется высокая скорость выполнения и писать на этом ЯП быстрее и проще.

Первым делом был сделан узел для составления бора в алгоритме Ахо-Корасика. Он содержит transitions для хранения ключей-символов и значений-других узлов. Здесь и в дальнейшем использованы аннотации типов для большего удобства чтения и подсветки в редакторе кода; suffix_link содержит следующий по размеру суффикс, имеющийся в боре, output – терминальный узел, чтобы оптимально найти следующий нужный элемент, не проходя по всем суффиксным узлам, pattern_indices, замененный потом на offsets – индексы шаблонов, заканчивающихся в узле.

Затем был реализован алгоритм для построения бора в функции build_aho_corasick. В нем сначала заполняется “скелет” бора, а потом с помощью прохода в ширину, который также можно было заменить на ленивый проход в длину, добавляются суффиксальные ссылки вместе с выходными (аналог алгоритма KMP для бора).

Далее был сделан сам поиск Ахо-Корасика. Используя созданный автомат состояний из бора, при нахождении терминального узла добавляются вхождения в список. После этого была сделана проверка.

Затем pattern_indices были заменены, так как для решения второй задачи, нужны смещения подшаблонов в основном шаблоне вместо индексов образцов. В качестве шаблонов, было использовано разделение основного шаблона по символу джокера в функции split_pattern. Там же к паттернам добавляются отступы, которые после используются вместо индексов в build_aho_corasick.

В самом поиске были сделаны следующие изменения: теперь результатом search_aho_corasick стал список из элементов формата: отступ в шаблоне, позиция начала подшаблона в тексте. А для формирования конечного ответа была сделана функция find_wildcard_matches. В ней организуются найденные позиции по смещениям подшаблонов и для каждой группы вхождений вычисляется возможное начало полного шаблона с постепенной проверкой каждого символа.

Описание функций.

build_aho_corasick, используя шаблоны, строит по ним бор с нужными ссылками, возвращая корневой узел.

search_aho_corasick возвращает список отступов в шаблоне с позициями начал подшаблонов в тексте, используя исходный текст, корневой узел бора и шаблоны со смещениями, чтобы по перовому находить второе.

split_pattern делит по джокеру (разделителю) основной шаблон на подшаблоны, добавляя отступы.

find_wildcard_matches, используя предыдущие функции, по тексту, шаблону и разделителю возвращает отсортированный список всех позиций, где найден полный шаблон.

В main`е вызываются созданные функции.

Выводы.

В ходе выполнения работы основной целью было разработать обработчики строк с помощью алгоритма Ахо-Корасика. Реализованные программы успешно решают поставленные задачи.

Сначала был сделан код для первой задачи, а потом логично модернизирован в код для второй. В main каждого файла был вынесен результат с решением соответствующей задачи. Разработанный программный код см. в приложении А.

Код получился лаконичным и работающим эффективно в естественных ограничениях языка. В работе из предыдущих тем были использованы граф (для бора) и обход в ширину.

Решения поставленных задач также были протестированы. Результаты тестирования см. в приложении Б.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: classic_aho_corasick.py

```
import sys
from collections import deque


class AhoCorasickNode:
    def __init__(self):
        self.transitions: dict[str, AhoCorasickNode] = {}
        self.suffix_link = None
        self.output = None
        self.pattern_indices = []


def build_aho_corasick(patterns):
    root = AhoCorasickNode()

    for i, pattern in enumerate(patterns):
        node = root
        for c in pattern:
            if c not in node.transitions:
                node.transitions[c] = AhoCorasickNode()
            node = node.transitions[c]
        node.pattern_indices.append(i + 1)

    root.suffix_link = root
    queue = deque()
    for child in root.transitions.values():
        child.suffix_link = root
        queue.append(child)
    while queue:
        current_node: AhoCorasickNode = queue.popleft()
        for c, child in current_node.transitions.items():
            queue.append(child)
            suffix_node: AhoCorasickNode = current_node.suffix_link
            while suffix_node is not root and c not in
suffix_node.transitions:
                suffix_node = suffix_node.suffix_link
            child.suffix_link = suffix_node.transitions.get(c, root)
            child.output = child.suffix_link if
child.suffix_link.pattern_indices else child.suffix_link.output

    return root


def search_aho_corasick(text, root, patterns):
    occurrences = []
    current_node = root
    for i, c in enumerate(text, 1):
        while current_node is not root and c not in
current_node.transitions:
            current_node: AhoCorasickNode = current_node.suffix_link
```

```
            if c in current_node.transitions:
                current_node = current_node.transitions[c]
            else:
                current_node = root

            temp_node = current_node
            while temp_node != None and temp_node is not root:
                for pattern_idx in temp_node.pattern_indices:
                    pattern_length = len(patterns[pattern_idx - 1])
                    start_pos = i - pattern_length + 1
                    occurrences.append((start_pos, pattern_idx))
                temp_node = temp_node.output
        return occurrences


    def main():
        input_lines = sys.stdin.read().splitlines()
        text = input_lines[0].strip()
        n_patterns = int(input_lines[1].strip())
        patterns = [line.strip() for line in input_lines[2:2 +
n_patterns]]

        root = build_aho_corasick(patterns)
        occurrences = search_aho_corasick(text, root, patterns)

        occurrences.sort()
        for pos, pattern_idx in occurrences:
            print(pos, pattern_idx)

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

Название файла: aho_corasick_with_joker.py

```
    from collections import deque, defaultdict


    class AhoCorasickNode:
        def __init__(self):
            self.transitions: dict[str, AhoCorasickNode] = {}
            self.suffix_link = None
            self.output = None
            self.offsets = []


    def build_aho_corasick(patterns):
        root = AhoCorasickNode()

        for pattern, offset in patterns:
            node = root
            for c in pattern:
                if c not in node.transitions:
                    node.transitions[c] = AhoCorasickNode()
                node = node.transitions[c]
            node.offsets.append(offset)

        root.suffix_link = root
        queue = deque()
```

```
        for child in root.transitions.values():
            child.suffix_link = root
            queue.append(child)
        while queue:
            current_node: AhoCorasickNode = queue.popleft()
            for c, child in current_node.transitions.items():
                queue.append(child)
                suffix_node: AhoCorasickNode = current_node.suffix_link
                while suffix_node is not root and c not in
suffix_node.transitions:
                    suffix_node = suffix_node.suffix_link
                child.suffix_link = suffix_node.transitions.get(c, root)
                child.output = child.suffix_link if
child.suffix_link.offsets else child.suffix_link.output

        return root


    def search_aho_corasick(text, root, patterns):
        occurrences = []
        current_node = root

        for i, c in enumerate(text, 1):
            while current_node is not root and c not in
current_node.transitions:
                current_node: AhoCorasickNode = current_node.suffix_link
            if c in current_node.transitions:
                current_node = current_node.transitions[c]
            else:
                current_node = root

            temp_node = current_node
            while temp_node != None and temp_node is not root:
                for offset in temp_node.offsets:
                    sub = next(p for p, o in patterns if o == offset)
                    sub_len = len(sub)
                    start_pos = i - sub_len + 1
                    occurrences.append((offset, start_pos))
                temp_node = temp_node.output

        return occurrences


    def split_pattern(pattern, wildcard):
        subpatterns = []
        current_sub = []
        for i, c in enumerate(pattern):
            if c == wildcard:
                if current_sub:
                    offset = i - len(current_sub)
                    subpatterns.append((''.join(current_sub), offset))
                    current_sub = []
            else:
                current_sub.append(c)
        if current_sub:
            offset = len(pattern) - len(current_sub)
            subpatterns.append((''.join(current_sub), offset))
```

```
        return subpatterns


    def find_wildcard_matches(text, pattern, wildcard):
        subpatterns = split_pattern(pattern, wildcard)

        if not subpatterns:
            return []

        root = build_aho_corasick(subpatterns)
        sub_occurrences = search_aho_corasick(text, root, subpatterns)

        total_matches = set()
        len_pattern = len(pattern)
        len_text = len(text)

        offset_groups = defaultdict(list)
        for offset, start in sub_occurrences:
            offset_groups[offset].append(start)

        for offset, starts in offset_groups.items():

            for start in starts:
                full_start = start - offset
                if full_start < 1 or full_start + len_pattern - 1 >
len_text:
                    continue

                match = True
                for j in range(len_pattern):
                    p_char = pattern[j]
                    if p_char != wildcard and text[full_start - 1 + j] !=
p_char:
                        match = False
                        break

                if match:
                    total_matches.add(full_start)

        return sorted(total_matches)


    def main():
        text = input().strip()
        pattern = input().strip()
        joker = input().strip()

        matches = find_wildcard_matches(text, pattern, joker)
        for pos in matches:
            print(pos)


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

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

В таблице Б.1 указаны результаты тестирования решения задач.

Таблица Б.1 - Тестовые случаи

№ п/п	Выходные и входные данные	Комментарии
1.	NTAG 3 TAGT TAG T ^Z 2 2 2 3	classic_aho_corasick.py
2.	ACGTACN 2 ACG GTA ^Z 1 1 3 2	classic_aho_corasick.py
3.	ACTANCA A$$A$ $ 1	aho_corasick_with_joker.py
4.	ACTANCA $$A $ 2 5	aho_corasick_with_joker.py
5.	ACTANCA CAT $	aho_corasick_with_joker.py