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: Поиск набора подстрок в строке
Вариант: 3

Студентка гр. 3343 _____ Лобова Е. И.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2025

Цель работы

Целью работы является изучение задачи точного поиска набора образцов в тексте и реализация обычного алгоритма Ахо-Корасик и с модификацией - поиск шаблона с масками.

Задание

Задание 1

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 $P = \{p1,..., pn\}$, $1 \leq |pi| \leq 75$.
- Все строки содержат символы из алфавита {A, C, G, T, N}.

Выход:

- Все вхождения образцов из P в T.
- 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 шаблону образцу P необходимо найти все вхождения P в текст T.

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

Символ джокер не входит в алфавит, символы которого используются в T. Каждый джокер соответствует одному символу, а не подстроке неопределённой длины. В шаблоне 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.

Sample Input:

ACTANCA

A$$A$

$

Sample Output:

1

Описание алгоритмов

Для решения задания лабораторной работы использовался алгоритм Ахо-Корасик.

Для построения бора используется класс *Node*, хранящий всю информацию о вершине бора(детей, родителя, символ, который идет к родителю, суффиксную ссылку, сжатую суффиксную ссылку, является ли терминальным и для каких паттернов, id, глубину и длину цепочки суффиксных ссылок до корня, длину сжатых суффиксных ссылок до корня).

Первым этапом алгоритма является построение бора, оно осуществляется *add_string(s: str, pattern_number: int, root: Node, num_vertices: int) -> int* - функцией, которая добавляет образец в бор. Каждая вершина в боре представляет собой префикс этой строки. Если ребро для символа уже существует, оно используется, иначе создается новая вершина.

Вторым этапом алгоритма является преобразование бора, то есть добавление суффиксных и сжатых суффиксных ссылок.

Функция *getSuffLink(v: Node, root: Node) -> Node* вычисляет и возвращает суффиксную ссылку для вершины v в боре, используя корень root. Если суффиксная ссылка еще не вычислена, вычисляет ее рекурсивно на основе родительской вершины. Также вычисляет и сохраняет длину цепочки суффиксных ссылок от вершины v до root.

Функция *getNodeUp(v: Node, root: Node) -> Node* возвращает сжатую суффиксную ссылку для вершины v. CCC указывает на ближайшую вершину-терминал в цепочке суффиксных ссылок. Также вычисляет и сохраняет длину цепочки конечных ссылок от вершины v до корня.

Первый и второй этап осуществляются в функции *build_aho_corasick(patterns: list[str]) -> Node*, которая сначала строит бор, а затем с помощью поиска в ширину преобразует его.

Последний этап происходит в функции *process_text(text: str, root: Node, patterns: list[str]) -> list[tuple[int, int]]*, которая ищет все вхождения паттернов в тексте с использованием автомата Ахо-Корасик, представленного корнем root. Для каждого символа в тексте функция переходит по автомату, начиная с корня. Если достигнута терминальная вершина, то фиксирует найденный паттерн и его индекс в тексте, добавляя результаты в *results*. Также использует сжатые

суффиксные ссылки для обнаружения других потенциальных совпадений паттернов, заканчивающихся в текущей позиции.

Для перехода используется функция *getLink(v: Node, char: str, root: Node) -> Node*, которая либо переходит к ребенку вершины, если ребро с нужным символом существует, либо к корню, если текущая вершина корень и нет нужного ребенка, либо по суффиксной ссылке в остальных случаях.

Для решения задачи точного поиска для одного образца с джокером необходимо обнаружить вхождения в текст всех безмасочных кусков образца. Для этого функция *prepare_mask_pattern(pattern: str, wild_card: str) -> tuple[list[str], list[int]]* разбивает шаблон *pattern* на список подшаблонов *subpatterns*, разделенных символом-джокером *wild_card*. Также определяет начальные индексы каждого подшаблона в исходном шаблоне и возвращает их в списке *start_index_subpattern*s. В функции *process_text_masks(text: str, root: Node, subpatterns: list[str], start_index_subpatterns: list[int], pattern_len: int)* происходит поиск вхождения подшаблонов *subpatterns* в тексте с использованием автомата Ахо-Корасик, корень которого передан в *root*. Фиксирует позиции в тексте, где заканчиваются вхождения подшаблонов. Если все подшаблоны найдены в соответствующих позициях (согласно *start_index_subpatterns*), то считает, что в данной позиции начинается полное вхождение шаблона. Печатает все позиции, где обнаружено полное вхождение шаблона.

Оценка сложности алгоритма Ахо-Корасик:

По времени:

- Построение бора выполняется за время $O(m)$, где m— суммарная длина строк.
- Добавление суффиксных ссылок выполняется за O(m*k), так как поиск в ширину проходит по каждой вершине один раз, а вершин не могло получиться больше m+1, k — размер алфавита, то есть сколько детей может быть у вершины.

- Сам же поиск в тексте выполняется за O(n+t), так как мы проходимся по каждому символу текста длины n, а в случае если узел терминальная вершина, то переходим по сжатным суффиксным ссылкам, то есть t - число вхождений паттернов в строку.
- Итоговая оценка алгоритма по времени: O(m*k + n + t)

По памяти:

- Максимальное количество вершин в боре — m+1, где m - суммарная длина строк. Для каждой вершины может храниться k детей, где k — размер алфавита. Итоговая оценка алгоритма по памяти: O(m*k)

Оценка сложности модифицированного алгоритма Ахо-Корасик с масками:

По времени:

- Разбиение паттерна на безмасочные куски — O(p), где p — длина паттерна.
- Построение бора и суффиксных ссылок — O(m*k), где m — суммарная длина подшаблонов, k — размер алфавита.
- Поиск подшаблонов в тексте — O(n+t), где n — длина текста, а t — количество вхождений подшаблонов.
- Итоговый проход для поиска вхождений всего шаблона — O(n)
- Итоговая оценка - O(m*k + n + t)

По памяти:

- O(m*k), где k — размер алфавита, а m - суммарная длина подшаблонов без масок

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии

1.	NTAG 3 TAGT TAG T	2 2 2 3	Задание 1(ex1) работает корректно для пересекающихся подстрок.
2.	babushka 2 ab ka	2 1 7 2	ex1:работает корректно для непересекающихся подстрок.
3.	abababa 2 lob ka		ex1: работает корректно на подстроках, которые не входят в текст
4.	xabvccababcax ab**c* *	2 7	Задание 2(ex2) работает корректно на обычном примере.
5.	ACTANCA A$$A $	1 4	ex2: работает корректно для шаблона имеющего одинаковые куски без масок
6.	ACTANCA A$$A$ $	1	ex2: работает корректно для шаблона, который оканчивается джокером
7	barabulka ve** *		ex2: работает корректно для шаблона, который не встречается в тексте

Выводы

В ходе работы была успешно изучена задача точного поиска набора образцов в тексте и реализованы обычный алгоритм Ахо-Корасик и с модификацией - поиск шаблона с масками.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from automat import *


def process_text(text: str, root: Node, patterns: list[str]) ->
list[tuple[int, int]]:
    cur = root
    results = []
    print("\nПоиск в тексте паттернов...")
    for i, char in enumerate(text):
        last_id = cur.id
        cur = getLink(cur, char, root)
        print(f"\n{i})Переход на символ в тексте {char} -> переход в
автомате с вершины с id = {last_id} на вершину с id = {cur.id}")
        print("Проверка на соответствие паттерну:")
        node_up = cur
        while node_up != root:
            if node_up.isLeaf:
                for pattern_index in node_up.leafPatternNumber:
                j = i - len(patterns[pattern_index]) + 1
                results.append((j, pattern_index))
                print(f"Найден паттерн №{pattern_index} -
{patterns[pattern_index]}, который начинается с индекса {j}")
            print(f"Переход по сжатой суффиксной ссылке с вершины с
id = {node_up.id} на вершину с id = {node_up.up.id}")
            node_up = node_up.up

    return results


    if __name__ == "__main__":
    text = input()
    n = int(input())
    patterns = []
    for i in range(n):
        patterns.append(input())
    root = build_aho_corasick(patterns)
    results = process_text(text, root, patterns)
    result_sorted = sorted(results, key=lambda x: (x[0], x[1]))

    print("Найденные образцы:")
    for position, pattern_index in result_sorted:
        print(f"{position + 1} {pattern_index + 1}")
```

Название файла: alg_with_masks.py

```
from automat import *


def prepare_mask_pattern(pattern: str, wild_card: str):
subpatterns = []
start_index_subpatterns = []
current_start = 0
```

```
    pattern += wild_card
    for i in range(len(pattern)):
        if pattern[i] == wild_card:
            if i > 0 and pattern[i-1] != wild_card:
                subpatterns.append(pattern[current_start:i])
                start_index_subpatterns.append(current_start + 1)
            current_start = i
        else:
            if pattern[i-1] == wild_card:
                current_start = i
    print(f"Из шаблона получили {len(subpatterns)} безмасочных куска: {'
'.join(subpatterns)}, которые встречаются в шаблоне на позициях {'
'.join(map(str,start_index_subpatterns))}")
    return subpatterns, start_index_subpatterns


    def process_text_masks(text: str, root: Node, subpatterns: list[str],
start_index_subpatterns: list[int], pattern_len: int):
    c = [0] * (len(text) + 1)
    cur = root
    print("\nПоиск в тексте шаблона...")
    for i, char in enumerate(text):
        last_id = cur.id
        cur = getLink(cur, char, root)
        print(f"\n{i}) Переход на символ в тексте {char} -> переход в
автомате с вершины с id = {last_id} на вершину с id = {cur.id}")
        print("Проверка на соответствие подшаблону:")
        node_up = cur
        while node_up != root:
            if node_up.isLeaf:
                for pattern_index in node_up.leafPatternNumber:
                j = i - len(subpatterns[pattern_index]) + 2
                print(f"Найден подшаблон № {pattern_index} -
{subpatterns[pattern_index]}, который начинается с индекса {j}")
                l_i = start_index_subpatterns[pattern_index]
                c[j - l_i + 1] += 1
            print(f"Переход по сжатой суффиксной ссылке с вершины с
id = {node_up.id} на вершину с id = {node_up.up.id}")
            node_up = node_up.up

    for i in range(len(c)):
        if c[i] == len(subpatterns) and i + pattern_len - 1 <=
len(text):
            print(f"\nШаблон начинается с индекса {i}")


    if __name__ == "__main__":
    text = input()
    pattern = input()
    wild_card = input()
    subpatterns, start_index_subpatterns = prepare_mask_pattern(pattern,
wild_card)
    root = build_aho_corasick(subpatterns)
    process_text_masks(text, root, subpatterns, start_index_subpatterns,
len(pattern))
```

Название файла: automat.py

```
class Node:
def __init__(self):
    self.son: dict = {}  # Массив сыновей
    self.go: dict = {} # Массив переходов
    self.parent: Node | None = None  # Вершина родитель
    self.suffLink: Node | None = None  # Суффиксная ссылка
    self.len_sufflink_chain: int = 0 # Длина цепочки из суффиксных
ссылок до корня
    self.up: Node | None = None  # Сжатая суффиксная ссылка
    self.len_uplink_chain: int = 0 # Длина цепочки из сжатых
суффиксных ссылок до корня
    self.charToParent: str | None = None  # Символ, ведущий к
родителю
    self.isLeaf: bool = False  # Флаг, является ли вершина
терминалом
    self.depth: int = 0 # Уровень, на котором находимя вершина
    self.leafPatternNumber: list[int] = []
    self.id: int = 0 # Идентификатор вершины

def __str__(self):
    if not self.parent:
        return "Вершина - корень."
    sons_str = ", ".join(self.son.keys())
    res_str = f"Вершина с id {self.id}:\n - Находится на уровне
{self.depth}"
    res_str += f"\n - У которой {len(self.son)} сыновей"
    if len(self.son)>0:
        res_str += f", от которых идут символы: {sons_str}."
    if self.charToParent:
        res_str += f"\n - Символ, ведущий к родителю(id родителя
= {self.parent.id}) - {self.charToParent}"
    if self.isLeaf:
        res_str += f"\n - Является терминальной для паттерна(ов)
под номером(ами) {self.leafPatternNumber}"
    return res_str


def getSuffLink(v: Node, root: Node) -> Node:
"""Функция для вычисления суффиксной ссылки"""
if v.suffLink is None:  # Если суффиксная ссылка еще не вычислена
    if v == root or v.parent == root:
        v.suffLink = root
    else:
        v.suffLink = getLink(getSuffLink(v.parent, root),
v.charToParent, root)
v.len_sufflink_chain = v.suffLink.len_sufflink_chain + 1
return v.suffLink


def getLink(v: Node, char: str, root: Node) -> Node:
"""Функция для вычисления перехода"""
if char not in v.go:
    if char in v.son:
        v.go[char] = v.son[char]
    elif v == root:
        v.go[char] = root
    else:
        v.go[char] = getLink(getSuffLink(v, root), char, root)
```

```
    return v.go[char]


    def getNodeUp(v: Node, root: Node) -> Node:
    """Функция для вычисления сжатой суффиксной ссылки"""
    if v.up is None:
        if getSuffLink(v,root).isLeaf:
            v.up = getSuffLink(v, root)
        elif getSuffLink(v, root) == root:
            v.up = root
        else:
            v.up = getNodeUp(getSuffLink(v,root), root)
    v.len_uplink_chain = v.up.len_uplink_chain + 1
    return v.up


    def add_string(s: str, pattern_number: int, root: Node, num_vertices:
int) -> int:
    """Добавляет строку в бор."""
    cur = root
    for char in s:
        if char not in cur.son:
            new_node = Node()
            new_node.parent = cur
            new_node.charToParent = char
            new_node.depth = cur.depth + 1
            num_vertices += 1
            new_node.id = num_vertices
            cur.son[char] = new_node
            print(f"Для символа {char} строим новое ребро и вершину с
id = {num_vertices}")
        else: print(f"Для символа {char} уже построено ребро")
        cur = cur.son[char]

    cur.isLeaf = True
    cur.leafPatternNumber.append(pattern_number)
    print(f"Последнюю вершину с id = {cur.id} помечаем как терминальную
для паттерна №{pattern_number}\n")
    return num_vertices

    def build_aho_corasick(patterns: list[str]) -> Node:
    """Строит автомат Ахо-Корасик для заданных образцов."""
    root = Node()
    root.suffLink = root
    root.up = root
    root.charToParent = None
    root.parent = None
    num_vertices = 0
    for i, pattern in enumerate(patterns):
        print(f"Добавление паттерна {pattern} в бор...")
        num_vertices = add_string(pattern, i, root, num_vertices)

    print(f"Количество вершин в построенном боре: {num_vertices}")
    queue = [root]
    max_len_sufflink_chain = 0
    max_len_uplink_chain = 0
    print("\nПостроение суффиксных и сжатых суффиксных ссылок для всех
вершин")
```

```
    while queue:
        v = queue.pop(0)
        for char, u in v.son.items():
            print(f"\nТЕКУЩАЯ ВЕРШИНА:\n{u}")
            print(f"\nСтроим суффиксную ссылку для текущей вершины...")
            node_sufflink = getSuffLink(u,root)
            max_len_sufflink_chain = max(max_len_sufflink_chain, u.len_sufflink_chain)
            print(f"Суффиксной ссылкой является вершина с id = {node_sufflink.id}")
            print(f"\nСтроим сжатую суффиксную ссылку для текущей вершины...")
            node_up = getNodeUp(u,root)
            max_len_uplink_chain = max(max_len_uplink_chain, u.len_uplink_chain)
            print(f"Сжатой суффиксной ссылкой является вершина с id = {node_up.id}")
            queue.append(u)
    print(f"\nПосле построения автомата получено:\nМаксимальная длина цепочки суффиксных ссылок = {max_len_sufflink_chain}\nМаксимальная длина цепочки сжатых суффиксных ссылок = {max_len_uplink_chain}")
    return root
```