

**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 3.**

Студент гр. 3343

Старков С. А.

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 ( $T, 1 \leq |T| \leq 100000$ ,  $T, 1 \leq |T| \leq 100000$  ).

Вторая - число  $n$  ( $1 \leq n \leq 3000$ ), каждая следующая из  $n$  строк содержит шаблон из набора  $P = \{p_1, \dots, p_n\}$   $1 \leq |p_i| \leq 75$ .  $P = \{p_1, \dots, p_n\}$   $1 \leq |p_i| \leq 75$ . Все строки содержат символы из алфавита  $\{A, C, G, T, N\}$ .

Выход:

Все вхождения образцов из  $P$  в  $T$ . Каждое вхождение 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 РР необходимо найти все вхождения Р в текст Т. Например, образец ab??c?ab??c? с джокером ? встречается дважды в тексте xabvccbabbabcax. Символ джокер не входит в алфавит, символы которого используются в ТТ. 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.

Sample Input:

ACTANCA

A\$\$A\$

\$

Sample Output:

1

Вариант 3: Вычислить длину самой длинной цепочки из суффиксных ссылок и самой длинной цепочки из конечных ссылок в автомате.

## **Выполнение работы**

Для выполнения задачи №1 и задачи №2 был реализован алгоритм Ахо-Корасика с отладкой возможных состояний автомата.

Задача №1: Поиск точных вхождений шаблонов

Класс *AhoCorasick*

- `__init__(self)`

*Инициализирует автомат:*

- создаёт корневой узел
- инициализирует суффиксную и терминальную ссылки корня `create_node(self)`

- `add_pattern(self, pattern, index)`

Добавляет шаблон `pattern` с номером `index` в автомат:

- по символам шаблона последовательно создаёт новые узлы, если их ещё нет
- запоминает индекс шаблона в списке `terminals` конечного узла
- `get_sufflink(self, node)`

Вычисляет (лениво, по требованию) суффиксную ссылку для узла:

- если родитель — корень, ссылка идёт в корень
- иначе рекурсивно ищет sufflink родителя и делает переход по `get_go` по символу перехода из родителя в текущий узел
- `get_go(self, node, c)`

Возвращает переход автомата из `node` по символу `c`:

- если есть прямой переход — его
- если в корне — в корень

- иначе через suflink родителя
  - переходы кешируются в go
- `get_up(self, node)`

Возвращает ближайшую вверх по suflink узел с терминалами или корень.

Используется для быстрого поиска совпадений при обходе текста.

- `search(self, text, pattern_lengths)`

Поиск вхождений всех добавленных шаблонов в тексте:

- проходит по тексту, выполняя переходы по `get_go`
- при попадании в терминальный узел или по up-ссылкам вверх, регистрирует совпадения
- возвращает список пар (позиция окончания вхождения, индекс шаблона)

Задача №2: Поиск шаблона с джокерами

Функция `wildcard(text, pattern, joker)`

Шаг 1: Разбиение шаблона

- разбивает шаблон на непустые подстроки между джокерами
- сохраняет пары (подстрока, её позиция в шаблоне)

Шаг 2: Построение автомата

- для каждой подстроки создаёт AhoCorasick
- добавляет подстроки как отдельные шаблоны с уникальными индексами
- сохраняет длину каждой подстроки для дальнейшего пересчёта позиций совпадений

Шаг 3: Поиск совпадений

- выполняет `search()` по тексту
- собирает позиции всех вхождений подстрок

— вычисляет предполагаемые позиции начала полного шаблона

с учётом позиции подстроки в шаблоне

#### Шаг 4: Подсчёт полных совпадений

— для каждой позиции текста накапливает количество совпавших подстрок

— позиции, где число совпадений равно числу подстрок — это позиции полных совпадений шаблона с джокерами

#### Шаг 5: Вывод результата

— выводит позиции начала всех полных совпадений

**Вариант 3:** Вычислить длину самой длинной цепочки из суффиксных ссылок и самой длинной цепочки из конечных ссылок в автомате.

- `max_sufflink(self)` – подсчет длины самой длинной цепочки из суффиксных ссылок с помощью `bfs` (обхода в ширину)
- `max_uplink(self)` – подсчет длины самой длинной цепочки из терминальных ссылок в автомате с помощью `bfs` (обхода в ширину)

### Оценка сложности алгоритма

Добавление шаблонов в автомат (`add_pattern`):

- Для каждого шаблона длины  $p$  проход по всем символам:  $O(p)$
- Для  $n$  подстрок суммарной длины  $S = \sum p$  суммарно:  $O(S)$

Построение суффиксных и up-ссылок (ленивое, в `get_sufflink()` и `get_up()`):

- Узлов в автомате получается не более  $S+1$
- Для каждого перехода по символу выполняется не более одного вызова `get_sufflink()` (благодаря мемоизации)
- Построение ссылок ленивое: каждый sufflink/up считается не более одного раза суммарно  $O(S)$  (переходов не больше количества ребер автомата, то есть  $S$ )

## Поиск в тексте (search)

Для длины текста N:

- Каждый символ обрабатывается за  $O(1)$
- Обход up-ссылок в сумме за все выполнение максимум пропорционален количеству найденных совпадений
- Если всего найдено Z совпадений, суммарное время поиска:  $O(N + Z)$

Итог для работы автомата:

- Добавление подстрок:  $O(S)$
- Построение ссылок:  $O(S)$
- Поиск в тексте:  $O(N + Z)$

**Итого по времени:**

$$O(S + N + Z)$$

Память:

Узлов автомата:  $O(S)$

Переходы children и кешированные go:  $O(S)$

Списки терминалов в узлах:  $O(S)$

Ссылки sufflink и up: два указателя на каждый узел –  $O(S)$

**Итого по памяти:**

$$O(S)$$

Сложность для функции wildcard(text,pattern, joker)

- $|P|$  — длина исходного шаблона
- $N$  — длина текста
- $K$  — количество подстрок после разбиения шаблона по джокерам
- $S$  — суммарная длина подстрок,  $S \leq |P|$

Шаг 1: Разбиение шаблона:

$O(|P|)$

Шаг 2: Построение автомата для сегментов

- Добавление К подстрок суммарной длины S:  $O(S)$
- Построение суффиксных и ip-ссылок( ленивое ):  $O(S)$

Шаг 3: Поиск в тексте:

- Поиск всех вхождений сегментов:  $O(N + Z')$   
 $Z'$  — общее число вхождений всех подстрок в тексте

Шаг 4: Подсчёт полных совпадений:

- Пробег по всем найденным совпадениям из возможным позициям текста:  
 $O(N)$

**Итоговое время:**

$$O(|P|+S+N+Z')=O(N+|P|+Z')$$

Память:

- Память на подстроки шаблона:  $O(|P|)$
- Память на автомат:  $O(S) \leq O(|P|)$
- Служебные структуры (Counter, списки совпадений) —  $O(N)$

**Итого по памяти:**

$$O(|P|+N)O(|P| + N)O(|P|+N)$$

### 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, а также программа поиска подстроки с джокером.

## ПРИЛОЖЕНИЕ А

```
from collections import deque, Counter

DEBUG = True # включить True для отладки


class Node:

    __slots__ = ['children', 'parent', 'charToParent', 'go', 'suffLink',
    'up', 'terminals']

    def __init__(self):
        self.children = {}
        self.parent = None
        self.charToParent = None
        self.go = {}
        self.suffLink = None
        self.up = None
        self.terminals = []


class AhoCorasick:

    def __init__(self):
        self.root = Node()
        self.root.parent = self.root
        self.root.suffLink = self.root
        self.root.up = self.root
        if DEBUG:
            print("Инициализирован корневой узел автомата.")

    # добавление паттерна через children
    def add_pattern(self, pattern, index):
        if DEBUG:
            print(f"\nДобавление шаблона {index}: '{pattern}'")
        node = self.root
        for char in pattern:
            if char not in node.children:
                new_node = Node()
                new_node.parent = node
                new_node.charToParent = char
                node.children[char] = new_node
            node = node.children[char]
        node.terminals.append(index)
```

```

# просчет суффлинки
def get_sufflink(self, node):
    if node.suffLink is not None:
        return node.suffLink

    if node.parent == self.root:
        node.suffLink = self.root
    else:
        # рекурсивная связь между get_sufflink и get_go
        parent_sufflink = self.get_sufflink(node.parent)
        node.suffLink = self.get_go(parent_sufflink, node.charToParent)

    return node.suffLink

# переход по автомату
def get_go(self, node, c):
    if c in node.go:
        return node.go[c]

    if c in node.children:
        node.go[c] = node.children[c]
    elif node == self.root:
        node.go[c] = self.root
    else:
        # рекурсивная связь между get_sufflink и get_go
        sufflink = self.get_sufflink(node)
        node.go[c] = self.get_go(sufflink, c)

    return node.go[c]

# просчет терминальных ссылок
def get_up(self, node):
    if node.up is not None:
        return node.up

    sufflink = self.get_sufflink(node)
    if sufflink == self.root or sufflink.terminals:
        node.up = sufflink
    else:
        node.up = self.get_up(sufflink)

    return node.up

```

```

# поиск
def search(self, text, pattern_lengths):
    result = []
    node = self.root
    for i, char in enumerate(text):
        node = self.get_go(node, char) # переход по автомату
        v = node
        while v != self.root:
            # если нашли терминал надо их всех добавить
            if v.terminals:
                for pattern_index in v.terminals:
                    # просчет позиции в тексте
                    position = i - pattern_lengths[pattern_index] + 2
                    result.append((position, pattern_index))
            # идем по терминальной ссылке
            v = self.get_up(v)
    return result

# нахождение самой длинной суффиксной ссылки bfs
def max_sufflink(self):
    max_length = 0
    queue = deque()
    queue.append(self.root)
    while queue:
        node = queue.popleft()
        length = 0
        v = node
        while v != self.root:
            v = self.get_sufflink(v)
            length += 1
        max_length = max(max_length, length)
        for child in node.children.values():
            queue.append(child)
    return max_length

# нахождение самой длинной терминальной ссылки bfs
def max_uplink(self):
    max_len = 0
    queue = deque()
    queue.append(self.root)
    while queue:

```

```

        node = queue.popleft()
        length = 0
        v = node
        while v != self.root:
            v = self.get_up(v)
            length+=1
        max_len = max(max_len, length)
        for child in node.children.values():
            queue.append(child)

    return max_len


def wildcard(text, pattern, joker):
    n = len(text)
    m = len(pattern)
    parts = []

    aho = AhoCorasick()
    pattern_lengths = {}

    # разбиваем шаблон на подстроки без джокера и запоминаем их позиции в
шаблоне
    current = ""
    for i, c in enumerate(pattern):
        if c == joker:
            if current:
                parts.append((current, i - len(current)))
                current = ""
            else:
                current += c
        if current:
            parts.append((current, m - len(current)))

    if not parts:
        return

    # добавляем части в автомат
    for pat_index, (subpattern, pos_in_pattern) in enumerate(parts, 1):
        pattern_lengths[pat_index] = len(subpattern)
        aho.add_pattern(subpattern, pat_index)

    # поиск всех вхождений подстрок

```

```

matches = aho.search(text, pattern_lengths)
matches.sort()

# для каждой позиции начала текста считаем количество совпавших частей
шаблона

count = Counter()
for position, pattern_index in matches:
    pos_in_pattern = parts[pattern_index - 1][1]
    start_pos = position - pos_in_pattern
    if 1 <= start_pos <= n - m + 1:
        count[start_pos] += 1

# если для позиции набралось столько совпадений, сколько частей — значит
полное совпадение

result = []
for pos in sorted(count):
    if count[pos] == len(parts):
        result.append(pos)

for p in result:
    print(p)

#
# text = input()
# pattern = input()
# joker = input()
#
# wildcard(text, pattern, joker)

text = "NTAG"
n = 3
patterns = ["TAGT", "TAG", "T"]
pattern_lengths = {}
# text = "AB"
# n = 2
# patterns = ["AB", "A"]
# pattern_lengths = {}

aho = AhoCorasick()

for idx in range(1, n + 1):
    # pattern = input().strip()
    # patterns.append(pattern)

```

```
pattern_lengths[idx] = len(patterns[idx-1])
aho.add_pattern(patterns[idx-1], idx)

matches = aho.search(text, pattern_lengths)
matches.sort()
for position, pattern_index in matches:
    print(position, pattern_index)
print(f"--max_uplink: " + str(aho.max_uplink()))
print(f"--max_sufflink_chain: " + str(aho.max_sufflink()))
```