**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 ≤ |T| ≤ 100000).

• Вторая - число n (1 ≤ n ≤ 3000), каждая следующая из n строк содержит шаблон из набора P = {p1,..., pn}, 1 ≤ |pi| ≤ 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 ≤ |T| ≤ 100000)

• Шаблон (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

## 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. ССС 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: О(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 - О(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 |
|  | NTAG  3  TAGT  TAG  T | 2 2  2 3 | Задание 1(ex1) работает корректно для пересекающихся подстрок. |
|  | babushka  2  ab  ka | 2 1  7 2 | ex1:работает корректно для непересекающихся подстрок. |
|  | abababa  2  lob  ka |  | ex1: работает корректно на подстроках, которые не входят в текст |
|  | xabvccababcax  ab\*\*c\*  \* | 2  7 | Задание 2(ex2) работает корректно на обычном примере. |
|  | ACTANCA  A$$A  $ | 1  4 | ex2: работает корректно для шаблона имеющего одинаковые куски без масок |
|  | ACTANCA  A$$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 c 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"\nCтроим 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 c id = {node\_sufflink.id}")

print(f"\nCтроим 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 c 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