# 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: АЛГОРИТМ АХО-КОРАСИКА.**

|  |  |  |
| --- | --- | --- |
| Студент гр.1304 |  | Мамин Р.А. |
| Преподаватель |  | Шевелева А.М. |

Санкт-Петербург 2023

# Цель работы.

Изучить алгоритм Ахо – Карасика. Используя его, написать две программы по вхождению подстрок в строку.

# Задание.

Задание 1.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

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

Вторая - число n (≤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 шаблоны образцу PP необходимо найти все вхождения РР в текст ТТ.

Например, образец аb??с с джокером ? встречается дважды в тексте xabvccbababcaxxabvccbababcax.

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}

# Основные теоретические положения.

Пусть дан набор строк в алфавите размера k суммарной длины n. Алгоритм Ахо-Корасик за O(nk) времени и памяти строит префиксное дерево для этого набора строк, а затем по этому дереву строит автомат, который может использоваться в различных строковых задачах — например, для нахождения всех вхождений каждой строки из данного набора в произвольный текст за линейное время.

# Выполнение работы.

Для двух задач строился бор *Trie*. Элементы которого *TrieNode* хранят дочерние элементы (поле *children*), суффиксные ссылки (*sufflink*) и список индексов шаблонов, для которых они являются терминальными.

Суффиксная ссылка для вершины v – это вершина, в которой оканчивается длиннейший суффикс строки, соответствующей вершине v.

Класс Trie. Реализация бора для работы алгоритма Ахо – Корасика. Методы Trie:

*create\_trie(patterns)* – принимает на вход список шаблонов, по которым строит бор.

*create\_links()* – создает для каждого узла бора суффиксную ссылку. Просматриваются все вершины обходом в ширину. Для каждой вершины проходим сначала по суффиксной ссылке родителя, затем по суффиксным ссылкам , пока не дойдем до корня или не найдем необходимый символ.

*aho\_korasick(text,patterns)* – функция реализации алгоритма. Внутри себя строит бор и обходит его. Возвращает массив кортежей (<индекс вхождения в тексте>, <номер в массиве шаблонов>)

Ответы к тестированию программы представлены в табл. 1.

Таблица – lab5\_1.py.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Данные: строка, массив подстрок | Вывод: <индекс вхождения в тексте>,  <номер в массиве шаблонов>) | Результат |
| 1. | ['NTAG', ['TAGT',  'TAG', 'T']],# Тест из  задания | [(2,2) , (2,3)], | Программа работает верно |
| 2. | ['ERG', ['ERG']], #  На одно вхождение | [(1,1)], | Программа работает верно |
| 3. | ['ejrhfbuhbrrr', ['hbr',  'hbrr', 'hbrrr']], #пересечение подстрок | [(8,1), (8,2), (8,3)], | Программа работает верно |

В задании 2 реализована дополнительно фнукция: *generate\_patterns(pattern, wild\_card)* – принимает на вход шаблон и

символ "джокера". Разделяет паттерн на подстроки, не содержащие джокеров и запоминает индексы начала этих подстрок в паттерне в списке start\_indices.

Ответы к тестированию программы представлены в табл. 2.

Таблица 2 – lab5\_2.py.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Данные: строка, подстрока | Вывод: строки с номерами позиций вхождений шаблона | Результат |
| 1. | ['ACTANCA',  'A$$A$', '$'],# Тест из задания | [1], | Программа работает верно |
| 2. | ['yuioyui', 'o', '\*'],# Граничный случай | [4], | Программа работает верно |
| 3. | ['abcdeffbcbfbdbfbbh', | [3, 4, 8, 12], | Программа работает верно |

# 

# Выводы.

# В рамках данной лабораторной работы был изучен алгоритм Ахо-Карасика, который является одним из наиболее эффективных алгоритмов для поиска всех заданных паттернов в тексте.

# Была написана программа на языке Python, использующая структуру данных "Дерево три" и суффиксные ссылки для быстрого поиска всех вхождений паттернов в тексте. Алгоритм Ахо-Карасика в этой программе реализуется с помощью создания дерева три для заданных паттернов и последующего создания суффиксных ссылок для узлов дерева три. Функция aho\_corasick реализует поиск всех вхождений паттернов в тексте с помощью созданного дерева три.

# Также была написана вторая программа, использующая регулярные выражения для поиска заданных подстрок в тексте. В этой программе поиск подстрок в тексте осуществляется с помощью библиотеки re, которая позволяет использовать регулярные выражения для поиска заданных подстрок в тексте.

# Обе программы позволяют быстро и эффективно решать задачу поиска подстрок в тексте. Однако, программа на основе алгоритма Ахо-Карасика имеет более высокую производительность при больших объемах текста и/или большом количестве паттернов.

# ПРИЛОЖЕНИЕ А

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

**Название файла:** lab5\_1.py

class TrieNode:

def init (self, \_sufflink=None): self.children = {} self.sufflink = \_sufflink self.patterns = []

class Trie:

def init (self, patterns): self.create\_trie(patterns) self.links()

def create\_trie(self, patterns): self.root = TrieNode()

for i, pattern in enumerate(patterns): node = self.root

for symbol in pattern:

node=node.children.setdefault(symbol, TrieNode(self.root))

node.patterns.append(i)

def links(self):

queue = [i for i in self.root.children.values()] while queue:

cur = queue.pop(0)

for alpha, node in cur.children.items(): queue.append(node)

link = cur.sufflink

while not (link is None or alpha in link.children): link = link.sufflink

node.sufflink = link.children[alpha] if link else

self.root

node.patterns += node.sufflink.patterns

def aho\_korasick(text, patterns): trie = Trie(patterns)

result = []

v = trie.root

for i in range(len(text)):

while v is not None and text[i] not in v.children: v = v.sufflink

if v is None:

v = trie.root continue

v = v.children[text[i]] for pattern in v.patterns:

result.append((i - len(patterns[pattern]) + 2, pattern + 1)) result = sorted(result)

return result

if name == " main ": text = input()

n = int(input()) patterns = []

for i in range(n): patterns.append(input())

result = aho\_korasick(text,patterns) for i in result:

print(i[0], i[1])

**Название файла:** lab5\_2.py

class TrieNode:

def init (self, \_sufflink=None): self.children = {} self.sufflink = \_sufflink self.patterns = []

class Trie:

def init (self, patterns): self.create\_trie(patterns) self.links()

def create\_trie(self, patterns): self.root = TrieNode()

for i, pattern in enumerate(patterns): node = self.root

for symbol in pattern:

node = node.children.setdefault(symbol, TrieNode(self.root))

node.patterns.append(i)

def links(self):

queue = [i for i in self.root.children.values()] while queue:

cur = queue.pop(0)

for alpha, node in cur.children.items(): queue.append(node)

link = cur.sufflink

while not (link is None or alpha in link.children): link = link.sufflink

self.root

node.sufflink = link.children[alpha] if link else node.patterns += node.sufflink.patterns

def aho\_korasick(text, patterns): trie = Trie(patterns)

result = [] v = trie.root

for i in range(len(text)):

while v is not None and text[i] not in v.children: v = v.sufflink

if v is None:

v = trie.root continue

v = v.children[text[i]] for pattern in v.patterns:

result.append((i - len(patterns[pattern]) + 1, pattern)) result = sorted(result)

return result

def generate\_patterns(pattern, wild\_card):

parts = list(filter(bool, pattern.split(wild\_card))) start\_indices = []

flag = 1

for i, c in enumerate(pattern): if c == wild\_card:

flag = 1 continue

if flag:

start\_indices.append(i) flag = 0

return parts, start\_indices

def solve(text, pattern, wild\_card):

patterns, starts = generate\_patterns(pattern, wild\_card) indices = aho\_korasick(text, patterns)

c = [0] \* len(text) for i, p\_i in indices:

index = i - starts[p\_i] if 0 <= index < len(c):

c[index] += 1

res = []

for i in range(len(c) - len(pattern) + 1): if c[i] == len(patterns):

res.append(i + 1) return res

if name == " main ": txt = input()

p = input() wc = input()

ans = solve(txt, p, wc) print(\*ans, sep="\n")