**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-Корасик

**Вариант – 2**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Березовский М.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2025

## Цель работы

Изучить и реализовать алгоритм Ахо-Корасика — эффективный метод поиска подстрок (образцов) в тексте

## Задание

Подсчитать количество вершин в автомате; вывести список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска.

## Выполнение работы

**Описание алгоритма для решения задачи**

**1. Построение бора (префиксного дерева)**

На этом этапе из всех шаблонов создаётся дерево, где каждая вершина соответствует некоторому префиксу шаблонов. Переходы по символам создаются по мере необходимости. В каждой конечной вершине сохраняется номер шаблона, который заканчивается в этой вершине. Это так называемое поле output.

**2. Построение суффиксных (fail-) ссылок**

После построения бора происходит построение ссылок, по которым можно перейти в случае, если текущий символ текста не соответствует ни одному из переходов. Эти ссылки указывают на наиболее длинный возможный суффикс, совпадающий с началом какого-либо шаблона. Построение происходит с помощью обхода в ширину (BFS), начиная с детей корня.

**3. Объединение выходных списков**

При построении fail-ссылок выходные списки (списки номеров шаблонов) из вершин объединяются. Таким образом, если мы попадаем в вершину через fail-ссылку, мы всё равно получаем корректный результат по шаблонам.

**4. Поиск в тексте**

Текст обрабатывается последовательно по символам. При наличии перехода по текущему символу — происходит переход. Если перехода нет — переходим по fail-ссылке, пока не найдём подходящий переход или не вернёмся в корень. При попадании в вершину с непустым output— фиксируем вхождение шаблона.

**5. Анализ пересечений вхождений**

После завершения поиска сравниваются все пары найденных вхождений. Если два шаблона пересекаются по позициям в тексте, их номера добавляются в результат.

**Оценка сложности алгоритма во времени и памяти в нотации O.**

Пусть T - количество шаблонов, |P| - суммарная длина всех шаблонов, n - длина текста, k -— количество вхождений.

Построение бора: O(|P|) - каждое добавление символа в бор создаёт не более одной новой вершины.

Построение суффиксных ссылок: O(|P|) - каждая вершина и каждое ребро обрабатываются один раз.

Поиск в тексте: O(n) - каждый символ обрабатывается с возможными fail-переходами, но суммарно не более 2n операций.

Обработка пересечений вхождений: O(k²) в худшем случае - перебираются все пары совпадений.

Итоговая оценка:

O(|P| + n + k²)

**Тестирование. Демонстрация граничных случаев алгоритма.**

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | xabcd  2  abc  cbd | 7  2 1 | Верный вывод |
|  | abcabc  3  abc  bca  cab | 10  1  2  3  1 1  2 2  3 3  4 1 | Верный вывод |
|  | banana  3  ana  na  nana | 8  1  2  3  2 1  3 2  3 3  4 1  5 2 | Верный вывод |

**Выводы**

Реализованный алгоритм Ахо-Корасика эффективно строит конечный автомат для поиска множества шаблонов в тексте за время, линейное от суммы длины текста и всех шаблонов. В автомате создаётся бор (Trie), к вершинам добавляются суффиксные ссылки для быстрого перехода при несовпадениях, что значительно ускоряет поиск. Таким образом, алгоритм демонстрирует высокую производительность и точность для поиска нескольких образцов одновременно, что делает его полезным в задачах анализа текстов и обработки строк.

Разработанный программный код см. в приложении А

# Приложение А Исходный код программы

Название файла: try.py

class Node:

def \_\_init\_\_(self):

self.children = {}

self.fail = None

self.output = []

node\_count = 0

def build\_trie(patterns):

global node\_count

root = Node()

node\_count = 1

for pattern in patterns:

node = root

for char in pattern:

if char not in node.children:

node.children[char] = Node()

node\_count += 1

node = node.children[char]

node.output.append(patterns.index(pattern))

return root

def build\_links(root):

queue = []

for child in root.children.values():

child.fail = root

queue.append(child)

while queue:

current = queue.pop(0)

for char, child in current.children.items():

fail\_node = current.fail

while fail\_node and char not in fail\_node.children:

fail\_node = fail\_node.fail

child.fail = fail\_node.children[char] if fail\_node and char in fail\_node.children else root

child.output += child.fail.output

queue.append(child)

def aho\_corasick(text, patterns):

root = build\_trie(patterns)

build\_links(root)

node = root

result = []

for i, char in enumerate(text):

while node and char not in node.children:

node = node.fail

node = node.children[char] if node and char in node.children else root

for pattern\_id in node.output:

start = i - len(patterns[pattern\_id]) + 1

end = i

result.append((start, end, pattern\_id + 1))

return result

text = input().strip()

n = int(input())

patterns = [input().strip() for \_ in range(n)]

matches = aho\_corasick(text, patterns)

print(node\_count)

intersecting = set()

matches.sort()

for i in range(len(matches)):

for j in range(i + 1, len(matches)):

s1, e1, id1 = matches[i]

s2, e2, id2 = matches[j]

if s2 > e1:

break

if e2 >= s1:

intersecting.add(id1)

intersecting.add(id2)

for id in sorted(intersecting):

print(id)

for s, e, id in sorted(matches):

print(s + 1, id)