**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе 6**

**по дисциплине «Построение и анализ алгоритмов»**

Тема: **Ахо-Корасик**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3384 |  | Козьмин Н.В. |
| Преподаватель |  | Шевелева А.М. |

Санкт-Петербург

2025

## Цель работы.

Разработать программы, которые обрабатывают строки с помощью алгоритма Ахо-Корасика.

## Задание.

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 представить в виде двух чисел - i p, где i - позиция в тексте (нумерация с 1), с которой начинается вхождение образца с номером p (нумерация с 1). Вывод отсортировать по возрастанию: сначала по позиции, затем по номеру шаблона.

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 P необходимо найти все вхождения P в текст T.

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

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}

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.

## Описание работы.

Был выбран Python, так как не требуется высокая скорость выполнения и писать на этом ЯП быстрее и проще.

Первым делом был сделан узел для составления бора в алгоритме Ахо-Корасика. Он содержит transitions для хранения ключей-символов и значений-других узлов. Здесь и в дальнейшем использованы аннотации типов для большего удобства чтения и подсветки в редакторе кода; suffix\_link содержит следующий по размеру суффикс, имеющийся в боре, output – терминальный узел, чтобы оптимально найти следующий нужный элемент, не проходя по всем суффиксным узлам, pattern\_indices, замененный потом на offsets – индексы шаблонов, заканчивающихся в узле.

Затем был реализован алгоритм для построения бора в функции build\_aho\_corasick. В нем сначала заполняется “скелет” бора, а потом с помощью прохода в ширину, который также можно было заменить на ленивый проход в длину, добавляются суффиксальные ссылки вместе с выходными (аналог алгоритма KMP для бора).

Далее был сделан сам поиск Ахо-Корасика. Используя созданный автомат состояний из бора, при нахождении терминального узла добавляются вхождения в список. После этого была сделана проверка.

Затем pattern\_indices были заменены, так как для решения второй задачи, нужны смещения подшаблонов в основном шаблоне вместо индексов образцов. В качестве шаблонов, было использовано разделение основного шаблона по символу джокера в функции split\_pattern. Там же к паттернам добавляются отступы, которые после используются вместо индексов в build\_aho\_corasick.

В самом поиске были сделаны следующие изменения: теперь результатом search\_aho\_corasick стал список из элементов формата: отступ в шаблоне, позиция начала подшаблона в тексте. А для формирования конечного ответа была сделана функция find\_wildcard\_matches. В ней организуются найденные позиции по смещениям подшаблонов и для каждой группы вхождений вычисляется возможное начало полного шаблона с постепенной проверкой каждого символа.

## Описание функций.

build\_aho\_corasick, используя шаблоны, строит по ним бор с нужными ссылками, возвращая корневой узел.

search\_aho\_corasick возвращает список отступов в шаблоне с позициями начал подшаблонов в тексте, используя исходный текст, корневой узел бора и шаблоны со смещениями, чтобы по перовому находить второе.

split\_pattern делит по джокеру (разделителю) основной шаблон на подшаблоны, добавляя отступы.

find\_wildcard\_matches, используя предыдущие функции, по тексту, шаблону и разделителю возвращает отсортированный список всех позиций, где найден полный шаблон.

В main`е вызываются созданные функции.

## Выводы.

В ходе выполнения работы основной целью было разработать обработчики строк с помощью алгоритма Ахо-Корасика. Реализованные программы успешно решают поставленные задачи.

Сначала был сделан код для первой задачи, а потом логично модернизирован в код для второй. В main каждого файла был вынесен результат с решением соответствующей задачи. Разработанный программный код см. в приложении А.

Код получился лаконичным и работающим эффективно в естественных ограничениях языка. В работе из предыдущих тем были использованы граф (для бора) и обход в ширину.

Решения поставленных задач также были протестированы. Результаты тестирования см. в приложении Б.

# Приложение А Исходный код программы

Название файла: classic\_aho\_corasick.py

import sys

from collections import deque

class AhoCorasickNode:

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

self.transitions: dict[str, AhoCorasickNode] = {}

self.suffix\_link = None

self.output = None

self.pattern\_indices = []

def build\_aho\_corasick(patterns):

root = AhoCorasickNode()

for i, pattern in enumerate(patterns):

node = root

for c in pattern:

if c not in node.transitions:

node.transitions[c] = AhoCorasickNode()

node = node.transitions[c]

node.pattern\_indices.append(i + 1)

root.suffix\_link = root

queue = deque()

for child in root.transitions.values():

child.suffix\_link = root

queue.append(child)

while queue:

current\_node: AhoCorasickNode = queue.popleft()

for c, child in current\_node.transitions.items():

queue.append(child)

suffix\_node: AhoCorasickNode = current\_node.suffix\_link

while suffix\_node is not root and c not in suffix\_node.transitions:

suffix\_node = suffix\_node.suffix\_link

child.suffix\_link = suffix\_node.transitions.get(c, root)

child.output = child.suffix\_link if child.suffix\_link.pattern\_indices else child.suffix\_link.output

return root

def search\_aho\_corasick(text, root, patterns):

occurrences = []

current\_node = root

for i, c in enumerate(text, 1):

while current\_node is not root and c not in current\_node.transitions:

current\_node: AhoCorasickNode = current\_node.suffix\_link

if c in current\_node.transitions:

current\_node = current\_node.transitions[c]

else:

current\_node = root

temp\_node = current\_node

while temp\_node != None and temp\_node is not root:

for pattern\_idx in temp\_node.pattern\_indices:

pattern\_length = len(patterns[pattern\_idx - 1])

start\_pos = i - pattern\_length + 1

occurrences.append((start\_pos, pattern\_idx))

temp\_node = temp\_node.output

return occurrences

def main():

input\_lines = sys.stdin.read().splitlines()

text = input\_lines[0].strip()

n\_patterns = int(input\_lines[1].strip())

patterns = [line.strip() for line in input\_lines[2:2 + n\_patterns]]

root = build\_aho\_corasick(patterns)

occurrences = search\_aho\_corasick(text, root, patterns)

occurrences.sort()

for pos, pattern\_idx in occurrences:

print(pos, pattern\_idx)

if \_\_name\_\_ == "\_\_main\_\_":

main()

Название файла: aho\_corasick\_with\_joker.py

from collections import deque, defaultdict

class AhoCorasickNode:

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

self.transitions: dict[str, AhoCorasickNode] = {}

self.suffix\_link = None

self.output = None

self.offsets = []

def build\_aho\_corasick(patterns):

root = AhoCorasickNode()

for pattern, offset in patterns:

node = root

for c in pattern:

if c not in node.transitions:

node.transitions[c] = AhoCorasickNode()

node = node.transitions[c]

node.offsets.append(offset)

root.suffix\_link = root

queue = deque()

for child in root.transitions.values():

child.suffix\_link = root

queue.append(child)

while queue:

current\_node: AhoCorasickNode = queue.popleft()

for c, child in current\_node.transitions.items():

queue.append(child)

suffix\_node: AhoCorasickNode = current\_node.suffix\_link

while suffix\_node is not root and c not in suffix\_node.transitions:

suffix\_node = suffix\_node.suffix\_link

child.suffix\_link = suffix\_node.transitions.get(c, root)

child.output = child.suffix\_link if child.suffix\_link.offsets else child.suffix\_link.output

return root

def search\_aho\_corasick(text, root, patterns):

occurrences = []

current\_node = root

for i, c in enumerate(text, 1):

while current\_node is not root and c not in current\_node.transitions:

current\_node: AhoCorasickNode = current\_node.suffix\_link

if c in current\_node.transitions:

current\_node = current\_node.transitions[c]

else:

current\_node = root

temp\_node = current\_node

while temp\_node != None and temp\_node is not root:

for offset in temp\_node.offsets:

sub = next(p for p, o in patterns if o == offset)

sub\_len = len(sub)

start\_pos = i - sub\_len + 1

occurrences.append((offset, start\_pos))

temp\_node = temp\_node.output

return occurrences

def split\_pattern(pattern, wildcard):

subpatterns = []

current\_sub = []

for i, c in enumerate(pattern):

if c == wildcard:

if current\_sub:

offset = i - len(current\_sub)

subpatterns.append((''.join(current\_sub), offset))

current\_sub = []

else:

current\_sub.append(c)

if current\_sub:

offset = len(pattern) - len(current\_sub)

subpatterns.append((''.join(current\_sub), offset))

return subpatterns

def find\_wildcard\_matches(text, pattern, wildcard):

subpatterns = split\_pattern(pattern, wildcard)

if not subpatterns:

return []

root = build\_aho\_corasick(subpatterns)

sub\_occurrences = search\_aho\_corasick(text, root, subpatterns)

total\_matches = set()

len\_pattern = len(pattern)

len\_text = len(text)

offset\_groups = defaultdict(list)

for offset, start in sub\_occurrences:

offset\_groups[offset].append(start)

for offset, starts in offset\_groups.items():

for start in starts:

full\_start = start - offset

if full\_start < 1 or full\_start + len\_pattern - 1 > len\_text:

continue

match = True

for j in range(len\_pattern):

p\_char = pattern[j]

if p\_char != wildcard and text[full\_start - 1 + j] != p\_char:

match = False

break

if match:

total\_matches.add(full\_start)

return sorted(total\_matches)

def main():

text = input().strip()

pattern = input().strip()

joker = input().strip()

matches = find\_wildcard\_matches(text, pattern, joker)

for pos in matches:

print(pos)

if \_\_name\_\_ == "\_\_main\_\_":

main()

# Приложение Б Тестирование

В таблице Б.1 указаны результаты тестирования решения задач.

Таблица Б.1 - Тестовые случаи

|  |  |  |
| --- | --- | --- |
| № п/п | Выходные и входные данные | Комментарии |
|  | NTAG  3  TAGT  TAG  T  ^Z  2 2  2 3 | classic\_aho\_corasick.py |
|  | ACGTACN  2  ACG  GTA  ^Z  1 1  3 2 | classic\_aho\_corasick.py |
|  | ACTANCA  A$$A$  $  1 | aho\_corasick\_with\_joker.py |
|  | ACTANCA  $$A  $  2  5 | aho\_corasick\_with\_joker.py |
|  | ACTANCA  CAT  $ | aho\_corasick\_with\_joker.py |