**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**

| Студент гр. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
| Преподаватель |  | Фирсов М. А. |

Санкт-Петербург

2023

**Цель работы.**

Ознакомиться с алгоритмом Ахо-Корасик поиска набора образцов в

строке, применить его в решении поставленных задач.

**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

шаблоны образцу P необходимо найти все вхождения Р в текст Т.

Например, образец аb??с? 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.

**Sample Input:**

ACTANCA

A$$A$

$

**Sample Output:**

1

Вариант 6. Подготовка к распараллеливанию (1-ое задание на Stepik): работа по поиску разделяется на k равных частей, пригодных для обработки k потоками (при этом максимальная длина образца гораздо меньше длины строки поиска).

**Основные теоретические положения.**

Алгоритм Ахо-Корасик (АК) - классическое решение задачи точного

сопоставления множеств. АК основан на структуре данных "дерево ключевых слов".

Поиск строки S в бору: начинаем в корне, идем по ребрам, отмеченным символами S, пока возможно.

Если с последним символом S мы приходим в вершину с сохраненным идентификатором, то S - слово из словаря.

Если в какой-то момент ребра, отмеченного нужным символом, не находится, то строки S в словаре нет.

Ясно, что это занимает O (|S|) времени. Таким образом, бор - это эффективный способ хранить словарь и искать в нем слова.

Теперь перейдем от бора к автомату, чтобы добиться поиска шаблонов в тексте за линейное время.

Действия автомата определяются тремя функциями, определенными для всех состояний:

Функция goto g(s, a) указывает, в какое состояние переходить из данного состояния s при просмотре символа a.

Функция неудачи f(s) указывает, в какое состояние переходить при

просмотре неподходящего символа.

Выходная функция out(s) выдает множество шаблонов, которые

обнаруживаются при переходе в состояние s.

Теорема. Поиск шаблонов в тексте T [1…m] с помощью автомата АК

занимает O(m + z) времени, где z - количество появлений шаблонов.

**Выполнение работы.**

Для решения поставленных задач определения вхождения подстроки в строку был реализован алгоритм Ахо-Корасик.

Для реализации алгоритма был создан вспомогательный класс,

описывающий узлы дерева ключевых слов – бора.

*Class AhoNode.* Структура описывает узлы для построения дерева ключевых слов. Имеет поля *leaves* (указывает, в какие узлы переходить), *plenty\_patterns* (множество шаблонов, обнаруживающихся при переходе по символу), *suf\_link* (узел для перехода при просмотре неподходящего символа).

Основные функции:

1. *create\_tree(patterns).* Функция создает бор – дерево паттернов. На вход подается набор шаблонов, по которым будет строиться дерево. С помощью вспомогательного класса, описанного ранее, создается узел для бора. Соответствующие поля класса обновляются для каждого узла: в словарь *leaves* добавляются символы из шаблонов, по которым будут храниться другие узлы для перехода, в список *plenty\_patterns* добавляется номер обрабатываемого на данном этапе шаблона. Функция возвращает корень бора.
2. *create\_aho\_statemachine(patterns).* Функция, создающая автомат Ахо-Корасик, вычисляет суффиксные ссылки для перехода при просмотре неподходящих символов. Переменная *suf\_link,* узел для перехода при просмотре неподходящего символа и список *plenty\_patterns* с множеством шаблонов вычисляются для всех вершин в порядке обхода. Данная функция находит указатель на строку, которая есть в боре и при этом является самым длинным суффиксом для текущей обрабатываемой подстроки. Функция возвращает корень бора.
3. *getIntervals(text, patterns, thread\_amount).* Функция, которая разбивает исходный текст на заданное число подстрок, пригодных для многопоточного решения задачи. В ней вычисляется размер одной подстроки, который будет равен размеру исходного текста, деленного на количество требуемых потоков, + размер самого длинного шаблона, чтобы подстроки пересекались и мы не потеряли возможное решение при разбиении. Функция принимает на вход исходный текст, шаблоны и количество потоков. На выходе функция возвращает список из интервалов, на которые нужно разбивать текст.
4. *find\_all\_substrings(str, root, patterns).* Функция находит все вхождения шаблонов, хранящихся в боре в текст. Для этого мы посимвольно сопоставляем строку с переходами между состояниями в боре. Если происходит совпадение полученного результата с одним из шаблонов, мы запоминаем индекс начала его местонахождения в строке и его порядковый номер. Если очередной символ не является одним из возможных путей перехода на текущем узле бора, мы переходим по суффиксной ссылке до тех пор, пока не найдем подходящий переход или не вернемся в корень бора. Функция принимает текст, корень бора и набор шаблонов, на выходе мы получаем список, содержащий шаблоны с их индексами вхождения в текст.
5. *create\_patterns(pattern, joker\_symbol).* Функция создающая наборы шаблонов для задания с символом-джокером, принимает на вход единый шаблон и символ, который является джокером. Функция возвращает набор шаблонов и соответствующие им стартовые индексы в едином шаблоне.
6. *calculate\_substrings(text, start\_indices\_in\_pattern, substrings\_in\_text).* Функция, считающая, количество безмасочных подстрок, встретившихся в тексте. Принимает на вход текст, стартовые индексы шаблонов в едином шаблоне, список, хранящий шаблоны и их индексы вхождения в тексте. Возвращает список, хранящий по индексу количество встретившихся подстрок.
7. *find\_all\_substrings\_aho\_second(text, root, patterns).* Функция, находящая индексы вхождения шаблона в строке. Принимает на вход строку, корень бора и набор шаблонов, полученный с помощью *create\_patterns(pattern, joker\_symbol)*. Результат работы функции – список вхождений шаблона в строку.

Сложность написанного алгоритма можно оценить как O(M+N+t), где М – размер бора, N – длина исходной строки, t – количество всех возможных вхождений всех сток-образцов в текст,

Разработанный код см. в приложении А.

**Тестирование.**

Пример вывода дополнительной информации для заданий 1-2, представлен на рисунках 1-2 соответственно.

Тестирование для задания 1:

| **Входные данные** | **Выходные данные** | **Комментарий** |
| --- | --- | --- |
| oogaboogaboo  3  oo  gab  boo  3 | 1 1  3 2  5 3  6 1  8 2  10 3  11 1 | Верно |
| abcdefghijabcdefjisfidjflsd  3  abc  j  cde  3 | 1 1  3 3  10 2  10 2  11 1  13 3  17 2  23 2 | Верно |
| alohahiolahohellohi  2  hi  l  3 | 2 2  6 1  9 2  15 2  16 2  18 1 | Верно |

Тестирование для задания 2:

| **Входные данные** | **Выходные данные** | **Комментарий** |
| --- | --- | --- |
| oogaboogaboo  oo$$  $  2 | 1  6 | Верно |
| weithiorghidfhghrfairjgioerjgijdiofjsiofa  $$fa  $  3 | 19  44 | Верно |

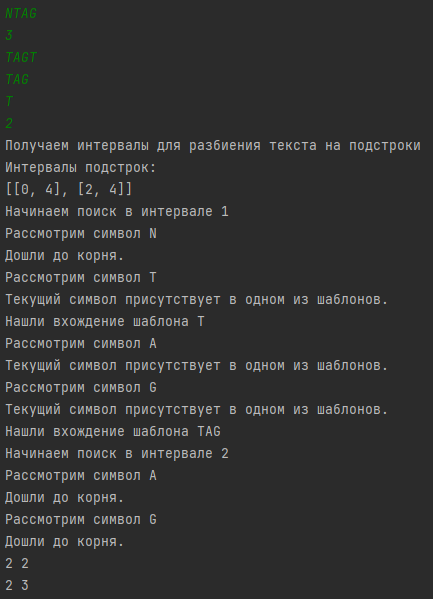


Рисунок 1 – Дополнительный вывод для задания 1

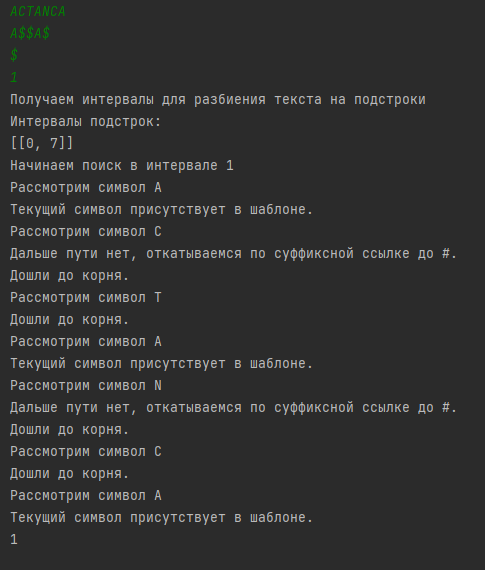


Рисунок 2 – Дополнительный вывод для задания 2

**Вывод.**

В ходе лабораторной работы был изучен алгоритм Ахо-Корасик.

Разработан программный код, позволяющий решить следующие задачи с помощью данного алгоритма: точный поиск набора образцов в тексте и поиск для одного образца с джокером. На языке программирования Python реализованы функции, представляющие собой решение поставленных задач.

Для работы алгоритма Ахо-Корасик изучена структура данных «дерево ключевых слов» или бор. Реализован класс, описывающий узел такого дерева, и функция, строящее такое дерево. Для нахождения шаблонов в тексте реализован автомат Ахо-Корасик, состояниями которого являются узлы бора.

Для решения задачи поиска одного образца с джокером была осуществлена предобработка образца перед началом алгоритма. Образец

делился на подстроки по разделителю, которым являлся символ джокера, а

также для получившихся подстрок сохранялись их стартовые позиции в образце. Это позволяет решать данную задачу по такому же принципу, как и первую.

Разработанный программный код для решения поставленных задач успешно прошел тестирование на онлайн платформе Stepik

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММ**

Название файла: aho-korasick.py

# Класс, описывающий узлы для построения дерева ключевых слов(узлы бора)

# Поля класса: leaves указывает, в какие узлы переходить

# plenty\_patterns множество шаблонов, обнаруживающиеся при переходе по символу

# suf\_link узел для перехода при просмотре неподходящего символа

class AhoNode:

# Метод, инициализирующий поля класса

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

self.leaves = {}

self.plenty\_patterns = []

self.suf\_link = None

self.way = way

# Функция, создающая бор - дерево паттернов

# Принимает на вход набор шаблонов

# Возвращает корень дерева паттернов - бора

def create\_tree(patterns: list) -> AhoNode:

root = AhoNode("#")

for index, string in enumerate(patterns):

node = root

for symbol in string:

node = node.leaves.setdefault(symbol, AhoNode(symbol))

node.plenty\_patterns += [index]

return root

# Функция, реализующая автомат Ахо-Корасик, а именно

# создается дерево паттернов и инициализируются

# суффиксные ссылки

# Принимает на вход набор шаблонов

# Возвращает корень дерева паттернов - бора

def create\_aho\_statemachine(patterns: list) -> AhoNode:

root = create\_tree(patterns)

nodes\_queue = []

for node in root.leaves.values():

nodes\_queue.append(node)

node.suf\_link = root

while len(nodes\_queue):

current\_node = nodes\_queue.pop(0)

for symbol, leaf in current\_node.leaves.items():

nodes\_queue.append(leaf)

fail\_node = current\_node.suf\_link

while fail\_node is not None and symbol not in fail\_node.leaves:

fail\_node = fail\_node.suf\_link

leaf.suf\_link = fail\_node.leaves[symbol] if fail\_node else root

leaf.plenty\_patterns += leaf.suf\_link.plenty\_patterns

return root

def getIntervals(text, patterns, thread\_amount):

maxlen = len(max(patterns, key=len))

print("Получаем интервалы для разбиения текста на подстроки")

partSize = int(len(text) / thread\_amount)

def interval(i, partSize, maxlen):

return [i \* partSize, i \* partSize + partSize + maxlen - 1] if len(text) >= i \* partSize + partSize + maxlen - 1 \

else [i \* partSize, len(text)]

intervals = [interval(i, partSize, maxlen) for i in range(thread\_amount - 1) if

len(text) > i \* partSize + maxlen - 1]

intervals.append([(thread\_amount - 1) \* partSize, len(text)])

print("Интервалы подстрок:")

print(intervals)

return intervals, partSize

# Функция, находящая все возможные вхождения шаблонов в текст(1 задание)

# Принимает на вход текст, корень бора и набор шаблонов

# Возвращает список, содержащий шаблоны с их индексами вхождения в текст

def find\_all\_substrings(text: str, root: AhoNode, patterns: list, partSize, numb) -> list:

current\_node = root

substrings\_in\_text = []

for i, sym in enumerate(text):

print(f"Рассмотрим символ {sym}")

while current\_node is not None and sym not in current\_node.leaves:

current\_node = current\_node.suf\_link

if current\_node is not None:

print(f"Дальше пути нет, откатываемся по суффиксной ссылке до {current\_node.way}.")

if current\_node is None:

print("Дошли до корня.")

current\_node = root

continue

current\_node = current\_node.leaves[text[i]]

print("Текущий символ присутствует в одном из шаблонов.")

for pattern in current\_node.plenty\_patterns:

print(f"Нашли вхождение шаблона {patterns[pattern]}")

substrings\_in\_text += [[i - len(patterns[pattern]) + 2 + partSize \* numb, pattern + 1]]

return substrings\_in\_text

# Функция, создающая наборы шаблонов для задания с символом джокером

# Принимает на вход единый шаблон и символ джокер

# Возвращает набор шаблонов и соответствующие им стартовые индексы в едином шаблоне

def create\_patterns(pattern, joker\_symbol):

patterns = list(filter(None, pattern.split(joker\_symbol)))

start\_indices\_in\_pattern = []

start\_index = 0

for string in patterns:

current\_section = pattern[start\_index:]

current\_index = current\_section.index(string)

start\_indices\_in\_pattern += [current\_index + (len(pattern) - len(current\_section))]

start\_index = current\_index + len(string) + (len(pattern) - len(current\_section))

return patterns, start\_indices\_in\_pattern

# Функция, считающая количество безмасочных подстрок, встретившихся в тексте

# Принимает на вход текст, стартовые индексы шаблонов в едином шаблоне,

# список, хранящий шаблоны и их индексы вхождения в тексте

# Возвращает список, хранящий по индексу количество встретившихся подстрок

def calculate\_substrings(text, start\_indices\_in\_pattern, substrings\_in\_text):

count\_substring\_without\_mask = [0] \* len(text)

for index\_in\_text, pattern in substrings\_in\_text:

current\_index = index\_in\_text - start\_indices\_in\_pattern[pattern]

if current\_index >= 0 and current\_index < len(count\_substring\_without\_mask):

count\_substring\_without\_mask[current\_index] += 1

# print(count\_substring\_without\_mask)

return count\_substring\_without\_mask

# Функция, выводящая ответ на первое задание

# Принимает на вход список, содержащий

# шаблоны с их индексами вхождения в текст

def print\_first\_answer(patterns\_in\_text):

patterns\_in\_text.sort(key=lambda value: (value[0], value[1]))

for i in range(len(patterns\_in\_text)):

print(\*patterns\_in\_text[i], sep=' ')

# Функция, выводящая ответ на второе задание

# Принимает на вход список, хранящий по индексу

# количество встретившихся подстрок

def print\_second\_answer(count\_substrings\_by\_index, pattern, all\_patterns):

for i in range(len(count\_substrings\_by\_index) - len(pattern) + 1):

if count\_substrings\_by\_index[i] == len(all\_patterns):

print(i + 1)

# Функция, запускающая решение первого задания

def solve\_first\_task():

text = input()

count\_pattern = int(input())

patterns = []

for \_ in range(count\_pattern):

patterns += [input()]

thread\_amount = int(input())

substrings\_list, partSize = getIntervals(text, patterns, thread\_amount)

result = []

root\_pattern\_tree = create\_aho\_statemachine(patterns)

i = 0

for elem in substrings\_list:

print(f"Начинаем поиск в интервале {i+1}")

patterns\_in\_text = sorted(find\_all\_substrings(text[elem[0]:elem[1]], root\_pattern\_tree, patterns, partSize, i))

result += patterns\_in\_text

i += 1

print\_first\_answer(sorted(result))

def find\_all\_substrings\_aho\_second(text, root, patterns):

current\_node = root

substrings\_in\_text = []

for i in range(len(text)):

print(f"Рассмотрим символ {text[i]}")

while current\_node is not None and text[i] not in current\_node.leaves:

current\_node = current\_node.suf\_link

if current\_node is not None:

print(f"Дальше пути нет, откатываемся по суффиксной ссылке до {current\_node.way}.")

if current\_node is None:

print("Дошли до корня.")

current\_node = root

continue

current\_node = current\_node.leaves[text[i]]

print("Текущий символ присутствует в шаблоне.")

for pattern in current\_node.plenty\_patterns:

substrings\_in\_text += [[i - len(patterns[pattern]) + 1, pattern]]

return substrings\_in\_text

# Функция, запускающая решение второго задания

def solve\_second\_task():

text = input()

pattern = input()

joker\_symbol = input()

thread\_amount = int(input())

all\_patterns, start\_indices\_in\_pattern = create\_patterns(pattern, joker\_symbol)

pattern\_len = [pattern]

substrings\_list, partSize = getIntervals(text, pattern\_len, thread\_amount)

root\_pattern\_tree = create\_aho\_statemachine(all\_patterns)

result = []

count = []

i = 0

for elem in substrings\_list:

print(f"Начинаем поиск в интервале {i + 1}")

substrings\_in\_text = sorted(find\_all\_substrings\_aho\_second(text[elem[0]:elem[1]], root\_pattern\_tree, all\_patterns))

count\_substrings\_without\_mask = calculate\_substrings(text[elem[0]:elem[1]], start\_indices\_in\_pattern, substrings\_in\_text)

result += substrings\_in\_text

count += count\_substrings\_without\_mask

i += 1

print\_second\_answer(count, pattern, all\_patterns)

# Условие для запуска программы

#solve\_first\_task()

solve\_second\_task()