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. 9383 _____ Соседков К.С.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Ахо-Корасик.

Задание 1.

Разработайте программу, решающую задачу точного поиска набора образцов.

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 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}

Задание 3(Вариант 5).

Вычислить максимальное количество дуг, исходящих из одной вершины в боре; вы́резать из строки поиска все найденные образцы и вывести остаток строки поиска.

Описание работы алгоритма.

Входные данные: текст(text) и массив подстрок(patterns).

1) По массиву patterns строится бор.

2) Выполняется цикл по тексту

2.1) Поиск текущего символа в боре

2.2) Если символ присутствует в боре и этот символ является окончанием какой-либо подстроки – подстрока найдена.

Для второго задания алгоритм следующий:

1) Каждая подстрока массива patterns разбивается по символу джокера.

Пример: подстрока ‘aba$$asd’ с джокером ‘$’ будет разбита на две подстроки ‘aba’ и ‘asd’.

2) С помощью алгоритма из Задания 1 выполняется поиск подстрок полученных в пункте 1.

3) Строится массив C. C[i] = количество встретившихся в тексте безмасочных подстрок шаблона, который начинается в тексте на позиции i.

4) Если C[i] равно длине подстроки – подстрока найдена.

Анализ алгоритма.

Вычислительная сложность алгоритма в первом и втором задании равна O(a+h+k), где a — суммарная длина подстрок, h — длина текста, k — общая длина всех совпадений.

Описание основных функций и переменных.

Переменные:

text— исходный текст

patterns — массив подстрок

n — количество подстрок.

Tree — бор.

Классы:

Node — описание узла дерева.

У класса Node есть следующие поля:

children — словарь дочерних узлов

suffix_link — суффиксная ссылка на узел

words — массив слов на которых заканчивается узел.

Функции:

create_tree(patterns) — по подстрокам создает дерево

create_links(tree) — добавляет в дерево tree суффиксные ссылки

Aho_Corasick(string, patterns) — поиск подстрок в тексте

Результаты тестирования представлены в таблице 1.

Тестирование.

Для основных функций Aho_Corasick и cut_patterns_from_string были написаны тесты. Для тестирования был написан Python-скрипт - run_tests.py. Результаты тестирования представлены на Рисунке 1.

```
hp-pro@hppro-laptop:~/Desktop/lab5piaa$ python3 ./run_tests.py
<Aho_Corasick> tests:
[OK] Test #1. abcdasdawdawdawd abc,sd 1,1,6,2
[OK] Test #2. asdflhasdfjkhasdjkf w,bc,sd 2,3,8,3,15,3
[OK] Test #3. asdfasdfasdffdf fd,b,as 1,3,5,3,9,3,13,1
<cut_patterns_from_string> tests:
[OK] Test #1. abcabababababa 0,1,3,4,5,6,7,8,9,10 ca
[OK] Test #2. sdfgsdfgsdfgsdfgfsdg 0,1,4,5,8,9,12,13,17,18 fgfgfgfgfg
[OK] Test #3. ejrngkresngserjhbgserjh 0,1,7,8 rngkrngserjhbgserjh
```

Таблица 1. Результаты тестирования

Ввод	Вывод
qwuieyqwuiyeqwuie 3 qw u zz	1 1 3 2 7 1 9 2 13 1 15 2
abcabcabcabcabcabc 2 abc ca	1 1 3 2 4 1 6 2 7 1 9 2 10 1 12 2 13 1 15 2 16 1
sjadhviaebvireavuaervhuiaerhfbhkbafuaerf 3	

qwe bxc sdf	

Выводы.

При выполнении работы был изучен и реализован алгоритм Ахо-Корасик, а также с помощью данного алгоритма была разработана программа для точного поиска одного образца с джокером.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД.

Название файла: lab5.py

```
import sys

class Node:
    def __init__(self, link=None):
        self.children = {}
        self.suffix_link = link
        self.words = []

    def print(self, v='', offset=0):
        print(offset*'.',v)
        for i in self.children:
            self.children[i].print(i, offset+2)


def create_tree(patterns):
    root = Node()
    for index, pattern in enumerate(patterns):
        node = root
        for c in pattern:
            node = node.children.setdefault(c, Node(root))
        node.words.append(index)
    return root


def create_links(tree):
    queue = [value for key, value in tree.children.items()]

    while queue:
        current_node = queue.pop(0)

        for key, value_node in current_node.children.items():
            queue.append(value_node)
```

```
            link_node = current_node.suffix_link

            while link_node is not None and key not in link_node.children:
                link_node = link_node.suffix_link

            value_node.suffix_link = link_node.children[key] if link_node else tree
            value_node.words +=  value_node.suffix_link.words

    return tree



def Aho_Corasick(string, patterns):
    tree_root = create_links(create_tree(patterns))
    answer = []
    node = tree_root
    for i in range(len(string)):
        while node is not None and string[i] not in node.children:
            node = node.suffix_link
        if node is None:
            node = tree_root
            continue
        node = node.children[string[i]]
        for pattern in node.words:
            answer.append((i - len(patterns[pattern]) + 2, pattern+1))
    return answer



def max_num_edges_from_vertex(tree):
    max_n = -1
    queue = [tree]
    while queue:
        current = queue.pop()
        if len(current.children.keys()) > max_n:
            max_n = len(current.children.keys())
        for i in current.children:
            queue.append(current.children[i])
    return max_n
```

```
def cut_patterns_from_string(string, segments):
    if segments:
        segment_without_intersections = []
        current_segment = segments[0]
        for index in range(len(segments)-1):
            first_segment = segments[index]
            second_segment = segments[index+1]
            if second_segment[0] > first_segment[1]:
                segment_without_intersections.append(current_segment)
                current_segment = (second_segment[0], second_segment[1])
            else:
                current_segment = (current_segment[0], max(first_segment[1], second_segment[1]))
        else:
            segment_without_intersections.append(current_segment)

        index = 0
        result_string = string

        for segment in segment_without_intersections:
            result_string = result_string[0:segment[0]-index] + result_string[segment[1]+1-index:]
            index += segment[1] - segment[0] + 1
        return result_string
    else:
        return string

if __name__ == '__main__':
    # text = input()
    # n = int(input())
    # patterns = []
    # for i in range(n):
    #     patterns.append(input())
    text = 'ejrngkresngserjhbgserjh'
    patterns = ['e', 's']
    segments = []
    result = sorted(Aho_Corasick(text, patterns))
```

```
for i in result:
    segments.append((i[0]-1, i[0]-1+len(patterns[i[1]-1])-1))


print('Maximum number of edges from one vertex:', max_num_edges_from_vertex(create_tree(patterns)))
print('Before:', text)


print(text, segments)
print('After:',cut_patterns_from_string(text, segments))
```