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 _____ Звега А.Р.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Ахо-Корасик для точного поиска набора образцов.

Задание.

Разработайте программу, решающую задачу точного поиска набора образцов. Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с джокером. В шаблоне встречается специальный символ, именуемый джокером (wildcard), 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 (Вариант 4).

Реализовать режим поиска, при котором все найденные образцы не пересекаются в строке поиска (т.е. некоторые вхождения не будут найдены; решение задачи неоднозначно).

Выполнение работы.

По массиву подстрок(patterns) строится бор. Затем выполняется цикл по тексту в котором ведется поиск текущего символа в боре, и если символ присутствует в боре и этот символ является окончанием какой-либо подстроки то, подстрока найдена.

Для второго задания каждая подстрока массива patterns разбивается по символу джокера. Например, подстрока ‘ab*c’ с джокером ‘*’ будет разбита на две подстроки ‘ab’ и ‘c’. Затем с помощью алгоритма из Задания 1 выполняется

поиск подстрок, полученных путем разбиения. Строится массив C. C[i] = количество встретившихся в тексте безмасочных подстрок шаблона, который начинается в тексте на позиции i. 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.

Таблица 1. Результаты тестирования алгоритма

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

Таблица 2. Результаты тестирования точного поиска для одного образца с джокером.

Ввод	Вывод
ACTANCA A$$A$ $	1
aaaaaaaaaaa a*a *	1 2 3 4 5 6 7 8 9

Выводы.

При выполнении работы был изучен и реализован алгоритм Ахо-Корасик, а также с помощью данного алгоритма были разработаны программы точного поиска набора образцов и одного образца с джокером.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД.

Название файла lab51.py

```
class AhoNode:

    def __init__(self):
        self.goto = {}
        self.out = []
        self.fail = None


def aho_create_forest(patterns):

    root = AhoNode()

    for path in patterns:
        node = root
        for symbol in path:
            node = node.goto.setdefault(symbol, AhoNode())
        node.out.append(path)
    return root


def aho_create_statemachine(patterns):

    root = aho_create_forest(patterns)
```

```
    queue = []
    for node in root.goto.values():
        queue.append(node)
        node.fail = root

    while len(queue) > 0:
        rnode = queue.pop(0)

        for key, unode in zip(rnode.goto.keys(), rnode.goto.values()):
            queue.append(unode)
            fnode = rnode.fail
            while fnode is not None and key not in fnode.goto:
                fnode = fnode.fail
            unode.fail = fnode.goto[key] if fnode else root
            unode.out += unode.fail.out

    return root


def aho_find_all(s, root, callback):

    node = root
    ans = []
    for i in range(len(s)):
        while node is not None and s[i] not in node.goto:
            node = node.fail
        if node is None:
            node = root
            continue
```

```
            node = node.goto[s[i]]

            for pattern in node.out:
                ans.append([i - len(pattern) + 2, patterns.index(pattern)+1])
                #callback(i - len(pattern) + 1, patterns.index(pattern))
    return ans


def on_occurence(pos, patterns):
    print (pos + 1, patterns+1)

def sort_col(i):
    return (i[0]*10)+i[1]

s = input()
n = int(input())
patterns = []
while n > 0:
    patterns.append(input())
    n -= 1
root = aho_create_statemachine(patterns)
ans = aho_find_all(s, root, on_occurence)
ans.sort(key=sort_col)
for enter in ans:
    print(enter[0], enter[1])
```