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. 9303 _____ Павлов Д.Р.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2021

Цель работы.

Ознакомиться с алгоритмом Ахо-Корасик. Научиться применять его для решения задач, а также оценивать временную сложность алгоритма.

Задание.

1. Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст (T, $1 \leq |T| \leq 1000000$).

Вторая - число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P=\{p1,\ldots,pn\}$ $1 \leq |pi| \leq 75$

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.

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 TT.

   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.

Выполнение работы.

Для решения поставленной задачи были реализованы следующие классы и их методы:

`AhoNode` – основной класс для вершины бора, а следовательно и конечного автомата.

Имеет три поля:

`destination` – словарь, который отражает суф-фиксные ссылки,

`out` – шаблон, который мы находим при достижении вершины,

`fail` – поле, хранящее терминальную вершину.

`AhoCorasik` – основной класс решения поставленной задачи.

`reading` – для чтения входных данных из файла или из stdin.

`aho_create_forest` – создает бор.

`aho_create_statemachine` – создает конечный автомат.

`aho_find` – находит решение.

`output_result` **--** печатает частичное решение.
Для решения задачи поиска подстроки с джокером был введен метод aho_find_joker.

Разработанный программный код см. в приложении А.

Таблица 1 - Примеры тестовых случаев.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	NTAG 3 TAGT TAG T	2 2 2 3	Программа работает корректно
2.	ACTANCA A$$A$ $	1	Программа работает корректно

Выводы.

В ходе лабораторной работы был реализован алгоритм Ахо-Корасик для поиска вхождений подстрок в исходном тексте согласно заданным паттернам.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: AhoCorasik.py

```
import sys


def output_result(pos, pattern):
    print(pos, pattern)


class AhoNode:
    def __init__(self):
        self.destination = {}
        self.out = []
        self.fail = None


class AhoCorasik:
    def __init__(self):
        self.T = None
        self.P = []
        self.N = 0
        self.pattern_indices = {}
        self.alphabet = ['A', 'C', 'T', 'G', 'N']
        self.result = []

    def reading(self, path):
        if not path:
            self.T = input()
            self.N = int(input())
            for i in range(self.N):
                line = input()
                self.P.append(line)
                self.pattern_indices[self.P[-1]] = len(self.P)
        else:
            with open("".join(path), 'r') as file:
                file = file.read().splitlines()
                self.T = file[0].strip()
                self.N = int(file[1])
                for i in range(2, self.N + 2):
                    line = file[i]
                    self.P.append(line)
                    self.pattern_indices[self.P[-1]] = len(self.P)

    def aho_create_forest(self):
        root = AhoNode()
        for path in self.P:
            node = root
            for character in path:
                node = node.destination.setdefault(character, AhoNode())
            node.out.append(path)
        return root

    def aho_create_statemachine(self):
        bohr = self.aho_create_forest()
        queue = []
```

```
        for node in bohr.destination.values():
            queue.append(node)
            node.fail = bohr

        while len(queue) > 0:
            cur_node = queue.pop(0)
            # print('Current node in BFS: ', cur_node.destination, cur_node.out)
            for key, next_node in iter(cur_node.destination.items()):
                queue.append(next_node)
                fail_node = cur_node.fail

                while fail_node is not None and key not in fail_node.destina-
tion:
                    fail_node = fail_node.fail

                next_node.fail = fail_node.destination[key] if fail_node else
bohr
                next_node.out.extend(next_node.fail.out)
        return bohr

    def aho_find(self):
        root = self.aho_create_statemachine()
        node = root

        for i in range(len(self.T)):
            while node is not None and self.T[i] not in node.destination:
                node = node.fail
            if node is None:
                node = root
                continue
            node = node.destination[self.T[i]]
            for pattern in node.out:
                self.result.append([i - len(pattern) + 2, self.pattern_indi-
ces[pattern]])

        self.result.sort()
        for value in self.result:
            output_result(value[0], value[1])


if __name__ == '__main__':
    result = AhoCorasik()
    result.reading(sys.argv[1:])
    result.aho_find()
```

Название файла: Joker.py

```
import sys


class AhoNode:
    def __init__(self):
        self.destination = {}
        self.out = []
        self.fail = None


class Joker:
    def __init__(self):
        self.T = None
```

```
        self.P = []
        self.Q = None
        self.joker = None
        self.C = None
        self.result = []
        self.index = -1

    def reading(self, path):
        if not path:
            self.T = input()
            self.Q = input()
            self.joker = input()
        else:
            with open("".join(path), 'r') as file:
                file = file.read().splitlines()
                self.T = file[0].strip()
                self.Q = file[1].strip()
                self.joker = file[2].strip()
        self.aho_find_joker()

    def aho_find_joker(self):
        tmp = self.Q.split(self.joker)
        self.C = [0 for _ in range(len(self.T))]

        while '' in tmp:
            tmp.remove('')

        for line in tmp:
            self.P.append(line)

            prev = self.index + 1
            ind = self.Q[prev:len(self.Q)].find(line)
            if (ind + prev) == 0:
                self.index = 0
            else:
                self.index = ind + prev
            print("Subline:", line, " index in pattern:", self.index)
            self.find_pattern()
            self.P.remove(line)
        print("List of coefficients: ", self.C)
        for i in range(len(self.C)):
            if self.C[i] == len(tmp) and (i + len(self.Q) <= len(self.C)):
                print(int(i + 1))

    def aho_create_forest(self):
        root = AhoNode()
        for path in self.P:
            node = root
            for character in path:
                print('Appending character: ', character)
                node = node.destination.setdefault(character, AhoNode())
            node.out.append(path)
        return root

    def aho_create_statemachine(self):
        bohr = self.aho_create_forest()
        queue = []
        for node in bohr.destination.values():
            queue.append(node)
            node.fail = bohr

        while len(queue) > 0:
            cur_node = queue.pop(0)
```

```
            for key, next_node in iter(cur_node.destination.items()):
                queue.append(next_node)
                fail_node = cur_node.fail

                while fail_node is not None and key not in fail_node.destina-
tion:
                    fail_node = fail_node.fail

                next_node.fail = fail_node.destination[key] if fail_node else
bohr
                next_node.out.extend(next_node.fail.out)
        return bohr

    def find_pattern(self):
        root = self.aho_create_statemachine()
        node = root

        for i in range(len(self.T)):
            while node is not None and self.T[i] not in node.destination:
                node = node.fail
            if node is None:
                node = root
                continue
            node = node.destination[self.T[i]]
            for pattern in node.out:
                j = i - len(pattern) + 1
                if j - self.index >= 0:
                    self.C[j - self.index] += 1
        self.result.sort()


if __name__ == '__main__':
    Joker().reading(sys.argv[1:])
```