**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≤∣T∣≤100000*T*).

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.

1. 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 ТТ.  
     
   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.

## 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 |
|  | NTAG  3  TAGT  TAG  T | 2 2  2 3 | Программа работает корректно |
|  | 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.destination:

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\_indices[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.destination:

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:])