**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №5**

**по дисциплине «Проектирование и анализ алгоритмов»**

Тема: Алгоритм Ахо-Корасик

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Запевалов А.И. |
| Преподаватель |  | Филатов А.Ю. |

Санкт-Петербург

2019

**Цель работы.**

Разработать программу, реализующую алгоритм Ахо-Корасик. Программа должна решать задачу поиска вхождения образов в исходный текст.

На вход программе подаются шаблон и текст, в котором следует найти все вхождения данного шаблона.

1.**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*={*p**1*,…,*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.

2.Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с *джокером*.  
  
В шаблоне встречается специальный символ, именуемого джокером (wild card), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу *P* необходимо найти все вхождения Р в текст Т.

**Основные теоретические положения.**

Алгоритм строит конечный автомат, которому затем передаёт строку поиска. Автомат получает по очереди все символы строки и переходит по соответствующим рёбрам. Если автомат пришёл в конечное состояние, соответствующая строка словаря присутствует в строке поиска.

Несколько строк поиска можно объединить в дерево поиска, так называемый бор (префиксное дерево). Бор является конечным автоматом, распознающим одну строку из m — но при условии, что начало строки известно.

**Экспериментальные результаты.**

1.Входные данные:

ababa

1

aba

Результат:

1 1

3 1

2.Входные данные:

ACT

A$

$

Результат:

1

**Обработка результатов эксперимента.**

Подход к выполнению задачи.

1. Используемые структуры.

Основная структура, используемая в программе — суффиксное дерево, называемое бором. Каждый узел содержит информацию о родителе, детях, всех переходах, простую и сжатую суффиксную ссылки. Также необходим флаг для определения того, является ли символ последним. Для вывода результата использована структура, содержащая позицию в тексте и номер шаблона.

2. Основная функция и принцип ее работы.

Во время выполнения работы, программа производит переходы по ребрам. Если следующий символ в тексте соответствует символу ребенка, то происходит переход к ребенку, если нет — переход по суффиксной ссылке и повторение этих же действий до тех пор пока не доходит до корня. Сжатые суффиксные ссылки необходимы для проверки, если данный символ является завершаюшим для нескольких шалблонов.

При работе с шаблоном, содержащим джокеры, сначала происходит разделение шаблона на подстроки и в выводе учитывается длина всего шаблона.

**Выводы.**

В процессе выполнения работы был изучен алгоритм Ахо-Корасик и найдено решение задачи точного поиска набора шаблонов в тексте. В результате выполнения лабораторной работы разработаны программы, вычисляющие позиций вхождения простых шаблонов и шаблона с символом джокера в заданном тексте. Были решены проблемы построения конечного автомата, нахождения суффиксных ссылок и переходов по ним.

**ПРИЛОЖЕНИЕ А**

**Исходный код программы.**

#include <iostream>

#include <vector>

#include <string>

#include <queue>

using namespace std;

vector<int> prefix\_function (string s) {

int n = (int) s.length();

vector<int> p (n);

for (int i=1; i<n; ++i) {

int j = p[i-1];

while (j > 0 && s[i] != s[j])

j = p[j-1];

if (s[i] == s[j]) ++j;

p[i] = j;

}

return p;

}

void kmp(queue <int> &answer, string P, string T)

{

int pl = P.length();

int tl = T.length();

vector<int> p = prefix\_function(P + "#" + T);

for (int i=0; i<tl; i++){

if (p[pl+i+1] == pl)

answer.push(i - pl + 1);

}

}

int main()

{

string P, T;

queue <int> result;

cin >> P;

cin >> T;

kmp(result, P, T);

if (result.size())

{

cout << result.front();

result.pop();

while (!result.empty())

{

cout << ',' << result.front();

result.pop();

}

}

else

cout << -1;

return 0;

}