**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. 7304 | |  | Давыдов А.А. |
| Преподаватель |  |  | Филатов А.Ю. |

Санкт-Петербург

2019

**Цель работы.**

Изучение алгоритма точного поиска набора образцов в строке – алгоритм Ахо-Корасик.

**Задача.**

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

**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={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.



**Sample Input:**

CCCA

1

CC



**Sample Output:**

1. 1
2. 1
   1. Используя реализацию точного множественного поиска, решите задачу

точного поиска для одного образца с *джокером*.

В шаблоне встречается специальный символ, именуемого джокером (wildcard), который "совпадает" с любым символом. По заданному

2

содержащему шаблоны образцу P необходимо найти все вхождения Р в текст Т.

Например, образец аb??с?

с

джокером ? встречается

дважды

тексте xabvccbababcax

Символ джокер не входит в алфавит, символы которого используются в T. Каждый джокер соответствует одному символу, а не подстроке неопределенной длины. В шаблоне входит хотя бы один символ не джокер, те 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.



**Sample Input:**

ACT

A$

$



**Sample Output:**

1

3

**Описание алгоритма.**

1. **Алгоритм Ахо-Корасик**
   1. На основе набора паттернов строим префиксное дерево.
   2. Рассматриваем префиксное дерево как конечный детерминированный автомат. Стартовая позиция в корне.
   3. Считываем первый символ текста.
   4. Переходим в следующее состояние по ребру, обозначающему этот символ. Если такого ребра нет, то идем по суффиксной ссылке. Если суффиксной ссылки нет, то запоминаем символ, по которому пришли в данный узел, берем суффиксную ссылку родителя и пытаемся

перейти по этому символу. Данный шаг выполняется рекурсивно, пока не найден такой переход или не достигнут корень.

* 1. Выполняем шаг 4 до тех пор, пока не найдем валидный переход по текущему символу текста или пока не достигнем корня.
  2. Проверяем является ли текущее состояние каким-либо паттерном. Для этого переходим по суффиксным ссылкам, проверяя соответствующий флаг. Если это паттерн, выводим его номер и позицию в тексте в консоль.
  3. Если не конец текста переходим к шагу 3.

1. **Алгоритм Ахо-Корасик + паттерн с джокером**
   1. Разбиваем паттерн на части, разделенные джокерами. Запоминаем их позицию в паттерне (индекс).
   2. Используя алгоритм Ахо-Корасик ищем позицию этих паттернов в тексте.
   3. Создаем массив нулей, размер которого совпадает с длиной текста.

4

1. Для каждого вхождения паттерна инкрементируем *i* *–* *j + 1* позицию в массиве, где *i* – индекс вхождения, *j* – позиция данного паттерна в исходном паттерне.
2. Таким образом, в тех позициях массива, где значение совпадает с количеством паттернов, присутствует совпадение с исходным паттерном.

5

**Вывод**

* ходе лабораторной работы был изучен алгоритм Ахо-Корасик для точного поиска набора образцов в строке. Данный алгоритм был реализован на

C++. Также, этот алгоритм был использован для поиска в строке паттерна, содержащего символ джокер. Для данного случая было решено разбить строку на паттерны по символам джокерам и добавить их в бор, сохраняя индексы этих шаблонов в исходной строке паттерне, также в программу добавлена структура для хранения индекса совпадения паттерна в тексте и номера паттерна. Затем применяется модифицированный алгоритм Ахо-Корасика и выводятся индексы совпадений.

6

**Исходный код программы**

#include <iostream>

#include <map>

#include <vector>

#include <cstring>

using namespace std;

struct **Vertex**

{

int next\_vrtx[5];

int auto\_move[5]; //

int par; // parrent of this vertex

char symb; // transfer symbol from parrent

int pat\_num[40]; // number of pattern designated by this vertex

bool flag;

int suff\_link;

int suff\_flink; // good suffix link which has flag = true

};

struct **EntryPattern**

{

**EntryPattern**(int index, int pat\_num) : index(index), pat\_num(pat\_num) {}

int index;

int pat\_num;

};

vector<Vertex> bohr;

vector<string> pattern;

map<char, char> alpha\_idx{

{'A', 0},

{'C', 1},

{'G', 2},

{'N', 3},

{'T', 4},

};

//p - parrent, c - symbol of transfer

Vertex **make\_bohr\_vrtx**(int p, char c)

{

Vertex v;

memset(v.next\_vrtx, 255, sizeof(v.next\_vrtx)); // filling all bytes of next\_vrtx by -1

memset(v.auto\_move, 255, sizeof(v.auto\_move));

memset(v.pat\_num, 255, sizeof(v.pat\_num));

v.flag=false;

v.suff\_link=-1; //изначально - суф. ссылки нет

v.par=p;

v.symb=c;

v.suff\_flink=-1;

return v;

}

void **bohr\_ini**()

{

//добавляем единственную вершину - корень

bohr.push\_back(make\_bohr\_vrtx(-1, -1));

}

void **add\_string\_to\_bohr**(const string& s)

{

int num=0; //начинаем с корня

for(int i = 0; i < s.length(); i++)

{

char ch = alpha\_idx[s[i]]; //получаем номер в алфавите

if(bohr[num].next\_vrtx[ch]==-1)

{

bohr.push\_back(make\_bohr\_vrtx(num, ch));

bohr[num].next\_vrtx[ch]=bohr.size()-1;

}

num=bohr[num].next\_vrtx[ch];

}

bohr[num].flag=true;

pattern.push\_back(s);

// if pattern has same subpatterns

for(int i = 0; i < 40; ++i)

if(bohr[num].pat\_num[i] == -1)

{

bohr[num].pat\_num[i] = pattern.size() - 1;

break;

}

}

int **get\_auto\_move**(int v, char ch);

int **get\_suff\_link**(int v)

{

if (bohr[v].suff\_link==-1) //если еще не считали

if (v==0||bohr[v].par==0) //если v - корень или предок v - корень

bohr[v].suff\_link=0;

else

bohr[v].suff\_link=get\_auto\_move(get\_suff\_link(bohr[v].par), bohr[v].symb);

return bohr[v].suff\_link;

}

int **get\_auto\_move**(int v, char ch)

{

if (bohr[v].auto\_move[ch]==-1)

{

if (bohr[v].next\_vrtx[ch]!=-1)

bohr[v].auto\_move[ch]=bohr[v].next\_vrtx[ch];

else

{

if (v==0)

bohr[v].auto\_move[ch]=0;

else

bohr[v].auto\_move[ch]=get\_auto\_move(get\_suff\_link(v), ch);

}

}

return bohr[v].auto\_move[ch];

}

int **get\_suff\_flink**(int v)

{

if (bohr[v].suff\_flink==-1)

{

int u=get\_suff\_link(v);

if (u==0) //либо v - корень, либо суф. ссылка v указывает на корень

bohr[v].suff\_flink=0;

else

bohr[v].suff\_flink=(bohr[u].flag) ? u : get\_suff\_flink(u);

}

return bohr[v].suff\_flink;

}

//implemenation of searching by automation

void **check**(int v, int i, vector<EntryPattern> &ep)

{

for(int u=v; u!=0; u=get\_suff\_flink(u))

if (bohr[u].flag)

for(int j = 0; j < 40; ++j)

{

if(bohr[u].pat\_num[j]!= -1)

ep.push\_back(EntryPattern(i - pattern[bohr[u].pat\_num[j]].length(), bohr[u].pat\_num[j]));

else

break;

}

}

void **aho\_corasik**(const string& s, vector<EntryPattern> &ep){

int u=0;

for(int i=0 ; i < s.length(); i++)

{

u=get\_auto\_move(u, alpha\_idx[s[i]]);

check(u, i+1, *ep*);

}

}

int **main**()

{

bohr\_ini();

string text, pattern\_str;

char joker;

cin >> text >> pattern\_str >> joker;

std::vector<int> matches(text.length(), 0);

std::vector<std::string> patterns;

std::vector<int> patterns\_pos;

for (int i = 0; i < pattern\_str.length(); i++)

{

std::string pat;

if (pattern\_str[i] != joker)

{

patterns\_pos.push\_back(i + 1);

for (int j = i; pattern\_str[j] != joker && j != pattern\_str.length(); j++)

{

pat += pattern\_str[j];

i++;

}

patterns.push\_back(pat);

}

}

for(string &el : patterns)

add\_string\_to\_bohr(el);

vector<EntryPattern> ep;

aho\_corasik(text, *ep*);

for(int i = 0; i < ep.size(); ++i)

{

if(ep[i].index < patterns\_pos[ep[i].pat\_num] - 1)

continue;

//increment i + j - 1 position in vector of mathes

matches[ep[i].index - patterns\_pos[ep[i].pat\_num] + 1]++;

/\*

\* if in position i + j - 1 has count of matches = count of subpatterns of pattern\_str

\* second condition check that pattern does not go beyond the text

\*/

if(matches[ep[i].index - patterns\_pos[ep[i].pat\_num] + 1] == patterns.size() &&

ep[i].index - patterns\_pos[ep[i].pat\_num] + 1 <= text.length() - pattern\_str.length())

cout << ep[i].index - patterns\_pos[ep[i].pat\_num] + 2 << endl;

}

return 0;

}