**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по практической работе №5**

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

Тема: **Алгоритм Ахо — Корасик**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7383 |  | Кирсанов А.Я. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

**Постановка задачи.**

**Цель работы.**

Разработать программу, решающую задачу точного поиска набора образцов.

Задание 1. Первая строка содержит текст .

Вторая - число ***n*** , каждая следующая из ***n*** строк содержит шаблон из набора . Выход: все вхождения образцов из ***P*** в ***T***.

Задание 2. 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***.

**Реализация задачи.**

Была создана структура **node**, описывающая вершину бора, со следующими полями:

**map<char, node\*> go** – контейнер, хранящий указатель на вершину, в которую мы можем перейти по соответствующему символу.

**map<char, node\*> son –** контейнер, хранящий сыновей вершины.

**node\* suffLink –** суффикс-ссылка вершины.

**node\* parent –** указатель на вершину-родителя.

**node\* up –** сжатая суффикс-ссылка вершины.

**char charToParent –** символ, по которому был совершен переход из вершины- родителя.

**bool isEnd –** соответствует-ли вершина какому-либо шаблону.

**size\_t endNumber –** номер шаблона, которому соответствует вершина.

**size\_t patternLeng –** длина шаблона.

Также был создан класс **Aho** со следующими полями:

**node\* root** – указатель на корень бора.

**char joker** – символ джокера.

**vector<char> alph** – алфавит, использующийся в задании.

**map<size\_t, map<size\_t, bool>> result** – контейнер, отсортированный по индексам вхождения шаблонов и их номеров.

**vector<node\*> vertexes** – контейнер, содержащий указатели на созданные вершины.

В классе **Aho** реализованы следующие функции:

**void makeTree(const string & s, size\_t patternNumber)** – функция, строящая бор по переданной строке. Начальная позиция – корень бора. На каждом шагу функции берется очередной символ ***c*** из строки ***s*** и, если по нему из данной вершины еще нет перехода, создается новая вершина. Если взятый символ – джокер, то в контейнер **go** кладутся все символы алфавита, то есть из текущей вершины мы можем перейти в следующую по любому символу. Затем осуществляется переход из текущей вершины по символу ***с***. Конечному состоянию шаблона присваивается его номер **patternNumber**, длина и метка о том, что состояние конечное.

**node\* getSuffLink(node\* v**) – возвращает указатель на вершину, соответствующую максимально возможной длине суффикса текущего состояния. Суффиксная ссылка на корень есть корень.

**node\* getLink(node\* v, char c)** – функция осуществляет переход по из вершины ***v*** по символу ***c***. Если прямого перехода нет, для перехода используется суффиксная ссылка.

**node\* getUp(node\* v)** – сжатая суффиксная ссылка. Действует также, как и суффиксная ссылка, но возвращает ближайшую к ***v*** конечную вершину. (Наибольший суффикс, являющийся шаблоном).

**void processText(const string & t)** – осуществляет поиск всех вхождений шаблонов в текст. Начальное состояние – корень бора. На каждом шаге функции производится переход из текущего состояния по очередному символу текста ***t***, для этого вызывается функция **getLink**. Если функция приходит в какое-либо конечное состояние, в контейнер result записывается индекс вхождения шаблона (индекс строки минус длина шаблона + 2) и его номер, затем, пока мы не придем в корень, осуществляется переход из данного состояния по сжатой суффиксной ссылке, чтобы сразу обработать все шаблоны, являющиеся подстроками найденного шаблона.

**Описание работы программы.**

Функция **main()** считывает текст и следующую строку. Если следующая строка является числом ***N***, то считываются ещё ***N*** шаблонов и создается объект класса **Aho** путем вызова конструктора для задания 1. Если строка не является числом, считывается символ джокера и вызывается конструктор **Aho** для   
задания 2. Конструкторы создают вершину-корень, вызывают функцию **makeTree** для всех шаблонов, затем функцию **processText**. Затем в **main** у созданного объекта вызывается метод **print()**, выводящий пары (индекс вхождения шаблона в текст, номер шаблона) для задания 1, или только индексы вхождения шаблона для задания 2.

Исходный код программы представлен в Приложении Б.

**Исследование сложности алгоритма.**

Функция **makeTree** строит красно-черное дерево, время обращения элементов в котором составляет , где *T* – число элементов. Таким образом вычислительная сложность алгоритма составляет , где *H* – длина текста для поиска, *n* – общая длина всех слов в словаре, *k* – суммарная длина всех совпадений.   
Сложность по памяти - .

**Тестирование.**

Программа тестировалась в среде разработки Qt с помощью компилятора MinGW 5.3.0 в операционной системе Windows 10.

Тестовые случаи представлены в Приложении А.

**Выводы.**

В ходе выполнения задания был реализован алгоритм   
Ахо-Корасик для поиска множества шаблонов в строке. Оценена сложность алгоритма.

**ПРИЛОЖЕНИЕ А**

**ТЕСТОВЫЕ СЛУЧАИ**

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| ACT  A$  $ | 1 |
| CCCA  1  CC | 1 1  2 1 |
| AAAAAA  2  AA  AAA | 1 1  1 2  2 1  2 2  3 1  3 2  4 1  4 2  5 1 |

**ПРИЛОЖЕНИЕ Б**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

#include <iostream>

#include <string>

#include <map>

#include <vector>

#include <cstdlib>

#define N 3000

using namespace std;

struct node{

map<char, node\*> go;

map<char, node\*> son;

node\* suffLink = nullptr;

node\* parent = nullptr;

node\* up = nullptr;

char charToParent;

bool isEnd;

size\_t endNumber;

size\_t patternLeng;

};

class Aho{

public:

Aho(const string & t, const string & str, char j){

root = new node;

root->isEnd = false;

root->suffLink = root;

joker = j;

makeTree(str, 0);

processText(t);

}

Aho(const string & t, const string str[], size\_t n){

root = new node;

root->isEnd = false;

root->suffLink = root;

joker = 0;

for (size\_t i = 0; i < n; i++) {

makeTree(str[i], i + 1);

}

processText(t);

}

~Aho(){

delete root;

for(auto it : vertexes)

delete it;

}

void print(){

for (auto it : result) {

for (auto tr : it.second) {

if(tr.first == 0)

cout << it.first << endl;

else

cout << it.first << ' ' << tr.first << endl;

}

}

}

private:

void makeTree(const string & s, size\_t patternNumber){

node\* cur = root;

size\_t i;

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

char c = s[i];

if(cur->son[c] == nullptr){

cur->son[c] = new node;

vertexes.push\_back(cur->son[c]);

cur->son[c]->parent = cur;

cur->son[c]->charToParent = c;

cur->son[c]->isEnd = false;

cur->son[c]->go[c] = nullptr;

if(c == joker){

for (auto it : alph) {

cur->go[it] = cur->son[c];

}

}

}

cur = cur->son[c];

}

cur->isEnd = true;

cur->patternLeng = i;

cur->endNumber = patternNumber;

}

node\* getSuffLink(node\* v){

if (v->suffLink == nullptr){

if(v == root || v->parent == root)

v->suffLink = root;

else {

v->suffLink = getLink(getSuffLink(v->parent), v->charToParent);

}

}

return v->suffLink;

}

node\* getLink(node\* v, char c){

if(v->go[c] == nullptr){

if(v->son[c])

v->go[c] = v->son[c];

else if(v == root)

v->go[c] = root;

else

v->go[c] = getLink(getSuffLink(v),c);

}

return v->go[c];

}

node\* getUp(node\* v){

if(v->up == nullptr){

if(getSuffLink(v)->isEnd)

v->up = getSuffLink(v);

else if(getSuffLink(v) == root)

v->up = root;

else

v->up = getUp(getSuffLink(v));

}

return v->up;

}

void processText(const string & t){

node\* cur = root;

for(size\_t i = 0; i < t.length(); i++){

cur = getLink(cur, t[i]);

if(cur->isEnd == true){

result[i - cur->patternLeng + 2].insert(pair<size\_t, bool>(cur->endNumber, 1));

}

node\* tmpcur = cur;

while(getUp(tmpcur) != root){

tmpcur = getUp(tmpcur);

if(tmpcur->isEnd == true)

result[i - tmpcur->patternLeng + 2].insert(pair<size\_t, bool>(tmpcur->endNumber, 1));

}

}

}

node\* root;

char joker;

vector<char> alph = {'A', 'C', 'G', 'T', 'N'};

map<size\_t, map<size\_t, bool>> result;

vector<node\*> vertexes;

};

int main()

{

string text, pattern[N];

char joker;

size\_t n;

cin >> text >> pattern[0];

n = static\_cast<size\_t>(atoi(pattern[0].c\_str()));

if(n){

for (size\_t i = 0; i < n; i++) {

cin >> pattern[i];

}

Aho aho(text, pattern, n);

aho.print();

}

else{

cin >> joker;

Aho aho(text, pattern[0], joker);

aho.print();

}

return 0;

}