МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо–Корасик

Студент гр. 8303 Логинов Е.А.

Преподаватель Фирсов М.А.

Санкт-Петербург

2020

Цель работы

Изучение алгоритма Ахо-Корасика для поиска всех вхождений каждой строки из данного набора.

Вариант 2.

Подсчитать количество вершин в автомате; вывести список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска

Задача 1.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст (T,$1 \leq |T| \leq 100000$).

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.

Sample Input:

CCCA

1

CC

Sample Output:

1 1

2 1

Задача 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 T.

Например, образец a*b*??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,$1 \le |T| \le 100000$)

Шаблон (P,$1 \le |P| \le 40$)

Символ джокера

Выход:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).
Номера должны выводиться в порядке возрастания.

Sample Input:

ACTANCA
A$$A$
$

Sample Output:

1

Описание алгоритмов.

1) **Алгоритм Ахо-Корасик**

Несколько строк поиска можно объеденить в дерево поиска (бор или префиксное дерево). Бор является конечным автоматом.

Алгоритм получает на вход набор строк-шаблонов, по которым осуществляется построение бора. Для построения бора сначала создается корень. После этого поочередно рассматриваются символы строки. В случае, если в боре существует переход по данному символу, то выполняется переход в следующее состояние. Если же перехода нет, создается новый узел и добавляется в бор.

Далее алгоритм строит конечный автомат из бора при помощи добавления суффиксальных ссылок, которому затем передает строку поиска.

В автомат, полученный из построенного бора, подается символ. Если существует переход по текущему символу, то автомат переходит в следующее состояние. Если прямого перехода из текущей вершины нет, то переход производится по суффиксальной ссылке. Для потомков корня суффиксальная вершина является корнем. При достижении

конечной(терминальной) вершины записывается индекс шаблона и вычисляется его позиция в тексте.

Для вычисления суффиксальной ссылки производится проверка на то, является ли данная вершина корнем. Если вершина – корень, то ссылка ссылается на этот корень и считается равной 0. Иначе происходит переход по суффиксальной ссылке предка и переход по текущему символу. В случае, если предок не имеет суффиксальной ссылки, то для него она определяется рекурсивно таким же образом.

Из текущей вершины алгоритм переходит по суффиксальной ссылке, пока не найдет терминальную вершину или корень. Если вершина по суффиксальной ссылке будет терминальной, то это и есть искомая вершина.

Сложности алгоритма.

Сложность по операциям: O(n+m), где n – количество символов в тексте, m – суммарная длина шаблонов.

Сложность по памяти: O(m + n), где n – количество символов в тексте, m – суммарная длина шаблонов.

2) Алгоритм поиска по строке с джокером

В начале происходит считывание текста строки с джокерами. Строка с джокерами разбивается на подстроки без них, фиксируются их позиции в исходной строке. Затем данные подстроки-шаблона заносятся в бор.

Выполняется поиск всех вхождений подстроки в строку при помощи алгоритма Ахо-Корасик. Для каждого совпадения фиксируется индекс начала совпадения. Если индекс попадает в заданный диапазон, то счетчик в векторе по данному индексу увеличивается на единицу.

После этого ищется место в тексте, в котором число совпадений совпадает с количеством подстрок. Если удалось найти совпадения, то выводится позиция вхождения.

Сложности алгоритма.

Сложность по операциям: O(n+m), где n – количество символов в тексте, m – суммарная длина шаблонов.

Сложность по памяти: O(m + 2 + n + k), где n – количество символов в тексте, m – суммарная длина шаблонов, k — количество подстрок.

Описание структур данных и функций алгоритма Ахо-Корасик.

```
class Vertex {
std::map<char,int> nextV; // множество ребер бора
std::map<char,int> machineSt;   // состояния автомата
bool terminalTop;  // флаг для проверки терминальной вершины
int parentI;  //индекс родительской вершины
char parentSymb; // значение родительской вершины
int link; // суффиксная ссылка
int patternNum; //номер шаблона в терминальной вершине.
 };
```

Класс предназначен для хранения вершин бора.

nextV – контейнер map, содержит переходы по ребрам в боре

machineSt - контейнер map, хранит переходы в автомате

terminalTop – флаг для проверки на терминальную вершину

parentI – индекс родительской вершины

parentSymb – значение родительской вершины

link – суффиксная ссылка

patternNum – номер шаблона в терминальной вершине

```
class Trie {
public:
```

std::vector<Vertex> vertexes; *// контейнер для хранения бора*

std::vector<std::string> patterns; *//подстроки*

std::vector<std::pair<int,int>> result; *//результат* };

Класс предназначен для хранения бора.

vertexes – контейнер vector для хранения вершин бора

patterns - контейнер vector для хранения шаблонов

result - контейнер vector для записи результата работы программы

void init() – метод для инициализации бора.

void addStr (const std::string & s) – метод добавляет новую строку в бор. Получает на вход строку s, которую необходимо добавить.

int nextSt(int curSt, char symb) – метод переводит автомат в новое состояние. Получает на вход текущее состояние curSt и символ перехода symb. Возвращает номер нового состояния.

int getLink (int curTop) – метод вычисления суффиксной сыллки. Получает на вход текущую вершину curTop. Возвращает номер вершины, на которую указывает ссылка curTop.

void print() – метод выводит на экран результат работы программы.

void ahoCorasick(std::string& text) – основной метод программы, реализует алгоритм Ахо-Корасик. Получает на вход строку text, в которой будут искаться совпадения.

Описание структур данных и функций алгоритма с им.

```
class Vertex {
public:
   std::map<char,int> nextV; //множество ребер бора
```

```
    std::map<char,int> machineSt; // состояния автомата
    bool terminalTop; // флаг для проверки терминальной вершины
    int parentI; // индекс родительской вершины
    char parentSymb; // символ родительской вершины
    int link;  // суффиксная ссылка
    std::vector<int> listPatterns;  // множество индексов шаблонов
};
```

Класс предназначен для хранения вершин бора.
nextV – контейнер map, содержит переходы по ребрам в боре
machineSt - контейнер map, хранит переходы в автомате
terminalTop – флаг для проверки на терминальную вершину
parentI – индекс родительской вершины
parentSymb – значение родительской вершины
link – суффиксная ссылка
patternNum – номер шаблона в терминальной вершине
listPatterns – контейнер для хранения множества индексов подстрок

```
class Trie
{
    std::vector<Vertex> vertexes; //контейнер для хранения бор
    std::vector<std::string> patterns; // множество подстрок
    std::vector<int> result;  //результат
    std::vector<int> patternsI; //индексы вхождения подстрок в строке с
джокером};
```

Класс предназначен для хранения бора.
vertexes – контейнер vector для хранения вершин бора
patterns - контейнер vector для хранения шаблонов
result - контейнер vector для записи результата работы программы

patternsIndex — вектор vector для хранения индексов подстрок в исходной строке с джокерами

void init() - метод для инициализации бора.

void addStr (const std::string & s) – метод добавляет новую строку в бор. Получает на вход строку s, которую необходимо добавить.
int nextSt(int curSt, char symb) – метод переводит автомат в новое состояние. Получает на вход текущее состояние curSt и символ перехода symb. Возвращает номер нового состояния.

void print() – метод выводит на экран результат работы программы.

void splitStr(std::string& pattern, char symbJ) – метод для разбиения строки с джокерами. Получает на вход исходную строку с джокерами pattern и символ джокера symbJ.

void ahoCorasickJ(std::string& text, std::string& pattern) – основной метод программы, выполняет поиск с джокером при помощи алгоритма Ахо-Корасик. Получает на вход строку text, в которой будут искаться совпадения и строку с джокерами pattern.

Тестирование.

1) Алгоритм Ахо-Корасик

```
NTAG
3
TAGT
TAG
T
2 2
2 3
```

Пример 1

```
CCCA
1
CC
1 1
2 1
```

Пример 2

```
abcabcb
5
abcabcb
abcb
cabcb
cb
b
1 1
2 5
3 3
4 2
5 5
6 4
7 5
```

Пример 3

Пример с отладкой:

```
CCCA
1
CC

<added new str to trie: CC>
symbol C
add new vertex C
symbol C
add new vertex C

<current symbol: C>
next state: 1
state: 1

get reference to root:
calculate reference to state: 0

<current symbol: C>
next state: 2
state: 2
found pattern -1: CC
calculate reference through parent:
calculate reference to state: 0

<current symbol: C>
next state: 1
calculate reference to state: 1
calculate reference to state: 0

<current symbol: C>
calculate reference to state: 1
```

Пример 4

```
<current symbol: C>
next state: 2
go through reference: 2
next state: 2
state: 2
found pattern 0: CC
calculate reference to state: 1
calculate reference to state: 0

<current symbol: A>
calculate reference to state: 1

<current symbol: A>
calculate reference to state: 0

<current symbol: A>
next state: 0
go through reference: 0
next state: 0
go through reference: 0
next state: 0
state: 0
Result of programm:
1 1
2 1
vertexes: 3
crossing of patterns CC and CC. <1>, <1> at ind 1,2
```

2) Алгоритм поиска с джокером

ACTANCA	*axxaxxabcabcx*	*axaxax*
A$$A$	*a??a??abc?*	*a$a*
$	*?*	*$*
1	1	1
	4	3
Пример 1	Пример 2	Пример 3

Пример с отладкой:

```
ACTANCA
A$$A$
$

<added new str to trie: A>
symbol A
add new vertex A

<added new str to trie: A>
symbol A

<current symbol: A>
next state: 1
state: 1
found pattern
coincides in position: 0

get reference to root:
calculate reference to state: 0

<current symbol: C>
calculate reference to state: 0

<current symbol: C>
go through reference: 0
next state: 0
go through reference: 0
next state: 0
state: 0

<current symbol: T>
go through reference: 0
next state: 0
state: 0
```

Пример 4

```
<current symbol: A>
next state: 1
state: 1
found pattern
coincides in position: 0
calculate reference to state: 0

<current symbol: N>
calculate reference to state: 0

<current symbol: N>
go through reference: 0
next state: 0
go through reference: 0
next state: 0
state: 0

<current symbol: C>
next state: 0
state: 0

<current symbol: A>
next state: 1
state: 1
found pattern
calculate reference to state: 0
Result of programm:
1
```

Пример 4 продолжение

Выводы.

В ходе выполнения лабораторной работы были получены навыки работы с алгоритмом Ахо-Корасик и алгоритмом поиска с «джокером». Были написаны программы, реализующую эти алгоритмы работы со строками.

ПРИЛОЖНИЕ А

ИСХОДНЫЙ КОД

1. Ахо-Корасик

```
#include <vector>
#include <map>
#include <iostream>
#include <algorithm>

#define show_debug

bool cmp(std::pair<int, int> i, std::pair<int, int> j)
{
    if(i.second == j.second)
        return i.first < j.first;
    return i.second < j.second;
}

class Vertex {
public:
    std::map<char,int> nextV; // множество ребер бора
    std::map<char,int> machineSt;   // состояния автомата
    bool terminalTop;  // флаг для проверки терминальной вершины
    int parentI;  //индекс родительской вершины
    char parentSymb; // значение родительской вершины
    int link; // суффиксная ссылка
    int patternNum; //номер шаблона в терминальной вершине

    Vertex() {
    }
    Vertex(bool leaf, int parent, char parentChar, int link):
            terminalTop(leaf), parentI(parent), parentSymb(parentChar), link(link)
    { }
};

class Trie
{
    std::vector<Vertex> vertexes; // контейнер для хранения бор
    std::vector<std::string> patterns; //подстроки
    std::vector<std::pair<int,int>> result; //результат
public:

    void init() {  // инициализация бора
        Vertex head;
        head.parentI = head.link = -1;
        head.terminalTop = false;
        vertexes.push_back(head);
    }

    void addStr (const std::string & s) {
#ifdef show_debug
        std::cout<<std::endl;
        std::cout << "<added new str to trie: " << s << ">"<< std::endl;
#endif
        patterns.push_back(s);
        int v = 0;
        for (auto c : s) {  //просмотр по всему шаблону
#ifdef show_debug
            std::cout << "symbol " << c << std::endl;
#endif
```

```
            if (vertexes[v].nextV.find(c) == vertexes[v].nextV.end()) { // если
вершины в боре нет, добавляем
#ifdef show_debug
                std::cout << "add new vertex " << c << "\n";
#endif
                Vertex buffer(false, v, c, -1);
                vertexes.push_back(buffer);
                vertexes[v].nextV[c] = vertexes.size() - 1;
            }
            v = vertexes[v].nextV[c];
        }
        vertexes[v].terminalTop = true; // помечаем вершну терминальной
        vertexes[v].patternNum = patterns.size() - 1; //добавляем индекс шаблона
    }

    int nextSt(int curSt, char symb) { //метод перехода в новое состояние автомата
#ifdef show_debug
        std::cout<<std::endl;
        std::cout << "<current symbol: " << symb <<">" << "\n";
#endif
        if (vertexes[curSt].machineSt.find(symb) == vertexes[curSt].machineSt.end())
//переход в новое состояние
        {
            if (vertexes[curSt].nextV.find(symb) == vertexes[curSt].nextV.end())
            {
                if(curSt == 0)
                    vertexes[curSt].machineSt[symb] = 0;
                else {
                    vertexes[curSt].machineSt[symb] = nextSt(getLink(curSt), symb); //
переход по ссылке
#ifdef show_debug
                    std::cout << "go through reference: " <<
vertexes[curSt].machineSt[symb] << std::endl;
#endif
                }
            }
            else
            {
                vertexes[curSt].machineSt[symb] = vertexes[curSt].nextV[symb];
            }
        }
#ifdef show_debug
        std::cout << "next state: " << vertexes[curSt].machineSt[symb] << std::endl;
#endif
        return vertexes[curSt].machineSt[symb];
    }


    int getLink (int curTop) { // метод вычисления суффиксной ссылки
        if (vertexes[curTop].link == -1) // при отстутствии сслыки вычисляем ее
        {
            if (curTop == 0 || vertexes[curTop].parentI == 0) // добавлем ссылку на
корень
            {
#ifdef show_debug
                std::cout<<std::endl;
                std::cout << "get reference to root: " <<" \n";
#endif
                vertexes[curTop].link = 0;
            }
            else// вычисление ссылки через родительскую вершину
            {
#ifdef show_debug
                std::cout << "calculate reference through parent: " << "\n";
```

```
#endif
                vertexes[curTop].link = nextSt(getLink(vertexes[curTop].parentI),
vertexes[curTop].parentSymb);
            }
        }
#ifdef show_debug
        std::cout << "calculate reference to state: " << vertexes[curTop].link <<" \
n";
#endif
        return vertexes[curTop].link;
    }

    void print()
    {
        std::sort(result.begin(), result.end(), cmp);
#ifdef show_debug
        std::cout << "Result of programm: " <<std::endl;
#endif
        for(auto iter : result)
        {
            std::cout << iter.second + 2 << " " << iter.first + 1 << std::endl;
        }

#ifdef show_debug
        std::cout << "vertexes: " << vertexes.size() << std::endl;
#endif
        for(size_t i = 0 ; i < result.size() - 1; i++) {
            for(size_t j = i + 1 ; j < result.size(); j++)
            {
                size_t first, second;
                first = patterns[result[i].first].size() - 1 + result[i].second;
                second =  result[j].second;
                if(first >= second)
                {
                    std::cout << "crossing of patterns " << patterns[result[i].first]
                    << " and " << patterns[result[j].first] << ". <" <<
result[i].first + 1 <<">"
                    << ", <" << result[j].first + 1 <<">" << " at ind " <<
result[i].second + 2
                    << "," << result[j].second + 2 << "\n";
                }
            }
        }
    }

    void ahoCorasick(std::string& text)
    {
        int state = 0;
        int i = 0;
        for(auto c : text)
        {
            state = nextSt(state, c); // переход в новое состояние
#ifdef show_debug
            std::cout << "state: " << state <<" \n";
#endif
            size_t tmp = state;
            while(tmp!=0) // цикл для поиска всех возможных вхождений
            {
                if(vertexes[tmp].terminalTop)
                {
                    std::pair<int,int> buffer;  // добавление в результат найденного
шаблонаи его позиции
                    buffer.first = vertexes[tmp].patternNum;
                    buffer.second = i - patterns[buffer.first].size();
```

```
                    result.push_back(buffer);
#ifdef show_debug
                    std::cout << "found pattern " << buffer.second << ": " <<
patterns[buffer.first] << " \n";
#endif
                }
                tmp = getLink(tmp);
            }
            i++;
        }
    }


};

/*
NTAG
3
TAGT
TAG
T

CCCA
1
CC
*/

int main() {

    Trie trie;
    trie.init();
    std::string text, pattern;
    int n = 0;
    std::cin >> text>>n;
    for(int i = 0; i < n ;i++)
    {
        std::cin >> pattern;
        trie.addStr(pattern);
    }

    trie.ahoCorasick(text);
    trie.print();
    return 0;
}
```

2. Поиск по строке с джокером

```
#include <iostream>
#include <map>
#include <vector>
#define debug

class Vertex {
public:
    std::map<char,int> nextV; //множество ребер бора
    std::map<char,int> machineSt; // состояния автомата
    bool terminalTop; // флаг для проверки терминальной вершины
    int parentI; // индекс родительской вершины
    char parentSymb; // символ родительской вершины
    int link;  // суффиксная ссылка
```

```
    std::vector<int> listPatterns;  // множество индексов шаблонов
};

class Trie
{
    std::vector<Vertex> vertexes; //контейнер для хранения бор
    std::vector<std::string> patterns; // множество подстрок
    std::vector<int> result;  //результат
    std::vector<int> patternsI; //индексы вхождения подстрок в строке с джокером
public:

    void init() { // инициализация бора
        Vertex head;
        head.parentI = head.link = -1;
        head.terminalTop = false;
        vertexes.push_back(head);
    }

    void addStr (std::string s) {
#ifdef debug
        std::cout<<std::endl;
        std::cout << "<added new str to trie: " << s << ">"<< std::endl;
#endif
        patterns.push_back(s);
        int v = 0;
        for (auto c : s) {
#ifdef debug
            std::cout << "symbol " << c << std::endl;
#endif
            if (vertexes[v].nextV.find(c) == vertexes[v].nextV.end()) {  // если
вершины в боре нет, добавляем
#ifdef debug
                std::cout << "add new vertex " << c << "\n";
#endif
                Vertex buffer;
                buffer.terminalTop = false;
                buffer.link = -1;
                buffer.parentI = v;
                buffer.parentSymb = c;

                vertexes.push_back(buffer);
                vertexes[v].nextV[c] = vertexes.size() - 1;
            }
            v = vertexes[v].nextV[c];
        }
        vertexes[v].terminalTop = true;  // помечаем вершну терминальной
        vertexes[v].listPatterns.push_back(patterns.size() - 1); // добавляем индекс
шаблона
    }

    int nextSt(int curSt, char symb) { //метод перехода в новое состояние автомата
#ifdef debug
        std::cout<<std::endl;
        std::cout << "<current symbol: " << symb <<">" << "\n";
#endif
        if (vertexes[curSt].machineSt.find(symb) == vertexes[curSt].machineSt.end())
//переход в новое состояние
        {
            if (vertexes[curSt].nextV.find(symb) == vertexes[curSt].nextV.end())
            {
                if(curSt == 0)
                    vertexes[curSt].machineSt[symb] = 0;
                else
                    vertexes[curSt].machineSt[symb] = nextSt(getLink(curSt), symb); //
```

```
переход по ссылке
#ifdef debug
                std::cout << "go through reference: " <<
vertexes[curSt].machineSt[symb] << std::endl;
#endif
            }
            else
            {
                vertexes[curSt].machineSt[symb] = vertexes[curSt].nextV[symb];
            }
        }
#ifdef debug
        std::cout << "next state: " << vertexes[curSt].machineSt[symb] << std::endl;
#endif
        return vertexes[curSt].machineSt[symb];
    }

    int getLink (int curTop) {                                                  //
метод получения суффиксной ссылки
        if (vertexes[curTop].link == -1)                                        //
при отстутствии сслыки вычисляем ее
        {
            if (curTop == 0 || vertexes[curTop].parentI == 0)
// добавлем ссылку на корень
            {
#ifdef debug
                std::cout<<std::endl;
                std::cout << "get reference to root: " <<" \n";
#endif
                vertexes[curTop].link = 0;
            }
            else                                                                //
вычисление ссылки через родительскую вершину
            {
#ifdef debug
                std::cout << "calculate reference through parent: " << "\n";
#endif
                vertexes[curTop].link = nextSt(getLink(vertexes[curTop].parentI),
vertexes[curTop].parentSymb);
            }
        }

#ifdef debug
        std::cout << "calculate reference to state: " << vertexes[curTop].link <<" \
n";
#endif
        return vertexes[curTop].link;
    }

    void ahoCorasickJ(std::string& text, std::string& pattern)
    {
        int state = 0;
        result.resize(text.size()); //вектор результата
        for(int i = 0; i < text.size();i++) {
            state = nextSt(state, text[i]); // переход в новое состояние
#ifdef debug
            std::cout << "state: " << state <<" \n";
#endif
            size_t tmp = state;
            while (tmp!=0){
                if(vertexes[tmp].terminalTop)
                {
#ifdef debug
                    std::cout << "found pattern "  << std::endl;
```

```
#endif
                    for(auto Li : vertexes[tmp].listPatterns) // вычисление индексов
вхождений
                    {
                        int buffer = i + 1 - patterns[Li].size() - patternsI[Li];
                        if(buffer >=0 && buffer <= text.size() - pattern.size())
                        {
#ifdef debug
                            std::cout << "coincides in position: " << buffer <<
std::endl;
#endif

                            result[buffer]++;
                        }
                    }
                }
                tmp = getLink(tmp);
            }
        }
    }

    void splitStr(std::string& pattern, char symbJ) //метод разбиения строки
    {
        size_t currentPos, prevPos;
        for(size_t i = 0; i < pattern.size() && currentPos != std::string::npos;)
        {
            std::string buffer;
            while(pattern[i] == symbJ) i++;  // пропуск джокера в строке
            prevPos = i;
            currentPos = pattern.find(symbJ, i); //поиск следующего джокера
            if(currentPos == std::string::npos) //создание подстроки
                buffer = pattern.substr(i, pattern.size() - i);
            else
                buffer = pattern.substr( prevPos,currentPos - prevPos);
            if(!buffer.empty())
            {
                patternsI.push_back(prevPos); // запись индекса подстроки и добавление
ее в бор
                addStr(buffer);
            }
            i = currentPos;
        }
    }

    void print(std::string pattern)
    {
#ifdef debug
        std::cout << "Result of programm: " <<std::endl;
#endif
        for(size_t i = 0; i < result.size(); i++)
        {
            if(result[i] == patterns.size())
                std::cout << i + 1<< "\n";
        }
    }
};

/*
ACTANCA
A$$A$
$


ахахах
а$а
```

```
$
*/

int main() {

    Trie trie;
    trie.init();
    std::string text, pattern;
    char joker;
    std::cin >> text>> pattern>> joker;

    trie.splitStr(pattern, joker);
    trie.ahoCorasickJ(text, pattern);
    trie.print(pattern);

    return 0;
}
```