

**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. 8303

\_\_\_\_\_

Данилов А.В.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

## **Цель работы.**

Изучить алгоритм Ахо-Корасик и алгоритм поиска вхождений шаблонов с “джокерами” в строку. Написать программу, реализующую эти алгоритмы работы со строками.

**Вариант 1.** На месте джокера может быть любой символ, за исключением заданного.

## **Алгоритм Ахо-Корасик**

### **Задание.**

Разработайте программу, решающую задачу точного поиска набора образцов.

### **Вход:**

Первая строка содержит текст ( $T$ ,  $1 \leq |T| \leq 1000000$ ).

Вторая - число  $n$  ( $1 \leq n \leq 3000$ ), каждая следующая из  $n$  строк содержит шаблон из набора  $P = \{p_1, \dots, p_n\}$   $1 \leq |p_i| \leq 75$

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.

## **Пример входных данных**

СССА

1

СС

## **Пример выходных данных**

1 1

2 1

### **Описание алгоритма.**

В начале алгоритма бор заполняется символами шаблонов. Для этого поочередно обрабатывается каждый символ шаблона. Если перехода в боре для текущей вершины нет, то вершина создается, добавляется в бор и в нее совершается переход по текущему символу. Если вершина с переходом по текущему символу уже существует, то в нее совершается переход.

Далее осуществляется поиск шаблонов в текстовой строке. Для этого обрабатывается автомат, полученный из созданного бора путем добавления суффиксных ссылок.

Обрабатывается текущий символ текстовой строки. Если в автомате уже существует ребро-переход по символу в вершину, то осуществляется переход в эту вершину. Если ребра-перехода в автомате еще нет, но существует переход по текущему символу в вершину-сына, то этот переход осуществляется и добавляется в ребра автомата. Если такого перехода также не существует, то переход осуществляется по суффиксной ссылке и также заносится в ребра автомата.

Для нахождения суффиксной ссылки для вершины, осуществляется переход в предка вершины, затем переход по суффиксной ссылке предка и переход по текущему символу. Если предок не имеет суффиксной ссылки, то для него она определяется аналогичным образом рекурсивно.

Если во время перехода в автомате встречается терминальная вершина, это означает, что шаблон в подстроке найден. Вычисляется индекс его в строке и заносится в вектор результата.

Для вывода максимального числа дуг, исходящих из одной вершины бора перебираются вершины-дети бора. Если число дуг для текущей вершины больше переменной, хранящей это максимальное число, то в

переменную заносится это новое значение. Результатом является значение, хранящееся в этой переменной.

Для вывода строки, из которой были удалены найденные шаблоны заводится булевский вектор. Индексы, соответствующие индексам с символами шаблона в строке, помечаются. Строка формируется путем добавления в нее символов, индексы которых не были помечены.

### **Сложность алгоритма по операциям:**

Таблица переходов автомата хранится в структуре `std::map`, которая реализована как красно-черное дерево. Тогда сложность алгоритма по операциям будет равна  $O((M+N)*\log(k)+t)$ ,  $M$  – длина всех символов слов шаблонов,  $N$  – длина текста, в котором осуществляется поиск,  $k$  – размер алфавита,  $t$  – длина всех возможных вхождений всех строк-образцов.

### **Сложность алгоритма по памяти:**

$O(M+N)$ ,  $M$  – длина всех символов слов шаблонов,  $N$  – длина текста, в котором осуществляется поиск.

### **Описание функций и структур данных.**

#### **Структура вершины**

```
class BorNode {
public:
    LinksMap links;
    BorNode *fail; // Предыдущее состояние для функции отката. Только для
root равно NULL.
    BorNode *term; // Ближайшее терминальное состояние. Если отсутствует -
NULL
    int out;
}
```

```
class AhoCorasick
{
public:
    typedef void (*Callback) (const char* substr);
    BorNode root;//корень
    vector<string> words;//массив паттернов
```

```
BorNode* current_state{ };//текущее состояние
Answer answer;//ответ
}
```

```
void addString(const char* const str)
```

Функция добавления символов шаблона в бор

str – шаблон для добавления в бор

```
void init()
```

Инициализация данных в AhoCorasick

```
bool step(const char c)
```

Очередной шаг алгоритма

```
void search(const char* str, Callback callback)
```

Основная функция поиска

### **Алгоритм поиска шаблона с джокером.**

#### **Задание.**

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 Р необходимо найти все вхождения Р в текст Т.

Например, образец *ab??с?* с джокером *?* встречается дважды в тексте *xabvссbababcax*.

Символ джокер не входит в алфавит, символы которого используются в Т.

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 (Т,  $1 \leq |T| \leq 100000$  )

Шаблон (Р,  $1 \leq |P| \leq 40$ )

Символ джокера

**Выход:**

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).

Номера должны выводиться в порядке возрастания.

**Пример выходных данных**

ACT

A\$

\$

**Пример выходных данных**

1

**Описание алгоритма.**

В начале работы алгоритма считывается шаблон, поиск которого будет осуществляться. Этот шаблон разделяется функцией на подшаблоны, которые были разделены друг от друга символом джокера в строке-шаблоне. Также запоминаются индексы этих подшаблонов в строке-шаблоне для дальнейшей работы алгоритма.

Далее с помощью алгоритма Ахо-Корасик подшаблоны заносятся в бор и осуществляется их поиск в строке. Когда подшаблон находится в строке поиска, то инкрементируется значение, находящееся в индексе вектора совпадений подшаблонов. Этот индекс определяется как индекс вхождения подшаблона в строку минус индекс подшаблона в строке-шаблоне.

После того, как вся строка поиска будет обработана и все подшаблоны найдены, то проверяются значения вектора вхождения подшаблонов. Если в каком-либо индексе этого вектора хранится число, равное количеству всех подшаблонов шаблона, значит строка-шаблон входит в строку поиска на

этом индексе полностью. Индекс вхождения этого шаблона запоминается и заносится в вектор результата.

### **Сложность алгоритма по операциям:**

Аналогично алгоритму Ахо-Корасик и проход по вектору совпадений подшаблонов в тексте:  $O((M+N)*\log(k)+t+N)$ ,  $M$  – длина всех символов слов шаблона,  $N$  – длина текста, в котором осуществляется поиск,  $k$  – размер алфавита,  $t$  – длина всех возможных вхождений всех строк-образцов.

### **Сложность алгоритма по памяти:**

Помимо данных, которые хранятся в алгоритме Ахо-Корасик, еще необходимо хранить массив подшаблонов, массив длин подшаблонов и массив, в котором отмечается количество входящих подшаблонов в каждый символ текста-поиска. Длина этого массива будет равна количеству символов текста-поиска:  $O(2*M+2*N+W)$ ,  $M$  – длина всех символов слов шаблона,  $N$  – длина текста, в котором осуществляется поиск,  $W$  – количество подшаблонов

## **Описание функций и структур данных.**

### **Структура вершины**

```
struct BorNode
{
    std::map<char, int> next;//потомки вершины
    std::map<char, int> go;// путь автомата
    std::vector<int> number;// массив номеров шаблонов
    int prev = 0;// индекс предка
    int deep = 0;// глубина вершины
    int suffix = -1;// индекс суффиксного перехода
    bool isLeaf = false;// является ли вершина листом
    char prevChar = 0;// символ предка
};
```

```
void addPattern(const std::string& str)
```

Функция добавления символов шаблона в бор

str – шаблон для добавления в бор

```
void search(const std::string& str)
```

Функция поиска шаблонов в строке

str – текст, в котором будет осуществляться поиск

```
void printResult(const std::string& text) const
```

Функция вывода результата работы алгоритма и строки, из которой были удалены найденные шаблоны.

text – текст, в котором осуществляется поиск шаблонов.

```
int getSuffix(int index)
```

Функция получения вершины, доступной по суффиксной ссылке.

index – индекс вершины, для которой осуществляется поиск по суффиксной ссылке.

Возвращаемым значением является индекс вершины, доступной по суффиксной ссылке, в векторе всех вершин автомата.

```
int getLink(int index, char ch)
```

Функция получения вершины, для перехода в нее.

index - индекс вершины, из которой осуществляется переход

ch – символ, по которому осуществляется переход

Возвращаемым значением является индекс вершины для перехода в векторе всех вершин автомата.



void readPattern(std::string& str)

Функция обработки считанного шаблона

str - считанный шаблон

void split(std::string str)

Функция разбиения шаблонов на подшаблоны

str – шаблон, который будет разбит на подшаблоны

### Тестирование.

#### 1.cpp

```
helloworld!  
3  
ll  
el  
word  
2 2  
3 1  
Process finished with exit code 0
```

#### 2.cpp

```
Enter string:  
helloworld!  
Enter pattern:  
#l#  
Enter joker:  
#  
Enter forbidden character:  
e  
3  
8  
  
Process finished with exit code 0
```

Входные данные	Вывод
ACTANC A\$\$\$A\$ \$ 0	1
ACCANS\$G A\$ \$	4

C	
helloworld	3 3
3	6 1
wor	9 2
ld	
llo	
IMMAGEN	1 1
2	
IMM	
MMG	
AAAAAAAAAAAAAAAAAAAA	1
\$A\$	2
O	3
	4
	5
	6
	7
	8
	9
	10
	11
	12
	13
	14

### **Выводы.**

В ходе выполнения лабораторной работы были получены навыки работы с алгоритмом Ахо-Корасик и алгоритмом поиска подстроки с джокером. Были написаны программы, реализующую эти алгоритмы работы со строками, а также добавлена реализация с запрещающим символом у джокера.

## **ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД**

### **АЛГОРИТМ АХО-КОРАСИК**

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

```
using std::string;
using std::map;
using std::vector;
using std::queue;
using std::cin;
using std::cout;
using std::endl;
using std::pair;
```

```
class BorNode;
typedef map<const char, BorNode *> LinksMap;
```

```
class BorNode {
public:
    LinksMap links;
```

```
BorNode *fail; // Предыдущее состояние для функции отката. Только для root равно NULL.  
BorNode *term; // Ближайшее терминальное состояние. Если отсутствует - NULL  
int out;
```

```
public:
```

```
explicit BorNode(BorNode *fail_node = nullptr)  
    : fail(fail_node)  
    , term(nullptr)  
    , out(-1)  
{ }
```

```
BorNode* getLink(const char c) const
```

```
{  
    auto iter = links.find(c);  
    if (iter != links.cend()) {  
        return iter->second;  
    }  
    else {  
        return nullptr;  
    }  
}
```

```
bool isTerminal() const
```

```
{  
    return (out >= 0);  
}
```

```
void showBor(){
```

```
}
```

```
};
```

```
typedef vector< pair< int, pair< int, string > > > Answer;
```

```
typedef vector< pair< int, pair< int, string > > >::iterator AnswerIterator;
```

```
bool operator < (AnswerIterator a, AnswerIterator b){
```

```
    if (a->first < b->first)
```

```
        return true;
```

```
    else if(a->first == b->first)
```

```

        return a->second.first < b->second.first;
    else return false;
}

```

```

class AhoCorasick

```

```

{

```

```

public:

```

```

    typedef void (*Callback) (const char* substr);

```

```

    BorNode root;

```

```

    vector<string> words;

```

```

    BorNode* current_state{};

```

```

    Answer answer;

```

```

public:

```

```

    void addString(const char* const str)

```

```

    {

```

```

        BorNode *current_node = &root;

```

```

        for(int i = 0; *(str + i); ++i) {

```

```

            BorNode *child_node = current_node->getLink(*(str + i));

```

```

            if (!child_node) {

```

```

                child_node = new BorNode(&root);

```

```

                current_node->links[*(str + i)] = child_node;

```

```

            }

```

```

            current_node = child_node;

```

```

        }

```

```

        current_node->out = words.size();

```

```

        words.push_back(str);

```

```

    }

```

```

void init()

```

```

{

```

```

    queue<BorNode*> q;

```

```

    q.push(&root);

```

```

    while (!q.empty()) {

```

```

        BorNode *current_node = q.front();

```

```

q.pop();
for (auto iter = current_node->links.cbegin(); iter != current_node->links.cend(); ++iter)
{
    const char symbol = iter->first;
    BorNode *child = iter->second;

    BorNode *temp_node = current_node->fail;
    while (temp_node) {
        BorNode *fail_candidate = temp_node->getLink(symbol);
        if (fail_candidate) {
            child->fail = fail_candidate;
            break;
        }
        temp_node = temp_node->fail;
    }
    if (child->fail->isTerminal()) {
        child->term = child->fail;
    }
    else {
        child->term = child->fail->term;
    }
    q.push(child);
}
}
}

```

```

bool step(const char c)
{
    while (current_state) {
        BorNode *candidate = current_state->getLink(c);
        if (candidate) {
            current_state = candidate;
            return true;
        }
        current_state = current_state->fail;
    }
    current_state = &root;
}

```

```

        return false;
    }

    void printTermsForCurrentState(Callback callback) const
    {
        if (current_state->isTerminal()) {
            callback(words[current_state->out].c_str());
        }
        BorNode *temp_node = current_state->term;
        while (temp_node) {
            callback(words[temp_node->out].c_str());
            temp_node = temp_node->term;
        }
    }

    void search(const char* str, Callback callback)
    {
        current_state = &root;
        for(int i = 0; *(str + i); i++) {
            bool flag = step(str[i]);
            if(current_state->isTerminal()) {
                answer.push_back({i - words[current_state->out].size() + 2, {current_state->out + 1,
words[current_state->out]}});
                //printTermsForCurrentState(callback);
            }
            BorNode *temp_node = current_state->term;
            while (temp_node) {
                answer.push_back({i - words[temp_node->out].size() + 2, {temp_node->out + 1,
words[temp_node->out]}});
                temp_node = temp_node->term;
            }
        }
    }
};

void print(const char* str)
{
    cout << "found substring " << str << "\n";
}

```

```

}

int main()
{
    AhoCorasick ak;

    char* search = new char[100000];
    cin >> search;
    int n;
    cin >> n;
    for(int i = 0; i < n; i++){
        char *temp = new char[100000];
        cin >> temp;
        ak.addString(temp);
    }

    ak.init();
    ak.search(search, print);
    std::sort(ak.answer.begin(), ak.answer.end());
    for(int i = 0; i < ak.answer.size(); i++){
        cout << ak.answer[i].first << " " << ak.answer[i].second.first;
        if (i != ak.answer.size() - 1){
            cout << endl;
        }
    }

    return 0;
}

```

## АЛГОРИТМ ПОИСКА ШАБЛОНОВ С МАСКАМИ С ИСПОЛЬЗОВАНИЕМ АЛГОРИТМА АХО-КОРАСИК

```

#include <iostream>
#include <string>
#include <vector>
#include <map>

```



```

struct BorNode
{
    std::map<char, int> next;//потомки вершины
    std::map<char, int> go;// путь автомата
    std::vector<int> number;// массив номеров шаблонов
    int prev = 0;// индекс предка
    int deep = 0;// глубина вершины
    int suffix = -1;// индекс суффиксного перехода
    bool isLeaf = false;// является ли вершина листом
    char prevChar = 0;// символ предка
};

```

```

class AhoCorasick// класс реализующий алгоритм Ахо-Корасик
{
private:
    std::vector<BorNode> nodes;// вектор вершин
    std::string pattern;//паттерн
    char joker;// символ джокера
    char forbidden;//запрещенный символ
    int countTermNodes;// количество терминальных вершин
    std::vector<std::string> pattenArray;// вектор подпаттернов
    int patternLen{ };// длина паттерна
    std::vector<int> matchPatterns;// вектор совпадений подпаттернов
    std::vector<int> patternsLength;// вектор длин подпаттернов образца

    void split (std::string str);// функция разделения паттернов на подпаттерны

    void addPattern (const std::string &str);// добавление символов паттерна в бор

    int getSuffix (int index); // получение вершины перехода по суффиксной ссылке

    int getLink (int index, char ch);// получить путь автомата из текущей вершины

public:
    explicit AhoCorasick (char joker, char forbidden);

    void readPattern (std::string &str);

    void search (const std::string &str);

```

```

void printResult (const std::string &text) const;

};

AhoCorasick::AhoCorasick (char joker, char forbidden) : matchPatterns(10000000) {
    BorNode root;
    root.prev = -1;
    nodes.push_back(root);
    this->joker = joker;
    this->forbidden = forbidden;
    countTermNodes = 0;
}

void AhoCorasick::readPattern (std::string &str) {
    this->pattern = str;
    patternLen = str.size();
    split(str);
    for (const auto &patern : pattenArray) { // pattern
        addPattern(pattern);
    }
}

void AhoCorasick::search (const std::string &str) {
    int curr = 0;
    for (int i = 0; i < str.size(); i++) {
        curr = getLink(curr, str[i]); // по каждому символу переходим в новую вершину бора
        for (int tmp = curr; tmp != 0; tmp = getSuffix(tmp)) { // также осуществляем
переходы по суффиксным ссылкам
            if (nodes[tmp].isLeaf) {
                for (int j = 0; j < nodes[tmp].number.size(); j++) {
                    if (i + 1 - patternsLength[nodes[tmp].number[j] - 1] -
nodes[tmp].deep >= 0 &&
                    i + 1 - patternsLength[nodes[tmp].number[j] - 1] -
nodes[tmp].deep <=
                    str.size() - patternLen) { //если паттерн не выходит за
границы (слева и справа)
                        matchPatterns[i + 1 -
patternsLength[nodes[tmp].number[j] - 1] -
nodes[tmp].deep]++; // добавляем индекс
совпадения в вектор совпадений подпаттернов
                        break;
                    }
                }
            }
        }
    }
}

```

```

    }
}
}

```

```

void AhoCorasick::printResult (const std::string &text) const {

```

```

    std::vector<bool> cutStr(text.size()); // вектор попадания символов, вошедших в паттерн
    std::string str; // входной текст без паттернов

```

```

    bool result_exist = false;

```

```

    for (int i = 0; i < matchPatterns.size(); i++) {
        if (matchPatterns[i] ==
            patternsLength.size()) { // если число вошедших подпаттернов в индексе
            совпадает с числом всех подпаттернов, то это индекс вхождения паттерна
                bool is_correct = true;
                for (int k = 0; k < i + patternLen; k++) {
                    if (pattern[k - i] && text[k] == forbidden && pattern[k - i] == joker) {
                        is_correct = false;
                        break;
                    }
                }

                if (is_correct) {
                    std::cout << i + 1 << "\n";
                    result_exist = true;
                    for (int j = 0; j < patternLen; j++)
                        cutStr[i + j] = true; // помечаем, что символ вошел в паттерн
                }
            }
        }
    }

    for (int i = 0; i < cutStr.size(); i++) {
        if (!cutStr[i])
            str.push_back(text[i]); // заполняем строку символов, которые не вошли в
            паттерн
        }
    }

    if (!result_exist) { // проверка наличия ответа
        std::cout << "\nNo result. Template forbidden!";
    }
}

```

```

void AhoCorasick::split(std::string str){// функция разделения паттернов на подпаттерны
    std::string buf;
    for (int i=0; i<str.size(); i++){
        if (str[i] == joker){
            if (!buf.empty()) {
                pattenArray.push_back(buf);           //заполняет массив
подпаттернов
                patternsLength.push_back(i - buf.size());    //и массив их вхождения
в паттерне
                buf = "";
            }
        }
        else {
            buf.push_back(str[i]);
            if (i == str.size() - 1){
                pattenArray.push_back(buf);
                patternsLength.push_back(i - buf.size() + 1);
            }
        }
    }
}

```

```

void AhoCorasick::addPattern(const std::string& str)// добавление символов паттерна в бор
{
    int current = 0;
    for (char i : str) {
        if (nodes[current].next.find(i) == nodes[current].next.end()) {// если для текущей
вершины нет перехода по символу
            BorNode ver;// вершина создается и добавляется в бор
            ver.suffix = -1;
            ver.prev = current;
            ver.prevChar = i;
            nodes.push_back(ver);
            nodes[current].next[i] = nodes.size() - 1;
        }
        current = nodes[current].next[i];
    }
    countTermNodes++;
    nodes[current].number.push_back(countTermNodes); //номера подпаттернов
    nodes[current].isLeaf = true;           // вершина объявляется терминальной
    nodes[current].deep = str.size();
}

```

```

int AhoCorasick::getSuffix(int index)// получение вершины перехода по суффиксной ссылке
{
    if (nodes[index].suffix == -1) {// если суффиксная ссылка еще не определена
        if (index == 0 || nodes[index].prev == 0) {
            nodes[index].suffix = 0;// если корень или родитель корень - то
            суффиксная ссылка ведет в корень
        }
        else {
            nodes[index].suffix = getLink(getSuffix(nodes[index].prev),
            nodes[index].prevChar);// иначе переходим ищем суффикс через суффикс родителя
        }
    }
    return nodes[index].suffix;// возвращаем индекс суффиксной вершины в векторе вершин
}

```

```

int AhoCorasick::getLink(int index, char ch)// получить путь автомата из текущей вершины
{
    if (nodes[index].go.find(ch) == nodes[index].go.end()) {// если пути по символу из текущей
    вершины нет
        if (nodes[index].next.find(ch) != nodes[index].next.end()) {
            nodes[index].go[ch] = nodes[index].next[ch];// если из вершины есть дети,
            то путь прокладывается через них
        }
        else {
            if (index == 0){
                nodes[index].go[ch] = 0;
            }
            else{
                nodes[index].go[ch] = getLink(getSuffix(index), ch);// иначе путь
                прокладывается через суффиксную ссылку
            }
        }
    }
    return nodes[index].go[ch];// возвращаем индекс вершины пути в векторе вершин
}

```

```

int main() {
    std::string str;
    std::string pattern;
    char joker;
    char forbidden;
    std::cout << "Enter string:" << std::endl;

```

```
std::cin >> str;
std::cout << "Enter pattern:" << std::endl;
std::cin >> pattern;
std::cout << "Enter joker:" << std::endl;
std::cin >> joker;
std::cout << "Enter forbidden character:" << std::endl;
std::cin >> forbidden;

auto* ahoCorasick = new AhoCorasick(joker, forbidden);
ahoCorasick->readPattern(pattern);
ahoCorasick->search(str);
ahoCorasick->printResult(str);

return 0;
}
```