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-Корасика

Студент гр. 7304 _____ Пэтайчук Н.Г.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2019

Цель работы

Исследование алгоритмов поиска всех вхождений нескольких подстрок в строку на примере алгоритма Ахо-Корасика.

Постановка задачи

- Разработайте программу, решающую задачу точного поиска набора образцов.
  - Входные данные: текст T, затем число подстрок N, далее N строк, которые нужно найти в T;
  - Выходные данные: 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;
- 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 P в текст T.
  - Входные данные: текст (T, $1 \leq |T| \leq 100000$), далее шаблон (P, $1 \leq |P| \leq 40$), после чего символ джокера;
  - Выходные данные: Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер). Номера должны выводиться в порядке возрастания;

Ход работы

1. Объявление структуры элемента бора (структуры данных, используемой в алгоритме Ахо-Корасика, представляющее из себя дерево, где за каждым ребром закреплён символ перехода по нему), включающая в себя индексы следующих вершин, суффиксальную ссылку, переходы по автомату, сооружённому из бора, а также индекс родительской вершины, символ, по которому перешли в данную вершину, номер шаблона и флаг того, является ли данный символ концом какого-либо шаблона. Также была реализована структура элемента результирующего массива, хранящая индекс вхождения и номер шаблона;
2. Реализация функций работы с бором, а именно создание элемента бора, инициализация бора, а также процедура добавления шаблона в бор;
3. Реализация функций, превращающих бор в автомат, а именно функций получения суффиксальной ссылки и автоматного перехода, рекурсивно вызывающих друг друга, и функции получения хорошей суффиксальной ссылки, которые необходимы для быстрой работы алгоритма;
4. Реализация двух функций, отвечающих за работу алгоритма Ахо-Корасика, первая из которых проходит по строке как по построенному из бора конечному автомату, а другая проверяет, не содержатся ли в текущей пройденной подстроке какие-нибудь шаблоны;
5. Реализация головной функции, где происходит ввод исходных данных, деление шаблона на подшаблоны (если говорить о второй задаче) и вывод результатов работы алгоритма;

Весь исходный код программы представлен в Приложении 1.

Тестирование программы

1) Демонстрация поиска нескольких шаблонов:

```
AGGAGNTCCATCGGTTTCAGCTNN
5
GG
CAT
TT
TNT
A
1 5
2 1
4 5
10 5
9 2
13 1
15 3
16 3
19 5
```

2) Демонстрация поиска шаблона с джокерами:

```
ACCNGGGNGAANNTTGANTC
N%%%
%
4
8
12
13
```

Вывод

В ходе данной лабораторной работы был исследован алгоритм Ахо-Корасика поиска всех вхождений нескольких подстрок в строку. Были получены знания о том, что такое бор, как его преобразовать в конечный автомат, что такое суффиксальная ссылка и какие суффиксальные ссылки называются хорошими. Помимо этого, с помощью алгоритма Ахо-Корасика была решена задача нечёткого поиска шаблона в строке, иначе говоря поиска в исходном тексте подстроки, содержащей джокеры.

Приложение 1: Исходный код программы

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

using namespace std;

map<char, char> Alphabet
{
    {'A', 0},
    {'C', 1},
    {'G', 2},
    {'N', 3},
    {'T', 4}
};

struct Bohr_Vertex
{
    int next_vertex[5];
    int machine_transition[5];
    int parent;
    int pattern_number[40];
    int suffix_link;
    int suffix_good_link;
    bool is_already_pattern;
    char symbol;
};
struct Pattern_Entry
{
    unsigned long long int index;
    unsigned int pattern_number;
};
typedef Bohr_Vertex bohr_vertex;
typedef vector<bohr_vertex> Bohr;
typedef vector<Pattern_Entry> pattern_entry_table;

Bohr bohr;

bohr_vertex create_BohrVertex(int parent, char symbol)
{
    bohr_vertex vertex;
    memset(vertex.next_vertex, 255, sizeof(vertex.next_vertex));
    memset(vertex.machine_transition, 255,
sizeof(vertex.machine_transition));
    memset(vertex.pattern_number, 255,
sizeof(vertex.pattern_number));
    vertex.is_already_pattern = false;
    vertex.suffix_link = -1;
```

```
    vertex.suffix_good_link = -1;
    vertex.parent = parent;
    vertex.symbol = symbol;
    return vertex;
}

void initialize_Bohr(Bohr &bohr)
{
    bohr.push_back(create_BohrVertex(-1, -1));
}

void add_StringToBohr(Bohr &bohr, const string &str,
vector<string> &patterns)
{
    int number = 0;
    for (char symbol : str)
    {
        char alphabet_index = Alphabet[symbol];
        if (bohr[number].next_vertex[alphabet_index] == -1)
        {
            bohr.push_back(create_BohrVertex(number,
alphabet_index));
            bohr[number].next_vertex[alphabet_index] = bohr.size()
- 1;
        }
        number = bohr[number].next_vertex[alphabet_index];
    }
    bohr[number].is_already_pattern = true;
    patterns.push_back(str);
    for (int i = 0; i < 40; i++)
        if (bohr[number].pattern_number[i] == -1)
        {
            bohr[number].pattern_number[i] = patterns.size() - 1;
            break;
        }
}

int get_MachineTransition(Bohr &bohr, int vertex_index, char
alphabet_index);

int get_SuffixLink(Bohr &bohr, int vertex_index)
{
    if (bohr[vertex_index].suffix_link == -1)
    {
        if (vertex_index == 0 || bohr[vertex_index].parent == 0)
            bohr[vertex_index].suffix_link = 0;
        else
            bohr[vertex_index].suffix_link =
get_MachineTransition(bohr,

get_SuffixLink(bohr,

bohr[vertex_index].parent),
```

```
bohr[vertex_index].symbol);
    }
    return bohr[vertex_index].suffix_link;
}

int get_MachineTransition(Bohr &bohr, int vertex_index, char
alphabet_index)
{
    if (bohr[vertex_index].machine_transition[alphabet_index] == -
1)
    {
        if (bohr[vertex_index].next_vertex[alphabet_index] != -1)
            bohr[vertex_index].machine_transition[alphabet_index]
= bohr[vertex_index].next_vertex[alphabet_index];
        else
        {
            if (vertex_index == 0)

bohr[vertex_index].machine_transition[alphabet_index] = 0;
            else

bohr[vertex_index].machine_transition[alphabet_index] =
get_MachineTransition(bohr,

get_SuffixLink(bohr,

vertex_index),

alphabet_index);
        }
    }
    return bohr[vertex_index].machine_transition[alphabet_index];
}

int get_SuffixGoodLink(Bohr &bohr, int vertex_index){
    if (bohr[vertex_index].suffix_good_link == -1)
    {
        int next_vertex = get_SuffixLink(bohr, vertex_index);
        if (next_vertex == 0)
            bohr[vertex_index].suffix_good_link = 0;
        else
            bohr[vertex_index].suffix_good_link =
(bohr[next_vertex].is_already_pattern) ?
                        next_vertex : get_SuffixGoodLink(bohr,
next_vertex);
    }
    return bohr[vertex_index].suffix_good_link;
}

void check_IsPattern(Bohr &bohr, vector<string> &patterns,
pattern_entry_table &result_table,
                     int vertex_index, int symbol_index)
```

```
{
    Pattern_Entry pattern_table_element;
    for (int current_index = vertex_index; current_index != 0;
current_index = get_SuffixGoodLink(bohr,

current_index))
        if (bohr[current_index].is_already_pattern)
        {
            for (int j = 0; j < 40; j++)
            {
                if (bohr[current_index].pattern_number[j] != -1)
                {
                    pattern_table_element.index = symbol_index -
patterns[bohr[current_index].pattern_number[j]].length();
                    pattern_table_element.pattern_number =
bohr[current_index].pattern_number[j];
                    result_table.push_back(pattern_table_element);
                }
                else
                    break;
            }
        }
}

void findAllSubstrs_AchoCorasick(Bohr &bohr, const string &text,
vector<string> &patterns,
                                 pattern_entry_table &result_table)
{
    int now_vertex = 0;
    for (int i = 0; i < text.size(); i++)
    {
        now_vertex = get_MachineTransition(bohr, now_vertex,
Alphabet[text[i]]);
        check_IsPattern(bohr, patterns, result_table, now_vertex,
i + 1);
    }
}

int main()
{
    Bohr bohr;
    pattern_entry_table result_table;
    vector<string> patterns;
    vector<string> pattern_part_list;
    vector<int> patterns_position;
    vector<int> pattern_parts_counter;
    string text, pattern_string;
    char joker;

    cin >> text >> pattern_string >> joker;
    initialize_Bohr(bohr);
    for (int i = 0; i < pattern_string.size(); i++)
    {
```

```
        string pattern_part;
        if (pattern_string[i] != joker)
        {
            patterns_position.push_back(i + 1);
            for (int j = i; pattern_string[j] != joker && j !=
pattern_string.size(); j++, i++)
                pattern_part += pattern_string[j];
            pattern_part_list.push_back(pattern_part);
        }
    }
    for (const string pattern_part : pattern_part_list)
        add_StringToBohr(bohr, pattern_part, patterns);
    findAllSubstrs_AchoCorasick(bohr, text, patterns,
result_table);

    pattern_parts_counter = vector<int>(text.size(), 0);
    for (int i = 0; i < result_table.size(); i++)
    {
        if(result_table[i].index <
patterns_position[result_table[i].pattern_number] - 1)
            continue;
        int counter_index = result_table[i].index -
patterns_position[result_table[i].pattern_number] + 1;
        pattern_parts_counter[counter_index]++;
        if (pattern_parts_counter[counter_index] ==
pattern_part_list.size() &&
                counter_index <= text.length() -
pattern_string.length())
            cout << counter_index + 1 << endl;
    }
    return 0;
}
```