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 _____ Есиков О.И.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2019

Цель работы.

Изучить и реализовать на языке программирования c++ алгоритм Ахо–Корасик, который осуществляет поиск множества подстрок в тексте с помощью построения бора.

Формулировка задания.

1) Разработать 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 $P = \{p_1, \dots, p_n\}$ $1 \leq |pi| \leq 75$

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.

2) Используя реализацию точного множественного поиска, решить 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.

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, $1 \leq |T| \leq 100000$)

Шаблон (P, $1 \leq |P| \leq 40$)

Символ джокера

Выход:

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

Номера должны выводиться в порядке возрастания.

Теоретические сведения.

Бор (англ. trie, луч, нагруженное дерево) – структура данных для хранения набора строк, представляющая из себя подвешенное дерево с символами на рёбрах. Строки получаются последовательной записью всех символов, хранящихся на рёбрах между корнем бора и терминальной вершиной. Размер бора линейно зависит от суммы длин всех строк, а поиск в бору занимает время, пропорциональное длине образца.

Детерминированный конечный автомат (ДКА) (англ. deterministic finite automaton (DFA)) — набор из пяти элементов $\langle \Sigma, Q, s \in Q, T \subset Q, \delta: Q \times \Sigma \rightarrow Q \rangle$, где Σ – алфавит (англ. alphabet), Q – множество состояний (англ. finite set of states), s – начальное (стартовое) состояние (англ. start state), T – множество допускающих состояний (англ. set of accept states), δ – функция переходов (англ. transition function).

Алгоритм.

1) Ахо–Корасик.

Шаг 1: выбирается образец, если образцов не осталось, то переход на шаг 3.

Шаг 2: выбранный образец добавляется в бор, переход на шаг 1.

Шаг 3: текущий символ = первый символ текста; текущая вершина = корень.
Шаг 4: переход из текущей вершины по текущему символу с помощью перехода по автомату.
Шаг 5: проверка на встретившееся шаблоны в вершине, выбранной на шаге 4, с помощью перехода по хорошим суффиксальным ссылкам.
Шаг 6: если текст не закончился, то текущая вершина = вершина, выбранная на шаге 4, текущий символ = следующий символ текста, переход на шаг 4.

2) Образец с джокером.

Шаг 1: образец с джокером делится на подстроки, которые не содержат джокеры.
Шаг 2: запоминаются позиции начала этих подстрок в шаблоне с джокером.
Шаг 3: выполняется алгоритм Ахо–Корасик, когда обнаруживается подстрока, то в массиве C элемент $C[j - l_i]$ увеличивается на 1, где j – позиция в тексте, l_i – позиция начала подстроки в шаблоне с джокером.
Шаг 4: каждое i, для которого C[i] = k, является стартовой позицией появления шаблона с джокером в тексте, где k – количество подстрок без джокеров.

Пример работы программы.

1) Ахо–Корасик.

Входные данные:

CCCA
1
CC

Выходные данные:

1 1
2 1

2) Образец с джокером.

Входные данные:

ACT
A$

$

Выходные данные:

1

Выводы.

В ходе выполнения данной лабораторной работы были изучен и реализован на языке программирования c++ алгоритм Ахо–Корасик, результатом работы которого являются индексы вхождений подстроки в тексте и номер этой подстроки. Для работы этого алгоритма понадобилась реализовать бор, содержащий все подстроки, которые необходимо найти. Бор был реализован в качестве конечного детерминированного автомата, для прохода по которому используются суффиксальные ссылки. Вычислительная сложность данного алгоритма O(nσ + H + k), где n – общая длина всех подстрок, σ – размер алфавита, H – длина текста, k – общая длина всех совпадений.

Приложение А

Исходный код.

Ахо–Корасик.

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

using namespace std;

struct bohr_vertex
{
    int next_vertex[5]; //{A, C, G, T, N} алфавит
    bool flag;          //является ли концом подстроки
    int num;            //номер подстроки
    int parent;         //индекс родителя
    int sufflink;       //индекс перехода по суффиксальной ссылке
    int gotosymbol[5];  //индекс перехода по каждому символу
    int symboltoparent; //возвращает индекс символа, по которому переходится из
родителя
    int goodsufflink;   //индекс перехода по хорошей суффиксальной ссылке
};

class Bohr
{
private:
    vector<bohr_vertex> bohr;
    string text;


public:
    Bohr()
    {
        bohr.push_back({{-1, -1, -1, -1, -1}, false, 0, 0, -1, {-1, -1, -1, -1,
-1}, -1, -1});
        cin >> text;
        int N;
        cin >> N;
        for(int i(0); i < N; i++)
        {
            string temp;
            cin >> temp;
            push(temp, i+1);
        }
    }

    void push(string str, int number)
    {
        int len = str.size();
        int index = 0;
        int symbol;
        for(int i(0); i < len; i++)
        {
            switch(str.at(i))
            {
                case 'A':
                            symbol = 0;
                            break;
                case 'C':
                            symbol = 1;
                            break;
```

```
                case 'G':
                            symbol = 2;
                            break;
                case 'T':
                            symbol = 3;
                            break;
                case 'N':
                            symbol = 4;
                            break;
            }
            if(bohr[index].next_vertex[symbol] == -1)
            {
                bool isEnd = false;
                if(i == len-1)
                    isEnd = true;
                bohr.push_back({{-1, -1, -1, -1, -1}, isEnd, number, index, -1,
{-1, -1, -1, -1, -1}, symbol, -1});
                bohr[index].next_vertex[symbol] = bohr.size() - 1;
            }
            index = bohr[index].next_vertex[symbol];
        }
    }

    int getSuffLink(int vertex)
    {
        if(bohr.at(vertex).sufflink == -1)
        {
            if(vertex == 0 || bohr.at(vertex).parent == 0)
                bohr.at(vertex).sufflink = 0;
            else
                bohr.at(vertex).sufflink =
getLink(getSuffLink(bohr.at(vertex).parent), bohr.at(vertex).symboltoparent);
        }
        return bohr.at(vertex).sufflink;
    }

    int getLink(int vertex, int symbol)
    {
        if(bohr.at(vertex).gotosymbol[symbol] == -1)
        {
            if(bohr.at(vertex).next_vertex[symbol] != -1)
                bohr.at(vertex).gotosymbol[symbol] =
bohr.at(vertex).next_vertex[symbol];
            else
                bohr.at(vertex).gotosymbol[symbol] = (vertex == 0) ? 0 :
getLink(getSuffLink(vertex), symbol);
        }
        return bohr.at(vertex).gotosymbol[symbol];
    }

    int getGoodSuffLink(int vertex)
    {
        if(bohr.at(vertex).goodsufflink == -1)
        {
            int temp = getSuffLink(vertex);
            if(temp == 0)
                bohr.at(vertex).goodsufflink = 0;
            else
                bohr.at(vertex).goodsufflink = (bohr.at(temp).flag) ? temp :
getGoodSuffLink(temp);
        }
        return bohr.at(vertex).goodsufflink;
    }
```

```
    void check(int v, int i)
    {
        for(int u(v); u != 0; u = getGoodSuffLink(u))
        {
            if(bohr.at(u).flag)
            {
                int delta = 0;
                int temp = u;
                while(bohr.at(temp).parent != 0)
                {
                    temp = bohr.at(temp).parent;
                    delta++;
                }
                cout << i - delta << " " << bohr.at(u).num << endl;
            }
        }
    }

    void AHO()
    {
        int vertex = 0, symbol = 0;
        for(int i(0); i < text.length(); i++)
        {
            switch(text.at(i))
            {
                case 'A':
                            symbol = 0;
                            break;
                case 'C':
                            symbol = 1;
                            break;
                case 'G':
                            symbol = 2;
                            break;
                case 'T':
                            symbol = 3;
                            break;
                case 'N':
                            symbol = 4;
                            break;
            }
            vertex = getLink(vertex, symbol);
            check(vertex, i + 1);
        }
    }
};

int main()
{
    Bohr object;
    object.AHO();
    return 0;
}
```

Образец с джокерами.

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

using namespace std;

const string sym = "ACGTN";

struct numbers
{
    long long int index;
    int pattern_num;
};

struct bohr_vrtx
{
    int next_vrtx[5];
    int pattern_num[40];
    int suff_link;
    int auto_move[5];
    int par;
    int suff_flink;
    bool flag;
    char symb;
};

class Bohr
{
private:
    vector <bohr_vrtx> bohr;
    string text;
    string pattern_str;
    vector <string> patterns;
    vector <int> patterns_pos;
    vector<numbers> num;

public:
    Bohr()
    {
        bohr.push_back(make_bohr_vrtx(-1,-1));
        char joker;
        cin >> text >> pattern_str >> joker;
        for(int i(0); i < pattern_str.size(); i++)
        {
            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);
                add_string_to_bohr(pat);
            }
        }
    }
```

```
    bohr_vrtx make_bohr_vrtx(int p, char c)
    {
        bohr_vrtx v;
        memset(v.next_vrtx, 255, sizeof(v.next_vrtx));
        memset(v.auto_move, 255, sizeof(v.auto_move));
        memset(v.pattern_num, 255, sizeof(v.pattern_num));
        v.flag = false;
        v.suff_link = -1;
        v.par = p;
        v.symb = c;
        v.suff_flink = -1;
        return v;
    }

    void add_string_to_bohr(string& s)
    {
        int num = 0;
        for(int i(0); i < s.size(); i++)
        {
            char ch = sym.find(s[i]);
            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;
        for(int i(0); i < 40; i++)
        {
            if(bohr[num].pattern_num[i] == -1)
            {
                bohr[num].pattern_num[i] = patterns.size() - 1;
                break;
            }
        }
    }

    int get_suff_link(int v)
    {
        if(bohr[v].suff_link == -1)
            if(v == 0 || bohr[v].par == 0)
                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)
                bohr[v].suff_flink = 0;
            else
                bohr[v].suff_flink = (bohr[u].flag) ? u : get_suff_flink(u);
        }
        return bohr[v].suff_flink;
    }

    void check(int v, int i)
    {
        struct numbers s;
        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].pattern_num[j] != -1)
                    {
                        s.index = i - patterns[bohr[u].pattern_num[j]].size();
                        s.pattern_num = bohr[u].pattern_num[j];
                        num.push_back(s);
                    }
                    else
                        break;
                }
            }
        }
    }

    void find_all_pos()
    {
        int u = 0;
        for(int i(0); i < text.size(); i++)
        {
            u = get_auto_move(u, sym.find(text[i]));
            check(u, i + 1);
        }
    }

    void out()
    {
        vector <int> c(text.size(), 0);
        for(int i(0); i < num.size(); i++)
        {
            if(num[i].index < patterns_pos[num[i].pattern_num] - 1)
                continue;
            c[num[i].index - patterns_pos[num[i].pattern_num] + 1]++;
            if(c[num[i].index - patterns_pos[num[i].pattern_num] + 1] ==
patterns.size() &&
               num[i].index - patterns_pos[num[i].pattern_num] + 1 <=
text.size() - pattern_str.size())
                cout << num[i].index - patterns_pos[num[i].pattern_num] + 2 <<
endl;
        }
    }
};

int main()
```

```
{
    Bohr obj;
    obj.find_all_pos();
    obj.out();
    return 0;
}
```